SAND2017-3513C

Foundations of Generalized Reversible Computing

Michael P. Frank*

Center for Computing Research, Sandia National Laboratories,
P.O. Box 5800, Mail Stop 1322, Albuquerque, NM 87185
mpfrank@sandia.gov
http://www.cs.sandia.gov/cr-mpfrank

Abstract. Landauer's Principle that information loss from a computation implies energy dissipation can be expressed as a rigorous theorem of mathematical physics. Thus, increasing the amount of useful computational work that can be accomplished for a given energy cost will eventually require increasing the degree to which our computing technologies avoid information loss, *i.e.*, are logically reversible. But the traditional definition of logical reversibility is more restrictive than is actually necessary to avoid information loss and energy dissipation due to Landauer's Principle. As a result, the operations that are usually thought of as the atomic elements of reversible logic, such as Toffoli gates, are not the simplest primitives one could use for the design of real reversible hardware. A complete theoretical framework for reversible computing should provide a more general, parsimonious foundation for practical engineering. To this end, we can use a rigorous quantitative formulation of Landauer's Principle to develop a new theory of *Generalized Reversible Computing* (GRC), which precisely characterizes the minimum requirements for a computation to avoid information loss and energy dissipation under Landauer's Principle, showing that, in fact, a much broader range of computations are reversible than is acknowledged by traditional reversible computing theory. This paper summarizes the foundations of GRC theory and briefly presents a few of its applications.

Keywords: Landauer's Principle, foundations of reversible computing, logical reversibility, reversible logic models, reversible hardware design, conditional reversibility, generalized reversible computing

1 Introduction

As we approach the end of the semiconductor roadmap [1], there is a growing realization that new computing paradigms will be required to continue improving

* This work was supported by the Laboratory Directed Research and Development program at Sandia National Laboratories, and by the Advanced Simulation and Computing program under the U.S. Department of Energy's National Nuclear Security Administration (NNSA). Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for NNSA under contract DE-AC04-94AL85000.

the energy efficiency (and thus, cost efficiency) of computing technology beyond the expected final CMOS node, when signal energies will reach a minimum practical level due to thermal noise and architectural overheads.[1] Sustained progress thus requires recovering and reusing signal energies with efficiency approaching 100%, which implies we must carry out logically reversible transformations of the local digital state, due to Landauer's Principle [2], which tells us that performing computational operations that are *irreversible* (*i.e.*, that lose information) necessarily generates entropy, and results in energy dissipation. Thus, it's essential for the designers of future computing technologies to clearly and correctly understand the meaning of and rationale for Landauer's Principle, and the consequent requirements, at the logical level, for computational operations to be reversible—meaning, both not information-losing, and also capable of being physically carried out in an asymptotically thermodynamically reversible way.

Although Landauer's Principle is valid, his original definition of what it meant for a computation to be "logically reversible" was not general enough to encompass all of the abstract logical structures that a computation can have while still avoiding information loss and being able to be carried out via (asymptotically) thermodynamically reversible physical processes. It turns out that a much larger set of computational operations can be reversible *at the logical level* than Landauer's traditional definition of logical reversibility acknowledges, which opens up many possibilities for engineering reversible devices and circuits that could never have been understood using the traditional definition, although some of those opportunities were discovered anyway by the designers of historical concepts for hardware implementation of reversible computing, such as Drexler's rod logic ([3], ch. 12) and Younis and Knight's charge recovery logic [4].

Yet, there remains today a widespread disconnect between standard reversible computing theory and the engineering principles required for the design of efficient reversible hardware. This disconnect has contributed to an ongoing debate (*e.g.*, [5]) regarding the question of whether logical reversibility is really required for physical reversibility. Indeed it is, but *not* if the standard definition of logical reversibility is used. A useful response from the theory side would be to *update* the standard definition of logical reversibility to reflect the *exact* logical-level requirements for physical reversibility. Upon that firmer foundation, we can construct a more general theoretical model for reversible computing, which can then help bridge the historical disconnect between theory and engineering in this field. It is the goal of this paper to develop such a model from first principles, and show exactly why it is necessary and useful.

The rest of this paper is structured as follows. In §2, we review some physical foundations and derive a general formulation of Landauer's Principle, which we then use in §3 as the basis for systematically reconstructing reversible computing theory to produce a new theoretical framework that we call General-

[1] Per [1], minimum gate energies are expected to bottom out at around the 40–80 $k_B T$ (1–2 eV) level (where k_B is Boltzmann's constant, and T is operating temperature); while typical total CV^2 node energies (where C is node capacitance, and V is logic swing voltage) may level off at a corresponding higher range of 1–2 keV.

ized Reversible Computing (GRC), which formalizes the essential but often-overlooked concept of *conditional reversibility* (previously mentioned in [6]). In §4, we present a few examples of conditionally-reversible operations that are useful building blocks for reversible hardware design, and are straightforwardly physically implementable. §5 briefly discusses why GRC is the appropriate model for asymptotically thermodynamically reversible hardware such as adiabatic switching circuits. §6 contrasts GRC's concept of conditional reversibility with existing concepts of conditions for correctness of reversible computations. §7 concludes with an outline of directions for future work.

The present version of this paper has been limited to a summary of results, omitting the proofs, due to conference page limits. A longer, more comprehensive version will be published as a journal article at a later time.

2 Formulating Landauer's Principle

Landauer's Principle is essentially the observation that the loss of information from a computation corresponds to an increase in physical entropy, implying a certain associated dissipation of energy to heat in the environment. But, articulating the meaning of and justification for the Principle in a more detailed way will help clarify what *information loss* really means, and under what conditions, precisely, information is lost in the course of carrying out a given computation.

