Reversible Space Equals Deterministic Space

Klaus-Jörn Lange * Univ. Tübingen
Pierre McKenzie † Univ. de Montréal
Alain Tapp † Univ. de Montréal

October 28, 1998

Abstract

This paper describes the simulation of an $S(n)$ space-bounded deterministic Turing machine by a reversible Turing machine operating in space $S(n)$. It thus answers a question posed by Bennett in 1989 and refutes the conjecture, made by Li and Vitanyi in 1996, that any reversible simulation of an irreversible computation must obey Bennett's reversible pebble game rules.

1 Introduction

A Turing Machine M is reversible iff the infinite graph of *all* configurations of M has indegree and outdegree one. Interest in reversibility arose at first in connection with the thermodynamics of computation, following Landauer's demonstration in 1961 that, contrary to earlier intuition (see [Ne66]), physical laws do not preclude using an arbitrarily small amount of energy to perform logically reversible computing steps [La61], see [Fey96, Chapter 5]. More recently, renewed interest in the notion of reversibility was sparked by the prospect of quantum computers, whose observation-free computational steps are intrinsically reversible [De85, Sh94, Br95].

Early strategies to make a Turing machine reversible were terribly wasteful in terms of space: Lecerf's method [Le63], rediscovered by Bennett [Be73], required space T to simulate a $T(n)$-time $S(n)$-space machine reversibly in time $O(T)$. Bennett then greatly improved on this by reducing the space to $O(S \log T)$ at the expense of an increase in time to $T^{1+\epsilon}$ [Be89]. Levine and Sherman refined the analysis of Bennett's algorithm and characterized the tradeoff between time and space even more precisely [LeSh90].

Bennett questioned [Be89] whether the reversible simulation of an irreversible computation necessarily incurs a non constant factor increase in space usage. Bennett offered both a pebble game formalizing his intuition that the space increase

*Wilhelm-Schickard-Institut für Informatik, Universität Tübingen, Sand 13, D–72076 Tübingen, Germany. lange@informatik.uni-tuebingen.de

†Dép. d'informatique et de recherche opérationnelle, Université de Montréal, C.P. 6128 succursale Centre-Ville, Montréal, H3C 3J7 Canada. Work supported by NSERC of Canada and by FCAR du Québec. {mckenzie,tappa}@iro.umontreal.ca

is unavoidable, and a possible source of contrary evidence arising from the suprisingly width-efficient reversible simulations of irreversible circuits. At the 1996 IEEE Computational Complexity conference, Li and Vitanyi took up Bennett's suggestion and performed an in-depth analysis of Bennett's pebble game [LiVi96a, LiVi96b]. They proved that any strategy obeying Bennett's game rules indeed requires the extra $\Omega(\log T)$ multiplicative space factor, and they exhibited a trade-off between the need for extra space and the amount of irreversibility (in the form of irreversibly erased bits) which might be tolerated from the simulation. Li and Vitanyi then conjectured that all reversible simulations of an irreversible computation obey Bennett's pebble game rules, hence that all such simulations require $\Omega(S \log T)$ space.

Here we refute Li and Vitanyi's conjecture: Using a strategy which of course does not obey Bennett's game rules, we reversibly simulate irreversible space S computations in space S. Our strategy is the extreme opposite of Lecerf's "space-hungry" method: While we scrupulously preserve space, time becomes exponential in S. We offer two reasons to justify interest in such a "time-hungry" method. First, the new method is proof that Bennett's game rules do not capture all strategies, leaving open the possibility that, unobstructed by Li and Vitanyi's lower bounds, more efficient reversible simulations of irreversible computation should exist. Secondly, for problems in DSPACE(log), Bennett's simulation uses space ($\log^2$), while our new method uses only $\log n$ space and polynomial time. This could be interesting in the context of quantum computing, for then, space seems to be more of a concern than time. (Storing many entangled q-bits seems more difficult than applying basic unitary transformations.)

As a side effect, our result implies the reversible $O(S^2(n))$ space simulation of $NSPACE(S(n))$ by Crescenzi and Papadimitriou [CrPa95].

Section 2 in this paper contains preliminaries and discusses the notion of reversibility. Section 3 presents our main result, first in detail in the context of linear space, and then in the more general context of any space bound. Section 4 concludes with a discussion.

