SIAM J. COMPUT.
Vol. 18, No. 4, pp. 766-776, August 1989

010

TIME/SPACE TRADE-OFFS FOR REVERSIBLE COMPUTATION*

CHARLES H. BENNETT†

Abstract. A reversible Turing machine is one whose transition function is 1:1, so that no instantaneous description (ID) has more than one predecessor. Using a pebbling argument, this paper shows that, for any $\varepsilon > 0$, ordinary multitape Turing machines using time T and space S can be simulated by reversible ones using time $O(T^{1+\varepsilon})$ and space $O(S \log T)$ or in linear time and space $O(ST^{\varepsilon})$. The former result implies in particular that reversible machines can simulate ordinary ones in quadratic space. These results refer to reversible machines that save their input, thereby insuring a global 1:1 relation between initial and final IDs, even when the function being computed is many-to-one. Reversible machines that instead erase their input can of course compute only 1:1 partial recursive functions and indeed provide a Gödel numbering of such functions. The time/space cost of computing a 1:1 function on such a machine is equal within a small polynomial to the cost of computing the function and its inverse on an ordinary Turing machine.

Key words. reversibility, invertibility, time, space, Turing machine, TISP

AMS(MOS) subject classifications. 68Q05, 68Q15

1. Introduction.

A deterministic computer is called reversible if it is also backwards deterministic, having no more than one logical predecessor for each whole-machine state. Reversible computers must avoid such common irreversible operations as assignments and transfers of control that throw away information about the previous state of the data or program. Reversible computers of various kinds (Turing machines, cellular automata, combinational logic) have been considered [1], [11], [12], [13], [6], [2], [14] especially in connection with the physical question of the thermodynamic cost of computation; and it has been known for some time that they can simulate the corresponding species of irreversible computers in linear time [1] (or linear circuit complexity [13]), provided they are allowed to leave behind at the end of the computation a copy of the input (thereby rendering the mapping between initial and final states 1:1 even though the input-output mapping may be many-to-one).

The physical significance of reversible simulation stems from the fact [7], [1] that many-to-one data operations have an irreducible thermodynamic cost, while one-to-one operations do not. The ability to program any computation as a sequence of 1:1 operations therefore implies that, using appropriate hardware, an arbitrarily large amount of "computational work" (in conventional units of time/space complexity) can in principle be performed per unit of physical energy dissipated by the computer. This result contradicted an earlier widely held belief [15] that any computer operating at temperature T must dissipate at least $kT \ln 2$ (the heat equivalent to one bit of entropy, approximately equal to the kinetic energy of a gas molecule at temperature T) per elementary binary operation performed. Needless to say, most real computing equipment, both electronic and biological, is very inefficient thermodynamically, dissipating many orders of magnitude more than kT per step. However, a few thermodynamically efficient data processing systems do exist, notably genetic enzymes such as RNA polymerase, which, under appropriate reactant concentrations, can transcribe information from DNA to RNA at a thermodynamic cost considerably less than kT per step.

* Received by the editors January 27, 1988; accepted for publication October 18, 1988.
† IBM Thomas J. Watson Research Center, Room 21-125, Yorktown Heights, New York 10598.

The old linear-time simulation [1] is not at all economical of space: in the worst case, it uses exponentially more space than the irreversible computation being simulated.

Here we review this old result and present some new results on more space-efficient reversible computation using the formalism of multitape Turing machines. For simplicity we consider (deterministic) machines whose input tape and output tape, like the work tapes, are two-way, read-write, and included in the space accounting. As usual we restrict ourselves to "well-formed" machines (a recursive subset of machines) whose computations, if started in a standard initial ID will either fail to halt or will halt in a standard final ID. A standard initial ID is one with the Turing machine's control unit in the designated start state, with the input in standard format (input head scanning the blank square just left of a finite input string containing no embedded blanks) and with work and output tapes blank. A standard final ID is one in which the control unit is in a single designated terminal state, all work tapes are blank and all heads have been restored to their original locations, and the output tape contains a single string in standard format. Let $\{\varphi_i\}$ be a Gödel numbering of well-formed machines, and let $T_i(x)$ and $S_i(x)$ denote the time and space used by machine i on input x, the partial functions T_i and S_i having the same domain as φ_i.

A machine φ_i is *reversible* if its quintuples have disjoint ranges, thereby rendering the transition function 1 : 1. We define two special kinds of standard, reversible machine:

Input-saving machines whose input head is read-only, so that the input string remains unchanged throughout the computation.

