Relativized Separation of Reversible and Irreversible Space-Time Complexity Classes[1]

Michael P. Frank[†] and M. Josephine Ammer[‡]

[†] *CISE Department, University of Florida; and* [‡] *EECS Department, University of California, Berkeley*

E-mail: mpf@cise.ufl.edu; mjammer@eecs.berkeley.edu

DEDICATED TO THE MEMORY OF JON BARWISE, LOGICIAN AND TEACHER

Reversible computing can reduce the energy dissipation of computation, which can improve cost-efficiency in some contexts. But the practical applicability of this method depends sensitively on the space and time overhead required by reversible algorithms. Time and space complexity classes for reversible machines match conventional ones, but we conjecture that the joint *space-time* complexity classes are different, and that a particular reduction by Bennett minimizes the *space-time product* complexity of general reversible computations. We provide an oracle-relativized proof of the separation, and of a lower bound on space for linear-time reversible simulations. A non-oracle proof applies when a read-only input is omitted from the space accounting. Both constructions model one-way function iteration, conjectured to be a problem for which Bennett's algorithm is optimal.

Several versions of this paper are available on the World-Wide Web at `http://www.cise.ufl.edu/~mpf/rc/memos/M06_oracle.html`.

Key Words: complexity theory, physics of computing, reversible computing, space-time complexity, incompressibility methods, relativized results, oracles, lower bounds

[1] Draft of May 23, 2001. Initial submission to *Information and Computation.*

CONTENTS

1. INTRODUCTION

This paper deals with *reversible* models of computation, which differ from conventional models in that all operations in a reversible computation must be (locally) invertible. Some discussion of the background and motivation for such models is warranted, for the benefit of readers who may be unfamiliar with them.

Importance of energy dissipation limits for future computing. Over the history of computing, shrinking bit-device (*e.g.*, transistor) sizes have resulted in an energy dissipation per bit-operation that has decreased roughly in proportion to the ever-increasing rates of bit-operations achievable in a machine of given cost. As a result, total power dissipation is not overwhelmingly greater today for a machine of given cost than it was in the early days of computing, even though the computational power of machines has increased by many orders of magnitude over the same period. (*E.g.*, Compare today's order-100 Watt, 1 GIPS desktop computers with order-10 Watt, 1 IPS hand-cranked mechanical calculators of a hundred years ago.) So, although power requirements have always been relevant to computer performance, they have never been the *overwhelmingly* dominant limiting factor—the total cost of the energy needed to run a computer over its operational lifetime has never been much greater than the cost of the machine itself.

However, in the future this situation could change, if continuing improvements in manufacturing techniques, such as nano-mechanical assembly [1] or molecular self-assembly result in manufacturing costs per bit-device continuing to decrease even after fundamental thermodynamic or technological lower limits on bit-energies have been reached, and if techniques that recycle bit energies are not applied. The Moore's law trend-line (see fig. 1) for bit energies reaches the absolute thermodynamic minimum of about $k_{\mathrm{B}}T \cong 4\times10^{-21}$ J (for room-temperature operation) by around the year 2035, so bit energies *must* start to level off at that time, if not earlier. Decreasing temperature would permit lower bit-energies, but this would not by itself reduce total system power dissipation when the cooling system is included, even if an ideal Carnot-cycle refrigerator is used.

So, unless manufacturing costs start to level off at the same time or earlier, or the cost of operation of power and heavy-duty cooling systems decreases radically, we may face the problem that although we might be able to afford to *build* nanocomputers with ever-increasing numbers of bit-devices, we might not be able to *operate* them for long at anywhere close to their peak performance. This problem has been pointed out before by nanotechnology visionaries Drexler, Merkle, Hall, and others [1, 2, 3].

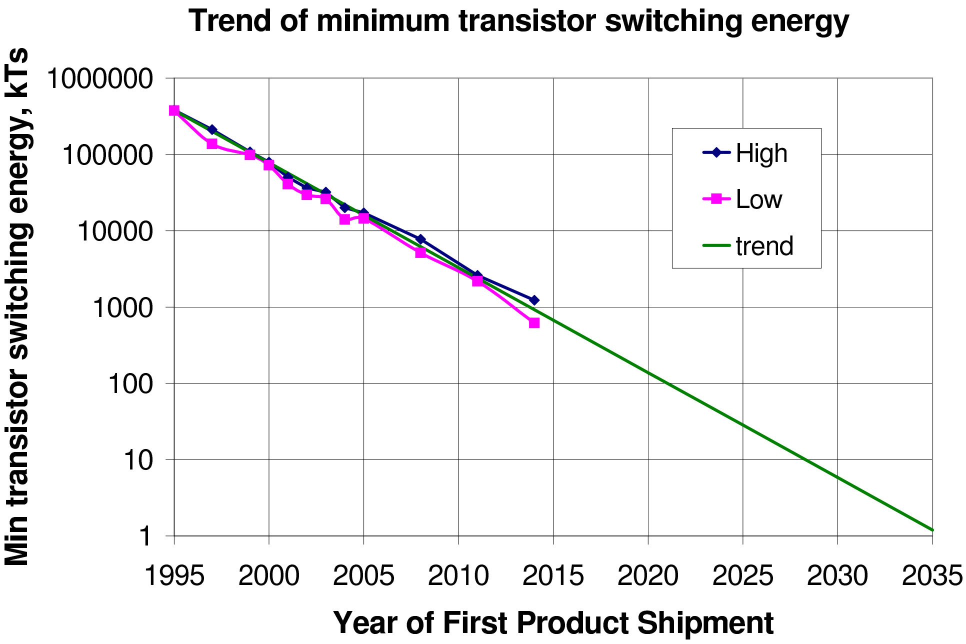

FIG. 1. This graph shows the $CV^2/2$ energy required to charge the gate of a minimum-sized transistor, computed from figures for power supply voltage, minimum transistor length, and gate oxide thickness listed in the 1994, 1997, and 1999 editions of the International Technology Roadmap for Semiconductors (formerly the National Semiconductor Technology Roadmap). Values for both high-performance and low-power design scenarios are shown. The 1999 roadmap specifies these quantities up through the year 2014. The trendline shown extrapolates the roadmap's trends, but the roadmap has itself historically been slightly more conservative than the technological reality of the steady Moore's Law trends which have held sway for more than 40 years.

Let us define the *power premium* P of a system to mean the ratio given by the lifetime cost of operation of a machine's power and cooling systems, divided by the cost of the computing hardware itself. Note that the power premium could potentially be much greater than 1 even today in specialized applications such as mobile computing (where there is a real, but difficult to quantify, added cost for power in the form of inconvenience to the user of carrying around heavy spare batteries) and space-borne systems (where the weight of solar panels and radiators incurs a high launch cost). But, as various bit-energy limits come into play in future decades, power premiums can be expected to increase for a wider range of computing systems, if cheaper manufacturing becomes available. Whenever $P \gg 1$, it can make sense to change the system design in ways that incur increased manufacturing costs in exchange for reduced power requirements, if as the sum of the system's manufacture and operation costs is thereby reduced.

Furthermore, under a reasonable set of physical assumptions (such as bounded heat flow density in the cooling system) one can show [4, 5, 6] that for a broad class of parallel computations that require frequent intercommunication between processing elements, asymptotically reducing energy dissipation per operation enables strictly superior asymptotic performance even if the cost of energy itself is negligible, since reducing heat flow enables packing devices more densely, with shorter round-trip communication delays.

Adiabatic computing techniques. For any given level of bit energies, the only way to avoid dissipating roughly one bit-energy with each bit-operation is to use *adiabatic* (*i.e.*, asymptotically thermodynamically reversible) physical mechanisms to conduct the bit-operation. It is a consequence of the second law of thermodynamics that such mechanisms are capable of performing only *logically reversible* (*i.e.*, bijective) transformations of a system's digital state [7, 8, 9]. Fortunately, it turns out that reversible operations are still computationally universal. Several fully-reversible universal processors have already been built [10, 11, 12].

By how large a factor can adiabatic/reversible techniques reduce the fraction of bit energies which is dissipated in practice? The precise answer for any given bit-device technology is, as of this writing, still unclear (although we are working on it). There are several independent limiting factors, including the rate of energy leakage of the bit-devices used, and the maximum efficiency (the Q quality factor) of the energy-recovering power supplies needed to drive adiabatic circuits.

In addition to these technological factors, there is also an important economic limiting factor. An adiabatic reduction in the energy dissipated per operation by a factor of F requires slowing devices down by a factor of (at least) F as well, so that F times as many devices are required to achieve a given level of raw processing performance. That is, the raw hardware efficiency or space-time efficiency (in physical units) of an adiabatic machine decreases in rough proportion to its increase in energy efficiency. As a result, adiabatics cannot cost-effectively reduce power dissipation by a factor greater than the power premium P, because this would raise the cost of the hardware to be greater than the original cost of the power, thereby nullifying any economic benefit of the decreased power consumption.

We do not yet know exactly which of these various limiting factors will dominate in a real adiabatic computing system implementation, because we do not yet have a sufficiently detailed adiabatic system design, including optimized logic and power supply designs, and accurate models of power supply dissipation and device leakage. But so far, we know of no fundamental reasons why these technological lower bounds on dissipation per bit-operation can not be reduced arbitrarily through "simple" engineering improvements, so it seems plausible that eventually, as power premiums increase and greater and greater degrees of reversibility can be implemented, the hardware efficiency of larger and larger reversible computations will come to be a dominant concern.

Complexity of reversible computations. It turns out that there is an important complexity-theoretic impact on this hardware efficiency issue. Beyond the immediate physical slowdown by F, the hardware efficiency of an adiabatic system will in general be further decreased as a result of the possibly greater *algorithmic* space-time cost (that is, bits of state required, times number of parallel state-update steps) for the reversible implementation of a specific computation or sub-computation within the machine. Many specific computations have reversible algorithms that incur no greater space-time cost than their traditional irreversible equivalents; some examples are mentioned in §9.5 of [6]. But what about other computations? In 1989, Bennett [13] proposed a general irreversible-to-reversible conversion technique that incurs only a modest polynomial increase in the spacetime cost for any computation. Although originally described as a software algorithm, it can be straightforwardly mapped to an equivalent logic-circuit construction.

Knowing of this polynomial reduction would be enough to satisfy many complexity theorists, but real-world concerns depend critically on such minutae as the degree of a polynomial, or the size of its constant coefficient. It could, for example, make the difference between adiabatic techniques yielding significant improvements in cost-efficiency in future generations of computing technology (or even in near-term power-limited applications), or, in contrast, yielding no improvements ever, depending on the absolute hardware efficiency of substantially reversible versions of the circuit algorithms required to implement a reasonable general-purpose microprocessor. As a result, the outcome of a more detailed study of reversible complexity theory is vitally relevant to planning future computing technologies.

Therefore, the question naturally arises as to whether Bennett's algorithm is the asymptotically optimal one for conversion of arbitrary irreversible algorithms to reversible ones, or whether a better algorithm (with still-reasonable constant factors) might be found. If the latter were to occur, the benefits of reversible computing might be much greater, and be realized much sooner, than would otherwise be the case.

Old and new complexity conjectures. Li and Vitányi conjectured in 1996 [14] that Bennett's algorithm was optimal, in terms of space complexity. Lange *et al.* disproved this conjecture in [15], but with a construction that incurred exponential increases in time complexity. However, we hypothesize that Bennett's algorithm *remains* optimal under the metric of *space-time* complexity, or space complexity *multiplied by* time complexity for a given algorithm, which is, anyway, the complexity measure that most directly relates to the goal of maximizing hardware efficiency (throughput per unit cost) in computer engineering.

Although our new conjecture is not yet proven, in this paper we provide suggestive evidence in support of it, in the form of an oracle construction that separates reversible and irreversible space-time complexity classes, together with lower bounds which are met by Bennett's algorithm.

Relevance of our relativized proof. We are well aware that relativized constructions have no general validity in drawing conclusions about non-relativized complexity classes, but we felt that presenting our construction might still be useful, for several reasons:

1. Our oracle is designed to be as realistic as possible: Although technically it is infeasible to physically realize exactly as defined, it is at least computable in principle. The oracle calls are also straightforwardly undo-able, as would be any real primitive operation in a reversible machine.
2. The structures of the oracle, and of the language that separates the classes, are designed to model a realistic type of real-world computation: Namely, the iteration of an arbitrary one-way function, such as a cryptographic hash function. We conjecture that if one-way functions do exist, then such iteration is a non-relativized example of a computational problem for which a spacetime-optimal reversible algorithm indeed results from Bennett's construction, and therefore our lower bounds still hold without the oracle. It is conceivable that some of the ideas or techniques used in our proof could be applied to this one-way-function iteration scenario, to prove a separation of the reversible and irreversible classes without resorting to an oracle, although we have not yet seen how to do so. But, perhaps someone with more familiarity with the theory of one-way functions would see the trick. Therefore, we thought it worthwhile to at least present this result to the community.
3. Finally, we feel that this entire field, which we call "physical computing theory," of working with new theoretical models of computation that are informed by increasingly-important physical constraints such as the energy cost of bit erasure, is deserving of more attention and we wish to help raise its visibility within the computer science community. The increasing need for models of computation that relate more closely to physics, and some proposed examples of such models, are discussed in more detail in [5] and in chapters 2, 5, and 6 of [6].

The results of this paper were first derived by the authors Frank and Ammer in 1997 at MIT, and were circulated in preprint form within the reversible computing community at that time.

2. TABLE OF SYMBOLS

The following table gives the meanings of most symbols used in this document. The third column gives the page number of the first appearance (often the definition) of the given symbol in the text. Please note that a symbol that has different meanings in different contexts within this paper correspondingly has multiple entries in this table.

See also table 2 on page 20 for the definitions of our order-of-growth notations.

