Reversible Computation

William Troiani

May 3, 2026

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 transferred from a cooler body to a warmer body, contradicting 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 inherently linked with physical irreversibility [1]. There, it is argued that practical computation fundamentally involves irreversible computation. Bennett proved that this pessimistic conclusion is not forced by the mathematics of computation [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 Theorem 0.3.3 for the precise statement.

What, though, makes a Turing machine M *reversible*? More is required than the bare recoverability of the input from the output. We want a reverse machine whose computation retraces the original computation. Because our convention is that a step first rewrites and then moves, the reverse machine may need a few administrative steps to move the head to the correct square before a previous tape update is undone. The definition below makes that convention explicit.

0.1 One-tape machines

The set of finite words over an alphabet Σ is denoted Σ^*. Throughout, $\sqcup \in \Sigma$ denotes the blank symbol, and input and output words are taken in $(\Sigma \setminus \{\sqcup\})^*$.

Notation 0.1.1. Let

$$\mathrm{Tape}(\Sigma) = \{t : \mathbb{Z} \to \Sigma \mid t(k) = \sqcup \text{ for all but finitely many } k\}.$$

Elements of $\mathrm{Tape}(\Sigma)$ are called **tapes**. If $w = w_0 \cdots w_{m-1} \in (\Sigma \setminus \{\sqcup\})^*$, define the tape $\widetilde{w} \in \mathrm{Tape}(\Sigma)$ by

$$\widetilde{w}(k) = \begin{cases} w_k, & 0 \le k < m, \\ \sqcup, & \text{otherwise.} \end{cases}$$

A tape t **represents** the word w if $t = \widetilde{w}$.

For $d \in \{\texttt{Left}, \texttt{Stay}, \texttt{Right}\}$, define

$$\Delta(d) = \begin{cases} -1, & d = \texttt{Left}, \\ 0, & d = \texttt{Stay}, \\ 1, & d = \texttt{Right}. \end{cases}$$

Also define the opposite direction by

$$\mathrm{op}(\texttt{Left}) = \texttt{Right}, \qquad \mathrm{op}(\texttt{Right}) = \texttt{Left}, \qquad \mathrm{op}(\texttt{Stay}) = \texttt{Stay}.$$

Definition 0.1.2. A **Turing machine** is a tuple

$$M = (\Sigma, Q, q_{\mathrm{in}}, q_{\mathrm{out}}, \delta)$$

consisting of the following data.

- A finite alphabet Σ containing a distinguished blank symbol $\sqcup$.
- A finite set Q of states, containing distinguished states q_{in} and q_{out}.
- A partial transition function

$$\delta : \Sigma \times (Q \setminus \{q_{\mathrm{out}}\}) \rightharpoonup \Sigma \times Q \times \{\texttt{Left}, \texttt{Stay}, \texttt{Right}\}.$$

The transition function is required to satisfy the following final-state convention: if

$$\delta(\sigma, q) = (\tau, q_{\mathrm{out}}, d),$$

then $d = \texttt{Stay}$. Thus every transition into the output state leaves the head fixed, and there are no transitions out of q_{out}.

When the distinguished states are called $q_{\texttt{Start}}$ and $q_{\texttt{Finish}}$, we write

$$M = (\Sigma, Q, q_{\texttt{Start}}, q_{\texttt{Finish}}, \delta).$$

Definition 0.1.3. A **configuration** of $M = (\Sigma, Q, q_{\mathrm{in}}, q_{\mathrm{out}}, \delta)$ is a triple

$$(t, q, p) \in \mathrm{Tape}(\Sigma) \times Q \times \mathbb{Z}.$$

The integer p is the head position. If $\delta(t(p), q)$ is defined and

$$\delta(t(p), q) = (\tau, q', d),$$

then the successor configuration is

$$(t', q', p + \Delta(d)),$$

where

$$t'(p) = \tau, \qquad t'(k) = t(k) \quad \text{for all } k \neq p.$$

We write

$$(t, q, p) \rightsquigarrow_M (t', q', p + \Delta(d))$$

for this one-step successor relation.