Input-erasing machines in which the input tape, like the work tapes, is required to be blank at the end of any halting computation on standard input.

These requirements, like the well-formedness requirements for ordinary Turing machines, can be enforced through recursive constraints on the quintuples, so that the input-saving and input-erasing reversible machines can be Gödel-numbered as recursives subsets of $\{\varphi_i\}$.

2. Input-saving reversible computations. As a preliminary to the new results we restate the old, linear-time simulation of [1].

LEMMA 1. *Each conventional multitape Turing machine running in time T and space S can be simulated by a reversible input-saving machine running in time $O(T)$ and space $O(S+T)$. More precisely, there exists a constant c such that for each ordinary Turing machine φ_i, an index j can be found effectively from i, such that φ_j is reversible and input-saving, $\varphi_j = \varphi_i$, and $T_j < c + c \cdot T_i$, and $S_j < c + c \cdot (S_i + T_i)$.*

Proof. As illustrated by Table 1 below, the simulation proceeds by three stages. The first stage uses an extra tape, initially blank, to record all the information that would have been thrown away by the irreversible computation being simulated. This history-taking renders the computation reversible, but also leaves a large amount of unwanted data on the history tape (hence the epithet "untidy"). The second stage copies the output produced by the first stagé onto a separate output tape, an action that is intrinsically reversible (without history-taking) if the output tape is initially blank. The third stage exactly reverses the work of the first stage, thereby restoring the whole machine, except for the now-written output tape, to its original condition. This cleanup is possible precisely because the first stage was reversible and deterministic. An apparent problem in constructing the cleanup stage, viz. that conventional read/write/shift quintuples have inverses of a different form (shift/read/write), can be avoided by expressing each read/write/shift of the untidy stage as a product of nonshifting read/writes and oblivious shifts, then inverting these. For more details, see the Appendix and [1].

TABLE 1
Basic linear-time reversible simulation using an extra tape to record, temporarily, a history of the irreversible computation being simulated (underbars denote head positions).

Stage	Input tape	Work tape(s)	History tape	Output tape
Untidy	_INPUT	–	–	–
	INPUT	WORK	HIST_	–
	_INPUT	_OUTPUT	HISTORY_	–
Copy output	_INPUT	_OUTPUT	HISTORY_	–
	_INPUT	OUTPUT	HISTORY_	OU_
	_INPUT	OUTPUT_	HISTORY_	OUTPUT_
	_INPUT	_OUTPUT	HISTORY_	_OUTPUT
Cleanup	_INPUT	_OUTPUT	HISTORY_	_OUTPUT
	INPUT	WORK	HIST_	_OUTPUT
	_INPUT	–	–	_OUTPUT

Note in passing that the operation of copying onto a blank tape serves in stage 2 as a substitute for ubiquitous assignment operation $a := b$, which is not permitted in reversible programming. Similarly, the inverse of copying onto blank tape, which reversibly erases one of two strings known to be identical, can take the place of the more general but irreversible operation of erasure. If the natural number 0 is identified with the empty string, reversible copying and erasure can be represented as incrementation and decrementation, with $a := a + b$ having the same effect as $a := b$, if $a = 0$ initially, and $a := a - b$ having the same effect as $a := 0$, if $a = b$ initially.

The above scheme for reversible simulation runs in linear time but sometimes uses as much as exponential space. The following theorem shows that by allowing slightly more than linear time a much more space-efficient simulation can be obtained.

THEOREM 1. *For any $\varepsilon > 0$, any multitape Turing machine running in time T and space S can be simulated by a reversible input-saving machine using time $O(T^{1+\varepsilon})$ and space $O(S \cdot \log T)$.*