As is standard in modern physics, we assume that any finite, closed physical system has only some finite number N of distinguishable physical states, thus a maximum entropy $\bar{S} = k_{\mathrm{B}} \ln N$. In quantum theory, N is also the dimensionality of the system's Hilbert space, *i.e.*, the cardinality of any basis set of orthogonal (distinguishable) state vectors that spans the space of all possible quantum states of the system. Let Σ denote any such maximal set of distinguishable states; we call this a *physical state space* for the system.

Furthermore, modern physics requires that the physical dynamics relating states at any time $t \in \mathbb{R}$ to the states that they may evolve to (or from) at any later (resp. earlier) time $t + \Delta t \in \mathbb{R}$ is a bijective (one-to-one and onto) functional relation. In quantum physics, this bijective dynamics is given by the unitary time-evolution operator $U(\Delta t) = \mathrm{e}^{-\mathrm{i}H\Delta t/\hbar}$, where H is the system's Hamiltonian operator (its total-energy observable).[2] Thus, physics is bijective, in the above sense, implying that it is deterministic (meaning, the present state determines the future) and reversible (the present determines the past).

Note that if fundamental physics were irreversible, then the Second Law of Thermodynamics (which states that the change in entropy over time is non-negative, $\Delta S \geq 0$) would be false, because two distinguishable states each with nonzero probability could merge, combining their probabilities, and reducing their contribution to the total entropy. Thus, the reversibility of fundamental physics follows from the empirically-observed validity of the Second Law.

[2] Although quantum physics does not yet incorporate a description of gravity, it's expected that even a full theory of quantum gravity would still exhibit unitarity.

In any event, if one accepts the bijectivity of dynamical evolution as a truism of mathematical physics, then, as we will see, the validity of Landauer's Principle follows rigorously from it, as a theorem.

Given a physical state space Σ, a *computational subspace* C of Σ can be identified with a partition of the set Σ. We say that a physical system Π is *in computational state* $c_j \in C$ whenever there is an $s_i \in c_j$ such that the physical state of the system is not reliably distinguishable from s_i. In other words, a computational state c_j is just an equivalence class of physical states that can be considered equivalent to each other, in terms of the computational information that we are intending them to represent. We assume that we can also identify an appropriate computational subspace $C(\Delta t)$ that is a partition of the evolved physical state space $\Sigma(\Delta t)$ at any past or future time $t_0 + \Delta t \in \mathbb{R}$.

Consider, now, any initial-state probability distribution p_0 over the complete state space $\Sigma = \Sigma(0)$ at time $t = t_0$. This then clearly induces an implied initial probability distribution P_{I} over the *computational* states at time t_0 as well:

$$P_{\mathrm{I}}(c_j) = \sum_{k=0}^{|c_j|} p_0(s_{j,k}),$$

where $s_{j,k}$ denotes the kth physical state in computational state $c_j \in C$.

For probability distributions p and P over physical and computational states, we can define corresponding entropy measures. Given any probability distribution p over a physical state space Σ, the *physical entropy* $S(p)$ is defined by

$$S(p) = \sum_{i=0}^{N=|\Sigma|} p(s_i) \log \frac{1}{p(s_i)},$$

where the logarithm can be considered to be an indefinite logarithm, dimensioned in generic logarithmic units.

The bijectivity of physical dynamics then implies the following theorem:

Theorem 1. ***Conservation of entropy.*** The physical entropy of any closed system, as determined for any initial state distribution p_0, is exactly conserved over time. *I.e.*, if the physical entropy of an initial-state distribution $p_0(s_i)$ at time t_0 is $S(0)$, and we evolve that system over an elapsed time $\Delta t \in \mathbb{R}$ according to its bijective dynamics, the physical entropy $S(\Delta t)$ of its final-state probability distribution $p_{\Delta t}$ at time $t_0 + \Delta t$ will be the exact same value, $S(\Delta t) = S(0)$.

Theorem 1 takes an ideal, theoretical perspective. In practice, entropy from any real observer's perspective increases, because the observer does not have exact knowledge of the dynamics, or the capability to track it exactly. But in principle, the ideal perspective with constant entropy still always exists.

We can also define the entropy of the computational state. Given any probability distribution P over a computational state space C, the *information entropy* or *computational entropy* $H(P)$ is defined by:

$$H(P) = \sum_{j=0}^{|C|} P(c_j) \log \frac{1}{P(c_j)},$$

which, like $S(p)$, is dimensioned in arbitrary logarithmic units.

Finally, we define the *non-computational entropy* as the remainder of the total physical entropy, other than the computational part; $S_{\mathrm{nc}} = S - H \geq 0$. This is the expected physical entropy conditioned on the computational state.

The above definitions let us derive Landauer's Principle, in its most general, quantitative form, as well as another form frequently seen in the literature.

Theorem 2. ***Landauer's Principle (general formulation).*** If the computational state of a system at initial time t_0 has entropy $H_{\mathrm{I}} = H(P_{\mathrm{I}})$, and we allow that system to evolve, according to its physical dynamics, to some other "final" time $t_0 + \Delta t$, at which its computational entropy becomes $H_{\mathrm{F}} = H(P_{\mathrm{F}})$ where $P_{\mathrm{F}} = P(\Delta t)$ is the induced probability distribution over the computational state set $C(\Delta t)$ at time $t_0 + \Delta t$, then the non-computational entropy is increased by

$$\Delta S_{\mathrm{nc}} = H_{\mathrm{I}} - H_{\mathrm{F}}.$$

Conventional digital devices are typically designed to locally reduce computational entropy, *e.g.*, by erasing or destructively overwriting "unknown" old bits obliviously, *i.e.*, ignoring any independent knowledge of their previous value. Thus, typical device operations necessarily eject entropy into the non-computational form, and so, over time, non-computational entropy typically accumulates in the system, manifesting as heating. But, systems cannot tolerate indefinite entropy build-up without overheating. So, the entropy must ultimately be moved out to some external environment at some temperature T, which involves the dissipation of energy $\Delta E_{\mathrm{diss}} = T\Delta S_{\mathrm{nc}}$ to the form of heat in that environment, by the definition of thermodynamic temperature. From Theorem 2 together with these facts and the logarithmic identity $1\,\mathrm{bit} = (1\,\mathrm{nat})/\log_2 \mathrm{e} = k_{\mathrm{B}} \ln 2$ follows the more commonly-seen statement of Landauer's Principle:

Corollary 1. ***Landauer's Principle (common form).*** For each bit's worth of computational information that is lost within a computer (*e.g.*, by obliviously erasing or destructively overwriting it), an amount of energy

$$\Delta E_{\mathrm{diss}} = k_{\mathrm{B}} T \ln 2$$

must eventually be dissipated to the form of heat added to some environment at temperature T.

3 Reformulating Reversible Computing Theory

We now carefully analyze the implications of the general Landauer's Principle (Theorem 2) for computation, and reformulate reversible computing theory on that basis. We begin by redeveloping the foundations of the traditional theory of unconditionally logically-reversible operations, using a language that we subsequently build upon to develop the generalized theory.

For our purposes, a computational *device* D will simply be any physical artifact that is capable of carrying out one or more different *computational operations*, by which the physical and computational state spaces Σ, C associated

with D's local state are transformed. If D has an associated local computational state space $C_\mathrm{I} = \{c_{\mathrm{I}1}, ..., c_{\mathrm{I}m}\}$ at some initial time t_0, a computational operation O on D that is applicable at t_0 is specified by giving a probabilistic transition rule, *i.e.*, a stochastic map from the initial computational state at t_0 to the final computational state at some later time $t_0 + \Delta t$ (with $\Delta t > 0$) by which the operation will have been completed. Let the computational state space at this later time be $C_\mathrm{F} = \{c_{\mathrm{F}1}, ..., c_{\mathrm{F}n}\}$. Then, the operation $O : C_\mathrm{I} \to \mathcal{P}(C_\mathrm{F})$ is a map from C_I to probability distributions over C_F; which is characterizable, in terms of random variables $c_\mathrm{I}, c_\mathrm{F}$ for the initial and final computational states, by a conditional probabilistic transition rule

$$r_i(j) = \Pr(c_\mathrm{F} = c_{\mathrm{F}j} | c_\mathrm{I} = c_{\mathrm{I}i}) = [O(c_{\mathrm{I}i})](c_{\mathrm{F}j}),$$

where $i \in \{1, ..., m\}$ and $j \in \{1, ..., n\}$. That is, $r_i(j)$ denotes the conditional probability that the final computational state is $c_{\mathrm{F}j}$, given that the initial computational state is $c_{\mathrm{I}i}$.

A computational operation O will be called *deterministic* if and only if all of the probability distributions r_i are single-valued. *I.e.*, for each possible value of the initial-state index $i \in \{1, ..., m\}$, there is exactly one corresponding value of the final-state index j such that $r_i(j) > 0$, and thus, for this value of j, it must be the case that $r_i(j) = 1$, while $r_i(k) = 0$ for all other $k \neq j$. If an operation O is not deterministic, we call it *nondeterministic*.[3] For a deterministic operation O, we can write $O(c_{\mathrm{I}i})$ to denote the unique $c_{\mathrm{F}j}$ such that $r_i(j) = 1$, that is, treating O as a simple transition function rather than a stochastic one.

A computational operation O will be called (unconditionally logically) *reversible* if and only if all of the probability distributions r_i have non-overlapping nonzero ranges. In other words, for each possible value of the final-state index $j \in \{1, ..., n\}$, there is at most one corresponding value of the initial-state index i such that $r_i(j) > 0$, while $r_k(j) = 0$ for all other $k \neq i$. If an operation O is not reversible, we call it *irreversible*.

For a computational operation O with an initial computational state space C_I, a *(statistical) operating context* for that operation is any probability distribution P_I over the initial computational states; for any $i \in \{1, ..., m\}$, the value of $P_\mathrm{I}(c_{\mathrm{I}i})$ gives the probability that the initial computational state is $c_{\mathrm{I}i}$.

A computational operation O will be called *(potentially) entropy-ejecting* if and only if there is some operating context P_I such that, when the operation O is applied within that context, the increase ΔS_nc in the non-computational entropy required by Landauer's Principle is greater than zero. If an operation O is not potentially entropy-ejecting, we call it *non-entropy-ejecting*.

[3] Note that this is a different sense of the word "nondeterministic" than is commonly used in computational complexity theory, when referring to, for example, nondeterministic Turing machines, which conceptually evaluate all of their possible future computational trajectories in parallel. Here, when we use the word "nondeterministic," we mean it simply in the physicist's sense, to refer to randomizing or stochastic operations; *i.e.*, those whose result is uncertain.

Now, we can derive Landauer's original result stating that only operations that are logically reversible (in his sense) can always avoid ejecting entropy from the computational state (independently of the operating context).

Theorem 3. ***Fundamental Theorem of Traditional Reversible Computing.*** Non-entropy-ejecting deterministic operations must be reversible. That is, if a given deterministic computational operation O is non-entropy-ejecting, then it is reversible in the sense defined above (its transition relation is injective).

The proof of the theorem involves showing that entropy is ejected when states with nonzero probability are merged by an operation. However, when states having *zero* probability are merged with other states, there is no increase in entropy. This is the key realization that sets us up to develop GRC.

To do this, we define a notion of a *computation* that fixes a specific statistical operating context for a computational operation, and then we examine the detailed requirements for a given computation to be non-entropy-ejecting. This leads to the concept of *conditional reversibility*, which is the most general concept of logical reversibility, and provides the appropriate foundation for GRC.