Sym.	Meaning	p.
A	A particular self-reversible oracle that separates two given corresponding **TISP** and **RTISP** complexity classes. Modeled as a function $A : \mathcal{C} \to \mathcal{C}$, where $A = A^{-1}$.	17
a	In §6, a bit-string of length b used as an address to reference the memory I.	33
B	A particular permutation oracle that equates two given corresponding **TISP** and **RTISP** complexity classes.	17
b	Some arbitrary bit-string.	19
b	In §6, a word length $b \geq 0$. Also, for $n = b2^b$, $b(n) \equiv b$.	33
C	Transistor gate capacitance.	3
C_τ	Machine configuration of machine M_i resulting after τ steps of execution on input $\mathtt{0}^n$.	27
c	Centi-, 10^{-2}.	7
c	In §6, a presumed constant such that reversible machine M decides L in no more than $c + c\mathsf{S}$ space and $c + c\mathsf{T}$ time.	33
c_i	The constant $c_i \in \mathbb{N}$ appearing in the ith pair (M_i, c_i) in an enumeration of all pairs of reversible oracle-querying machines & such constants.	24
$\mathcal{C}$	The space of possible oracle tape contents.	18
$\mathfrak{C}$	A variable standing for an arbitrary complexity class.	17
D	For a given time point τ, a direction ("forwards" or "backwards") in which queries lie that cause most of the nodes pebbled at τ to be pebbled.	30
d	A description; a bit string that describes another bit string under some description system s.	19
ϵ	An arbitrarily small positive real number; $\epsilon \in \mathbb{R}$; $\epsilon > 0$; $\epsilon \to 0$.	15
F	Factor reduction in energy dissipation per operation from adiabatics.	4
F	A set of functions f having a particular asymptotic relation ($\boldsymbol{\Theta}$, $\mathcal{O}$, $\boldsymbol{\Omega}$, $\boldsymbol{\omega}$, $\mathbf{o}$)	
f, g	In localized contexts, these are often complexity functions $f, g : \mathbb{N} \to \mathbb{N}$ mapping input lengths (in bits) to some quantity that is roughly proportional to a complexity measure (*e.g.*, space or time) for worst-case inputs of the given length.	20
f	In our main proof, f is a partial *successor function* $f\colon \{\mathtt{0},\mathtt{1}\}^* \to \{\mathtt{0},\mathtt{1}\}^*$ defining a directed graph on bit-strings that is represented by our *graph oracle* A.	22
g	Gram; unit of mass originally defined as the mass of 1 cm^3 of water.	8
h	From given time point τ, how many nodes are pebbled because of queries in direction D?	30
I	A random-access, reversible, read-only memory of 2^b b-bit words.	33
IPS	One instruction per second; measure of performance.	

Sym.	Meaning	p.
i	Except in localized contexts, i in this paper means the index of one of the possible pairs (M_i, c_i) of reversible machines & constants.	24
i	In §6, a node index, $1 \le i \le t$.	34
J	Joule; the SI unit of energy, defined as 1 N · m.	2
j	Index of a query string, $1 \le j \le t$.	25
k	Kilo-, 10^3.	8
k	Number of sublevel repetitions in Bennett's 1989 algorithm [13].	15
k	Index of a query string, $1 \le k \le t$.	26
k	Largest number of pebbles which is insufficient to pebble 2^k nodes in Bennett's pebble game.	27
k_B	Boltzmann's constant, $\sim 1.4 \times 10^{-23}$ J/K.	2
L	For given S, T, and A, the separator language $L(A)$ shows $\mathbf{RTISP}(\mathsf{T},\mathsf{S})^A \not\supseteq \mathbf{TISP}(\mathsf{T},\mathsf{S})^A$; it belongs to the latter class but not the former.	23
L	In §6, for given S, T, this is the (non-relativized) language showing that $\mathbf{RTISP}(\mathsf{T},\mathsf{S}) \not\supseteq \mathbf{TISP}(\mathsf{T},\mathsf{S})$.	33
ℓ	A natural number giving the length of a bit-string.	19
M	In §6, this is an (oracle-less) reversible machine presumed to decide the language L within $c + c\mathsf{S}$ space and $c + c\mathsf{T}$ time.	33
M_i	The reversible oracle-querying machine in the ith pair (M_i, c_i) of an enumeration of all pairs of such machines and constant factors.	24
m	Meter; unit of length originally defined as $\frac{1}{4} \times 10^{-7}$ of Earth's circumference.	8
N	Newton; the SI unit of force, defined as 1 m · kg/s^2.	8
$\mathbb{N}$	The set of the natural numbers, $\{0, 1, 2, ...\}$.	19
NP	The complexity class of languages decidable in polynomial time by nondeterministic Turing machines.	17
n	Number of levels in Bennett's 1989 algorithm [13].	15
n	Abbreviation of n_in or n_i.	24
n_i	The length of input strings for which machine M_i fails to decide L within the space-time bounds determined by S, T, and constant c_i.	24
n_in	Number of bits in an input string.	19
NEXT	Given time point τ, $\textsc{next}(q_j)$ is the next query in M_i's history involving q_j before time τ.	28
O	Some arbitrary oracle. (In our context, a self-reversible one.)	17
$\mathcal{O}$	The "at most" order-of-growth operator $\mathcal{O}$ maps any function $g : \mathbb{N} \to \mathbb{N}$ to the set $F = \mathcal{O}(g)$ of functions f that are asymptotically at most proportional to g.	20
$\mathbf{o}$	The "less than" order-of-growth operator $\mathbf{o}$ maps any function $g : \mathbb{N} \to \mathbb{N}$ to the set $F = \mathbf{o}(g)$ of functions f that are asymptotically strictly less than g.	20

Sym.	Meaning	p.
P	The complexity class of languages decidable in polynomial time in most traditional models of computation (*e.g.*, Turing machines).	17
P	Power premium; ratio of lifetime power cost to logic hardware cost.	4
p	The number of nodes that are pebbled at time τ.	28
PREV	Given time point τ, PREV(q_j) is the previous query in M_i's history involving q_j before time τ.	28
Q	Quality factor; ratio between energy transfered and energy dissipated during a system's cycle of operation.	4
q	A possible oracle query string, *i.e.*, an oracle tape contents, *i.e.*, a bit string, *i.e.*, a graph node identifier, *i.e.*, a graph node.	24
q'	An alternative final node in the chain, replacing our original choice of q_t.	26
q_0	Initial query string in a node chain. $q_0 = 0^{\mathsf{S}}$.	26
q_j	A particular query string in the sequence $q_1, \ldots, q_t$ formed from x, or if $j = 0$, see q_0 above.	25
$\mathbb{R}$	The (nonconstructive) set of all "real" numbers.	7
RTISP	**RTISP**(T, S) is the complexity class of problems solvable by reversible algorithms taking time $\mathcal{O}(\mathsf{T})$ and space $\mathcal{O}(\mathsf{S})$.	17
$r(I)$	In §6, the 1-bit result for a given input memory I, found by doing $\lfloor \mathsf{T}/\mathsf{S} \rfloor$ iterated pointer dereferences in I starting at address 0^b.	33
S	Space bounding function $\mathsf{S} : \mathbb{N} \to \mathbb{N}$, mapping an input length n_{in} to an upper bound $\mathsf{S}(n_{\text{in}})$ on the number of temporary state bits used at any time in processing any input of length n_{in}.	13
S'	A larger space bounding function, $\mathsf{S}' \prec \mathsf{S} \log(\mathsf{T}/\mathsf{S})$, which is *still* not enough to allow reversible machines to compute the same functions in linear time (in our oracle model).	32
s	Second; unit of time originally defined as 1/86,400 of Earth's solar day.	8
s	A description system. (In §6, a particular one that we are defining.)	19
s_i	The particular description system used to select the incompressible string x that defines the chain of nodes that foils M_i.	25
T	Absolute temperature.	2
T	Time bounding function $\mathsf{T} : \mathbb{N} \to \mathbb{N}$, mapping an input length n_{in} to an upper bound $\mathsf{T}(n_{\text{in}})$ on the number of state-update "ticks" to be used in processing any input of length n_{in}.	12
T'	Actual number of steps $\mathsf{T}' \le c_i + c_i \mathsf{T}(n)$ taken before halting in the case of a machine that does not exceed the time bound.	27
TISP	**TISP**(T, S) is the complexity class of problems solvable by ordinary algorithms taking time $\mathcal{O}(\mathsf{T})$ and space $\mathcal{O}(\mathsf{S})$.	17
t	$t(n)$ is the number of nodes, size $\mathsf{S}(n)$ each, in a chain of nodes that will take time $\Theta(\mathsf{T}(n))$ to traverse on a serial machine. $t(n) \equiv \lfloor \mathsf{T}(n)/\mathsf{S}(n) \rfloor$.	25

Sym.	Meaning	p.
τ	$0 \leq \tau \leq \mathsf{T}'$, an index of the machine configuration of M_i (running on the oracle graph) that results after τ steps (primitive operations) have taken place.	28
$\Delta\tau_j$	The number of steps between time τ and the query in direction D that causes node q_j to be pebbled at time τ.	30
$\mathbf{\Theta}$	The "exactly" order-of-growth operator $\mathbf{\Theta}$ maps any function $g : \mathbb{N} \to \mathbb{N}$ to the equivalence class $F = \mathbf{\Theta}(g)$ of functions f that are asymptotically proportional to g.	20
V	Logic swing voltage; absolute voltage difference between $\mathsf{0}$ and $\mathsf{1}$ logic levels.	3
w	An arbitrary bit-string input to our oracle-querying machines.	24
w_i	In §6, length-b bit string number i, where $1 \leq i \leq t$, in a linked list of bit strings formed from x.	34
x	A bit-string, $\lvert x\rvert = \mathsf{T}(n)$, incompressible in description system s_i, to be broken up into a chain of node bit-strings $q_0, \ldots, q_t$.	25
x'	x with a substring spliced out. (See explanations in text.)	26
y	A bit-string that is described by another bit-string d under some description system s.	19
z	The maximum length over all oracle queries asked by machines $M_0, \ldots, M_{i-1}$ running within their respective bounds $c_0, \ldots, c_{i-1}$ when given respective inputs $\mathsf{0}^{n_0}, \ldots, \mathsf{0}^{n_{i-1}}$.	24
$\mathbf{\Omega}$	The "at least" order-of-growth operator $\mathbf{\Omega}$ maps any function $g : \mathbb{N} \to \mathbb{N}$ to the set $F = \mathbf{\Omega}(g)$ of functions f that are asymptotically no less than proportional to g.	20
$\boldsymbol{\omega}$	The "more than" order-of-growth operator $\boldsymbol{\omega}$ maps any function $g : \mathbb{N} \to \mathbb{N}$ to the set $F = \boldsymbol{\omega}(g)$ of functions f that are asymptotically strictly greater than g.	20

3. REVIEW OF PREVIOUS RESULTS IN REVERSIBLE COMPUTING THEORY

In this section we briefly review previous results in the theory of computability and of computational complexity relating to reversible computation.

Reversible models of computation. Reversible models of computation can be easily defined in general as models of computation in which the transition function between machine configurations has a single-valued inverse. In other words, the directed graph showing allowed transitions between states has in-degree 1. In this paper we will always deal with machines that are deterministic, so that the configuration graph always has out-degree one as well. See figure 2.

3.1. Computability in reversible models

Unbounded-space reversible machines are Turing-universal. In his 1961 paper [7], Landauer had already pointed out that arbitrary irreversible computations could be embedded into reversible ones by simply saving a record of all the information that would otherwise be thrown away (*cf.* §3 of [7]). This observation

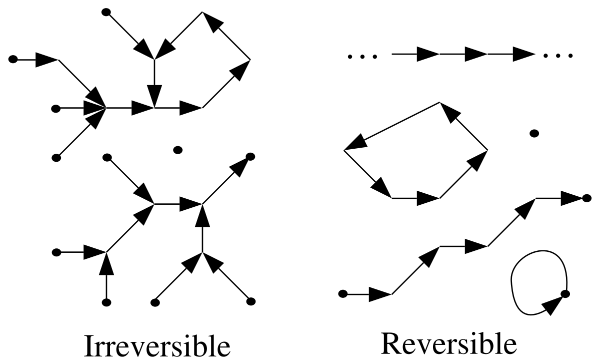

FIG. 2. Machine configuration graphs in (deterministic) reversible and irreversible models of computation.
In the configuration graphs of irreversible machines, configurations may have many different predecessor configurations. In reversible models of computation, each configuration may have at most one predecessor. The configuration graph therefore consists of disjoint loops and chains, which may be infinite. In both reversible and irreversible models we may, if we wish, permit configurations having 0 predecessors (initial states) and/or 0 successors (final states).

makes it obvious that reversible machines with unbounded memory can certainly compute all the Turing-computable functions.

We call this idea, of embedding an irreversible computation into a reversible one by saving a history of garbage, a "Landauer embedding," since Landauer seems to have been the first to suggest it.

A certain model of reversible finite automata is especially weak. In contrast to Landauer's result, in 1987 Pin [16] investigated reversible *finite* automata, which he defined as machines with fixed memory reading an unbounded-length *one-way* stream of data, and found that such cannot even decide all the regular languages, which means that technically they are strictly less powerful than normal irreversible finite automata.

So there are stream recognition tasks computable by an irreversible machine with fixed memory that no purely reversible machine with fixed memory can compute, given an external one-way stream of input. We should note, however, that this incapacity may be due to the non-reversible nature of the input flow, rather than to the reversibility of the finite automaton itself. Conceivably, if a finite reversible machine was permitted to read backwards as well as forwards through its read-only input, and perform some sort of "unread" operations, it might then be able to recognize any regular language. But we have not investigated that possibility in detail.

In any event, the finite automaton model is not generally considered to express the salient features of computation, since real computers are not designed as state machines with small fixed numbers of states, but rather as unbounded-memory machines that can be given as much external storage as needed to perform a particular task, and can explore an enormous state space, one that grows exponentially with the number of storage bits that are available. So, for the rest of this paper, we consider only models of computation that permit access to increasingly large amounts of memory as input sizes increase. For such machines, Landauer's result overrides

Pin's, and pure computability is no longer an issue. So we turn to questions of computational complexity.

3.2. Time complexity in reversible models

In a theoretical computer science context, "time complexity" T for serial machine models means essentially the number of primitive operations performed. Landauer's suggestion (*cf.* §3 of [7]) of embedding each irreversible operation into a reversible one makes it clear that the number of such operations in a reversible machine need not be larger than the number for an irreversible machine, as was demonstrated more explicitly in later embeddings by Lecerf [17] and Bennett [18]. So under the time complexity measure by itself, reversibility does not hurt.

Can a reversible machine perform a task using *fewer* computational operations than any irreversible machine? Obviously not, if we take reversible operations to just be a special case of irreversible operations. However, it is interesting to note that, physically speaking, actually it is the converse that is true: so-called "irreversible" operations, implemented physically, are really just a special case of reversible operations, since physics is believed to be *always* reversible at a low level. The implications of this fact for *physical* time complexity are discussed in more detail in [19]. But, using the usual computer-science definition of time as the number of *computational* operations required, clearly reversible machines can be no more "time"-efficient than irreversible ones.

Although Lecerf and Bennett explicitly discussed their time-efficient reversible simulations only in the context of Turing machines, the approach is easily generalized to any model of computation in which we can give each processing element access to an unbounded amount of auxiliary unit-access-time stack storage. For example, Toffoli [20] describes how one can use essentially the same trick to create a time-efficient simulation of irreversible cellular automata on reversible ones, by using an extra dimension in the cell array to serve as a garbage stack for each cell of the original machine.

3.3. Entropic complexity in reversible models

The original point of reversibility was not to reduce time but to reduce energy dissipation, or in other words entropy production. Can this be done by reversible machines? In 1961 Landauer [7] argued that it could not, since if we cannot get rid of the "garbage" bits that are accumulated in memory, they just constitute another form on entropy, no better in the long term than the kind produced if we just irreversibly dissipated those bits into physical entropy right away.