Definition 0.1.4. A **computation** of $M = (\Sigma, Q, q_{\mathrm{in}}, q_{\mathrm{out}}, \delta)$ on input $w \in (\Sigma \setminus \{\sqcup\})^*$ is a finite or infinite sequence of configurations

$$(t_0, q_0, p_0) \rightsquigarrow_M (t_1, q_1, p_1) \rightsquigarrow_M \cdots$$

with

$$(t_0, q_0, p_0) = (\widetilde{w}, q_{\mathrm{in}}, 0).$$

The computation **halts** if it is finite and has final configuration

$$(t_n, q_{\mathrm{out}}, 0).$$

If, in addition, $t_n = \widetilde{u}$ for some $u \in (\Sigma \setminus \{\sqcup\})^*$, then M **halts on** w **with output** u, and we write $M(w) = u$.

Remark 0.1.5. The hypotheses that a halting computation finishes with head position 0, and that every transition into the output state uses `Stay`, are normalization conventions. Lemma 0.1.11 records why they do not reduce the class of computable partial functions.

Definition 0.1.6. Two configurations (t, q, p) and (t', q', p') are **head-equivalent**, written

$$(t, q, p) \sim (t', q', p'),$$

if $t = t'$ and $q = q'$. Thus $\sim$ forgets only the head position.

Definition 0.1.7. A **run** of M from an arbitrary configuration C is any finite or infinite sequence

$$C = C_0 \rightsquigarrow_M C_1 \rightsquigarrow_M C_2 \rightsquigarrow_M \cdots$$

following the successor relation of Definition 0.1.3. Computations on inputs are the special runs whose first configuration is $(\widetilde{w}, q_{\text{in}}, 0)$.

Definition 0.1.8. A Turing machine $M = (\Sigma, Q, q_{\text{in}}, q_{\text{out}}, \delta)$ is **reversible** if there exists a Turing machine

$$M^\dagger = (\Sigma, Q^\dagger, q_{\text{out}}, q_{\text{in}}, \delta^\dagger)$$

with the same tape alphabet, with $Q \subseteq Q^\dagger$, and with input and output states swapped, such that the following condition holds.

For every halting computation of M

$$C_0 \rightsquigarrow_M C_1 \rightsquigarrow_M \cdots \rightsquigarrow_M C_n,$$

where $C_0 = (\widetilde{w}, q_{\text{in}}, 0)$ and $C_n = (t_n, q_{\text{out}}, 0)$, the run of $M^\dagger$ beginning at C_n is finite and contains the configurations of the forward computation in reverse order, up to head-equivalence. More precisely, if

$$D_0 \rightsquigarrow_{M^\dagger} D_1 \rightsquigarrow_{M^\dagger} \cdots \rightsquigarrow_{M^\dagger} D_m$$

is the run of $M^\dagger$ beginning at C_n, then there are indices

$$0 = r_0 < r_1 < \cdots < r_n = m$$

such that

$$D_{r_i} \sim C_{n-i} \qquad \text{for all } 0 \le i \le n.$$

The configurations between two successive checkpoints are administrative configurations used, for example, to move the head before undoing a previous rewrite.

Example 0.1.9. Consider the machine which scans the input and returns to the start. Its alphabet is

$$\Sigma = \{0, 1, \dot{0}, \dot{1}, \sqcup\},$$

where dotted symbols mark the first input square. Its states are

$$Q = \{q_{\texttt{Start}}, q_{\texttt{MoveRight}}, q_{\texttt{MoveLeft}}, q_{\texttt{Finish}}\}.$$