For us, a *computation* $\mathcal{C} = (O, P_{\mathrm{I}})$ performed by a device D is defined by specifying *both* a computational operation O to be carried out by that device, *and* a specific operating context P_{I} under which the operation O is to be performed.

A computation $\mathcal{C} = (O, P_{\mathrm{I}})$ is called *(specifically) entropy-ejecting* if and only if, when the operation O is applied within the specific operating context P_{I}, the increase ΔS_{nc} in the non-computational entropy required by Landauer's Principle is greater than zero. If $\mathcal{C}$ is not specifically entropy-ejecting, we call it *non-entropy-ejecting*.

A deterministic computational operation O is called *conditionally reversible* if and only if there is a non-empty subset $A \subseteq C_{\mathrm{I}}$ of initial computational states (the *assumed set* or *assumed precondition*) that O's transition rule maps onto an equal-sized set $B \subseteq C_{\mathrm{F}}$ of final states. That is, each $c_{\mathrm{I}i} \in A$ maps, one to one, to a unique $c_{\mathrm{F}j} \in B$ where $r_i(j) = 1$. We say that B is the *image of A under O*. We also say that O is *(conditionally) reversible under the precondition (that the initial state is in) A*.

It turns out that *all* deterministic computational operations are, in fact, conditionally reversible, under some sufficiently-restrictive preconditions.

Theorem 4. ***Conditional reversibility of all deterministic operations.*** All deterministic computational operations are conditionally reversible.

A trivial proof of Theorem 4 involves considering precondition sets A that are singletons. However, deterministic operations with any number $k > 1$ of reachable final computational states are also conditionally reversible under at least one precondition set A of size k.

Whenever we wish to fix a *specific* assumed precondition A for the reversibility of a conditionally-reversible operation O, we use the following concept:

Let O be any conditionally-reversible computational operation, and let A be any one of the preconditions under which O is reversible. Then the *conditioned*

reversible operation $O_A = (O, A)$ denotes the concept of performing operation O in the context of a requirement that precondition A is satisfied.

Restricting the set of initial states that may have nonzero probability to a specific proper subset $A \subset C_\mathrm{I}$ represents a change to the semantics of an operation, so generally, a conditioned reversible version of an arbitrary deterministic operation is, in effect, not exactly the same operation. But we will see that arbitrary computations can still be composed out of these restricted operations.

The central result of GRC theory (Theorem 5, below) is then that a deterministic computation $\mathcal{C} = (O, P_\mathrm{I})$ is specifically non-entropy-ejecting, and therefore avoids any requirement under Landauer's Principle to dissipate any energy $\Delta E_\mathrm{diss} > 0$ to its thermal environment, if and only if its operating context P_I assigns total probability 1 to some precondition A under which its computational operation O is reversible. Moreover (Theorem 6), even if the probability of satisfying some such precondition only *approaches* 1, this is sufficient for the entropy ejected (and energy dissipation required) to approach zero.

Theorem 5. ***Fundamental Theorem of Generalized Reversible Computing.*** Any deterministic computation is non-entropy-ejecting if and only if at least one of its preconditions for reversibility is satisfied. *I.e.*, let $\mathcal{C} = (O, P_\mathrm{I})$ be any deterministic computation (*i.e.*, any computation whose operation O is deterministic). Then, part (a): If there is some precondition A under which O is reversible, such that A is satisfied with certainty in the operating context P_I, then $\mathcal{C}$ is a non-entropy-ejecting computation. And, part (b): Alternatively, if no such precondition A is satisfied with certainty, then $\mathcal{C}$ is entropy-ejecting.

Theorem 6. ***Entropy ejection vanishes as precondition certainty approaches unity.*** Let O be any deterministic operation, and let A be any precondition under which O is reversible, and let $P_{\mathrm{I}1}, P_{\mathrm{I}2}, ...$ be any sequence of operation contexts for O within which the total probability mass assigned to A approaches 1. Then, in the corresponding sequence of computations, the entropy ejected ΔS_nc also approaches 0.

A numerical example illustrating how the ΔS_nc calculation comes out in a specific case where the probability of violating the precondition for reversibility is small can be found in [7].

It's important to note that in order for real hardware devices to apply Theorems 5 and 6 to avoid or reduce energy dissipation in practice, the device must be designed with implicit knowledge of not only what conditionally-reversible operation it should perform, but also which specific one of the preconditions for that operation's reversibility it should assume is satisfied.

As we saw in Theorem 4, any deterministic computational operation O is conditionally reversible with respect to any given one A of its suitable preconditions for reversibility. For any computation $\mathcal{C} = (O, P_\mathrm{I})$ that satisfies the conditions for reversibility of the conditioned reversible operation O_A with certainty, we can undo the effect of that computation exactly by applying any conditioned reversible operation that is what we call a *reversal* of O_A. The reversal of a conditioned reversible operation is simply an operation that maps the image of

the assumed set back onto the assumed set itself in a way that exactly inverts the original forward map.

The above framework can also be extended to work with nondeterministic computations. In fact, adding nondeterminism to an operation only makes it easier to avoid ejecting entropy to the non-computational state, since nondeterminism tends to increase the computational entropy, and thus tends to reduce the non-computational entropy. As a result, a nondeterministic operation can be non-entropy-ejecting (or even entropy-absorbing, *i.e.*, with $\Delta S_{\text{nc}} < 0$) even in computations where none of its preconditions for reversibility are satisfied, so long as the reduction in computational entropy caused by its irreversibility is compensated for by an equal or greater increase in computational entropy caused by its nondeterminism. However, we will not take the time, in the present paper, to flesh out detailed analyses of such cases.