Lecerf reversal. However, in 1963, Lecerf [17] formally described a construction in which an irreversible machine was embedded into a reversible one that first simulated the irreversible machine running forwards, then turned around and simulated the irreversible machine in reverse, uncomputing all of the history information and returning to a state corresponding to the starting state. If anyone familiar with Landauer's work had noticed Lecerf's paper in the 1960's, it would have seemed tantalizing, because here was Lecerf showing how to reversibly get rid of the garbage information that was accumulated in Landauer's reversible machine in lieu of entropy. So maybe the entropy production can be avoided after all!

Unfortunately, Lecerf was apparently unaware of the thermodynamic implications of reversibility; he was concerned only with determining whether certain questions about reversible transformations were decidable. Lecerf's paper did not address the issue of how to get useful results out of a reversible computation. In Lecerf's embedding, by the time the reversible machine finishes its simulation of the irreversible machine, any outputs from the computation have been uncomputed, just like the garbage. This is not very useful!

The Bennett trick. Fortunately, in 1973, Charles Bennett [18], who was unaware of Lecerf's work but knew of Landauer's, independently rediscovered Lecerf reversal, and moreover added the ability to retain useful output. The basic idea was simple: one can just reversibly copy the desired output into available memory before performing the Lecerf reversal! As far as we know, this simple trick had not previously occurred to anyone.

Bennett's idea suddenly implied that reversible computers could in principle be *more* efficient than irreversible machines under at least *one* cost measure, namely entropy production. To compute an output on an irreversible machine, one must produce an amount of entropy roughly equal to the total number of (irreversible) operations performed; whereas the reversible machine in principle can get by with *no* new entropy production, and with the accumulation of only the desired output in memory.

Entropy proportional to speed. Unfortunately, absolutely zero entropy generation per operation is achievable in principle only in the ideal limit of a perfectly-isolated ballistic (frictionless) system, or in a Brownian-motion-based system that makes zero progress forwards through the computation on average, and takes $\mathbf{\Theta}(n^2)$ expected time before visiting the nth computational step. In useful systems that progress forwards at a positive constant speed, the entropy generation per operation appears to be, at minimum, proportional to the speed. (We do not yet know how fundamental this relationship is, but it appears to be the case empirically.) A cost analysis that takes both speed and entropy into account will need to recognize this tradeoff. We do this in [4, 5] and in ch. 6 of [19].

3.4. Space complexity in reversible models

In computational complexity, "space complexity" refers to the number S of memory cells that are required to perform a computation.

Initial estimates of space complexity. As Landauer pointed out [7], his simple strategy of saving all the garbage information appears to suffer from the drawback that the amount of garbage that must be stored in digital form is as large as the amount of entropy that would otherwise have been generated. If the computation performs on average a constant number of irreversible bit-erasures per computational operation, then this means that the memory usage becomes proportional to the number of operations. This means a large asymptotic increase in memory usage for many problems; up to exponentially large. Even if the garbage is uncomputed using Lecerf reversal, this much space will still be needed temporarily during the computation.

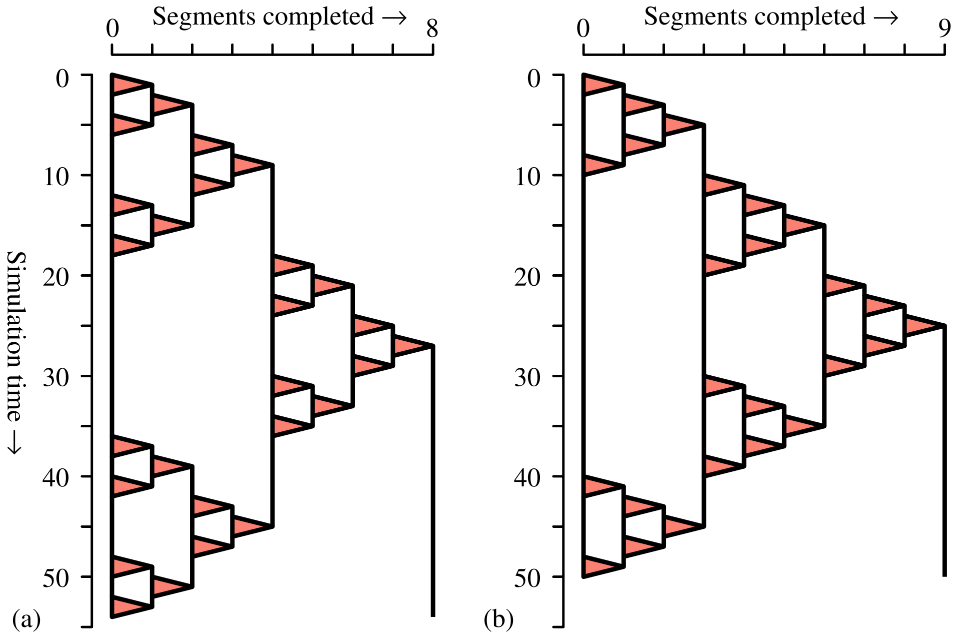

FIG. 3. Illustration of two versions of Bennett's 1989 algorithm for reversible simulation of irreversible machines. Diagram (a) illustrates the version with $k = 2$, diagram (b) the version with $k = 3$. (See text for explanation of k.)

In both diagrams, the horizontal axis indicates which segment of the original irreversible computation is being simulated, whereas the vertical axis tracks time taken by the simulation in terms of the time required to simulate one segment. The black vertical lines represent times during which memory is occupied by an image of the irreversible machine state at the indicated stage of the irreversible computation, whereas the shaded areas within the triangles represent memory occupied by the storage of garbage data for a particular segment of the irreversible computation being simulated.

Note that in (b), where $k = 3$, the 9th stage is reached after only 25 time units, whereas in (a) 27 time units are required to only reach stage 8. But note also that in (b), at time 25, five checkpoints (after the initial state) are stored simultaneously, whereas in (a) at most four are stored at any given time. This illustrates the general point that higher-k versions of the Bennett algorithm run faster, but require more memory.

Bennett's pebbling algorithm. In 1989, Bennett [13] introduced a new, more space-efficient reversible simulation for Turing machines. This new algorithm involved doing and undoing various-sized portions of the computation in a recursive, hierarchical fashion. Figure 3 is a schematic illustration of this process. We call this the "pebbling" algorithm because the algorithm can be seen as a solution to a sort of "pebble game" or puzzle played on a one-dimensional chain of nodes, as described in detail by Li and Vitányi '96 [21]. (Compare figure 3(a) with fig. 9 on page 27.) We will discuss the pebble game interpretation and its implications in more detail in §5.

The overall operation of the algorithm is as follows. The irreversible computation to be simulated is broken into fixed-size segments, whose run time is proportional to the memory required by the irreversible machine. The first segment is reversibly simulated using a Landauer embedding (as in [7]). Then the state of the irreversible machine being simulated is checkpointed using the Bennett trick of reversibly copying it to free memory. Then, we do a Lecerf reversal (§3.3, p. 12) to clean up the garbage from simulating the first segment.

We proceed the same way through the second segment, starting from the first checkpoint, to produce another checkpoint. After some number k of repetitions of this procedure, all the previous checkpoints are then removed by reversing everything done so far except the production of the final checkpoint. Now we have only a single checkpoint which is k segments along in the computation. We repeat the above procedure to create another checkpoint located another k segments farther along, and then again, and again k times, then reverse everything again at the higher level to proceed to a point where we only have checkpoint number k^2 in memory. The procedure can be applied indefinitely at higher and higher levels.

In general, for any number n of recursive higher-level applications of this procedure, k^n segments of irreversible computation are be simulated by $(2k-1)^n$ reversible forwards-and-backwards simulations of a single segment, while having at most $n(k-1)$ intermediate checkpoints in memory at any given time [13].

The upshot is that if the original irreversible computation takes time T and space S, then the reversible simulation via this algorithm takes time $O(\mathsf{T}^{1+\epsilon})$ and space $O(\mathsf{S}\log\mathsf{T}) = O(\mathsf{S}^2)$. As k increases, the ϵ approaches 0 (very gradually), but unfortunately the constant factor in the space usage increases at the same time [22].

Li and Vitányi '96 [21] proved that Bennett's algorithm (with $k=2$) is the most space-efficient possible pebble-game strategy for reversible simulation of irreversible machines. This result is central to our proof.

Crescenzi and Papadimitriou '95 [23] later extended Bennett's technique to provide space-efficient reversible simulation of *nondeterministic* Turing machines as well.

3.4.1. Achieving linear space complexity

Bennett's results stood for almost a decade as the most space-efficient reversible simulation technique known, but in 1997, Lange, McKenzie, and Tapp [15] showed how to simulate Turing machines reversibly in linear space—but using worst-case exponential time. Their technique is very clever, but simple in concept: Given a configuration of an irreversible machine, they show that one can reversibly enumerate its possible predecessors. Given this, starting with the initial state of the irreversible machine, the reversible machine can traverse the edges of the irreversible machine's tree of possible configurations in a reversible "Euler tour." (See figure 4.) This is analogous to using the "right-hand rule" technique (move forward while keeping your right hand on the wall) to find the exit of a planar non-cyclical maze. The search for the final state is kept finite, and the space usage is kept small, by cutting off exploration whenever the configuration size exceeds some limit. Unfortunately, the size of the pruned tree, and thus the time required for the search, is still, in the worst case, exponential in the space bound.

In a very recent result, Buhrman *et al.* '01 [24] show that it is actually possible to view the Bennett and Lange-McKenzie-Tapp techniques as extreme points on a continuous spectrum of simulation algorithms having intermediate asymptotic space and time requirements. (Unfortunately, all of the intermediate algorithms in this tradeoff space still suffer at least a polynomial increase in spacetime complexity.)

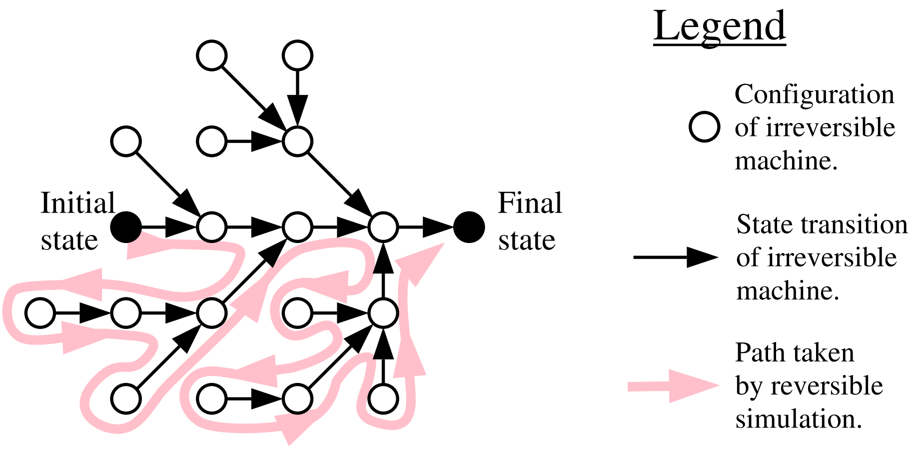

FIG. 4. Illustration of an Euler tour of an irreversible machine's computation tree. Although the tree has branches, the Euler tour is itself both forward- and reverse-deterministic, and so can be traversed in purely reversible fashion, using no more space than is needed to keep track of the current irreversible machine configuration [15].

As with Bennett's techniques, the Lange-McKenzie-Tapp technique was defined explicitly only in terms of Turing machines, but it is easily generalized to many different models of computation.

The above time and space complexity results for reversible simulation are very interesting in themselves, but to our knowledge, before our work no one had previously addressed the specific question of whether a single reversible simulation could run both in linear time like Bennett's 1973 technique *and* in linear space like the newer Lange *et al.* technique. Li and Vitányi's analysis [21] of Bennett's 1989 algorithm [13] leads to our proof in this paper that if such an ideal simulation exists, it would not relativize to oracles, or work in cases where the space bound is much less than the input length.

3.5. Miscellaneous developments

Here, we mention in passing a couple of other miscellaneous developments in reversible computing theory.

Coppersmith and Grossman (1975, [25]) proved a result in group theory which implies that reversible boolean circuits only 1 bit wider than a fixed-length input can compute arbitrary boolean functions of that input.

Toffoli (1977, [20]) showed that reversible cellular automata can simulate irreversible ones in linear time using an extra spatial dimension. Fredkin and Toffoli developed much reversible boolean-circuit theory (1980–1982, [26, 27, 28]).

4. GENERAL DEFINITIONS

In this section we set forth some general definitions that we will use in our proof, but that may also be useful for future proofs in reversible computing theory. Later, in section 5.2, we will give some additional, more specific definitions that are not anticipated to be widely useful outside of this paper.

4.1. Space-time complexity classes

Given any reversible model of computation (*e.g.*, reversible Turing machines), and given any computational space and time bounding functions $\mathsf{S}(n_{\text{in}}), \mathsf{T}(n_{\text{in}})$, we define (following Bennett [13] the *reversible space-time* S, T *complexity class*, abbreviated $\mathbf{RTISP}(\mathsf{T}, \mathsf{S})$, to be the set of languages that are accepted by reversible machines that take worst-case space of $\mathcal{O}(\mathsf{S}(n_{\text{in}}))$ memory bits and worst-case time $\mathcal{O}(\mathsf{T}(n_{\text{in}}))$ ticks, where n_{in} is the length of the input. Similarly, we define the (unrestricted) *space-time* S, T *complexity class*, abbreviated $\mathbf{TISP}(\mathsf{T}, \mathsf{S})$, to be the set of languages accepted in that same order of space and time on the corresponding normal machine model, without the restriction on the in-degree of the transition graph. For oracle-relativized complexity classes, we use the notation $\mathfrak{C}^O$, as is standard in complexity theory, to indicate the class of problems that can be solved by the machines that define the class $\mathfrak{C}$ if they are allowed to query oracle O.

We want to know whether $\mathbf{RTISP}(\mathsf{T}, \mathsf{S}) \stackrel{?}{=} \mathbf{TISP}(\mathsf{T}, \mathsf{S})$, for all S, T, in normal sorts of serial computational models such as multi-tape Turing machines or RAM machines.

Unfortunately, we have found this question, in its purest form, very difficult to definitively resolve. We do not see any general way to simulate normal machines on reversible machines without suffering asymptotic increases in either the time or space required. But neither do we know of a language that can be proven to require extra space or time to recognize reversibly in ordinary machine models. The difficulty is in constructing a proof that rules out all reversible algorithms, no matter how subtle or clever.

But is the $\mathbf{RTISP} \stackrel{?}{=} \mathbf{TISP}$ question truly difficult to resolve, or have we just been unlucky in our search for a proof? Often in computational complexity theory, we find ourselves unable to prove whether or not two complexity classes (for example, $\mathbf{P}$ and $\mathbf{NP}$) are equivalent. Traditionally (as in Baker *et al.* [29]), one way to indicate that such an equivalence might really be difficult to prove is to show that if the machine model defining each class is augmented with the ability to perform a new type of operation (a query to a so-called "oracle"), then the classes may be proven either equal or unequal, depending on the behavior of the particular oracle. This shows that any proof equating or separating the two classes must make use of the fact that normal machine models are only capable of performing a particular limited set of primitive operations. Otherwise, we could just add the appropriate oracle call as a new primitive operation, and invalidate the supposed proof.