2 Preliminaries

We assume familiarity with basic notions of complexity theory such as can be found in [HoUl79]. We refer to the finite set of states of a Turing machine as to its set of "local states". Turing machine tapes extend infinitely to the right, and unused tape squares contain the blank symbol B. We assume that, when computing a function f on input x, a Turing machine halts in a unique final configuration.

2.1 Reversible Turing machines

If a Turing machine M is deterministic, then each node in the configuration graph of M has outdegree at most one. A deterministic TM is reversible if this restriction holds also for the indegree:

Definition. A Turing machine is *reversible* iff the infinite graph of M's configurations, in which an arc (C, C') indicates that M can go from configuration C to configuration C' in a single transition, has indegree and outdegree at most one. ■

The above definition of reversibility does not address the question of whether the reverse computation necessarily halts. The usual assumption, sometimes implicitly made, is that the reverse computation is required to halt. But one could also consider a weaker notion of reversibility in which the reverse computation may run forever. In the sequel, we enforce halting of the reverse computations, unless we explicitly state otherwise.

Without restricting its computational power, we can separate in a deterministic machine the actions of moving and writing. A transition is either *moving* (i.e. the tape head moves), or *stationary*. Each moving transition must be oblivious (i.e. it depends only on the local state and not on the symbol read from the tape). That is, a moving transition has the form $p \to q, \pm 1$ meaning that from state p the machine makes one step to the right (+1) resp. to the left (-1) and changes into state q without reading or writing any tape cell. A stationary (or writing) transition has the form $p, a \to q, b$ meaning that in state p the machine overwrites an a by a b and changes into state q without moving the head.

Following Bennett [Be89], we impose the following restrictions on the transitions of a Turing machine with separated moving and writing steps and claim that these are sufficient to imply reversibility. We require that no pair of transitions *intersect*, in the following sense. We say that two stationary transitions intersect if their execution leaves the machine in the same local state with the same symbol under the tape head. We say that two moving transitions intersect if they lead from different local states to the same local state. Finally, we say that a stationary transition and a moving transition intersect if they lead to the same local state.

We can extend these syntactic restrictions on the transitions of a machine to the case of a multi-tape machine, but these become trickier to describe. Intuitively however, we only require that the local state and symbols under the heads uniquely determine the most recent transition.

2.2 Computing functions reversibly

Unlike in the deterministic computation of a function f on input x, the tape content beyond the value of $f(x)$ at the end of a reversible computation is a concern. In any reversible computation model (even other than Turing machines), one must know the content of a memory cell in order to use this cell in some useful computation later. This is because *erasing is a fundamentally irreversible action.* Since spoiled memory is no longer useful, it is important that the memory be restored to its initial state at the end of a reversible computation.

Two notions of memory "cleanliness" have been studied in the literature: These correspond to input-saving and to input-erasing Turing machines [Be89]. In an input-saving computation, the final tape content includes x as well as $f(x)$. In an input-erasing computation, only $f(x)$ remains on the tape.

In this paper, we observe that reversibly computing an arbitrary function f in an input-erasing way is possible, *without* first embedding f in an injective function. When discussing space bounds that are at least linear, we thus adopt the input-erasing mode of computing f, even when f is arbitrary. Our reversible machine on input x potentially cycles through all the inverse images of $f(x)$: the lack of injectivity of f only translates into the fact that the machine cannot tell which of these inverse images was its actual input x. (This is no problem in the weak model of reversibility which allows the reverse computation to run forever. In the stronger model, care is needed, as explained at the end of the next section.) When discussing sublinear space bounds, we are forced to equip our Turing machines with a read-only input tape, which in effect implies the input-saving mode of oomputation.

3 Main result

In Section 3.1, we prove in detail that bijective functions computable in space n can be computed reversibly with no additional space. Then, in Section 3.2, we generalize our simulation to the cases of non-bijective functions, and of general (even non-constructive) space bounds.

3.1 Bijective linear-space computable functions

Theorem 3.1 *Any bijective function computed in space equal to the input length can be computed reversibly in the same space.*

Proof. We begin by describing the idea intuitively. As with Bennett's reversible simulations of irreversible computations, our high level strategy is simple, but care is needed when filling in the details because the syntactic conditions required at the transition function level can be tricky to enforce and verify.

The main idea for simulating a machine without using more space is to reversibly cycle through the configuration tree of the machine. For our purposes, it will suffice to consider the configuration tree as an undirected tree, in which each edge is duplicated. We will then in effect perform an Euler tour of the resulting tree. A similar technique was used by Sipser [Si80] to simulate an $S(n)$ space-bounded Turing machine by another $S(n)$ space-bounded machine which never loops on a finite amount of tape.