Proof. This space-efficient simulation is obtained by breaking the original computation into segments of m steps, where $m \approx S$, then doing and undoing these segments in a hierarchical manner as shown in Table 2. This table shows that by hierarchically iterating the simulation of 2 segments of original computation by 3 stages of reversible computation, 2^n segments of the original computation can be simulated in 3^n stages of reversible computation. More generally, for any fixed $k > 1$, k^n segments of irreversible computation can be simulated by $(2k-1)^n$ stages of reversible computation. Each stage of reversible computation invokes the linear-time reversible simulation of Lemma 1 to create or destroy a checkpoint (a complete intermediate ID of the simulated computation) from the preceding, m-steps earlier, checkpoint, using time and space $O(S)$. Taking $n \approx \log(T/m)/\log(k)$, the number of intermediate checkpoints in storage at any one time is at most $n(k-1)$, which for fixed k is $O(\log T)$, and since each checkpoint uses storage $O(S)$, the space requirement is $O(S \cdot \log T)$, while the time requirement is $O(S \cdot (2k-1)^n) = O(T^{(1+1/\log k)})$. The simulation is applied to segments of length $m \approx S$ rather than to single steps ($m = 1$) to insure that the time $O(S)$ of copying or erasing a checkpoint does not dominate the time $O(m)$ of computing it from the previous checkpoint. The extra space $O(m)$ required for the history tape in

TABLE 2
Reversible simulation in time $O(T^{\log 3/\log 2})$ and space $O(S \cdot \log T)$.

Stage	Action	Checkpoints in storage (0 = initial ID, checkpoint $j = (jm)$th step ID)								
0	Start	0								
1	Do segment 1	0	1							
2	Do segment 2	0	1	2						
3	Undo segment 1	0		2						
4	Do segment 3	0		2	3					
5	Do segment 4	0		2	3	4				
6	Undo segment 3	0		2		4				
7	Do segment 1	0	1	2		4				
8	Undo segment 2	0	1			4				
9	Undo segment 1	0				4				
10	Do segment 5	0				4	5			
11	Do segment 6	0				4	5	6		
12	Undo segment 5	0				4		6		
13	Do segment 7	0				4		6	7	
14	Do segment 8	0				4		6	7	8
15	Undo segment 7	0				4		6		8
16	Do segment 5	0				4	5	6		8
17	Undo segment 6	0				4	5			8
18	Undo segment 5	0				4				8
19	Do segment 1	0	1			4				8
20	Do segment 2	0	1	2		4				8
21	Undo segment 1	0		2		4				8
22	Do segment 3	0		2	3	4				8
23	Undo segment 4	0		2	3					8
24	Undo segment 3	0		2						8
25	Do segment 1	0	1	2						8
26	Undo segment 2	0	1							8
27	Undo segment 1	0								8

the linear-time simulation between checkpoints does not increase the order of the overall space bound $O(S \cdot \log T)$.

To complete the proof, it remains to be shown how the reversible machine causes the above operations to be performed in correct sequence, how it organizes the storage of the checkpoints on tape, and finally how it arrives at satisfactory values for m and n without knowing T and S beforehand. The hierarchical sequencing of operations is conveniently performed by a recursive subroutine that, given m and n, reversibly calculates the $m \cdot k^n$th successor of an arbitrary ID, using $k-1$ local variables (each restricted to m squares of tape) at each lower level of recursion in n to store the necessary intermediate checkpoints. At the $n = 0$ level of recursion, the mth successor is calculated by the method of Lemma 1. The search for the correct m and n values is performed by a main procedure, which calls the recursive subroutine for successively larger m and n until the time $m \cdot k^n$ and space m allowances prove sufficient, meanwhile keeping a history of the sequence of m and n values to preserve reversibility. When the desired output has been obtained and copied onto the output tape, the search for m and n is undone to dispose of the history of this search. (For more details, see the Appendix.) □

Remark. The reversible creation and annihilation of checkpoints illustrated in Table 2 may be regarded as a case of *reversible pebbling* of a one-dimensional graph. In ordinary pebbling, addition of a pebble to a node requires that all predecessors of

the node already have pebbles, but removal of a pebble can be performed at any time. In reversible pebbling, both the addition and the removal of a pebble require that the predecessors have pebbles.

COROLLARY. *Each ordinary Turing machine running in space S can be simulated by a reversible, input-saving machine running in space $O(S^2)$.*

Related results. Using the same construction, but with k increasing as a function of T (and consequently having n increase less rapidly than $\log T$), one can obtain reversible simulations using less time and more space. For example, using fixed n, k increases linearly with T and one obtains a reversible simulation in time $O(T)$ and space $O(ST^{\varepsilon})$. Intermediate trade-offs can also be obtained, e.g., time $O(T \cdot 2^{\sqrt{\log T}})$ and space $O(S \cdot 2^{(\sqrt{\log T})+O(\log\log T)})$.