For $\sigma \in \{0, 1\}$, the relevant transitions are

$$\begin{aligned}
(\sigma, q_{\texttt{Start}}) &\longmapsto (\dot{\sigma}, q_{\texttt{MoveRight}}, \texttt{Right}),\\
(\sigma, q_{\texttt{MoveRight}}) &\longmapsto (\sigma, q_{\texttt{MoveRight}}, \texttt{Right}),\\
(\sqcup, q_{\texttt{MoveRight}}) &\longmapsto (\sqcup, q_{\texttt{MoveLeft}}, \texttt{Left}),\\
(\sigma, q_{\texttt{MoveLeft}}) &\longmapsto (\sigma, q_{\texttt{MoveLeft}}, \texttt{Left}),\\
(\dot{\sigma}, q_{\texttt{MoveLeft}}) &\longmapsto (\sigma, q_{\texttt{Finish}}, \texttt{Stay}),\\
(\sqcup, q_{\texttt{Start}}) &\longmapsto (\sqcup, q_{\texttt{Finish}}, \texttt{Stay}).
\end{aligned}$$

Unspecified transitions are undefined. This computes the identity function on binary words and returns the head to position 0. It is not literally its own reverse: a reverse machine starts in the state $q_{\texttt{Finish}}$ and retraces the displayed trajectory back to $q_{\texttt{Start}}$.

Example 0.1.10. The transition function of a reversible machine need not be injective on all of $\Sigma \times Q$. We only require reversibility along actual halting computations. For example, consider the machine which moves to the end of the input, appends ten zeros, returns the head to position 0, and halts. Let

$$\Sigma = \{0, 1, \sqcup\}$$

and

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

For $\sigma \in \{0, 1\}$, define the reachable transitions by

$$\begin{aligned}
(\sigma, q_{\texttt{Start}}) &\longmapsto (\sigma, q_{\texttt{Move}}, \texttt{Right}), & (\sigma, q_{\texttt{Move}}) &\longmapsto (\sigma, q_{\texttt{Move}}, \texttt{Right}),\\
(\sqcup, q_{\texttt{Start}}) &\longmapsto (0, q_2, \texttt{Right}), & (\sqcup, q_{\texttt{Move}}) &\longmapsto (0, q_2, \texttt{Right}),\\
(\sqcup, q_i) &\longmapsto (0, q_{i+1}, \texttt{Right}) \quad (2 \le i \le 9), & (\sqcup, q_{10}) &\longmapsto (0, q_{\texttt{Done}}, \texttt{Left}),\\
(\sigma, q_{\texttt{Done}}) &\longmapsto (\sigma, q_{\texttt{Done}}, \texttt{Left}), & (\sqcup, q_{\texttt{Done}}) &\longmapsto (\sqcup, q_{\texttt{Reset}}, \texttt{Right}),\\
(\sigma, q_{\texttt{Reset}}) &\longmapsto (\sigma, q_{\texttt{Finish}}, \texttt{Stay}).
\end{aligned}$$

Add the unreachable dummy transitions

$$(\rho, q_{\texttt{Dummy}}) \longmapsto (\sqcup, q_{\texttt{Dummy}}, \texttt{Stay}) \qquad (\rho \in \Sigma).$$

These dummy transitions make δ non-injective, but they are never encountered in a computation beginning at $q_{\texttt{Start}}$. They therefore do not affect the reversible behavior of the reachable halting computations.

Lemma 0.1.11 (Normalization)**.** *Every ordinary one-tape Turing machine can be converted into a Turing machine satisfying Definition 0.1.2, computing the same partial function under the output convention of Definition 0.1.4.*

Proof. We recall the standard construction. Add a fresh marker for the initial square, simulate the original machine, and, when the simulated machine halts, enter a cleanup phase. The cleanup phase moves the intended output into the block beginning at the marked origin, erases temporary workspace and the marker, returns the head to position 0, and enters the output state by a `Stay` transition. This adds only finitely many tape symbols and states and does not change the computed partial function. □