4 Examples of Conditioned Reversible Operations

Here, we define and illustrate a number of examples of conditionally reversible operations (including a specification of their assumed preconditions) that comprise natural primitives out of which arbitrary reversible algorithms may be composed. First, we introduce some textual and graphical notations for describing conditioned reversible operations.

Let the computational state space be factorizable into independent *state variables* $x, y, z, ...$, which are in general n-ary discrete variables. A common special case will be binary variables ($n = 2$). For simplicity, we assume here that the sets of state variables into which the initial and final computational state spaces are factorized are identical, although more generally this may not be the case.

Given a computational state space C that is factorizable into state variables $x, y, z, ...$, and given a precondition A on the initial state defined by

$$A = \{c_i \in C \,|\, P(x, y, ...)\},$$

where $P(x, y, ...)$ is some propositional (*i.e.*, Boolean-valued) function of the state variables $x, y, ...$, we can denote a conditionally-reversible operation O_A on C that is reversible under precondition A using notation like:

$$\texttt{OpName}(x, y, ... \,|\, P(x, y, ...))$$

which represents a conditionally-reversible operation named `OpName` that operates on and potentially transforms the state variables $x, y, ...$, and that is reversible under an assumed precondition A consisting of the set of initial states that satisfy the given proposition $P(x, y, ...)$.

A simple, generic graphical notation for a deterministic, conditionally reversible operation named `OpName`, operating on a state space that is decomposable into three state variables x, y, z, and possibly including an assumed precondition for reversibility $P(x, y, z)$, is the ordinary space-time diagram representation shown in Fig. 1(a).

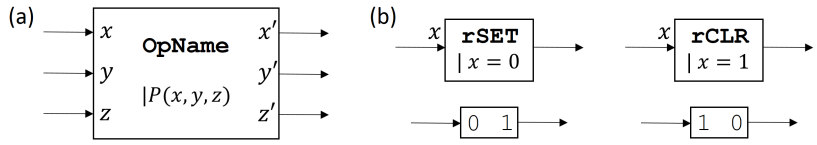

Fig. 1. (a) Generic graphical notation for a deterministic, conditioned reversible operation $\texttt{OpName}(x, y, z \,|\, P(x, y, z))$ on three state variables x, y, z, with an assumed precondition specified by the propositional function $P(x, y, z)$. (b) Left: Standard graphical notation (top) and simplified symbol (bottom) for the conditioned reversible operation $\texttt{rSET}(x \,|\, x = 0)$; Right: Likewise for $\texttt{rCLR}(x \,|\, x = 1)$.

In this representation, as in standard reversible logic networks, time is visualized as flowing from left to right, and the horizontal lines represent state variables. The primed versions x', y', z' going outwards represent the values of the state variables in the final computational state c_F after the operation.

As Landauer observed, operations such as "set to one" and "reset to zero" on binary state spaces are logically irreversible, under his definition; indeed, they constitute classic examples of *bit erasure* operations for which (assuming equiprobable initial states) an amount $k_\mathrm{B} \ln 2$ of entropy is ejected from the computational state. However, as per Theorem 4, these operations are in fact conditionally reversible, under suitably-restricted preconditions. A suitable precondition, in this case, is one in which one of the two initial states is excluded. Thus, the initial state is known with certainty in any operating context satisfying such a precondition. A known state can be transformed to any specific new state reversibly. If the new state is different from the old one, such an operation is non-vacuous. Thus, the following conditioned reversible operations are useful.

The deterministic operation `rSET` (*reversible set-to-one*) on a binary variable x, which (to be useful) is implicitly associated with an assumed precondition for reversibility of $x = 0$, is an operation that is defined to transform the initial state into the final state $x' = 1$; in other words, it performs the operation $x := 1$. Standard and simplified graphical notations for this operation are illustrated on the left-hand side of Fig. 1(b).

By Theorem 5, the conditioned reversible operation $\texttt{rSET}(x \,|\, x = 0)$ is specifically non-entropy-ejecting in operating contexts where the designated precondition for reversibility is satisfied. It can be implemented in a way that is asymptotically physically reversible (as the probability that its precondition is satisfied approaches 1) using any mechanism that is designed to adiabatically transform the state $x = 0$ to the state $x = 1$.

Similarly, we can consider a deterministic conditioned reversible operation $\texttt{rCLR}(x \,|\, x = 1)$ (*reversible clear* or *reversible reset-to-zero*) which has an assumed precondition for reversibility of $x = 1$ and which performs the operation $x := 0$, illustrated on the right-hand side of Fig. 1(b).

A very commonly-used computational operation is to copy one state variable to another. As with any other deterministic operation, such an operation is con-

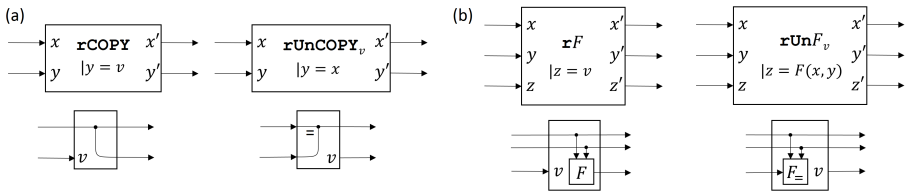

Fig. 2. (a) Left: Reversible copy of x onto $y = v$; Right: Reversible uncopy of y from x back to v. (b) Given any function $F(x, y) = z$ of n (here, $n = 2$) state variables, we can easily convert it to a pair of conditioned reversible operations $\mathtt{r}F(x, y, z \,|\, z = v)$ and $\mathtt{rUn}F_v(x, y, z \,|\, z = F(x, y))$ that are mutual reversals of each other that compute and decompute the value of F by reversibly transforming the output variable z from and to any predetermined value v. Top: standard notation, bottom: simplified symbols.

ditionally reversible under suitable preconditions. An appropriate precondition for the reversibility of this `rCOPY` operation is any in which the initial value of the target variable is known, so that it can be reversibly transformed to the new value. A standard reversal of a suitably-conditioned `rCOPY` operation, which we can call `rUnCOPY`, is simply a conditioned reversible operation that transforms the final states resulting from `rCOPY` back to the corresponding initial states.