The usual linear tape-compression and time speedup theorems apply to reversible machines; therefore, if $T(x)$ and $S(x)$ are total functions bounded below by $3|x|+3$ and $|x|+2$, respectively, then

$$\textbf{TISP}\,(T, S) \subseteq \textbf{RTISP}\,(T^{1+\varepsilon}, S \log T),$$

where **TISP** (T, S) denotes the class of languages accepted by ordinary Turing machines in time T and space S, and **RTISP** (T, S) denotes the analogous class for input-saving reversible machines. Similarly, the class **TISP** (T, S) is contained in

$$\textbf{RTISP}\,(T, ST^{\varepsilon}),$$

and in

$$\textbf{RTISP}\,(T \cdot 2^{\sqrt{\log T}}, S \cdot 2^{(\sqrt{\log T})+O(\log\log T)}).$$

It is interesting to compare the time/space trade-off in reversible simulation of ordinary Turing machines with the analogous depth/width trade-off in reversible simulation of ordinary combinatorial logic circuits. A depth-efficient construction due to Fredkin and Toffoli [6], paralleling Lemma 1, simulates ordinary circuits of depth d and width w, in an input-saving manner, by reversible circuits of depth $O(d)$ and width $O(dw)$, composed of reversible gates with no more than three inputs and outputs. A construction parallel to Theorem 1 above would yield a more width-efficient simulation, by reversible circuits of depth $O(d^{1+\varepsilon})$ and width $O(w \cdot \log d)$.

However, a still more width-efficient simulation is possible, based on Coppersmith and Grossman's result [5] that any even permutation on the space of n-bit strings can be computed by a circuit of width n, and depth exponential in n, composed of the reversible gates of $\leqq 3$ inputs and outputs. Width n is insufficient to realize odd permutations, but these can be computed by circuits of width $n+1$, using an extra "garbage" bit whose initial value does not matter, and which is returned unmodified in the output.

Accessing such garbage during a reversible computation might at first appear useless, like renting space in a warehouse already full of someone else's belongings, but in fact it serves the purpose of allowing the odd permutation to be embedded in an even permutation of higher order (any permutation on strings that is oblivious to the values of one or more bits is perforce even) [3]. In particular, any m-bit function of an n-bit argument, $F_m(X_n)$, can be embedded in an even permutation of the form $(X_n, 0^m, g) \to (X_n, F_m(X_n), g)$, where g is a garbage bit, and computed reversibly in width $n+m+1$. Cleve [4] has explored other aspects of the depth/width trade-off for reversible circuits.

The extreme width-efficiency of the Coppersmith–Grossman circuits suggests the possibility of more space-efficient reversible Turing machine simulations using, say,

linear space rather than $S \cdot \log T$. However, this may not be possible, since a deep-but-narrow logic circuit can perform a sequence of *different* transformations, while a Turing machine is forced to apply the *same* transition function at each step.

3. Input-erasing reversible computations. The schemes described so far achieve global reversibility by saving the input, in effect embedding an arbitrary partial recursive function $\varphi(x)$ into the 1:1 partial recursive function $\langle x, \varphi(x)\rangle$. We now consider reversible machines that destroy their input, as well as clearing all working storage, while producing the desired output. Because of unique terminal state of the finite control, such machines cannot destroy or hide even a finite amount of the information present in the input, and so can compute only 1:1 functions. We show that such machines provide a Gödel numbering of the 1:1 partial recursive functions and that the time/space cost of computing a 1:1 function on such a machine is polynomially related to the cost of computing and inverting the function on ordinary Turing machines.

First we need a definition of the cost of inverting a function. Let T^* and S^* be arbitrary partial recursive functions. A 1:1 partial recursive function $f(x)$ will be said to be *invertible in time* $T^*(x)$ *and space* $S^*(x)$ if and only if there exists an (in general irreversible) multitape Turing machine for computing f^{-1} that, for all x in the domain of f, uses time $\leqq T^*(x)$ and space $\leqq S^*(x)$ to compute x from $f(x)$. The theorem below extends an analogous but space-inefficient result [1], [2] on reversible computation of 1:1 functions.