0.2 Multi-tape machines and one-tape simulation

Definition 0.2.1. An **n-tape Turing machine** is a tuple

$$M = (\Sigma, Q, q_{\text{in}}, q_{\text{out}}, \delta),$$

where Σ, Q, q_{in}, and q_{out} are as before, and the transition function is partial:

$$\delta : \Sigma^n \times (Q \setminus \{q_{\text{out}}\}) \rightharpoonup \Sigma^n \times Q \times \{\texttt{Left}, \texttt{Stay}, \texttt{Right}\}^n.$$

If

$$\delta((\sigma_1, \ldots, \sigma_n), q) = ((\tau_1, \ldots, \tau_n), q_{\text{out}}, (d_1, \ldots, d_n)),$$

then $d_1 = \cdots = d_n = \texttt{Stay}$.

A configuration is a triple

$$(\mathbf{t}, q, \mathbf{p}) = \big((t^1, \ldots, t^n), q, (p^1, \ldots, p^n)\big) \in \text{Tape}(\Sigma)^n \times Q \times \mathbb{Z}^n.$$

If

$$\delta\big((t^1(p^1), \ldots, t^n(p^n)), q\big) = \big((\tau_1, \ldots, \tau_n), q', (d_1, \ldots, d_n)\big),$$

then the successor configuration is

$$(\mathbf{s}, q', (p^1 + \Delta(d_1), \dots, p^n + \Delta(d_n))),$$

where, for every j,

$$s^j(p^j) = \tau_j, \qquad s^j(k) = t^j(k) \quad \text{for all } k \neq p^j.$$

The computation on input $(w_1, \dots, w_n)$ starts at

$$((\widetilde{w}_1, \dots, \widetilde{w}_n), q_{\text{in}}, (0, \dots, 0)).$$

It halts only in state q_{out} with all heads at position 0. If the final tapes are $(\widetilde{o}_1, \dots, \widetilde{o}_n)$, then the output is $(o_1, \dots, o_n)$. Reversibility is defined by the evident n-tape analogue of Definition 0.1.8, with head-equivalence forgetting the tuple of head positions.

Lemma 0.2.2. *Let M be an n-tape Turing machine. There is a one-tape Turing machine $S(M)$ with the following property. If*

$$M(w_1, \dots, w_n) = (o_1, \dots, o_n),$$

then

$$S(M)(w_1 \# \cdots \# w_n) = o_1 \# \cdots \# o_n.$$

Moreover, if M is in normal form, then $S(M)$ may be chosen in normal form.

Proof. We describe the encoding and one simulated step. The one-tape machine first converts the delimiter input $w_1 \# \cdots \# w_n$ into a zipped encoding of the n tapes. The working alphabet for this encoded part is

$$\Gamma = (\Sigma \times \{\mathtt{Y}, \mathtt{N}\})^n.$$

A column

$$((\sigma_1, \epsilon_1), \dots, (\sigma_n, \epsilon_n)) \in \Gamma$$

represents the symbols $\sigma_1, \dots, \sigma_n$ appearing at the same integer position on the n tapes. The marker $\epsilon_j = \mathtt{Y}$ means that the head of tape j is currently at this column. Several components of the same column may be marked `Y`, since several simulated heads may occupy the same integer position.

Only finitely many columns are written. The simulator maintains one all-blank guard column on each side of the represented interval. If a simulated head moves into a guard column, the represented interval is extended by creating a fresh guard column beyond it. This handles moves beyond the current nonblank region.

The finite control stores the current simulated state q, a vector

$$(a_1, \dots, a_n) \in (\Sigma \coprod \{\mathtt{?}\})^n,$$

and a phase marker. A read phase sweeps across the encoded tape. Whenever it encounters a component $(\sigma_j, \mathtt{Y})$, it replaces the corresponding question mark `?` in the state by σ_j. Once all entries have been found, the state contains

$$((a_1, \dots, a_n), q).$$

The simulator then computes

$$\delta((a_1, \dots, a_n), q) = ((b_1, \dots, b_n), q', (d_1, \dots, d_n))$$

in its finite control. Notice that the current simulated state is q, not necessarily $q_{\mathtt{Start}}$.