Formally, let x, y be any two discrete state variables both with the same arity (number n of possible values, which without loss of generality we may label $0, 1, ...$), and let $v \in \{0, 1, ..., n-1\}$ be any fixed initial value. Then *reversible copy of x onto $y = v$* or

$$\mathtt{rCOPY}_v = \mathtt{rCOPY}(x, y \,|\, y = v)$$

is a conditioned reversible operation O with assumed precondition $y = v$ that maps any initial state where $x = i$ onto the final state $x = i, y = i$. In the language of ordinary pseudocode, the operation performed is simply $y := x$.

Given any conditioned reversible copy operation $\mathtt{rCOPY}_v$, there is a conditioned reversible operation which we hereby call *reversible uncopy of y from x back to v* or

$$\mathtt{rUnCOPY}_v = \mathtt{rUnCOPY}_v(x, y \,|\, y = x)$$

which, assuming (as its precondition for reversibility) that initially $x = y$, carries out the operation $y := v$, restoring the destination variable y to the same initial value v that was assumed by the `rCOPY` operation.

Fig. 2(a) shows graphical notations for $\mathtt{rCOPY}_v$ and $\mathtt{rUnCOPY}_v$.

It is easy to generalize `rCOPY` to more complex functions. In general, for any function $F(x, y, ...)$ of any number of variables, we can define a conditioned reversible operation $\mathtt{r}F(x, y, ..., z \,|\, z = v)$ which computes that function, and writes the result to an output variable z by transforming z from its initial value to $F(x, y, ...)$, which is reversible under the precondition that the initial value of z is some known value v. Its reversal $\mathtt{rUn}F_v(x, y, ..., z \,|\, z = F(x, y, ...))$ decomputes the result in the output variable z, restoring it back to the value v. See Fig. 2(b).

The F above may indeed be any function, including standard Boolean logic functions operating on binary variables, such as AND, OR, *etc.* Therefore, the above scheme leads us to consider conditioned reversible operations such as $\texttt{rAND}_0$, $\texttt{rAND}_1$, $\texttt{rOR}_0$, $\texttt{rOR}_1$; and their reversals $\texttt{rUnAND}_0$, $\texttt{rUnAND}_1$, $\texttt{rUnOR}_0$, $\texttt{rUnOR}_1$; which reversibly do and undo standard AND and OR logic operations with respect to output nodes that are expected to be a constant logic 0 or 1 initially before the operation is done (and also finally, after doing the reverse operations).

Clearly, one can compose arbitrary n-input Boolean functions out of such primitives using standard logic network constructions, and decompute intermediate results using the reverse (mirror-image) circuits (after rCOPYing the desired results), following the general approach pioneered by Bennett [8]. This results in an embedding of the desired function into a reversible function that preserves only the input and the final output.

One may wonder, however, what is the advantage of using operations such as rAND and rUnAND for this, compared to the traditional unconditionally reversible operation $\texttt{ccNOT}(x, y, z)$ (controlled-controlled-NOT, a.k.a. the Toffoli gate operation [9], $z := z \oplus xy$). Indeed, any device that implements $\texttt{ccNOT}(x, y, z)$ in a physically-reversible manner could be used in place of a device that implements $\texttt{rAND}(x, y, z \,|\, z = 0)$ and $\texttt{rUnAND}_0(x, y, z \,|\, z = xy)$, or in place of one that implements $\texttt{rNAND}(x, y, z \,|\, z = 1)$ and $\texttt{rUnNAND}_1(x, y, z \,|\, z = \overline{xy})$, in cases where the preconditions of those operations would be satisfied.

But, the converse is not true. In other words, there are devices that can asymptotically physically reversibly carry out $\texttt{rAND}_0$ and $\texttt{rUnAND}_0$ that do not also implement full Toffoli gate operations. Therefore, if what one really needs to do, in one's algorithm, is simply to do and undo Boolean AND operations reversibly, then to insist on doing this using Toffoli operations rather than conditioned reversible operations such as rAND and rUnAND is overkill, and amounts to tying one's hands with regards to the implementation possibilities, leading to hardware designs that can be expected to be more complex than necessary. Indeed, there are very simple adiabatic circuit implementations of devices capable of performing rAND/rUnAND and rOR/rUnOR operations (based on *e.g.* series/parallel combinations of CMOS transmission gates [10]), whereas, adiabatic implementations of ccNOT itself are typically much less simple. This illustrates our overall point that the GRC framework generally allows for simpler designs for reversible computational hardware than does the traditional reversible computing model based on unconditionally reversible operations.

5 Modeling Reversible Hardware

A broader motivation for the study of GRC derives from the following observation (not yet formalized as a theorem):

Assertion 1. ***General correspondence between truly, fully adiabatic circuits and conditioned reversible operations.*** Part (a): Whenever a switching circuit is operated deterministically in a truly, fully adiabatic way (*i.e.*, that

asymptotically approaches thermodynamic reversibility), transitioning among some discrete set of logic levels, the computation being performed by that circuit corresponds to a conditioned reversible operation O_A whose assumed precondition A is (asymptotically) satisfied. Part (b): Likewise, any conditioned reversible operation O_A can be implemented in an asymptotically thermodynamically reversible manner by using an appropriate switching circuit that is operated in a truly, fully adiabatic way, transitioning among some discrete set of logic levels.

Part (a) follows from our earlier observation in Theorem 5 that, in deterministic computations, conditional reversibility is the correct statement of the logical-level requirement for avoiding energy dissipation under Landauer's Principle, and therefore it is a necessity for approaching thermodynamic reversibility in any deterministic computational process, and therefore, more specifically, in the operation of adiabatic circuits.