Let $G_\infty(M)$ be the infinite configuration graph of a single worktape linear space deterministic Turing machine M. Write $C_0(w) = \binom{B}{\vec{q_0}} \dagger w_1 w_2 \cdots w_n \ddagger$ for the initial configuration of M on input $w = w_1 w_2 \ldots w_n \in \Sigma^*$, that is, the local initial state of M is $\vec{q_0}$, M's input is placed between a left marker $\dagger$ and a right marker $\ddagger$, and the tape head of M is initially scanning a blank symbol B immediately to the left of the left marker (these initial conditions are chosen for convenience later). By convention, M on input w will eventually halt in a unique local state with $B \dagger f(w) \ddagger$ on its tape. Consider the weakly connected component[1] G_w of $G_\infty(M)$ which contains $C_0(w)$

[1]Induced by the connected component in the associated undirected graph.

and which contains no configuration of M using more than linear space between the left and right markers. Without loss of generality we make the following assumptions concerning M and G_w:

- M's first transition from $q_0^{\rightarrow}$ is a right head motion into local state q_0, and from then on, M never moves to the left of †.
- M never moves to the right of ‡, hence the finite graph G_w contains all the relevant configurations of M on input w.
- The indegree of $C_0(w)$ is zero. This is done by preventing the deterministic machine M from ever transiting back into its initial local state $q_0^{\rightarrow}$.
- The outdegree of any final configuration of M is zero.
- G_w contains no directed cycle. In other words, we assume that M's computation on w necessarily ends in a final configuration. Note that some linear space configurations of M could participate in directed cycles, but that these could not be weakly connected to $C_0(w)$.
- G_w is acyclic even when its arcs are replaced by (undirected) edges. This follows from our previous assumption because M is deterministic and hence the directed G_w has outdegree one.

We think of G_w as a tree with a final configuration as root at the bottom, and with $C_0(w)$ as one of its leaves on top. Observe that the irreversibility of M translates precisely into the fact that G_w is a tree and not simply a line. Our idea is to design a reversible Turing machine R whose configurations on input w will form a line, which will include, among other configurations, a representation of sufficiently many configurations of G_w to reach the final configuration of M in G_w. An obvious idea to design such an R is to have R perform an Euler tour of G_w (more precisely, of the undirected tree obtained from G_w by replacing each directed arc by two undirected edges). The difficulty is to ensure reversibility of R at the transition function level.

To simplify the presentation of the machine R, we will make use of a construction developed in [CoMc87] for the purpose of showing that permutation representation problems are logspace-complete. We now recall this construction by an example. Denote by T the tree with nodes $\{1,2,3,4,5,6,7\}$ and with edges $\{a,b,c,d,e,f\}$ depicted on Figure 1. $|\Omega_T|$ equals twice the number of edges in the tree, Let $\Omega_T = \{e_1, e_3, f_2, f_4, b_3, b_6, c_4, c_6, d_5, d_6, a_6, a_7\}$ be the set of "edge ends". Note that each edge has two "ends", hence $|\Omega_T|$ equals twice the number of edges in the tree. Two permutations on Ω_T will be constructed. We first construct the "ROTATION permutation" π_T. To define π_T, fix, locally at each node N in the tree, an arbitrary ordering of the "edge ends incident with N". Then let π_T be the product, over each node N, of a cycle permuting the edge ends incident with N according to this

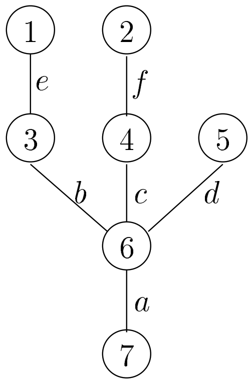

Figure 1: Example

ordering. In our example,

$$\pi_T = (b_3 e_3)(c_4 f_4)(a_6 b_6 c_6 d_6),$$

where we have chosen the alphabetical ordering of the incident edges as the local ordering at each node. We then construct the "SWAP permutation" σ_T, defined simply as a product of as many transpositions as there are edges in T:

$$\sigma_T = (e_1 e_3)(f_2 f_4)(b_3 b_6)(c_4 c_6)(d_5 d_6)(a_6 a_7).$$

Our interest lies in the permutation $\pi_T \sigma_T = (e_1 e_3 b_6 c_4 f_2 f_4 c_6 d_5 d_6 a_7 a_6 b_3)$. Although we have described the construction of π_T and σ_T in the context of a simple example, the construction as it applies to a general tree T should be clear. Among the properties of $\pi_T \sigma_T$ proved in [CoMc87], we will only make use of the property that $\pi_T \sigma_T$ is a single cycle which includes precisely all the elements in Ω_T. Our reversible Turing machine R will simulate the computation of $\pi_{G_w} \sigma_{G_w}$. In this way, R will traverse the computation tree of M on input w, by traversing the cycle $\pi_{G_w} \sigma_{G_w}$ (reversibly, $\pi_{G_w} \sigma_{G_w}$ being a permutation of Ω_{G_w}). Since the unique final configuration of M present in G_w is necessarily the final configuration reached by M on input w, R can reach the same final decision as that reached by M on input w.

In terms of the above construction, we now complete the description of the reversible machine R simulating the reversible machine M computing a bijective function $f : \Sigma^* \to \Sigma^*$ in linear space. We assume that R's input tape is also R's (only) worktape. The initial configuration of R will have $B \dagger w \ddagger$ on its tape, with R's tape head under the leftmost B, and the final configuration of R will have $B \dagger f(w) \ddagger$ on its tape.

To simplify the exposition of our construction, we further assume that the moving steps of M are distinguishable syntactically from the writing steps of M, and that these alternate. In other words, the state set of M consists in a set Q of stationary states together with two disjoint copies of Q, denoted $Q^{\rightarrow}$ and $Q^{\leftarrow}$, such that the transition function δ imposes an alternation between moving steps and stationary steps. Specifically, $\delta(q, a) \in (Q^{\rightarrow} \cup Q^{\leftarrow}) \times \Gamma$ for each $q \in Q$ and $a \in \Gamma$, and for all $q \in Q$ we have $\delta(q^{\rightarrow}) = q, +1$ as well as $\delta(q^{\leftarrow}) = q, -1$; hence, in a moving step, there is no "real" change of state. In the sequel we use names like p or q' to denote elements of Q.

Our reversible algorithm consists in iterating an "Euler stage", which we now describe. The simple setup of this process will be discussed afterwards. We first describe the local state set of R. The basic local states of R will consist in three components:

First component: symbol σ (for swap) or π (for rotation) indicating which of the two permutations between edge ends is to be applied next.

Third component: symbol s or t indicating whether it is the source or the target of the current directed edge, in the configuration tree G_w, which is the current edge end.

Second component: piece of local information which, together with the current tape of R and the third component just described, uniquely identifies the current directed edge (and thus the current edge end). This second component is an ordered pair of items, each item itself a state-symbol pair. The first item in the ordered pair pertains to the source configuration of the directed edge. The second item pertains to the target configuration of the directed edge. Redundancy is built into this representation in order to simplify our construction. The specifics of the second component are given below. A stationary (resp. moving) edge is an edge out of a configuration of M in which the state is stationary (resp. moving):

Second component, for a stationary directed edge. A stationary transition will be represented by the second component $[qb, q'^{\rightarrow}b']$ (resp. $[qb, q'^{\leftarrow}b']$), indicating that M in state q reading b would replace b with b' and enter state $q'^{\rightarrow}$ (resp. $q'^{\leftarrow}$). In both configurations of M participating in this transition, the head position is that of the head of R, and the tape symbols away from the head of M are given by the corresponding symbols away from the head of R.

Second component, for a moving directed edge. Recall that M alternates between moving and stationary states. A moving transition would thus be represented by a second component $[q^{\leftarrow}a, qb]$ or $[q^{\rightarrow}a, qb]$ indicating that M in state $q^{\leftarrow}$ (resp. $q^{\rightarrow}$) positioned over a symbol a moves (without reading the a) one cell to the left (resp. right) which contains a b. The representation is completed by equating the head position of M and the symbols away from the head of M, in the *current* edge end (specified by the third component of the basic state of R), to the head position of R and the symbols away from the head of R.

This completes the description of the basic local states of R. Additional auxiliary states will be introduced as needed in the course of the construction.

EULER STAGE: At the beginning of an Euler stage, R internally represents some edge end $ee \in \Omega_{G_w}$. The goal of the stage is to replace ee with $\sigma_{G_w}(\pi_{G_w}(ee))$. This is done in two substages distinguishing between the swapping of edge ends by σ and the rotation of edge ends around a configuration by π. In the following we will consider the cases where no left or right endmarker is involved. The appropriate changes to deal with these are straightforward[2].