THEOREM 2 (on input-erasing reversible computations). (a) *A function is partial recursive and* 1:1 *if and only if it is computed by an input-erasing reversible multitape Turing machine. These machines thus provide a Gödel-numbering of* 1:1 *partial recursive functions.*

(b) *If f is a* 1:1 *partial recursive function computable in time T and space S, and invertible in time T^* and space S^*, then f can be computed in time $O(T + T^*)$ and space $O(\max\{S \cdot T^{\varepsilon}, S^* \cdot T^{*\varepsilon}\})$ on an input-erasing reversible machine.*

(c) *If f is a* 1:1 *partial recursive function computable in time T and space S, and invertible in time T^* and space S^*, then f can be computed in time $O(T^{1+\varepsilon} + T^{*1+\varepsilon})$ and space $O(\max\{S \log T, S^* \log T^*\})$ on an input-erasing reversible machine.*

(d) *If f is computable by an input-erasing reversible machine in time T and space S, then f is computable by an ordinary machine in time T and space S, and f is invertible by an ordinary machine in time $O(T)$ and space $S + O(1)$.*

Proof. The "if" of part (a) above is immediate from the nature of an input-erasing reversible machine. The "only if" of (a) and the time/space bounds (b) and (c) depend on the construction below (Table 3), in which irreversible algorithms for f and f^{-1} are combined to yield an input-erasing reversible algorithm for f. The "only if" of part (a) in addition depends on the fact noted by McCarthy [10] that, if f is 1:1, then an algorithm for f^{-1} can be obtained effectively from an algorithm for f. (Levin's optimal inverting algorithm [8], [9] is a refinement of this idea). Part (d) is immediate from the fact that an input-saving reversible Turing machine is already a Turing machine, and the fact that the inverse of a reversible machine (obtained by inverting its quintuples as explained in the proof of Lemma 1) runs in the same space and linear time as the original reversible machine.

Table 3 shows how two irreversible algorithms, for a 1:1 partial recursive function f and its inverse f^{-1}, can be combined to perform an input-erasing reversible computation of f. In the first half of the computation the irreversible algorithm for f is used in the manner of Table 1 or Table 2 to construct a temporary bridge (history or checkpoints) from x to $f(x)$, thereby allowing a copy of $f(x)$ to be left on the output

TABLE 3
Input-erasing reversible computation of a 1:1 function f, given irreversible algorithms for f and f^{-1}.

Action	Input tape	History tape	Work tapes	Output tape
Do then undo f, via	x			
history or checkpoints,	x	f-HISTORY	$f(x)$	
to create $f(x)$ reversibly	x	f-HISTORY	$f(x)$	$f(x)$
in presence of x	x			$f(x)$
Do then undo f^{-1}, via	x			$f(x)$
history or checkpoints,	x	f^{-1}-HISTORY	x	$f(x)$
to destroy x reversibly		f^{-1}-HISTORY	x	$f(x)$
in presence of $f(x)$				$f(x)$

tape after the bridge is removed. In the last half of the computation the algorithm for f^{-1} is used similarly to generate a bridge from $f(x)$ back to x, but in this case the bridge is used to delete the copy of x already present on the input tape before the bridge was constructed. The time and space required are evidently as stated in (b) and (c) above. □

Remark. Theorem 2 implies Theorem 1, since for any function φ computable in time T and space S, the function $f(x) = x * \varphi(x)$, where $*$ is a distinctive punctuation, is 1:1 and both computable and invertible in time $O(T)$ and space $O(S)$.

Appendix. Details of proofs. Here we give further details of the proofs of Lemma 1 and Theorem 1.

Lemma 1. Here we describe the construction of the reversible machine's Untidy Stage and Cleanup Stage quintuples corresponding to a typical quintuple

$$q_j, S_j \to S_k, \sigma, q_l$$

of the irreversible machine being simulated. The elements of the quintuple before the arrow represent, respectively, the initial control state and scanned tape symbol(s); those after the arrow represent the written tape symbol(s), shift, and final control state. In the Untidy Stage, each execution of such a quintuple should leave a record on the history tape sufficient to undo the operation in the subsequent Cleanup Stage. Since a deterministic Turing machine's quintuples have disjoint domains, it is sufficient to write $\langle i, j \rangle$ on the history tape. This can be achieved by replacing the above quintuple by the following quintuples:

(1) A nonshifting quintuple of the form

$$q_j, (S_j, b) \to (S_k, b), (0, 0), q'_{ij},$$

which executes the original quintuple's read/write operation while reading and writing blank on the history tape (second element in parentheses), and remembering i and j in a special control state;

(2) A quintuple of the form

$$q'_{ij}, X \to X, (\sigma, +), q''_{ij}$$

for each scanned-symbol combination X. Together these quintuples implement an oblivious shift, in which the history tape is right-shifted (+), while the other tapes perform the shift σ called for by the simulated computation. The memory of i and j is passed on via another special control state.