In this section we will demonstrate, for any given S, T in a large class, an oracle A relative to which we prove $\mathbf{RTISP}(\mathsf{T}, \mathsf{S})^A \neq \mathbf{TISP}(\mathsf{T}, \mathsf{S})^A$, for the case of serial machine models with a certain kind of oracle interface. For these same S, T we have not yet found an alternative oracle B for which $\mathbf{RTISP}(\mathsf{T}, \mathsf{S})^B = \mathbf{TISP}(\mathsf{T}, \mathsf{S})^B$, except for irreversible oracles which make the equivalence trivial. It may be that no reversible oracle that equates the classes exists, but this is uncertain.

4.2. Reversible oracle interface

First, we define an oracle interface that allows a reversible machine to call an oracle. Ordinarily, oracle queries are irreversible, and thus impossible in reversible machines. For example, a bit of the oracle's answer cannot just overwrite some storage location, because regardless of whether the location contained `0` or `1` before

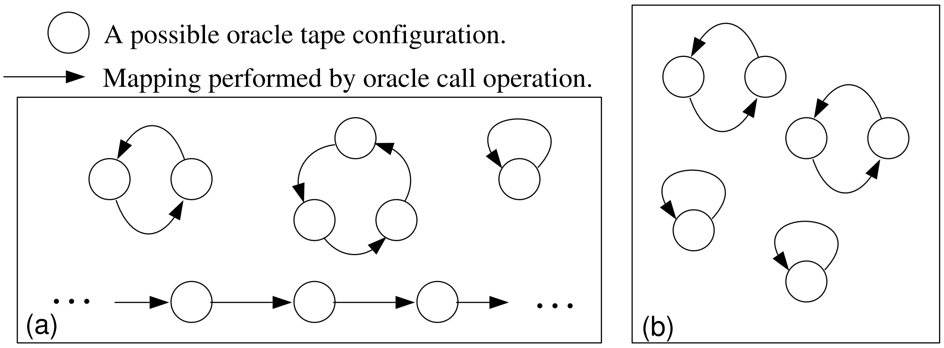

FIG. 5. Illustration of the structure of (a) a permutation oracle, and (b) a self-reversible permutation oracle.
In either case, the oracle call operation replaces the old contents of the oracle tape with new contents according to a transition function $A : \mathcal{C} \to \mathcal{C}$ that is a permutation mapping—a bijective function—over the space $\mathcal{C}$ of possible tape contents. The bijectivity of this function means that a call to a permutation oracle is always a reversible operation. After an oracle call, the previous oracle tape contents can be uniquely determined by applying the inverse mapping A^{-1}. In self-reversible oracles, $A = A^{-1}$.

the oracle call, after the call it would contain the oracle's answer. The resulting configuration would thus have two predecessors, and the machine would be irreversible.

Our reversible oracle-calling protocol is as follows. Machines will have reversible read and write access to a special *oracle tape* which has a definite start, unbounded length, and is initially clear. At any time, the machine is allowed to perform an *oracle call*, a special primitive operation which in a single step replaces the entire contents of the oracle tape with new contents, according to some fixed invertible mapping $A : \mathcal{C} \to \mathcal{C}$ over the space $\mathcal{C}$ of possible tape contents. The function A is called a *permutation oracle*. Further, if A is its own inverse, $A = A^{-1}$, it will be called *self-reversible*. Presented more formally:

Definition 4.1. A *permutation oracle* A is an invertible (bijective) function $A : \mathcal{C} \to \mathcal{C}$, where $\mathcal{C}$ is the space of possible contents of a semi-infinite *oracle tape*.

Definition 4.2. A *self-reversible (permutation) oracle* is a permutation oracle A such that $A = A^{-1}$.

In the below, we will deal only with self-reversible oracles. Self-reversibility ensures that machines can easily undo oracle operations, just as they can easily undo their own internal reversible primitives. (Since primitive operations such as bit-operations are by definition finite operations over small state-spaces, if those operations are invertible then their inverses must be easy to compute.) If oracle calls were much harder to undo than to do, then the oracle model would be unlikely to teach us anything meaningful about real machines.

4.3. ST-constructibility

In order for our proof to go through, we will need to restrict our attention to space and time functions $\mathsf{S}(n_{\text{in}}), \mathsf{T}(n_{\text{in}})$ which are *ST-constructible*, meaning that given any input of length $n_{\rm in}$, an irreversible machine can construct binary representations of the numbers $\mathsf{S}(n_{\rm in})$ and $\mathsf{T}(n_{\rm in})$ using only space $\mathcal{O}(\mathsf{S}(n_{\rm in}))$ and time $\mathcal{O}(\mathsf{T}(n_{\rm in}))$. We state here without proof that many reasonable pairs of functions are indeed ST-constructible. For example, $\mathsf{S} = n_{\rm in}^2$, $\mathsf{T} = n_{\rm in}^3$ can both be computed in time $\mathcal{O}(\log^2 n_{\rm in})$ plus $\mathcal{O}(n_{\rm in})$ to count the input bits, and space $\mathcal{O}(\log n_{\rm in})$ plus $\mathcal{O}(n_{\rm in})$ if we include the input.

Next, we need some basic definitions to support the notion of incompressibility that will be crucial to the proof of our theorem. The following definition and lemma follow the spirit of the discussions of incompressibility in Li and Vitányi's excellent book on Kolmogorov complexity [30].

4.4. Description systems and compressibility

DEFINITION 4.3. A *description system* s is any function $s\colon \{0,1\}^* \to \{0,1\}^*$ from bit-strings to bit-strings, that is, from *descriptions* to the bit-strings they describe. We say that a bit-string d *describes* bit-string y in description system s if $s(d) = y$. We say that a bit-string y is *compressible* in description system s if there is a shorter bit-string that describes it; *i.e.* if there exists a string d such that $s(d) = y$ and $|d| < |y|$, where the notation $|b|$ denotes the number of bits in bit-string b.

LEMMA 4.1 (Existence of incompressible strings). *For any description system s, and any string length $\ell \in \mathbb{N}$, there is at least one bit-string y of length ℓ that is not compressible in s.*

Proof (Trivial counting argument). There are 2^ℓ bit-strings of length ℓ, but there are only $\sum_{i=0}^{\ell-1} 2^i = 2^\ell - 1$ descriptions that are shorter than ℓ bits long. Each description d can describe at most one bit string of length ℓ, namely the string $s(d)$ if that string's length happens to be ℓ. Therefore there must be at least one remaining bit-string y of length ℓ that is not described by any shorter description. ■

In our main proof, we will be selecting incompressible strings from a series of computable description systems.

4.5. Notational conventions

In the following, we will often abbreviate the space and time function values $\mathsf{S}(n_{\rm in})$ and $\mathsf{T}(n_{\rm in})$ by just S and T, respectively; likewise for other functions of $n_{\rm in}$. For comparing orders of growth, we will use both the standard $\boldsymbol{\Theta}$, $\mathcal{O}$, $\boldsymbol{\Omega}$, $\mathbf{o}$, $\boldsymbol{\omega}$ notations, and our mnemonic $\asymp$, $\precsim$, $\succsim$, $\prec$, $\succ$ notation, defined in table 2.

5. MAIN THEOREM

Preliminary discussion. In this section we prove that reversible machine models require higher asymptotic space-time complexity on some problems than corresponding irreversible models, if a certain new reversible black-box operation (a self-reversible oracle A) is made available to both models. Thus, no *completely*

TABLE 2
Asymptotic order-of-growth notation. In addition to reviewing the standard notation, we introduce a simplified, more mnemonic notation that will be convenient in some contexts.

Cryptic standard notation	A more mnemonic notation	Mathematical definition; English explanation
$f = \boldsymbol{\Theta}(g)$ or $f \in \boldsymbol{\Theta}(g)$	$f \asymp g$	$\exists c_1, c_2, n_0 > 0 : \forall n > n_0 : 0 < c_1 g(n) < f(n) < c_2 g(n)$; f has the same asymptotic order of growth as g.
$f = \mathcal{O}(g)$ or $f \in \mathcal{O}(g)$	$f \precsim g$	$\exists c, n_0 > 0 : \forall n > n_0 : 0 < f(n) < cg(n)$; f has a lower asymptotic order of growth than g.
$f = \boldsymbol{\Omega}(g)$ or $f \in \boldsymbol{\Omega}(g)$	$f \succsim g$	$\exists c, n_0 > 0 : \forall n > n_0 : 0 < cg(n) < f(n)$; f has a greater asymptotic order of growth than g.
$f = \mathbf{o}(g)$ or $f \in \mathbf{o}(g)$	$f \prec g$	$\forall c > 0 : \exists n_0 > 0 : \forall n > n_0 : 0 < f(n) < cg(n)$; f has a strictly lower asymptotic order of growth than g.
$f = \boldsymbol{\omega}(g)$ or $f \in \boldsymbol{\omega}(g)$	$f \succ g$	$\forall c > 0 : \exists n_0 > 0 : \forall n > n_0 : 0 < cg(n) < f(n)$; f has a strictly greater asymptotic order of growth than g.

general technique can exist for simulating irreversible machines on reversible ones with no asymptotic overhead.

However, the new primitive operation that we defined in order to make this proof go through is not itself physically realistic. The operation implements a computable function, but the operation is modeled as taking constant ($\boldsymbol{\Theta}(1)$) time to perform independent of the size of its input, which violates physical locality and the asymptotically very large number of steps that it would take to compute the operation using the algorithm that corresponds directly to the operation's definition.

Therefore, technically, even given our proof, it is still an open question whether a perfectly efficient simulation technique might still exist that works in the case of reversible machines simulating irreversible machines that are composed only of primitives that are physically realistic.

Incidentally though, if one wishes to progress to *complete* physical realism, then to be completely fair, one should take into account the physical time and space costs associated with removing the physical entropy produced by irreversible operations from a machine, when comparing reversible and irreversible machine models. We do this in [19] and conclude that under certain reasonable assumptions, a variety of physically realistic reversible models are actually asymptotically strictly *more* spacetime-efficient on some problems than are the corresponding irreversible models, although an extremely large scale of machine may be required to realize that particular theoretical benefit.

The encroaching issue of lower limits on bit energies is more important. As we mentioned in §1, the exact magnitude of the purely *computational* asymptotic overheads incurred by reversible operation has an important role to play in helping to make an accurate comparison between the potential efficiency of reversible and irreversible machine designs in particular technologies. It is a key element that drastically affects the shape of the overall tradeoff function between energy costs and hardware costs in partially-adiabatic machine design spaces.

Below, we will prove our results in both oracle-relativized and non-oracle forms for serial (uniprocessor) machines. The oracle results cover a large family of possible asymptotic bounds on the joint space and time requirements of computations. For all bounding functions within this family, we show that there exist an oracle and a language such that the language is decidable within the given bounds by serial machines that can query the oracle only if the machines are *ir*reversible. This result is non-trivial (compared to Pin's, for example) because the individual oracle calls are themselves reversible and easy to undo.

In section 6, a similar result, not involving an oracle, covers cases where the space bound is much smaller than the length of the randomly (and reversibly) accessible input. Corollaries to both the oracle and non-oracle results give loose lower bounds on the amount of extra space a reversible machine will require to decide the language within the same time bounds as the irreversible machine, although one should keep in mind that this approach of meeting the time bounds will not necessarily minimize the real costs corresponding to the space-time *product*.

Another contribution of our proof is to illustrate ways to use incompressibility arguments in analyzing reversible machines. It is conceivable that similar techniques might increase the range of reversible and irreversible space-time complexity classes that we can separate without resorting to the oracle.

5.1. Statement of main theorem

THEOREM 5.1. (Relative separation of reversible and irreversible space-time complexity classes.) *Let* S,T *be any two non-decreasing functions over the non-negative integers. Then both of the following are true:*

(a) If $\mathsf{S} \gtrsim \mathsf{T}$ *or* $\mathsf{T} \gtrsim 2^{\mathsf{S}}$*, then* $\mathbf{RTISP}(\mathsf{T},\mathsf{S})^O = \mathbf{TISP}(\mathsf{T},\mathsf{S})^O$ *for any self-reversible oracle* O*.*

(b) If $\mathsf{S} \prec \mathsf{T} \prec 2^{\mathsf{S}}$*, and if* S,T *are* ST*-constructible, then there exists a computable, self-reversible oracle* A *such that* $\mathbf{RTISP}(\mathsf{T},\mathsf{S})^A \neq \mathbf{TISP}(\mathsf{T},\mathsf{S})^A$*.*

Proof.

Part (a). (Cases $\mathsf{S} \gtrsim \mathsf{T}$ and $\mathsf{T} \gtrsim 2^{\mathsf{S}}$.) First, if $\mathsf{S} \succ \mathsf{T}$, then obviously we have both $\mathbf{RTISP}(\mathsf{T},\mathsf{S})^O = \mathbf{RTISP}(\mathsf{T},\mathsf{T})^O$ and $\mathbf{TISP}(\mathsf{T},\mathsf{S})^O = \mathbf{TISP}(\mathsf{T},\mathsf{T})^O$ simply because in time T no more than $\mathsf{S} \asymp \mathsf{T}$ memory cells can be accessed on a machine that performs $\Theta(1)$ operations per time step. Similarly, if $\mathsf{T} \succ 2^{\mathsf{S}}$, then $\mathbf{RTISP}(\mathsf{T},\mathsf{S})^O = \mathbf{RTISP}(2^{\mathsf{S}},\mathsf{S})^O$ and $\mathbf{TISP}(\mathsf{T},\mathsf{S})^O = \mathbf{TISP}(2^{\mathsf{S}},\mathsf{S})^O$, because no computation using only S bits of memory can run for more than 2^{S} steps without repeating. So part (a) reduces to proving $\mathbf{RTISP}(\mathsf{T},\mathsf{S})^O = \mathbf{TISP}(\mathsf{T},\mathsf{S})^O$ only for the case where $\mathsf{S} \asymp \mathsf{T}$ or $\mathsf{T} \asymp 2^{\mathsf{S}}$.

From here, the result follows due to the existing relativizable simulations. When $\mathsf{S} \asymp \mathsf{T}$, Bennett's simple reversible simulation technique [18] can be applied because it takes time $\mathcal{O}(\mathsf{T})$ and space $\mathcal{O}(\mathsf{T})$. Similarly, when $\mathsf{T} \asymp 2^{\mathsf{S}}$ the simulation of Lange *et al.* [15] can be used because it takes time $\mathcal{O}(2^{\mathsf{S}})$ and space $\mathcal{O}(\mathsf{S})$. Both techniques can be easily seen to relativize to any self-reversible oracle O. Thus, in both cases, any irreversible machine can be simulated reversibly in $\mathcal{O}(\mathsf{T})$ and space $\mathcal{O}(\mathsf{S})$, and therefore $\mathbf{RTISP}(\mathsf{T},\mathsf{S})^O = \mathbf{TISP}(\mathsf{T},\mathsf{S})^O$.