PART 1: Realization of σ.

Case 1: Moving edges. The σ-permutation of an end of a moving edge simply requires exchanging s and t to reflect the fact that the ends (of the directed edge which itself remains the same) are swapped, and performing a head motion to maintain the correct representation of the edge. When the current edge end is the *source* of the edge, the head motion is that made when M traverses the edge. When the current edge end is the *target* of the edge, the direction of the move is reversed:

$$\begin{array}{rcl}
\langle \sigma, [q^{\leftarrow}a, qb], s\rangle & \rightarrow & \langle \pi, [q^{\leftarrow}a, qb], t\rangle, -1 \\
\langle \sigma, [q^{\rightarrow}a, qb], s\rangle & \rightarrow & \langle \pi, [q^{\rightarrow}a, qb], t\rangle, +1 \\
\langle \sigma, [q^{\leftarrow}a, qb], t\rangle & \rightarrow & \langle \pi, [q^{\leftarrow}a, qb], s\rangle, +1 \\
\langle \sigma, [q^{\rightarrow}a, qb], t\rangle & \rightarrow & \langle \pi, [q^{\rightarrow}a, qb], s\rangle, -1
\end{array}$$

Case 2: Stationary edges. The σ-permutation of either end of a stationary edge of M results in a stationary transition of R. Note that this is the only situation in which R actually modifies its tape:

$$\begin{array}{rcl}
\langle \sigma, [qb, q'^{\leftarrow}b'], s\rangle, b & \rightarrow & \langle \pi, [qb, q'^{\leftarrow}b'], t\rangle, b' \\
\langle \sigma, [qb, q'^{\rightarrow}b'], s\rangle, b & \rightarrow & \langle \pi, [qb, q'^{\rightarrow}b'], t\rangle, b' \\
\langle \sigma, [qb, q'^{\leftarrow}b'], t\rangle, b' & \rightarrow & \langle \pi, [qb, q'^{\leftarrow}b'], s\rangle, b \\
\langle \sigma, [qb, q'^{\rightarrow}b'], t\rangle, b' & \rightarrow & \langle \pi, [qb, q'^{\rightarrow}b'], s\rangle, b
\end{array}$$

PART 2: Realization of π. While realizing the permutation σ was very easy, the construction for π is more involved. In a way similar to above, here we distinguish between rotating around a moving edge end (i.e. an edge end containing a moving state), and rotating around a stationary edge end.

Case 1: Rotation around moving configurations. We only treat the case of a right-moving state $q'^{\rightarrow} \in Q^{\rightarrow}$. The left-moving case is dual. Rotation around a right-moving configuration involves the source end $q'^{\rightarrow}b'$ of the second component $[q'^{\rightarrow}b', q'c]$ of a basic state of R, if c is the tape symbol to the right of b', and it also involves the k edges $[q_ib_i, q'^{\rightarrow}b'], 1 \leq i \leq k$. Here $(q_ib_i)_{1\leq i\leq k}$ is the ordered sequence of all possible predecessors of $q'^{\rightarrow}b'$ w.r.t. δ (see Figure 2). Observe that $k = 0$ is possible. We then define the steps of R in the following way:

[2]For example, the assumption that M never moves to the right of ‡ implies that, when the b depicted on the right hand side diagram of Figure 2 equals †, R must be defined as if the transition from $q^{\leftarrow}c$ to qb were absent. This is required in order to prevent R from wandering away into an irrelevant and potentially infinite part of the configuration graph of M.