(3) A quintuple of the form

$$q''_{ij}, (Y, b) \to (Y, \langle i, j \rangle), (0, 0), q_l$$

for each scanned symbol combination Y on the nonhistory tapes. Together these quintuples implement a nonshifting read/write on the history tape, in which a blank is read and the information $\langle i, j \rangle$ representing the quintuple just executed is left in its place. Since each of these quintuples of types (1)-(3) has an inverse of the same form as itself, and since the history-taking and special starred control states guarantee disjointness of the quintuples' ranges, all operations of the Untidy Stage can be undone in the Cleanup Stage.

Theorem 1. We describe in more detail how the reversible machine causes operations to be performed in correct sequence, how it organizes the storage of the checkpoints on tape, and finally how it arrives at the satisfactory values for m and n without knowing T and S beforehand. We describe the case $k = 2$; generalization to larger k is straightforward. For the moment assuming m and n to be given, the hierarchical simulation is conveniently performed by a recursive routine such as RS(z,x,n,m,d) below, whose effect is to cause argument z, representing an ID of the simulated computation, to be incremented or decremented (according to whether argument d is +1 or −1) by the (m 2^n)th successor of argument x under the simulated computation's transition rule.

```
procedure RS(z,x,n,m,d)
 if (n=0)
  LINSIM(z,x,m,d)
 fi (n=0)
 if (n>0)
  RS(y,x,n-1,m,+1)
  RS(z,y,n-1,m,d)
  RS(y,x,n-1,m,-1)
 fi (n>0)
```

It should be noted that in reversible programs, the paths entering a merge point, like those leaving a branch point, must be labeled with mutually exclusive conditions. For example, in RS, the statement fi (n=0) denotes a merge point in which control may pass *from* the preceding block of code only if n=0. All assignments are done reversibly, presuming a knowledge of the previous value of the variable (initially zero).

All the actual simulation in RS above is done by the procedure LINSIM(z,x,m,d) below, which, according to the sign of d, increments or decrements argument z by the mth successor of argument x, computed by the method of Lemma 1, i.e., by creating, then clearing, an m-step history on a worktape reserved for that purpose.

```
procedure LINSIM(z,x,m,d)
 if (x is terminal or |x|>m-1)
  z:=z+d*x
 fi (x is terminal or |x|>m-1)
 if (x nonterminal and |x|<m)
  z:=z+d*[mth successor of x]
 fi (x nonterminal and |x|<m)
```

Note that LINSIM suspends forward computation, treating x as its own successor, when the computation being simulated terminates, or when the size of argument x exceeds the space bound m. This convention enables the main procedure, to be described later, to find the right segment size m $\approx S$ and recursion depth n $\approx \log(T/S)$.