Part (b). (Case $\mathsf{S} \prec \mathsf{T} \prec 2^{\mathsf{S}}$.) Here, we give only an outline of the full proof of part (b), which will be fleshed out in §§5.2–5.4 below. **Proof outline:** We will construct A to be a permutation oracle that can be interpreted as specifying an infinite directed graph of nodes with outdegree at most 1. We will also define a corresponding language-recognition problem, which will be to report the contents of a node that lies T/S nodes down an incompressible linear chain of nodes that have size-S identifiers, starting from a node that is determined by the input length. The oracle will be explicitly constructed via a diagonalization, so that for each possible reversible machine, there will be a particular input for which our oracle makes that particular reversible machine take too much space or else get the wrong answer. In the cases where the reversible machine takes too much space, we will prove this by equating the machine's operation with the "pebble game" for which Li and Vitányi [21] have already proven lower bounds, and by showing that if the machine does not take too much space, then we can build a shorter description of the chain of nodes using the machine's small intermediate configurations, thus contradicting our choice of an incompressible chain.

■

Before we can develop the proof of part (b) in full detail, we need some more definitions specialized to our needs.

5.2. Specialized definitions

DEFINITION 5.1. A *graph oracle* is a self-reversible permutation oracle with the following property: There exists a partial function $f\colon \{0,1\}^* \to \{0,1\}^*$, called a *successor function*, such that for any bit string (node) $b \in \{0,1\}^*$ for which f is defined, the oracle's permutation function maps the tape contents b to the tape contents $b\#f(b)$, and also maps $b\#f(b)$ back to b, where # is a special separator character in the oracle tape alphabet. For all tape contents x not of either of these forms, the oracle's permutation function maps them to themselves. See fig. 6.

Remark. The name "graph oracle" for this concept is really over-general; our graph oracles are capable of embodying only graphs of a special type, namely directed graphs in which all nodes are named by bit-strings and have out-degree 1. The unique node that is adjacent from node q is given by the successor function $f(q)$.

Given that we will be working only with graph oracles, we can now specify an oracle by specifying just the successor function f that it embodies. But before we

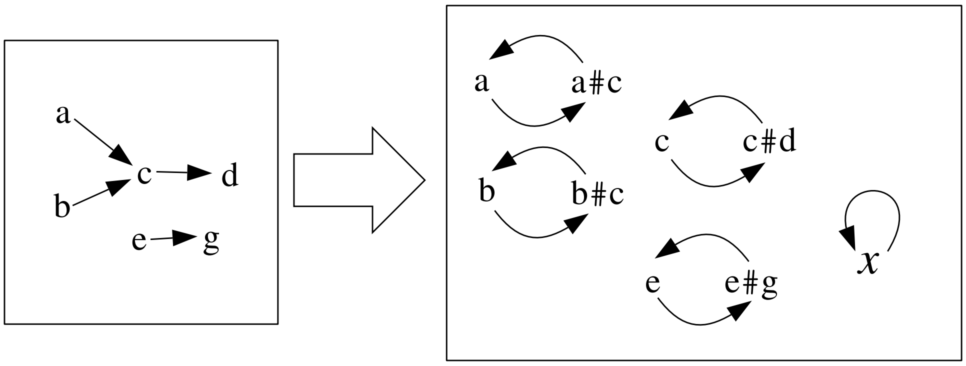

FIG. 6. Encoding outdegree-1 directed graphs in self-reversible permutation oracles. Letters stand for nodes represented as bit-strings, except for x which represents any other bit-string not explicitly shown. The # is a special separator character.

On the left, we show an example of an outdegree-1 directed graph with bit-string nodes abbreviated a,b,c,d,e,g. The graph function f gives the successor of each node: $f(\mathrm{a}) = \mathrm{c}$, $f(\mathrm{c}) = \mathrm{d}$, etc. This f is a partial function; *e.g.* $f(\mathrm{d})$ is undefined. For each edge in this graph, there is a corresponding pair of strings that are mapped to each other by the self-reversible oracle. To represent the edge a $\rightarrow$ c, for example, the permutation oracle maps tape contents "a" to "a#c" and maps "a#c" back to "a". Any other string x (including those for terminal nodes of the graph) is simply mapped to itself. In this way the permutation oracle allows easily and reversibly looking up a node's successor, or uncomputing a node's successor given the node and its successor. But finding a node's predecessor(s), given just the node itself, is designed to be hard. Thus the oracle call resembles the reversible computation of a "one-way" invertible function that is easy to compute, but whose inverse is difficult to compute.

actually construct the special oracle A that proves theorem 5.1, let us define, relative to A, the language that we claim separates $\mathbf{RTISP}(\mathsf{T},\mathsf{S})^A$ from $\mathbf{TISP}(\mathsf{T},\mathsf{S})^A$.

Definition 5.2. Given two ST-constructible functions $\mathsf{S}(n)$, $\mathsf{T}(n)$, and graph oracle A with successor function f, we define the *separator language* $L(A)$ to be the language decided by the irreversible machine described by algorithm 1 in figure 7.

The algorithm is essentially this: Given a string of length n, construct a string of zeros of length $\mathsf{S}(n)$. Treat this string as a node identifier, and use oracle queries to proceed down its chain of successors for up to $\lfloor \mathsf{T}/\mathsf{S} \rfloor$ nodes. Finally, return the first bit of the final node's bit-string identifier.

We will be explicitly constructing the successor function f so that it always returns a string of the same length as its input. Given the corresponding oracle, algorithm 1 obviously requires only space $\mathcal{O}(\mathsf{S})$ and time $\mathcal{O}(\mathsf{T})$ on on irreversible machine in any standard serial model of computation. (Recall that S,T are ST-constructible.) Therefore the language $L(A)$ will be in the class $\mathbf{TISP}(\mathsf{T},\mathsf{S})^A$.

In §5.3, we will show how to construct f so that the language $L(A)$ will not be computable by any reversible machine that takes space $\mathcal{O}(\mathsf{S})$ and time $\mathcal{O}(\mathsf{T})$. The way we will do this is to make each of the node identifiers be a different incompressible string. Intuition suggests that the only way to decide $L(A)$ is to actually follow the entire chain of nodes, to see what the final one is. But having obtained a node's successor, the reversible machine cannot easily get rid of its incompressible records of the prior nodes. The graph oracle provides no convenient way to compute f^{-1} and find a node's predecessor, even if the successor function f happens to be invertible. Thus (as we will show) the reversible machine will tend to

Algorithm 1 (SEPARATOR(w)).

Given input string w,
    Let $n = |w|$; compute $\mathsf{S} = \mathsf{S}(n), \mathsf{T} = \mathsf{T}(n)$.
    Let bit-string $b = \mathtt{0}^{\mathsf{S}}$.
    Repeat the following, $t = \lfloor \mathsf{T}/\mathsf{S} \rfloor$ times:
        Write b on the oracle tape, and call the oracle A.
        If result is of the form $b\#c$, with c a bit-string,
            assign $b \leftarrow c$ (note that $c = f(b)$),
        else, quit loop early.
    Accept iff $b[0] = \mathtt{1}$.

FIG. 7. Irreversible algorithm defining the language $L(A)$ that separates $\mathbf{TISP}(\mathsf{T}, \mathsf{S})$ from $\mathbf{RTISP}(\mathsf{T}, \mathsf{S})$, relative to our reversible oracle A. The essence of this algorithm is simply to interpret A as a graph oracle, construct an initial node (which is dependent on the input string), and follow the directed path leading away from the initial node for a certain number of steps.

accumulate records of previous nodes, of size $\mathsf{S}(n_{\text{in}})$ each, and thus, for sufficiently long enough chains, it will take more than a constant factor times $\mathsf{S}(n_{\text{in}})$ space. The reversible machine could conceivably find and uncompute predecessor nodes by searching them all exhaustively, but this would take too much time.

The situation with this oracle language resembles the non-oracle problem of iterating a one-way function, *i.e.* an invertible function whose inverse much is harder to compute than the function itself (*e.g.*, MD5). Public-key cryptography depends on the (unproven, but empirically reasonable) assumption that some functions are one-way. The same assumption might allow us to show that $\mathbf{RTISP}(\mathsf{T}, \mathsf{S}) \neq \mathbf{TISP}(\mathsf{T}, \mathsf{S})$ without an oracle, by using a one-way function instead.

5.3. Oracle construction

We now construct a particular oracle A (given any appropriate T,S) and prove that $L(A) \notin \mathbf{RTISP}(\mathsf{T}, \mathsf{S})^A$.

First, fix some standard enumeration of all reversible oracle-querying machines. The enumeration is possible because reversible Turing machines, for example, can be characterized by local syntactic restrictions on their transition function, as in Lange *et al.*, so we can enumerate all machines and pick out the reversible ones. Let $(M_1, c_1), (M_2, c_2), \ldots$ be this enumeration dovetailed together with an enumeration of the positive integers. If a given machine always runs in space $\mathcal{O}(\mathsf{S})$ and time $\mathcal{O}(\mathsf{T})$ then it will eventually appear in the enumeration paired with a large enough c_i so that the machine M_i takes space less than $c_i + c_i\mathsf{S}(n_{\text{in}})$ and time less than $c_i + c_i\mathsf{T}(n_{\text{in}})$ for any input length n_{in}.

We will construct the oracle A so that each machine M_i will fail to decide $L(A)$ within these bounds. When considering M_i, $f(q)$ will have already been specified for all oracle queries q asked by machines $M_1, M_2 \ldots, M_{i-1}$ when given certain inputs of lengths $n_1, n_2, \ldots, n_{i-1}$, respectively. Now, choose n_i (henceforth called n), the input length for which our oracle definition will foil M_i, to be such that $\mathsf{S}(n)$ is greater than the maximum length z of any of those earlier machines' oracle

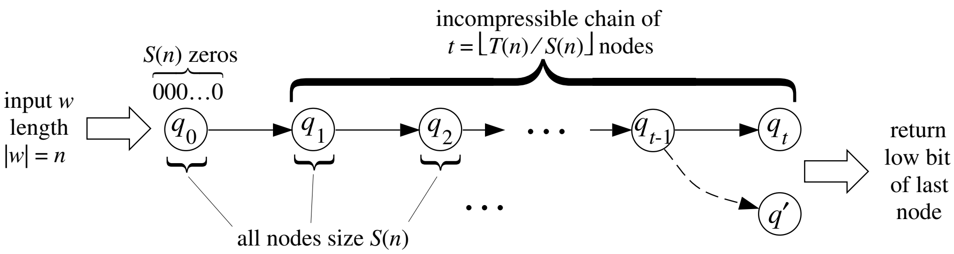

FIG. 8. The problem graph defined by our oracle for inputs of size n. The "correct answer" is just the first bit of the final node q_t. If the reversible machine M_i that we are trying to foil happens to get the right answer, but never asks for the successor of node q_{t-1}, we redefine q_{t-1}'s successor to be a new node q' having a different initial bit.

TABLE 3

Constraints on the input length n_i chosen to foil machine M_i running within bounds determined by S, T, and c_i.

Constraint on n_i	Introduced on
$\mathsf{S}(n_i) > z$	p. 24
$\lvert(j,k,x')\rvert < \lvert x\rvert$	p. 26
$\frac{1}{2}2^{\mathsf{S}(n_i)} > c_i + c_i\mathsf{T}(n_i)$	p. 26
$\lvert(j,\Delta\tau_j,k_j)\rvert < \frac{1}{4}\mathsf{S}(n_i)$	p. 30
$t(n_i) \geq 2^{4(c_i+1)}$	p. 32
$\mathsf{S}(n_i) \geq c_i$	p. 32

TABLE 4

Description formats needed in description system s_i.

Description format	Explained on
(j,k,x')	p. 26
(j,x')	p. 26
$(C_\tau, D, x'$, h triples $(j,\Delta\tau_j,k_j)$, extra bits)	p. 30
$(j,\Delta\tau_j,k_j,x')$	p. 31

queries. Some other lower bounds on the size of n will be mentioned as we go along, and are summarized in table 3.

Later we will specify a description system s_i, summarized in table 4, based on M_i, c_i, the value of n, and all the $f(q)$ values defined so far (for bit-strings smaller than $\mathsf{S}(n)$). The description system will be a total computable function, *i.e.*, there is an algorithm that computes $s_i(d)$ for any d and always halts. We will use this description system to define $f(q)$ for bit-strings q of length $\mathsf{S}(n)$, as follows:

Let x be a bit-string of length $\mathsf{T}(n)$ that is incompressible in description system s_i (to be defined as we go along). This x will be used as the sequence of size-$\mathsf{S}(n)$ node identifiers that will define our graph for inputs of size n.

Break x up into a sequence of $t(n) \equiv \lfloor \mathsf{T}/\mathsf{S} \rfloor$ bit-strings of length $\mathsf{S}(n)$ each; call these our graph nodes or *query strings* $q_1, \ldots, q_t$. (Due to the floor operation, up to $\mathsf{S} - 1$ bits may be left over; these aren't used in any query strings.) We will design our description system s_i so that all the q_j's must be different. We accomplish this by allowing descriptions of the form (j,k,x'), where j and k are the indices of two equal nodes $q_j = q_k$, $j < k$, and x' is x with the q_k substring spliced out. The description system would be defined to generate x from such a description by simply looking up the string q_j in x' and inserting a copy of it in the kth position. The indices j and k would take $\mathcal{O}(\log(\mathsf{T}/\mathsf{S}))$ space, which is $\mathcal{O}(\log \mathsf{T})$ space, which is $o(\mathsf{S})$ space, whereas we are saving $\mathsf{S}(n)$ space by not explicitly including the repetition of q_j. Therefore as long as n is sufficiently large, the total length of this description of x would be less than $|x|$. With x being incompressible in a description system that permits such descriptions, we know that $q_1, \ldots, q_t$ includes no repetitions.

Now we can specify exactly how the oracle defines our problem graph for inputs of size n, as follows. Define query string $q_0 = \mathtt{0}^{\mathsf{S}}$ (a string of S $\mathtt{0}$-bits). Provisionally, set $f(q_{j-1}) = q_j$ for all $1 \le j \le t$. These assignments are possible since all the q_j's are different, as we just proved. (They also must be different from q_0, but this is easy to ensure as well, using descriptions of the form (j, x').) Given these assignments, all strings of length n are in the language $L(A)$ if and only if $q_t[0] = \mathtt{1}$ (where $q_t[0]$ means the first bit of q_t), due to the earlier definition of $L(A)$. (Definition 5.2.)

Suppose temporarily that our oracle definition were completed by letting f remain undefined over all strings w for which we have not yet specified $f(w)$. (*I.e.*, let $A(w) = w$ for these strings.) Under that assumption, simulate M_i's behavior on the input $\mathtt{0}^n$. If M_i runs for more than $c_i + c_i T$ steps, then it takes too much time, and we are through addressing it. Otherwise, M_i either accepts ($\mathtt{1}$) or rejects ($\mathtt{0}$). If this answer is different from $q_t[0]$, then M_i already fails to accept the language $L(A)$, and we are through with it.