Meanwhile, part (b) follows from general constructions showing how to implement any desired conditioned reversible operation in an asymptotically thermodynamically reversible way using adiabatic switching circuits. For example, Fig. 3 illustrates how to implement an `rCOPY` operation using a simple four-transistor CMOS circuit. In contrast, implementing `rCOPY` by embedding it within an unconditionally-reversible `cNOT` would require including an `XOR` capability, and would require a much more complicated adiabatic circuit, whose operation would itself be composed from numerous more-primitive operations (such as adiabatic transformations of individual MOSFETs [11]) that are themselves only conditionally reversible.

In general, the traditional reversible computing framework of unconditionally reversible operations does not exhibit any correspondence such as that of Assertion 1 to any natural class of asymptotically physically-reversible hardware that we know of. In particular, the traditional unconditionally-reversible framework does not correspond to the class of truly/fully adiabatic switching circuits, because there are many such circuits that do not in fact perform unconditionally reversible operations, but only conditionally-reversible ones.

6 Comparison to Prior Work

The concept of conditional reversibility presented here is similar to, but distinct from, certain concepts that are already well known in the literature on the theory of reversible circuits and languages.

First, the concept of a reversible computation that is only semantically correct (for purposes of computing a desired function) when a certain precondition on the inputs is satisfied is one that was already implicit in Landauer's original paper [2], when he introduced the operation now known as the Toffoli gate, as a reversible operation within which Boolean `AND` may be embedded. Implicit in the description of that operation is that it only correctly computes `AND` if the control bit is initially 0; otherwise, it computes some other function (in this case, `NAND`). This is the origin of the concept of *ancilla* bits, which are required

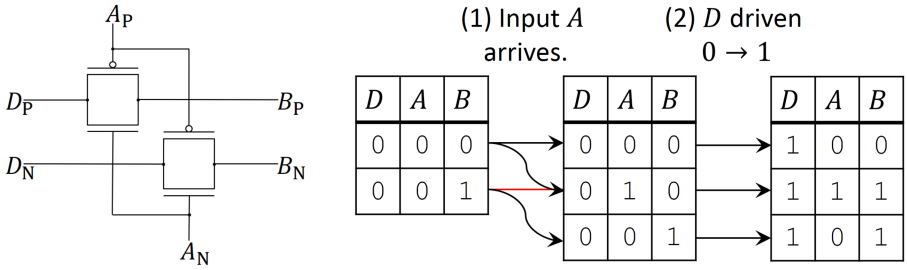

Fig. 3. (Left) A simple adiabatic CMOS circuit capable of carrying out a variant of the `rCOPY` operation. Here, computational states are represented using dual-rail complementary voltage coding, so that, for example, a logical state $A = \texttt{0}$ is represented using the voltage assignments $A_P = V_H, A_N = V_L$, where V_H, V_L are high and low voltage levels, respectively. The logical state $A = \texttt{1}$ would be represented using the opposite voltage assignments. The two CMOS transmission gates shown will thus be turned ON (conducting) only when $A = \texttt{1}$. In this circuit, A is the logic input, B is the output, and D is a driving signal. (Right) Sequence of operation. Assume initially that $D = \texttt{0}$ and $A = \texttt{0}$. Normally we would also have $B = \texttt{0}$ initially, but to illustrate the conditional reversibility of this circuit, we will also consider the case $B = \texttt{1}$. In step 1, some external circuit adiabatically transforms input A from logic `0` to a newly-computed value (`0` or `1`) to be copied, then in step 2, the drive signal D is unconditionally transformed adiabatically from logic `0` to `1`. Note that, in the course of this operation sequence, if B were `1` initially, then it would be dissipatively sourced to $D = \texttt{0}$ in step 1 if $A = \texttt{1}$. Thus, this particular operation sequence implements a conditioned reversible operation $\texttt{rCOPY}'(A, B \,|\, \overline{AB})$; it is reversible as long as we don't try to copy an input value $A = \texttt{1}$ onto an initial state where $B = \texttt{1}$. The prime there after `rCOPY` is denoting the variant semantics, namely that in the case $\bar{A}B$, the value $A = \texttt{0}$ is not copied to B.

to obey certain pre- and post-conditions (typically, being cleared to 0) in order for reversible circuits to be composable and still function as intended. The study of the circumstances under which such requirements may be satisfied has been extensively developed, *e.g.* as in [12]. However, any circuit composed from Toffoli gates is *still reversible* even if restoration of its ancillas is violated; it may yield nonsensical outputs in that case, when composed together with other circuits, but at no point is information erased. This distinguishes ancilla-preservation conditions from our preconditions for reversibility, which, when they are unsatisfied, necessarily yield actual (physical) irreversibility.

Similarly, the major historical examples of reversible high-level programming languages such as Janus ([13], [14]), Ψ-Lisp [15], the author's own R language [16], and RFUN ([17], [18]) have invoked various "preconditions for reversibility" in the defined semantics of many of their language constructs. But again, that concept really has more to do with the "correctness" or "well-definedness" of a high-level reversible program, and this notion is distinct from the requirements for actual physical reversibility during execution. For example, the R language compiler generated PISA assembly code in such a way that even if high-level language requirements were violated (*e.g.*, in the case of an `if` con-

dition changing its truth value during the `if` body), the resulting assembly code would still execute reversibly, if nonsensically, on the Pendulum processor [19].

In contrast, the notion of conditional reversibility explored in the present document ties directly to Landauer's principle, and to the possibility of the physical reversibility of the underlying hardware. Note, however, that it does not concern the semantic correctness of the computation, or lack thereof, and in general, the necessary preconditions for the physical reversibility and correctness of a given computation may be orthogonal to each other, as illustrated by the example in Fig. 3.