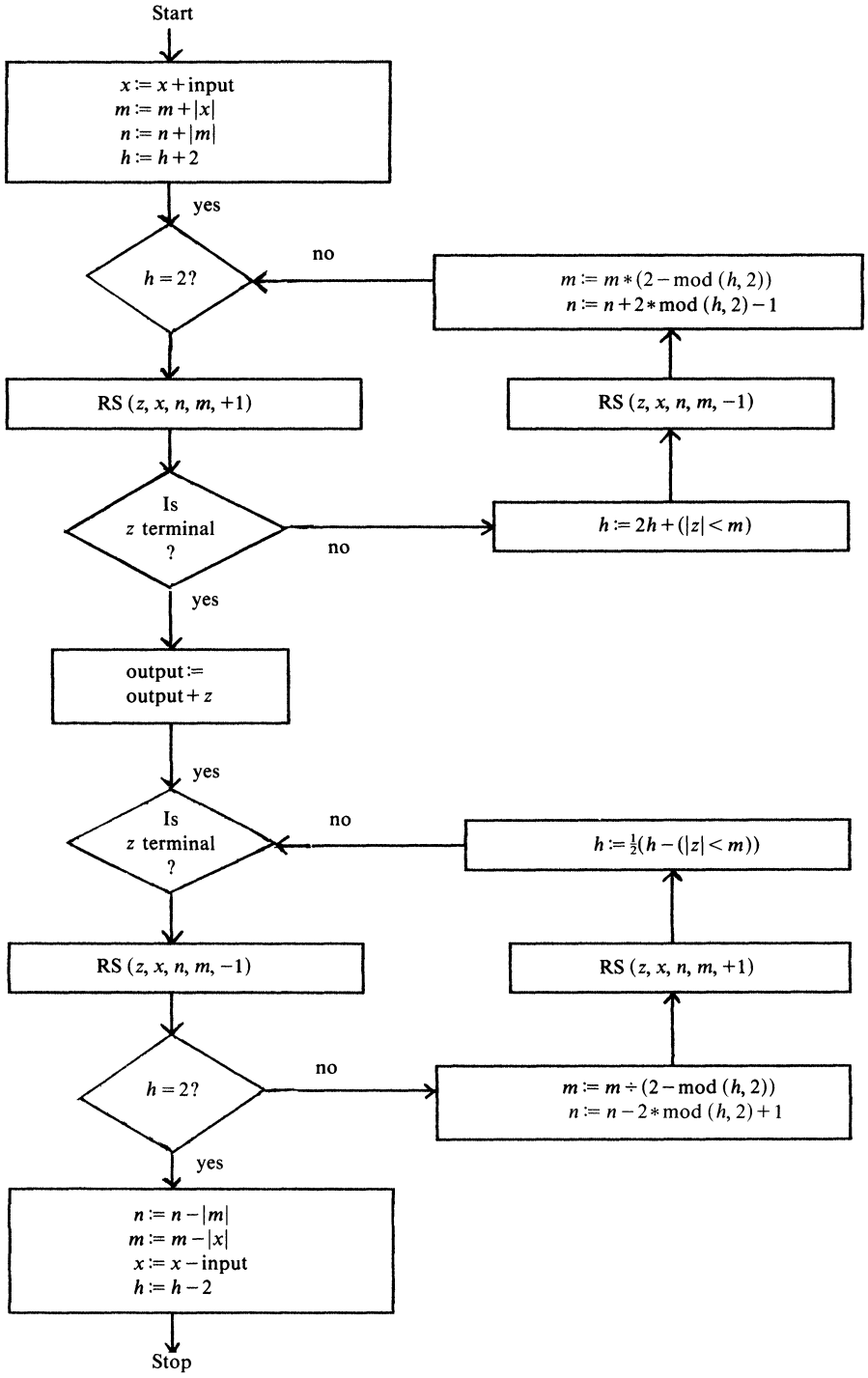

FIG. 1. MAIN *procedure for space-efficient reversible computation using recursive subroutine* "RS."

Proceeding now to the organization of data on tape, it can be seen that the procedure RS needs separate storage $O(\mathtt{m})$ at each level of recursion, for its arguments and for the local variable y. This storage can conveniently be organized as a sequence of n blocks of size $O(\mathtt{m})$ on a single "checkpoint" tape, demarcated by appropriate punctuation. Each call to RS would then involve reversibly passing arguments of size $O(\mathtt{m})$ back and forth between adjacent blocks, a job that can be done in time $O(\mathtt{m})$ if an extra blank tape is used as a buffer. The level of recursion would not have to be passed as an explicit argument, since that information would already be encoded by the head position on the checkpoint tape. At the 0 level of recursion, the linear time computation of each checkpoint from its predecessor would proceed using all the tapes of the original machine being simulated, plus an additional history tape.

We now go on to describe the main procedure, which tries successively larger m and n until the correct values are found, writes the final output (if the computation being simulated indeed terminates), and then reversibly disposes of all intermediate data generated in the course of the simulation. This procedure can be represented by a flow chart (Fig. 1). Here diamonds are used to represent both branch points and merge points since, as remarked before, it is necessary in reversible programming to label alternative paths into a merge point, like the paths out of a branch point, with mutually exclusive conditions. Note that the whole bottom half of the flow chart is devoted to undoing the work of the top half, so that all intermediate data are restored to their initial states after the desired output has been obtained.

In more detail, the main procedure begins by initializing m, n, and x, as well as a variable h, which will be used to encode the history of the search for correct m and n values. Each turn around the main loop in the upper half of the chart either increases n by one (thereby doubling the time allowance if it was exceeded before the space allowance), or doubles m and decreases n by one (thereby doubling the space allowance while keeping the time allowance fixed if the space allowance was exceeded first). In the former case a 0 is pushed into the least significant bit of h (regarded as a binary string); in the latter case a 1 is pushed. Since LINSIM simulates m steps of computation between space tests, the final m value may be less than S, but lies between $S/2$ and $2S$. The final n value is similarly equal to $\log(T/S)$ within a term of order unity. When the final m and n are found, the ID z generated by RS will be found to be terminal. It is then copied onto the output tape, and the entire computation up to that point is undone in the bottom half of the chart.