Alternatively, suppose M_i's answer is correct with the given q_j's and it halts within $c_i + c_i T$ steps. But now, suppose that M_i never asked any query that was dependent on our choice of $f(q_{t-1})$ during its run on input $\mathtt{0}^n$. That is, suppose M_i never asked either query q_{t-1} or query $q_{t-1}\mathtt{\#}q_t$. In that case, let us change our definition of $f(q_{t-1})$ as follows, to change the correct answer to be the opposite of what M_i gave. Let q' be a bit-string that is independent of all queries made by M_i in that simulation, and whose first bit is the opposite of M_i's answer. To ensure such strings exist, note there are $\frac{1}{2}2^{\mathsf{S}}$ bit-strings of length S having the desired initial bit, but M_i can make at most $c_i + c_i T$ queries since that is its running time. We know $\mathsf{T} \prec 2^{\mathsf{S}}$, so with sufficiently large n, $\frac{1}{2}2^{\mathsf{S}} > c_i + c_i\mathsf{T}$, and we can find our node q'. Now, given q', we change $f(q_{t-1})$ to be q'. This cannot possibly affect the behavior of M_i since it never asked about $f(q_{t-1})$. But the correct answer is changed to the first bit of q', the new node number t in the chain. Thus with this new partial specification of f, M_i fails to correctly decide $L(A)$, and we can go on to foil other machines.

Finally, suppose M_i does ask query q_{t-1}. We now show how to complete the definition of our description system s_i, source of our incompressible x, so that if M_i does ask query q_{t-1}, then it must at some point take more than $c_i + c_i\mathsf{S}$ space.

To do this, we show that M_i can always be interpreted as following the rules of Bennett's reversible "pebble game," introduced in [13] and analyzed by Li and Vitányi in [21].

Pebble game rules. The game is played on a linear chain of nodes, which we will identify with our query strings $q_1, \ldots, q_t$. At any time during the game some set

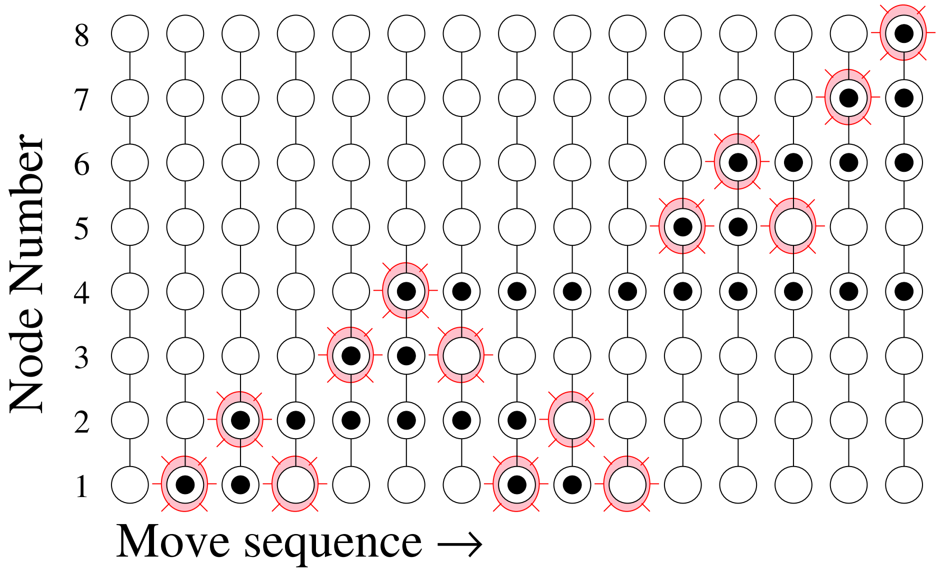

FIG. 9. Bennett's reversible pebble game strategy. Highlights point out the move made at each step. (Compare with fig. 3(a), page 14, rotated 90°.)

A node q_j can be pebbled or unpebbled only if it is node q_1 or if the previous node q_{j-1} is pebbled. The strategy invented by Bennett [13], illustrated here, was shown by Li and Vitányi to be optimal [14] in terms of the number of pebbles required. But even with this optimal strategy, to pebble node 2^k we must at some time have more than k nodes pebbled. In this example, we reach node $2^3 = 8$ but must use 4 pebbles to do so. (After pebbling node 8, we can remove all pebbles by undoing the sequence of moves.) The fact that a constant-size supply of pebbles can only reach upwards along the chain a constant distance is crucial to our proof.

of nodes is *pebbled.* Initially, no nodes are pebbled. At any time, the *player* (in our case, M_i) may, as a move in the game, change the pebbled vs. unpebbled status of node q_1 or any node q_j for which the previous node q_{j-1} is pebbled. Only one such move may be made at a time.

The idea of the pebbled set is that it corresponds to the set of nodes that is currently "stored in memory" by M_i. (We will show how to make this correspondence explicit.) We will show that pebbling or unpebbling node q_j will require querying the oracle with query string q_{j-1} or $q_{j-1}\#q_j$, respectively. The goal of the pebble game is to eventually place a pebble on the final node q_t. This corresponds to the fact (already established) that M_i must at some point ask query q_{t-1}, or the oracle we are constructing will foil it trivially.

Li and Vitányi's analysis of the pebble game [21] showed that no strategy can win the game for 2^k nodes or more without at some time having more than k nodes pebbled at once. We will show that our machine M_i and its space usage can be modeled using the pebble game, so that for some sufficiently large n, the space required to store the necessary number of pebbled nodes will exceed M_i's allowable storage capacity $c_i + c_i\mathsf{S}$.

For the oracle A as defined so far, consider the complete sequence of configurations of M_i given input $\mathtt{0}^n$, notated $C_0, C_2, \ldots, C_{\mathsf{T}'}$, where $\mathsf{T}' \leq c_i + c_i\mathsf{T}$ is M_i's total running time, in terms of the number of primitive operations (including oracle calls) performed.

Now, we need a couple of slightly more complex definitions.

DEFINITION 5.3. *(Previous and next queries involving a node.)* For any time point τ, where $0 \leq \tau \leq \mathsf{T}'$, and for any node q_j in the chain of nodes $q_1, \ldots, q_t$, define *the previous query involving* q_j (written $\text{PREV}(q_j)$) to mean the most recent oracle query in M_i's history before time τ in which the query string (the one that is present on the oracle tape at the start of the query) is either q_{j-1}, $q_{j-1}\#q_j$, q_j, or $q_j\#q_{j+1}$. There may of course be no such query, in which case $\text{PREV}(q_j)$ does not exist. Similarly, define *the next query involving* q_j (written $\text{NEXT}(q_j)$) to mean the most imminent such query in M_i's future after time τ.

DEFINITION 5.4. *(A node being pebbled at a point in time.)* Node q_j *is pebbled at time* τ iff at time τ either:

(a) $\text{PREV}(q_j)$ exists and is either

- (a.1) q_{j-1},
- (a.2) q_j, or
- (a.3) $q_j\#q_{j+1}$, or

(b) $\text{NEXT}(q_j)$ exists and is

- (b.1) q_j,
- (b.2) $q_j\#q_{j+1}$, or
- (b.3) $q_{j-1}\#q_j$.

(With the exception that the final node q_t is only considered pebbled in cases (a.1) and (b.3).)

Note that this definition implies that q_j is *not* pebbled iff both $\text{PREV}(q_j) = q_{j-1}\#q_j$ (or nonexistent) and $\text{NEXT}(q_j) = q_{j-1}$ (or nonexistent).

Figure 11 illustrates the intuition behind this definition using the graphical notation introduced in fig. 10. This graphical notation is especially nice because it evokes the image of playing the pebble game or running Bennett's algorithm (compare fig. 11 with figs. 9 and 3).

The times at which a node is to be considered "pebbled" during a machine's execution are indicated by the solid horizontal lines on 11. These times are determined, according to definition 5.4 above, solely by the arrangement of triangles (representing oracle queries, see fig. 10) on the chart. Each vertex of a triangle generates a line of pebbled times for the corresponding node, extending horizontally away from the triangle until it hits another triangle. Query string 0 is never considered pebbled because it is not considered to be a node.

5.4. Main Space-Bounding Lemma

Let p denote the number of distinct nodes out of $q_1, \ldots, q_t$ that are pebbled at time τ. We now lower bound the size of C_τ, *i.e.* M_i's space usage at time τ.

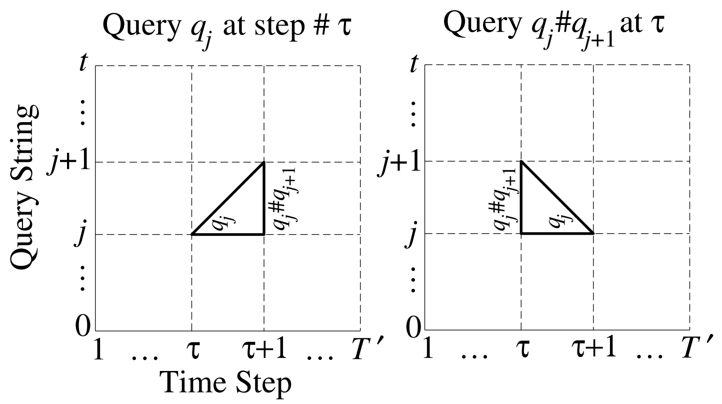

FIG. 10. Triangle representation of oracle queries.
The shape and direction of the triangle is meant to evoke the fact that at the times just before and after an oracle query, the oracle tape contains the shorter string q_j at one of the times, and the longer string $q_j\#q_{j+1}$ at the other time. The set of triangles defines the set of pebbled nodes at any time, as illustrated in figure 11.

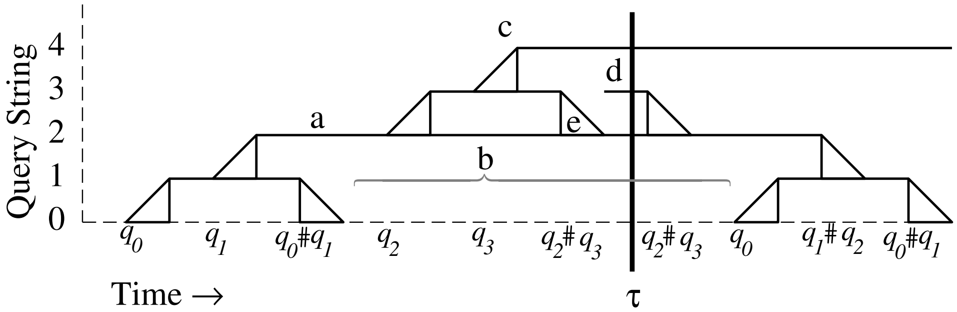

FIG. 11. Visualizing the definition of the set of pebbled nodes.
The times at which a node is pebbled (indicated by solid horizontal lines on the chart) are determined, by definition, solely by the identities and timing of oracle queries and the corresponding arrangement of triangles (see fig. 10) on the chart. Each vertex of a triangle generates a line of pebbled times for the corresponding node, extending horizontally away from the triangle until it encounters another triangle. (Except query string 0 is never pebbled, because it is not considered to be a node.)
The above example shows a pattern of queries similar to the one that would occur if one tried to apply Bennett's [13] optimal pebble game strategy. (Compare with figs. 9 and 3.)
Node 2 is considered pebbled at time (a) both because of the previous and next queries (triangles) involving node 2. Node 1 is not pebbled at times (b) because the previous and next queries are $q_0\#q_1$ and q_0 respectively. Node 4 is pebbled at all times after (c) because even though there is no next query involving node 4, the previous query involving node 4 exists and is of the right form (q_3). Node 3 is pebbled at time (d) because although the previous query (e) is of the wrong form ($q_2\#q_3$), the next query is okay.
Query (e) does not change the set of pebbled nodes and so is not considered to be a move in the pebble game. All the other queries are considered to be pebbling or unpebbling moves in the pebble game, depending on the direction of the corresponding triangle.
In the machine configuration C_τ at time τ, nodes 2, 3, and 4 are pebbled. But note that the query string for node 2 can be found by simulating the machine backwards from time τ until query (e), and reading q_2 off of the oracle tape. And if q_3 is given, we can continue simulating backwards until we get to time (c), and read q_4 off the oracle tape as well. The ability to perform this sort of simulation, for any arrangement of triangles, either forwards or backwards in time as needed to find out more than a constant number of the pebbled nodes is what makes our incompressibility argument work.

TABLE 5
Size accounting for the description format d used to prove lemma 5.1.

Component of description d	Length
C_τ	$\leq \frac{1}{4}p\mathsf{S}$
D	1 bit
x'	$(t-h)\mathsf{S} \leq (t-\frac{1}{2}p)\mathsf{S}$
h triples $(j, \Delta\tau_j, k_j)$	$h \cdot \mathbf{o}(\mathsf{S}) = \mathbf{o}(\frac{1}{4}p\mathsf{S}-1)$
extra bits	$\mathsf{T} - t\mathsf{S}$
TOTAL	$< \mathsf{T} = \lvert x \rvert$ (for sufficiently large n)

LEMMA 5.1 (Minimum space required to pebble p nodes). *Given the preceding definitions,* $|C_\tau| > \frac{1}{4}p\mathsf{S}$.

Remark. The constant $\frac{1}{4}$ here is somewhat arbitrary, and with straightforward generalization of the below proof this constant could instead be replaced by $\frac{1}{2}-\epsilon$ for any constant $\epsilon > 0$. We conjecture, but have not proven, that it could also be replaced by any constant $1-\epsilon$.)

Proof. Suppose C_τ were no larger than $\frac{1}{4}p\mathsf{S}$ bits. Then we can show that x (the sequence of all q_j's) is compressible to a shorter description d, which we will now specify. Our description system s_i will be defined to process descriptions of the required form.

First, note that for each node q_j that is pebbled at time τ, that node is pebbled either because of the previous query involving q_j, because of the next query involving q_j, or both. Therefore, either at least $\frac{1}{2}p$ nodes are pebbled because of their previous query, or at least $\frac{1}{2}p$ nodes are pebbled because of their next query. Let D be a direction (forwards or backwards) from time τ in which one can find queries causing $h \geq \frac{1}{2}p$ nodes to be pebbled.

We now specify the shorter description d that describes x. It will contain an explicit description of C_τ, which by our assumption is no longer than $\frac{1}{4}p\mathsf{S}$. It will also specify the direction D and contain a concatenation x' of all $t-h$ of the nodes q_j (for $1 \leq j \leq t$) that are *not* pebbled because of queries in direction D. The size of x' will be $(t-h)\mathsf{S}$. For each of the h nodes q_j that *are* pebbled because of a query in direction D, the description d will contain the node index j and an integer $\Delta\tau_j$ giving the number of steps from step τ to the time of the query. Also we include a short tag k_j indicating which of the 3 possible cases of queries causes the node to be pebbled. Each of the indices j takes space $\mathcal{O}(\log t) \prec \log \mathsf{T} \prec \mathsf{S}$, and similarly each $\Delta\tau_j$ takes space $\mathcal{O}(\log \mathsf{T}) \prec \mathsf{S}$. The tag is constant size. Thus for sufficiently large n, all $h \leq p$ of the $(j, \Delta\tau_j, k_j)$ tuples together take less than $\frac{1}{4}p\mathsf{S}$ space. The total space so far is less than $t\mathsf{S}$. If $t\mathsf{S} < \mathsf{T}$, then x will contain some additional bits beyond the concatenation of $q_1 q_2 \ldots q_t$, in which case d includes those extra bits as well. The total length of d will still be less than $\mathsf{T} = |x|$, as demonstrated in table 5.