7 Conclusion

In this paper, we presented the core foundations of a general theoretical framework for reversible computing. We considered the case of deterministic computational operations in detail, and presented results showing that the class of deterministic computations that are not required to eject any entropy from the computational state under Landauer's Principle is larger than the set of computations composed of the unconditionally-reversible operations considered by traditional reversible computing theory, because it also includes the set of conditionally-reversible operations whose preconditions for reversibility are satisfied with probability approaching unity. This is the most general possible characterization of the set of classical deterministic computations that can be physically implemented in an asymptotically thermodynamically reversible way.

We then illustrated some basic applications of the theory in modeling conditioned reversible operations that transform an output variable between a predetermined, known value and the computed result of the operation. Such operations can be implemented easily using *e.g.* adiabatic switching circuits, whose low-level computational function cannot in general be represented within the traditional theory of unconditionally-reversible computing. This substantiates that the GRC theory warrants further study.

Some promising directions for future work include: (1) Giving further examples of useful conditioned reversible operations; (2) illustrating detailed physical implementations of devices for performing such operations; (3) further extending the development of the new framework to address the nondeterministic case; and (4) developing further descriptive frameworks for reversible computing at higher levels (*e.g.*, hardware description languages, programming languages) building on top of the fundamental conceptual foundations that GRC theory provides.

Since GRC broadens the range of design possibilities for reversible computing devices in a clearly delineated, well-founded way, its study and further development will be essential for the computing industry to successfully transition, over the coming decades, to the point where it is dominantly utilizing the reversible computing paradigm. Due to the incontrovertible validity of Landauer's Principle, such a transition will be an absolute physical prerequisite for the energy efficiency (and cost efficiency) of general computing technology to continue growing by many orders of magnitude.

References

1. International Technology Roadmap for Semiconductors 2.0, 2015 Edition, Semiconductor Industry Association (2015)
2. Landauer, R.: Irreversibility and Heat Generation in the Computing Process. IBM J. Res. Dev. 5(3), 183–191 (1961)
3. Drexler, K.E.: Nanosystems: Molecular machinery, manufacturing, and computation. John Wiley & Sons, Inc. (1992)
4. Younis, S.G., Knight Jr., T.F.: Practical implementation of charge recovering asymptotically zero power CMOS. In: Proceedings of the 1993 Symposium on Research in Integrated Systems, pp. 234–250. MIT Press (1993)
5. López-Suárez, M., Neri, I., Gammaitoni, L.: Sub-k_BT micro-electromechanical irreversible logic gate. Nat. Comm. 7, 12068 (2016)
6. Frank, M.P.: Approaching the physical limits of computing. In: 35th International Symposium on Multiple-Valued Logic, pp. 168–185. IEEE Press, New York (2005)
7. DeBenedictis, E.P., Frank, M.P., Ganesh, N., Anderson, N.G.: A path toward ultra-low-energy computing. In: IEEE International Conference on Rebooting Computing. IEEE Press, New York (2016)
8. Bennett, C.H.: Logical reversibility of computation. IBM J. Res. Dev. 17(6), 525–532 (1973)
9. Toffoli, T: Reversible computing. In: International Colloquium on Automata, Languages, and Programming, pp. 632–644. Springer, Berlin (1980)
10. Anantharam, V., He, M., Natarajan, K., Xie, H., Frank, M.: Driving Fully-Adiabatic Logic Circuits Using Custom High-Q MEMS Resonators. In: Arabnia, H.R., Guo, M., Yang, L.T. (eds.) ESI/VLSI 2004, pp. 5–11. CSREA Press (2004)
11. Frank, M.P.: Towards a More General Model of Reversible Logic Hardware. Invited talk presented at the Superconducting Electronics Approaching the Landauer Limit and Reversibility (SEALeR) workshop, sponsored by NSA/ARO (2012)
12. Thomsen, M.K., Kaarsgaard, R., Soeken, M.: Ricercar: A Language for Describing and Rewriting Reversible Circuits with Ancillae and Its Permutation Semantics. In: Krivine, J., Stefani, J.-B. (eds.) RC 2015. LNCS, vol. 9138, pp. 200-215. Springer International Publishing (2015)
13. Lutz, C.: Janus: A time-reversible language. Letter from Chris Lutz to Rolf Landauer, reproduced at `http://tetsuo.jp/ref/janus.pdf` (1986).
14. Yokoyama, T.: Reversible Computation and Reversible Programming Languages. Elec. Notes Theor. Comput. Sci. 253(6), 71–81 (2010)
15. Baker, H.G.: NREVERSAL of fortune—The thermodynamics of garbage collection. In: Bekkers Y., Cohen J. (eds.) Memory Management. LNCS, vol. 637, pp. 507–524. Springer, Berlin, Heidelberg (1992)
16. Frank, M.: Reversibility for Efficient Computing. Doctoral dissertation, Massachusetts Institute of Technology, Dept. of Elec. Eng. and Comp. Sci. (1999)
17. Yokoyama, T., Axelsen, H.B., Glück, R.: Towards a Reversible Functional Language. In: De Vos, A., Wille, R. (eds.) RC 2011. LNCS, vol. 7165, pp. 14-29. Springer, Berlin, Heidelberg (2012)
18. Axelsen, H.B., Glück R.: Reversible Representation and Manipulation of Constructor Terms in the Heap. In: Dueck, G.W., Miller, D.M. (eds.) RC 2013. LNCS, vol. 7948, pp. 96-109. Springer, Berlin, Heidelberg (2013)
19. Vieri, C.J.: Reversible Computer Engineering and Architecture. Doctoral dissertation, Massachusetts Institute of Technology, Dept. of Elec. Eng. and Comp. Sci. (1999)