The space used by the algorithm is dominated by the $O(\mathtt{mn}) \leqq O(S \log(T))$ squares used on the checkpoint tape, all other terms being $O(S)$ or less. The time used in one turn around the main loop depends on m and n as $O(\mathtt{m} \cdot (2k-1)^{\mathtt{n}})$, from $O((2k-1)^{\mathtt{n}})$ recursive invocations of RS and $O((2k-1)^{\mathtt{n}})$ m-step linear-time simulations. Since n may be decreased at some turns around the loop (recall that when the space bound is exceeded, m is doubled and n is decreased by 1), the largest time cost is not necessarily associated with the last turn. Nevertheless, the total time cost of all the turns is bounded by a convergent geometric series

$$O\left((2k-1)^{\log T} \sum_i \left(\frac{k}{(2k-1)}\right)^i\right).$$

Executing the bottom half of the flow chart of course only introduces a factor of order unity, so the total time cost remains $O(T^{1+1/(\log k)})$ as claimed. □

Acknowledgments. I wish to thank Leonid Levin for urging and helping me to strengthen a previous, much weaker version of Theorem 1 and Martin Tompa for pointing out how to reduce the time exponent from 1.585 to $1+\varepsilon$.

Note added in proof. Robert Y. Levine and Alan T. Sherman, in a recent preprint, "A Note on Bennett's Time-Space Trade-off for Reversible Computation" (Department of Computer Science, Tufts University, Medford, Massachusetts 02155), give tighter time and space bounds for the reversible simulation as $\Theta(T^{1+\epsilon}/S^{\epsilon})$ and $\Theta(S \ln(T/S))$, respectively; and they point out that the space bound contains a constant factor diverging with small ϵ approximately as $\epsilon 2^{1/\epsilon}$.

REFERENCES

[1] C. H. BENNETT, *Logical reversibility of computation*, IBM J. Res. Develop., 17 (1973), pp. 525-532.

[2] ———, *The thermodynamics of computation—a review*, Internat. J. Theoret. Phys., 21 (1982), pp. 905-940.

[3] D. COPPERSMITH AND R. CLEVE, private communication.

[4] R. CLEVE, *Methodologies for designing block ciphers and cryptographic protocols*, Ph.D. thesis, Computer Science Department, University of Toronto, Toronto, Ontario, Canada, 1988.

[5] D. COPPERSMITH AND E. GROSSMAN, *Generators for certain alternating groups with applications to cryptography*, SIAM J. Appl. Math., 29 (1975), pp. 624-627.

[6] E. FREDKIN AND T. TOFFOLI, *Conservative Logic*, Internat. J. Theoret. Phys., 21 (1982), pp. 219-253.

[7] R. LANDAUER, *Irreversibility and heat generation in the computing process*, IBM J. Res. Develop., 5 (1961), pp. 183-191.

[8] L. LEVIN, *Universal sequential search problems*, Problems Inform. Transmission, 10 (1973), pp. 206-210.

[9] ———, *Randomness conservation inequalities; Information and independence in mathematical theories*, Inform. and Control, 61 (1984), pp. 15-37.

[10] J. MCCARTHY, *The inversion of functions defined by Turing machines*, in Automata Studies, C. E. Shannon and J. McCarthy, eds., Princeton University Press, Princeton, NJ, 1956.

[11] L. PRIESE, *On a simple combinatorial structure sufficient for sublying nontrivial self-reproduction*, J. Cybernetics, 6 (1976), pp. 101-137.

[12] T. TOFFOLI, "*Computation and construction universality of reversible cellular automata*, J. Comput. System Sci., 15 (1977), pp. 213-231.

[13] ———, *Reversible computing*, MIT Report MIT/LCS/TM-151, Mass. Inst. of Technology, Cambridge, MA, 1980.

[14] N. MARGOLUS, *Physics-like models of computation*, Phys. D., 10 (1984), pp. 81-95.

[15] J. VON NEUMANN, *Theory of Self-Reproducing Automata*, A. W. Burks, ed., University of Illinois Press, Urbana, IL, 1966.