We now demonstrate that the description d is sufficient to reconstruct x, and give an algorithm for doing so. The function computed by this algorithm tells how our description system s will handle descriptions of the form outlined above.

The algorithm will work by simulating M_i's operation in direction D starting from configuration C_τ, and reading the identifiers of pebbled nodes from M_i's simulated oracle tape as it proceeds. We can figure out which oracle queries correspond to which nodes by referring to the stored times $\Delta\tau_j$ and tags k_j. Once we have extracted the identifiers of all nodes pebbled in direction D, we print all the nodes out in the proper order.

As an example, refer again to fig. 11. In the machine configuration marked at time τ, nodes 2, 3, and 4 are pebbled. But note that the query string for node 2 can be found by simulating the machine backwards from time τ until query (e), and reading q_2 off of the oracle tape. And if q_3 is known, we can continue simulating backwards until we get to time (c), and read q_4 off the oracle tape as well. The ability to perform this sort of simulation, for any arrangement of triangles, either forwards or backwards in time as needed to find out at least half of the pebbled nodes is what makes our incompressibility argument work. The algorithm is described and verified in more detail in §7.

Given d, the algorithm produces x, and with n chosen large enough, the length of the description will be smaller than x itself, contradicting the assumption of x's incompressibility relative to s. Therefore for these sufficiently large n, all configurations in which p nodes are pebbled must actually be larger than $\frac{1}{4}p\mathsf{S}$. This completes the proof of lemma 5.1. ■

Interpreting any M_i as playing the pebble game. Now, given the definition of the set of pebbled nodes from earlier (defn. 5.4), it is easy to see how M_i's execution history can be interpreted as the playing of a pebble game. Whenever M_i performs a query q_j and node q_{j+1} was not already pebbled immediately prior to this query, we say that M_i *is pebbling node* q_{j+1} as a move in the pebble game. Similarly, whenever M_i performs a query $q_j\#q_{j+1}$ and node q_{j+1} is not pebbled immediately after this query, we say that M_i *is unpebbling node* q_{j+1}. All other oracle queries and computations by M_i are considered as pauses between pebble game moves of these two forms. For example, in fig. 11, query (e) (the first occurrence of $q_2\#q_3$) is not considered a move in the pebble game, since it doesn't change the set of pebbled nodes as defined by definition 5.4.

It is obvious that under the above interpretation, all moves must obey the main pebble game rule, *i.e.* that the pebbled status of node q_j can only change if $j = 1$ or if node q_{j-1} is pebbled during the change. The move is a query, and the presence of the query means the node q_{j-1} is pebbled both before and after the query, by definition 5.4, unless $j = 1$ (we consider q_0 not to be a node).

To show that no nodes are *initially* pebbled (another pebble game rule) takes only a little more work. Suppose that some nodes were pebbled in M_i's initial configuration, and consider a node q_j out of these that is pebbled due to the *earliest* query involving any of the initially-pebbled nodes. Then a shorter description of x (for sufficiently large n) can be given as $(j, \Delta\tau_j, k_j, x')$, where x' is x with q_j spliced out. This description could be processed via simulation of M_i to produce x in much the same way as in lemma 5.1, except that this time, the starting configuration C_1 can be produced directly from the known values of M_i and n, and need not be explicitly included in the description. Of course the description system s needs to be able to process descriptions of this form. Then the incompressibility of x in s shows that the assumption that q_j is initially pebbled is inconsistent.

Thus, M_i can be seen as exactly obeying all the rules of the Bennett pebble game. Now, Li and Vitányi have shown [21] that any strategy for the pebble game that eventually pebbles a node at or beyond node 2^k must at some time have at least $k+1$ nodes pebbled at once. So let us simply choose n large enough so that $t(n) \geq 2^k$ for some $k \geq 4(c_i + 1)$, and also so that $\mathsf{S} \geq c_i$. Then at times τ when p is maximum, M_i's space usage is (using lemma 5.1) $|C_\tau| > \frac{1}{4}p\mathsf{S} > \frac{1}{4}k\mathsf{S} \geq (c_i+1)\mathsf{S} \geq c_i + c_i\mathsf{S}$.

The above discussion establishes that machine M_i takes more than space $c_i + c_i\mathsf{S}$ if it correctly decides membership in $L(A)$ for inputs of length $n_i = n$ and takes only time $c_i + c_i\mathsf{T}$, so long as the oracle A is consistent with the definition above. Since machine M_i's behavior on the input $\mathtt{0}^n$ only depends on the values of the successor function $f(b)$ for bit-strings b up to a certain size (call it z), we are free to extend the oracle definition to similarly foil machine M_{i+1} by picking n_{i+1} so that $\mathsf{S}(n_{i+1}) > z$. If one continues the oracle definition process in this fashion for further M_i's *ad infinitum*, then for the resulting oracle, it will be the case that for any M_i and constant c_i in the entire infinite enumeration, the machine will either get the wrong answer or take more than time $c_i + c_i\mathsf{T}$ or space $c_i + c_i\mathsf{S}$ on input $\mathtt{0}^{n_i}$. Thus, no reversible machine can actually decide $L(A)$ in time $\mathcal{O}(\mathsf{T})$ and space $\mathcal{O}(\mathsf{S})$, and so $L(A) \notin \mathbf{RTISP}(\mathsf{T},\mathsf{S})^A$.

Note that this entire oracle construction, as described, is computable. If we are given procedures for computing $\mathsf{S}(n)$ and $\mathsf{T}(n)$, we can write an effective procedure that, given any finite oracle query, returns A's response to the query. The details of the oracle construction algorithm follow directly from the above definition of A, but would be too tedious to present here. This concludes our proof of theorem 5.1.

Note that in the above proof, we used the fact that the number of pebbles required to get to the final node grows larger than any constant as n increases. But the actual rate of growth can be used as well, to give us an interesting lower bound.

5.5. Lower Bound Corollary

COROLLARY 5.1. *(Lower bound on space for linear-time relativizable reversible simulation of irreversible machines.) For all* ST*-constructible* S,T *and computable* S' *such that* $\mathsf{S} \prec \mathsf{T} \prec 2^{\mathsf{S}}$ *and* $\mathsf{S}' \prec \mathsf{S}\log(\mathsf{T}/\mathsf{S})$*, there exists a computable, self-reversible oracle* A *such that* $\mathbf{RTISP}(\mathsf{T},\mathsf{S}')^A \not\supseteq \mathbf{TISP}(\mathsf{T},\mathsf{S})^A$.

Proof. (Sketch.) Essentially the same as theorem 5.1 part (b), but with S' in place of S in appropriate places. In the last part of the proof, M_i is shown to take more than $c_i + c_i\mathsf{S}'$ space by using lemma 5.1, together with the fact that $p > \lfloor \lg \lfloor \mathsf{T}/\mathsf{S} \rfloor \rfloor$ pebbles are required to reach the final node. ■

This result implies that any general linear-time simulation of irreversible machines by reversible ones that is relativizable with respect to all self-reversible oracles must take space $\mathbf{\Omega}(\mathsf{S}\log(\mathsf{T}/\mathsf{S}))$.

The most space-efficient linear-time reversible simulation technique that is currently known was provided by Bennett ([13], p. 770), and analyzed by Levine and Sherman [22] to take space $\mathcal{O}(\mathsf{S}(\mathsf{T}/\mathsf{S})^{1/(0.58 \lg(\mathsf{T}/\mathsf{S}))})$. Bennett's simulation can be easily seen to work with all self-reversible oracles, so it gives a relativizable upper bound on space. There is a gap between it and our lower bound, due to the fact that the space-optimal pebble-game strategy referred to in our proof takes *more* than linear time in the number of nodes. A lower bound on the number of pebbles used by *linear* time pebble game strategies would allow us to expand our lower bound on space, hopefully to converge with the existing upper bound.

6. NON-RELATIVIZED SEPARATION

We now explain how the same type of proof can be applied to show a non-relativized separation of $\mathbf{RTISP}(\mathsf{T}, \mathsf{S})$ and $\mathbf{TISP}(\mathsf{T}, \mathsf{S})$ for a certain slowly-growing space bound S, when inputs are accessed in a specialized way that is similar to an oracle query, and the input size is not included in the space usage.

Input framework. Machine inputs will be provided in the form of a random-access read-only memory I, which may consist of 2^b b-bit words for any integer $b \geq 0$. The length of this input may be considered to be $n(b) = b2^b$ bits; let $b(n)$ be the inverse of this function. The machine will have a special *input access tape* which is unbounded in one direction, initially empty, and is used for reversibly accessing the input ROM via the following special operations.

Get input size. If the input access tape is empty before this operation, after the operation it will contain b written as a binary string. If the tape contains b before the operation, afterwards it will be empty. In all other circumstances, the query is a no-op.

Access input word. If the input access tape contains a binary string a of length b before the operation, afterwards it will contain the pair $(a, I[a])$ where $I[a]$ is a length-b binary string giving the contents of the input word located at address a. If the tape contains this pair before the operation, afterwards it will contain just a. Otherwise, nothing happens.

THEOREM 6.1. (Non-relativized separation of reversible and irreversible space-time.) *For models using the above input framework, and for* $\mathsf{S}(n) = b(n)$ *and any* ST*-constructible* $\mathsf{T}(n)$ *such that* $\mathsf{S} \prec \mathsf{T} \prec 2^{\mathsf{S}}$, $\mathbf{RTISP}(\mathsf{T}, \mathsf{S}) \neq \mathbf{TISP}(\mathsf{T}, \mathsf{S})$.

Proof. *(Sketch following proof of theorem 5.1.)* For input I of length $n = b2^b$, define result bit $r(I)$ to be the first bit in the b-bit string given by

$$\underbrace{I[I[\ldots I[}_{\lfloor \mathsf{T}/\mathsf{S} \rfloor} \mathtt{0}^b] \ldots]].$$

Let language $L = \{I : r(I) = \mathtt{1}\}$. $L \in \mathbf{TISP}(\mathsf{T}, \mathsf{S})$ because an irreversible machine can simply follow the chain of $\lfloor \mathsf{T}/\mathsf{S} \rfloor$ pointers from address $\mathtt{0}^b$, using space $\mathcal{O}(\mathsf{S})$ (not counting the input) and time $\mathcal{O}(\mathsf{T})$.

Assume there is a reversible machine M that decides L in $c + c\mathsf{S}$ space and $c + c\mathsf{T}$ time for some c. Let b be sufficiently large for the proof below to work.

Let s be a certain description system to be defined. Let $t = \lfloor \mathsf{T}/\mathsf{S} \rfloor$. Let x be a length-$t\mathsf{S}$ string incompressible in s. Let $w_1 \ldots w_t = x$ where all w_i are size b. Restrict s so that all the words w_i must be different from each other and from 0^b. Let I be an input of length $n = b2^b$ such that $I[0^b] = w_1$, and $I[w_i] = w_{i+1}$ for $1 \leq i < t$, and $I[a] = 0^b$ for every other address a. M must at some time access $I[w_{t-1}]$ because otherwise we could change the first bit of $I[w_{t-1}]$ to be the opposite of whatever M's answer is, and M would give the wrong answer. Assign a set of pebbled nodes to each configuration of M's execution on input I like in the oracle proof, except that this time, input access operations take the place of oracle calls. Show, as in lemma 5.1, that the size of any of these configurations is at least $\frac{1}{4}p\mathsf{S}$ where p is the number of pebbled nodes, by defining s to allow descriptions that are interpreted by simulating M forwards or backwards and reading pebbled nodes from the input access tape. As before, the machine must therefore take space $\mathbf{\Omega}(\mathsf{S}\log(\mathsf{T}/\mathsf{S}))$, which for sufficiently large n contradicts our assumption that the space is bounded by $c + c\mathsf{S}$. Thus $L \notin \mathbf{RTISP}(\mathsf{T}, \mathsf{S})$. ■

COROLLARY 6.1. Non-relativized lower bound on space for linear-time reversible simulations. *For* $\mathsf{S} = b(n)$, *computable* $\mathsf{S}' \prec \mathsf{S}\log(\mathsf{T}/\mathsf{S})$, *and* ST*-constructible* $\mathsf{T}(n)$ *such that* $\mathsf{S} \prec \mathsf{T} \prec 2^{\mathsf{S}}$, $\mathbf{RTISP}(\mathsf{T}, \mathsf{S}') \not\supseteq \mathbf{TISP}(\mathsf{T}, \mathsf{S})$.

Proof. *(Sketch.)* As in corollary 5.1, but with theorem 6.1. ■

Such a T exists because b can be found in space and time $\mathcal{O}(\log b)$ using the "get input size" operation, after which $\mathsf{T} = b^2$, for example, can be found in space $\mathcal{O}(\log b)$ and time $\mathcal{O}(\log^2 b)$.

COROLLARY 6.2. *Thus, any reversible machine that simulates irreversible ones without asymptotic slowdown takes* $\mathbf{\Omega}(\mathsf{S}\log(\mathsf{T}/\mathsf{S}))$ *space in some cases, given the type of input model presented in this section.*

Again, we emphasize that this particular lower bound is probably not tight.

We should also note that this particular non-relativized result is not very compelling, because with a space bound that is much less than the input size, the space usage is unlikely to reflect a dominant component of system cost for real-world applications.

7. DECOMPRESSION ALGORITHM

It is probably not obvious to the reader that the algorithm that we briefly mentioned in the proof of lemma 5.1 in §5.4 can be made to work properly. In this section we give the complete algorithm and explain why it works.

The algorithm, shown in figure 12, essentially just simulates M_i's operation in direction D starting from configuration C_τ, and reads the identifiers of the pebbled nodes off of M_i's simulated oracle tape. The bulk of the algorithm is in the details showing how to simulate all oracle queries correctly.

There is a small subtlety in the fact that this algorithm has, built into it, some of the values of f that are defined by the oracle. Yet the algorithm is part of the definition of our description system s_i, which is used to pick x and define the $f(q_j)$

values. This would be a circularity that might prevent the oracle from being well-defined, if not for the fact that the portion of f that is built in, that is, $f(b)$ for $|b| < \mathsf{S}$, is disjoint from the portion of f that depends on this algorithm, that is, only values of $f(b)$ for $|b| \geq \mathsf{S}(n_i)$. Thus there is no circularity.