During the update phase, the simulator sweeps back across the encoded tape. For each component marked `Y`, it rewrites a_j as b_j. If $d_j = \mathtt{Stay}$, the marker `Y` remains on that component. If $d_j = \mathtt{Left}$ or $d_j = \mathtt{Right}$, the marker is removed from the old component and placed in the corresponding component of the neighboring column. If multiple heads move to the same neighboring column, all corresponding components are marked.

The finite control records which pending head moves have already been performed. After all pending moves are complete, the finite control is reset to

$$((?, \dots, ?), q'),$$

and the next read phase begins. If $q' = q_{\text{out}}$, the simulator enters its own output phase.

Finally, the simulator decodes the zipped representation back into delimiter form

$$o_1 \# \cdots \# o_n.$$

The initial delimiter-to-zipped conversion and the final zipped-to-delimiter conversion are fixed-format tape rearrangements. Since n is fixed and the delimiters determine the components, these conversions can be implemented by a finite sequence of scans, markings, and shifts. They are injective on the well-formed encodings used here, and they preserve the represented tuple of tapes. □

Lemma 0.2.3. *If M is a reversible n-tape Turing machine, then the one-tape simulator $S(M)$ in Lemma 0.2.2 can be chosen to be reversible.*

Proof. Let $M^\dagger$ be a reverse machine for M. The inverse of the simulator is obtained by applying the same simulation construction to $M^\dagger$, together with the reverse of the fixed-format encoding and decoding routines. Thus the inverse simulator does not try to recover the input of a local transition from its output by assuming that the transition table is injective. At the macro-step level it simulates the given reverse computation of M.

It remains only to ensure that the micro-steps used by the simulator can be reversed. The read sweeps, update sweeps, guard-column extensions, and delimiter conversions are all performed with phase states recording the finite amount of temporary information needed to undo them. Hence the reverse simulator traverses the same encoded macro-configurations in the opposite order, with only administrative configurations in between. Therefore $S(M)$ is reversible. □

0.3 Bennett's reversible simulation

Lemma 0.3.1. *For every Turing machine M there is a reversible four-tape Turing machine B_M such that, for every input w on which M halts with output $M(w)$, the machine B_M halts with final tape contents*

$$(w, \varepsilon, \varepsilon, M(w)).$$

All four heads finish at position 0.

Proof. By Lemma 0.1.11, we may replace M by an equivalent normal-form machine. Thus assume M is final-stationary and halts with head position 0.

The four tapes of B_M have the following roles:

Tape 1	simulates the tape of M,
Tape 2	records the symbol-state update at each step,
Tape 3	records the head movement at each step,
Tape 4	stores a protected copy of the final output.

The computation is divided into three reversible phases.

Phase 1: compute and record history. Suppose that at forward step j, where $0 \le j < m$, the simulated computation has head position p_j and

$$\delta(\sigma, q) = (\tau, q', d_j).$$

Tape 1 performs this simulated step. At the same time, tape 2 writes the history symbol

$$h_j = ((\sigma, q), (\tau, q')),$$

and tape 3 writes the direction d_j, both in the next unused history cell. Since h_j contains both the before-data and the after-data, this recorded step can be undone without requiring δ itself to be injective.

When the simulated machine enters q_{out}, the final direction is `Stay`, because M is final-stationary. Hence $d_{m-1} = \texttt{Stay}$. Erase this final `Stay`, shift the remaining directions one cell to the right, and write `Stay` in the first direction cell. The shifted sequence is

$$s_0 = \texttt{Stay}, \qquad s_j = d_{j-1} \quad (1 \le j \le m-1).$$

This is the sequence needed by the reverse pass: after undoing forward step j, the reverse machine must move to the site of forward step $j-1$, so it moves in the opposite direction to $s_j = d_{j-1}$.