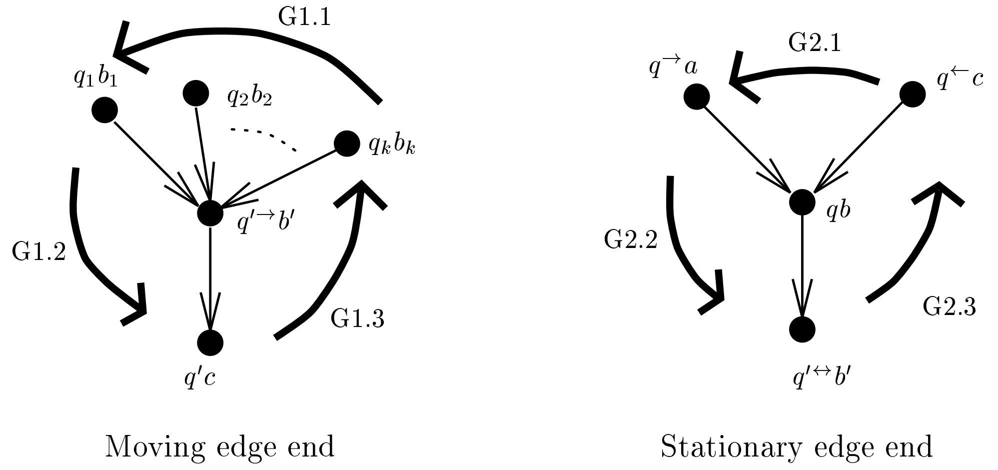

Figure 2: The cases arising in PART 2, handling the π-permutation (rotation). The diagram on the left depicts a rotation around a moving configuration. The diagram on the right depicts a rotation around a stationary configuration having a successor.

Subcase 1.1 : $k = 0$. Then the single incident edge end remains unchanged:

$$\langle \pi, [q'^{\rightarrow}b', q'c], s\rangle, b' \quad \rightarrow \quad \langle \sigma, [q'^{\rightarrow}b', q'c], s\rangle, b'$$

Subcase 1.2 : $k \geq 1$. We cycle through all target edge ends of edges originating from predecessors of $q'^{\rightarrow}b'$, all the way down to the target edge end of the edge from q_1b_1. The latter is mapped to the edge end of the edge leading from configuration $q'^{\rightarrow}b'$ to its successor, which in turn is mapped to the target edge end of the edge leading from q_kb_k to $q'^{\rightarrow}b'$. To realize this, we need some auxiliary states which are named according to our case analysis.

Group 1.1 : This group of instructions implements the core of the Euler tour idea, in that it is precisely here that a forward computation of the deterministic machine M is changed into a backward computation. For each "inner" edge end q_ib_i, $2 \leq i \leq k$ we set:

$$\langle \pi, [q_ib_i, q'^{\rightarrow}b'], t\rangle, b' \quad \rightarrow \quad \langle \sigma, [q_{i-1}b_{i-1}, q'^{\rightarrow}b'], t\rangle, b'$$

Group 1.2 : The "edge end" q_1b_1 is mapped to the "successor edge end":

$$\begin{array}{rcl} \langle \pi, [q_1b_1, q'^{\rightarrow}b'], t\rangle & \rightarrow & \langle \iota 1.2.1, q'^{\rightarrow}, b'\rangle, +1 \\ \langle \iota 1.2.1, q'^{\rightarrow}, b'\rangle, c & \rightarrow & \langle \iota 1.2.2, q'^{\rightarrow}, b', c\rangle, c \\ \langle \iota 1.2.2, q'^{\rightarrow}, b', c\rangle & \rightarrow & \langle \sigma, [q'^{\rightarrow}b', q'c], s\rangle, -1 \end{array}$$

Group 1.3 : The "successor edge end" is mapped to the edge end of the edge from the last predecessor $q_k b_k$:

$$\begin{array}{rcl} \langle\sigma,[q'^{\rightarrow}b',q'c],s\rangle & \rightarrow & \langle\iota 1.3.1,q'^{\rightarrow},b',c\rangle,+1 \\ \langle\iota 1.3.1,q'^{\rightarrow},b',c\rangle,c & \rightarrow & \langle\iota 1.3.2,q'^{\rightarrow},b'\rangle,c \\ \langle\iota 1.3.2,q'^{\rightarrow},b'\rangle & \rightarrow & \langle\sigma,[q_k b_k,q'^{\rightarrow}b'],t\rangle,-1 \end{array}$$

Case 2: Rotation around stationary configurations. We only define the transitions below for $q \in Q$ and $b \in \Gamma$ such that $\delta(q,b)$ is defined and b is not an endmarker (the Euler tours of inaccessible portions of the configuration graph of M may thus get broken up into incomplete segments). Then there are precisely three edge ends incident with qb, corresponding to the successor edge and to the two predecessor edges induced by $q^{\rightarrow}$ and $q^{\leftarrow}$ (see Figure 2). We map the target corresponding to the predecessor $q^{\leftarrow}$ to the target corresponding to the predecessor $q^{\rightarrow}$:

Group 2.1 :

$$\begin{array}{rcl} \langle\pi,[q^{\leftarrow}c,qb],t\rangle & \rightarrow & \langle\iota 2.1.1,q,b,c\rangle,+1 \\ \langle\iota 2.1.1,q,b,c\rangle,c & \rightarrow & \langle\iota 2.1.2,q,b\rangle,c \\ \langle\iota 2.1.2,q,b\rangle & \rightarrow & \langle\iota 2.1.3,q,b\rangle,-1 \\ \langle\iota 2.1.3,q,b\rangle & \rightarrow & \langle\iota 2.1.4,q,b\rangle,-1 \\ \langle\iota 2.1.4,q,b\rangle,a & \rightarrow & \langle\iota 2.1.5,q,b,a\rangle,a \\ \langle\iota 2.1.5,q,b,a\rangle & \rightarrow & \langle\sigma,[q^{\rightarrow}a,qb],t\rangle,+1 \end{array}$$

Let $\delta(q,b) = (q'^{\leftarrow},b')$ (resp. $\delta(q,b) = (q'^{\rightarrow},b')$) for some $b' \in \Gamma$ and some $q' \in Q$. Group 2.2 maps the target end of the edge from the predecessor $q^{\rightarrow}a$ to the source edge end of the edge towards the successor $q'^{\leftarrow}b'$ (resp. $q'^{\rightarrow}b'$):

Group 2.2 :

$$\begin{array}{rcl} \langle\pi,[q^{\rightarrow}a,qb],t\rangle & \rightarrow & \langle\iota 2.2.1,q,a,b\rangle,-1 \\ \langle\iota 2.2.1,q,a,b\rangle,a & \rightarrow & \langle\iota 2.2.2,q,b\rangle,a \\ \langle\iota 2.2.2,q,b\rangle & \rightarrow & \langle\sigma,[qb,q'^{\leftarrow}b'],s\rangle,+1 \\ (\text{resp. } \langle\iota 2.2.2,q,b\rangle & \rightarrow & \langle\sigma,[qb,q'^{\rightarrow}b'],s\rangle,+1) \end{array}$$

Group 2.3 finally maps the source edge end of the "successor edge" to the target edge end of the edge from $q^{\leftarrow}$:

Group 2.3 :

$$\begin{array}{rcl} \langle\pi,[qb,q'^{\leftarrow}b'],s\rangle & \rightarrow & \langle\iota 2.3.3,q,b\rangle,+1 \\ (\text{resp. } \langle\pi,[qb,q'^{\rightarrow}b'],s\rangle & \rightarrow & \langle\iota 2.3.3,q,b\rangle,+1) \\ \langle\iota 2.3.3,q,b\rangle,c & \rightarrow & \langle\iota 2.3.4,q,b,c\rangle,c \\ \langle\iota 2.3.4,q,b,c\rangle & \rightarrow & \langle\sigma,[q^{\leftarrow}c,qb],t\rangle,-1 \end{array}$$

SETUP: Our Turing machine input conventions and the assumption made about M's first transition were chosen in such a way as to yield a sufficient amount of information to determine an initial local state of R which, together with R's initial tape content, correctly represents an accessible edge end of M on input w. Given the choices made, "setting up" then merely reduces to defining the initial local state of R as $\langle \sigma, [\vec{q_0} B, q_0 \dagger], s \rangle$.

TERMINATION: R halts when the current edge end involves the unique final local state q_f of M. This necessarily happens. (To see this, note that by [CoMc87], an edge into the unique final configuration of M necessarily becomes the current *edge* at some time t during the computation of R. Now suppose that the current edge *end* of the current edge at time t does not involve q_f. Either the permutation simulated at time $t-1$ was the swap permutation σ, or the permutation to be applied at time t is σ. In the former case, the current edge end at time $t-1$ must have involved q_f, and in the latter case, the current edge end at time $t+1$ will involve q_f.) In this way, R cuts out of the Euler tour on G_w the chain beginning at the initial configuration of M and ending at the (unique) configuration in G_w without a successor which marks the end of the computation of M.