The $f()$ values for the entire infinite oracle can be enumerated by enumerating all values of i in sequence, and for each one, computing the appropriate values of M_i and c_i, and choosing an n_i that satisfies all the explicit and implicit lower bounds on n that we mentioned above. Then, n_i is used in the above algorithm to allow us to define s_i and choose the appropriate x, which determines $f(b)$ for all b where $|b| = \mathsf{S}(n_i)$; these values of f can then be added to the table for use in the algorithm later when running on higher values of i.

We now explain why the simulation carried out by the (oracle-less) decompression algorithm imitates the real oracle-calling program exactly. When we come to an oracle query operation where the queried bit-string(s) do not appear in our $q[j]$ array and do not have a matching $\Delta\tau_j$, then we know the bit-string(s) must not correspond to a real node in $q_1, \ldots, q_t$, because if they did, then either they were not pebbled due to queries in direction D, in which case they would have been in the description d and would have been present in the initial q array, or else the first query that involved them must have been before the current one (or else some $\Delta\tau_j$ would match), in which case they would have been added to the q array earlier.

Moreover, when we get to a single query q_j, we know we can look up q_{j+1} to answer the query, because it must already have been stored. Either q_{j+1} was not pebbled in direction D in which case it was stored originally, or it was pebbled in direction D in which case the first query involving it must have been before this one, since this query is not of the type that would have caused the node to be pebbled in direction D. In either case we will already have a value in array entry $q[j+1]$.

Given any description d derived from the execution history of a real M_i, the simulation will eventually find values for all nodes, since either they were given initially or they are found eventually as we simulate. Thus the algorithm prints x, as required for the proof of lemma 5.1.

8. BEYOND THIS PROOF

In light of the work above, an obviously desirable next step would be to show that $\mathbf{RTISP}(\mathsf{T},\mathsf{S}) \neq \mathbf{TISP}(\mathsf{T},\mathsf{S})$ (and demonstrate corresponding tight lower bounds) for a larger class of space-time functions S,T in a reasonable serial model of computation *without* an oracle or a black-box input. A similar problem of following a chain of nodes may still be useful for this. But when there is no oracle, and when the time bound is larger than the input length $\mathsf{T} \succ n$, there is no opportunity to specify an incompressible chain of nodes to follow. Instead, the function f mapping nodes to their successors must be provided by some actual computation that is specified by the relatively short input. It may be helpful in such a proof if f is non-invertible, or is a one-way invertible function, whose inverse might be hard to compute. But f will still have some structure in general, and so it may be very difficult to prove that there are no shortcuts that might allow the result of repeated applications of f to be computed reversibly using little time or space.

ALGORITHM 2 (DECOMPRESS(d)).

Given description d as described on p. 30,
Let $q[1] \ldots q[t]$ be a table of node values, initially all NULL.
Initialize all $q[j]$'s not pebbled in direction D, as specified by description d.
Simulate M_i in direction D starting from configuration C_τ, as follows:
  To simulate a single operation of M_i:
    If it's a non-query operation, then
      simulate it straightforwardly, and proceed.
    Otherwise, it's an oracle query; examine the oracle tape.
    If the tape is not of the form b or $b\#c$ for bit-strings b, c, where $|b| = |c|$,
      do nothing for this operation.
    Else, if $|b| < \mathsf{S}$, then look up $f(b)$ in a computable table,
      set the oracle tape appropriately, and proceed.
    Else, if $|b| > \mathsf{S}$,then do nothing for this operation.
    Else, if the oracle tape is of the form b, then
      If the current step count matches some $\Delta\tau_j$ in direction D,
        then set $q[j] = b$.
      If $b = q[j]$ for some $j < t$,
        then set the oracle tape to $b\#q[j+1]$,
      else, do nothing for this operation.
    Else, if the oracle tape is of the form $b\#c$, then
      For each $\Delta\tau_j$ in direction D matching the current step count,
        set $q[j]$ to b or c depending on tag k_j.
      If $b = q[j]$ and $c = q[j+1]$ for some j, set oracle tape to b,
      else do nothing for this operation.
  Increment count of the number of steps simulated.
  Continue simulating steps of M_i until step count exceeds largest $\Delta\tau_j$.
Print all $q[j]$'s.

FIG. 12. Algorithm to print the incompressible chain of nodes x via simulation of the reversible machine M_i.

9. CONCLUSION

Although the above results are inconclusive with respect to their real-world implications, it seems likely that reversible algorithms in the real world will indeed in many cases require algorithmic space-time costs that exceed those of traditional computations, by factors that are at least logarithmic and more likely small polynomials in the cost of the original computation.

However, this is not to say that reversible computing will never be useful. For contexts where the cost of energy is high compared to the cost of computation, or where the computation would benefit from a 3-D parallel architecture which would tend be difficult to cool effectively, we have shown elsewhere that the overall cost per performance of a partially-reversible solution may be lower than that of a traditional irreversible design, despite the higher algorithmic costs [5]. Also, someday we might carry out *quantum* computations which would be demonstrably much *more* efficient than traditional computation on some problems, despite their reversibility [31]. So, the exact magnitude of the algorithmic cost of reversibility is still important, because it affects the location of the optimal tradeoff points for a reversible design, within those contexts where it *is* useful.

We believe that the most fruitful direction for future work in reversible computing theory at this point is to optimize the parameters of Bennett's algorithm in a way that minimizes the hardware cost per unit performance of parallel reversible architectures, as a function of whatever upper bounds on power dissipation per unit performance may arise from the requirements presented by particular application contexts. Such analyses should take into account the asymptotic behavior of realistic physical implementations of reversible computing; for example, there is an additional asymptotic slowdown factor not accounted for in the present paper which is required for the quasi-adiabatic (*i.e.*, asymptotically reversible) physical operation of real logic devices. Additionally, in order for the analytical model to be useful for estimating the feasibility of real-world computer designs, the model would also need to incorporate the specific constant factors and limits on reversibility that would be incurred in a specific, feasible real-world reversible technology.

In the years since the first manuscripts of this paper were written and circulated (circa 1997), we have been developing elements of some practical reversible hardware technologies (*cf.* [11, 32, 33, 34]) and carrying out the accompanying tradeoff analysis. The results of the most recent (and still unpublished) work will be announced in future reports to be presented to the computer science & engineering community.

ACKNOWLEDGMENT

Thanks are due to Michael Sipser of MIT for suggesting the use of the incompressibility method, and to Alain Tapp and Pierre McKenzie of the University of Montreal for helpful discussions and detailed feedback on early drafts of this paper.

REFERENCES

1. K. Eric Drexler. *Nanosystems: Molecular Machinery, Manufacturing, and Computation.* John Wiley & Sons, Inc., 1992. `http://nano.xerox.com/nanotech/nanosystems.html`.
2. Ralph C. Merkle. Reversible electronic logic using switches. *Nanotechnology*, 4:21–40, 1993.
3. J. Storrs Hall. An electroid switching model for reversible computer architectures. In PhysComp92 [35], pages 237–247.

4. Michael P. Frank, Thomas F. Knight, Jr., and Norman H. Margolus. Reversibility in optimally scalable computer architectures. In Calude et al. [36], pages 165–182. http://www.ai.mit.edu/~mpf/rc/scaling_paper/scaling.html.

5. Michael P. Frank and Thomas F. Knight, Jr. Ultimate theoretical models of nanocomputers. *Nanotechnology*, 9(3):162–176, 1998. Presented at the Fifth Foresight Conference on Molecular Nanotechnology, Palo Alto, CA, November 1997. http://www.ai.mit.edu/~mpf/Nano97/paper.html.

6. Michael P. Frank. Reversibility for efficient computing. Course manuscript for the *Physical Limits of Computing* course at the University of Florida, Spring 2000[37], December 1999. http://www.cise.ufl.edu/~mpf/manuscript.

7. Rolf Landauer. Irreversibility and heat generation in the computing process. *IBM J. Research and Development*, 5:183–191, 1961. Reprinted in [8], ch. 4, pp. 188–196.

8. Harvey S. Leff and Andrew F. Rex, editors. *Maxwell's Demon: Entropy, Information, Computing*. Princeton series in physics. Princeton University Press, Princeton, NJ, 1990. May be ordered through http://www.ioppublishing.com/Books/Catalogue/020/__26/0750300566.

9. C. H. Bennett. The thermodynamics of computation, a review. *International Journal of Theoretical Physics*, 21(12):905–940, 1982.

10. W. Athas, N. Tzartzanis, L. Svensson, L. Peterson, H. Li, X. Jiang, P. Wang, and W-C. Liu. AC-1: A clock-powered microprocessor. In *Proc. of the International Symposium on Low-Power Electronics and Design*, Monterey, CA, 18–20 August 1997. http://www.isi.edu/acmos/papers/97-08.MontereyAC1.ps.

11. Michael P. Frank, Carlin Vieri, M. Josephine Ammer, Nicole Love, Norman H. Margolus, and Thomas F. Knight, Jr. A scalable reversible computer in silicon. In Calude et al. [36], pages 183–200. http://www.ai.mit.edu/~mpf/rc/flattop/ft.html.

12. Carlin Vieri. *Reversible Computer Engineering and Architecture*. PhD thesis, Massachusetts Institute of Technology, 1999.

13. C. H. Bennett. Time/space trade-offs for reversible computation. *SIAM J. Computing*, 18(4):766–776, 1989.

14. Ming Li and Paul M. B. Vitányi. Reversible simulation of irreversible computation. In *Proc. 11th IEEE Conference on Computational Complexity*, Philadelphia, Pennsylvania, May 24–27, 1996.

15. Klaus-Jörn Lange, Pierre McKenzie, and Alain Tapp. Reversible space equals deterministic space. In *Proc. 12th Annual IEEE Conf. on Computational Complexity (CCC '97)*, pages 45–50, June 1997. http://www.iro.umontreal.ca/~tappa/Publications/LMT'97_abstract.html.

16. J. E. Pin. On the languages accepted by finite reversible automata. In Thomas Ottman, editor, *Automata, Languages and Programming, Proc. 14th Int'l Colloq. (ICALP)*, volume 267 of *Lecture Notes in Computer Science*, pages 237–249. Springer-Verlag, 1987.

17. Y. Lecerf. Machines de Turing réversibles. Insolubilité récursive en $n \in N$ de l'équation $u = \theta^n$, où θ est un $\ll$ isomorphisme de codes $\gg$ [Reversible Turing machines. Recursive insolubility in $n \in N$ of the equation $u = \theta^n$, where θ is an "isomorphism of codes"]. *Comptes Rendus Hebdomadaires des Séances de L'académie des Sciences [Weekly Proceedings of the Academy of Science]*, 257:2597–2600, October 28, 1963. Unauthorized English translation at http://www.ai.mit.edu/~mpf/rc/Lecerf/lecerf.html.

18. C. H. Bennett. Logical reversibility of computation. *IBM J. Research and Development*, 17(6):525–532, 1973.

19. Michael P. Frank. *Reversibility for Efficient Computing*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, May 1999. http://www.ai.mit.edu/~mpf/rc/thesis/phdthesis.html.

20. Tommaso Toffoli. Computation and construction universality of reversible cellular automata. *J. Computer and System Sciences*, 15:213–231, 1977.

21. Ming Li and Paul M. B. Vitányi. Reversibility and adiabatic computation: trading time and space for energy. *Proceedings of the Royal Society of London Ser. A*, 452:1–21, 1996.

22. Robert Y. Levine and Alan T. Sherman. A note on Bennett's time-space tradeoff for reversible computation. *SIAM J. Computing*, 19(4):673–677, 1990.

23. Pierluigi Crescenzi and Christos H. Papadimitriou. Reversible simulation of space-bounded computation. *Theoretical Computer Science*, 143:159–165, 1995.

24. Harry Buhrman, John Tromp, and Paul Vitányi. Time and space bounds for reversible simulation. Proc ICALP 2001, Lecture Notes in Computer Science, Vol xxx Springer-Verlag, January 2001. `http://arxiv.org/abs/quant-ph/0101133`.

25. Don Coppersmith and Edna Grossman. Generators for certain alternating groups with applications to cryptography. *Society for Industrial and Applied Mathematics J. Appl. Math.*, 29(4):624–627, December 1975.

26. Tommaso Toffoli. Reversible computing. Technical memo MIT/LCS/TM-151, MIT Lab for Computer Science, February 1980. Out of print; available from NTIS. Abridged version available as [27].

27. Tommaso Toffoli. Reversible computing. In J. W. de Bakker and J. van Leeuwen, editors, *Automata, Languages and Programming (Seventh Colloquium, Noordwijkerhout, the Netherlands, July 14–18, 1980)*, volume 85 of *Lecture Notes in Computer Science*, pages 632–644. Springer-Verlag, 1980. Abridged version of [26].

28. E. Fredkin and T. Toffoli. Conservative logic. *International Journal of Theoretical Physics*, 21:219–253, 1982.

29. Theodore Baker, John Gill, and Robert Solovay. Relativizations of the $\mathcal{P} =? \mathcal{NP}$ question. *SIAM J. Computing*, 4(4):431–442, December 1975.

30. Ming Li and Paul Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications.* Graduate Texts in Computer Science. Springer-Verlag, 2nd edition, 1997.

31. Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. Los Alamos Physics Preprint Archive, `http://xxx.lanl.gov/abs/quant-ph/9508027`, August 1995. Expanded version of [38].

32. Carlin Vieri, M. Josephine Ammer, Amory Wakefield, Lars "Johnny" Svensson, William Athas, and Thomas F. Knight, Jr. Designing reversible memory. In Calude et al. [36], pages 386–405. (Proc. of the First International Conference on Unconventional Models of Computation (UMC'98), held at the University of Auckland, January 5–9, 1998).

33. C. Vieri, M. Josephine Ammer, Michael Frank, Norm Margolus, and Tom Knight. A fully reversible asymptotically zero energy processor. In *Proceedings of the ISCA Workshop*, Barcelona, Spain, 1998.

34. M. J. Ammer, M. Bolotski, P. Alvelda, and T. Knight. A 160×120 pixel liquid-crystal-on-silicon microdisplay with an adiabatic dac. In *International Solid-State Circuits Conference (ISSCC '99)*, San Francisco, CA, February 1999.

35. *PhysComp '92: Proceedings of the Workshop on Physics and Computation, October 2–4, 1992, Dallas, Texas*, Los Alamitos, CA, 1992. IEEE Computer Society Press.

36. C. S. Calude, J. Casti, and M. J. Dinneen, editors. *Unconventional Models of Computation.* Springer, 1998. (Proc. of the First International Conference on Unconventional Models of Computation (UMC'98), held at the University of Auckland, January 5–9, 1998).

37. Michael P. Frank. Physical limits of computing. Course web site, April 2000. `http://www.-cise.ufl.edu/~mpf/physlim`.

38. Peter W. Shor. Algorithms for quantum computation: Discrete log and factoring. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, pages 124–134. Institute of Electrical and Electronic Engineers Computer Society Press, November 1994. `ftp://netlib.att.com/netlib/att/math/shor/quantum.algorithms.ps.Z`.