Phase 2: copy the output. Tape 4 is blank on all reachable configurations entering this phase. Therefore the copying operation

$$(x, \sqcup) \longmapsto (x, x)$$

from tape 1 to tape 4 is reversible on the reachable configurations of this phase. The heads are then returned to their origins.

Phase 3: uncompute. Read the history symbols on tape 2 from right to left. If the current history symbol is

$$h_j = ((\sigma, q), (\tau, q')),$$

then tape 1, currently in simulated state q' and reading τ at the appropriate cell, rewrites τ as σ and changes the simulated state back to q. The movement performed after this inverse rewrite is the opposite of the shifted direction s_j on tape 3. As each history entry is used, the corresponding cells of tapes 2 and 3 are erased; the inverse phase re-creates exactly those entries. At the end of this phase, tape 1 again contains the original input w, tapes 2 and 3 are blank, and tape 4 still contains $M(w)$.

Each phase has an inverse on its reachable configurations, and the phase label is part of the finite control. Hence the composite four-tape machine is reversible. □

Example 0.3.2. If a computation has four simulated directions

$$d_0 = \texttt{Right}, \qquad d_1 = \texttt{Stay}, \qquad d_2 = \texttt{Left}, \qquad d_3 = \texttt{Stay},$$

then the shifted directions paired with the history entries are

j	0	1	2	3
h_j	h_0	h_1	h_2	h_3
d_j	`Right`	`Stay`	`Left`	`Stay`
s_j	`Stay`	`Right`	`Stay`	`Left`

The reverse phase reads h_3, h_2, h_1, h_0 and moves by $\mathrm{op}(s_3), \mathrm{op}(s_2), \mathrm{op}(s_1), \mathrm{op}(s_0)$ after the corresponding rewrites. This compensates for the fact that a forward step rewrites before it moves.

Theorem 0.3.3 (Bennett)**.** *For every Turing machine*

$$M = (\Sigma, Q, q_{\mathit{Start}}, q_{\mathit{Finish}}, \delta)$$

there exists a reversible one-tape Turing machine

$$N = (\Sigma', Q', q'_{\mathit{Start}}, q'_{\mathit{Finish}}, \delta')$$

such that $\Sigma \subseteq \Sigma'$ and there is a distinguished symbol $\# \in \Sigma' \setminus \Sigma$ with the following property: if M halts on input w with output $M(w) = u$, then N halts on input w with output

$$w\#\#u.$$

Proof. By Lemma 0.1.11, we may assume that M satisfies the head-position and final-`Stay` conventions. Apply Lemma 0.3.1 to obtain a reversible four-tape machine B_M which maps

$$(w, \varepsilon, \varepsilon, \varepsilon) \quad \text{to} \quad (w, \varepsilon, \varepsilon, u)$$

whenever $M(w) = u$.

By Lemma 0.2.3, this four-tape reversible computation can be simulated by a reversible one-tape machine. Prepend a reversible initialization routine which converts the one-tape input w into the four-tape encoding $(w, \varepsilon, \varepsilon, \varepsilon)$. Append a reversible formatting routine which converts the final four-tape encoding $(w, \varepsilon, \varepsilon, u)$ into the one-tape word

$$w\#\#u.$$

Both formatting routines are reversible on their canonical domains, and the composition of reversible machines is reversible by composing their reverse machines in the opposite order. Therefore the resulting one-tape machine N is reversible and satisfies $N(w) = w\#\#M(w)$ whenever $M(w)$ is defined. □

References

[1] R. Landauer. Irreversibility and heat generation in the computing process. *IBM Journal of Research and Development*, 5(3):183–191, 1961.

[2] C. H. Bennett. Logical reversibility of computation. *IBM Journal of Research and Development*, 17(6):525–532, 1973.

[3] W. Troiani. Reversible Turing Machines, 2019. https://williamtroiani.github.io/Notes/Reversible_computation_talk_notes.pdf.

[4] J. Clift. Universal Turing Machines. http://therisingsea.org/notes/talk-james-utm.pdf.