It can be (somewhat tediously) checked that the machine R constructed above satisfies Bennett's local reversibility conditions, implying that R is reversible. This completes the proof of Theorem 3.1. ■

3.2 Generalizations

The proof of Theorem 3.1 does not rely on the injectivity of the function being computed, as we now argue in the case of the weak model of reversibility allowing reverse computations to run forever:

Corollary 3.2 *Any function computable in space n can be computed in space n with a reversible TM.*

Proof. Recall the proof of Theorem 3.1. The potential difficulty which now arises is that G_w, and hence also the Euler tour of G_w, will in general contain more than one legal initial configuration of M (i.e. R may traverse more than one configuration of M of the form $\binom{B}{\vec{q_0}} \dagger x \ddagger$ for some string x). But such configurations are handled just like many other indegree-zero nodes in G_w (for example, when realizing the rotation π around $\binom{B}{\vec{q_0}} \dagger x \ddagger$, the case $k = 0$ in the left part of Figure 2 applies). Hence, having many initial configurations affects neither the reversibility of R, nor the reachability by R of the unique final configuration of M on input w. The only consequence of the non-injectivity is that, upon halting, R has lost track of w, and hence, of the true initial configuration from which it started. ■

We can also adapt the above simulation to the case of general (non linear and not necessarily constructible) space bounds. We make use of the standard technique for avoiding constructibility, as applied for example by Sipser [Si80]:

Theorem 3.3 *Any function f computable irreversibly in space $S(n)$ can be computed by a reversible Turing machine in the same space.*

Proof. Recall the proof of Theorem 3.1. To meaningfully discuss non-constructible space bounds, we must drop the assumption that the space allotted to the irreversible machine M is delimited by a right marker ‡. Hence M's initial configuration $C_0(w)$ on input w is now $\binom{B}{q_0^{\rightarrow}} \dagger w$, and G_w is defined as the weakly connected component containing $C_0(w)$ and pruned of any configuration of M which uses more than $S(|w|)$ space. We maintain all other assumptions on M however, in particular, those involving the left marker †, the initial state $q_0^{\rightarrow}$, and the acyclicity of G_w. Finally, we maintain R's input convention[3], that is, R's initial configuration includes the right marker ‡.

Then, R will proceed initially exactly as in Theorem 3.1, under the tentative assumption that M will never move past the first blank to the right of its input (R remembers the position of this blank using its own right marker ‡). If R's tentative assumption is correct, then R completes the simulation as in Theorem 3.1. Otherwise, at some point, M attempts to "overwrite R's right marker". In that case, R considers M's transition as undefined, but R does not halt. R instead remembers that this has occurred, bounces back from this dead end[4], and carries on until some initial configuration of M, which may or may not be the initial configuration of M on w, is encountered (i.e. until some current edge end is found to involve $q_0^{\rightarrow}$). At this point, R shifts its right marker ‡ one square to the right, and proceeds under the revised assumption that M will not exceed the new space bound (now enlarged by one).

Although R periodically restarts its computation (progressively enlarging its allotted space) from an initial configuration which may or may not be legal and which may or may not correspond to the "true" initial configuration of M on w, we claim that R eventually computes $f(w)$ correctly. This is because any one of the initial configurations encountered, while R unsuccessfully operated within a given space bound, necessarily led M to the same fateful configuration in which M tried to "write over R's right marker" for the first time within that round. Hence, all these initial configurations, upon restarting with a larger allotted space, lead M to the common fateful configuration (which now has a valid successor). All these initial configurations thus eventually lead M to the same final $f(w)$. Hence R computes $f(w)$ correctly, and thus, obviously, within space $S(|w|)$.

We leave the detailed verification that R is reversible to the reader. This completes the proof. ■

Remark. Theorem 3.3 applies as well when the machine is equipped with a read-only input tape (implying that the input-saving mode of computation is used), in which case we can assume for example that a work tape contains $B \dagger \ddagger$ initially. A read-only input tape is required, for example, to meaningfully discuss sublinear space bounds. Hence, the results of Theorem 3.3 hold as well when $S(n)$ is sublinear, and in particular, when the space bound is $\log n$.

[3] The reader should not be alarmed by the fact that M and R have different input conventions; a superficially different convention, common to both machines, could be devised.

[4] In other words, the edge just traversed, leading to qb on the right hand side diagram of Figure 2, is considered as the only edge into qb.

Remark. To fulfill the claim that we refute Li and Vitanyi's conjecture, we must use the same model of reversibility as they. Hence we must justify that Corollary 3.2 and Theorem 3.3 hold in the stronger model in which reverse computations are required to halt. Corollary 3.2 and Theorem 3.3 do hold, in the stronger model, when the input-saving mode is used, because the reverse computation can then detect the forward computation's initial configuration. This completely takes cares of sublinear space bounds. For space bounds $S(n) \geq n$, in the input-erasing mode, we arrange for the reversible machine operating on a length-n input to count, during the Euler tour, the number of initial configurations encountered involving inputs of length n. The reverse computation then decreases this counter, and halts when the counter reaches zero. This counting requires an additional $\log n$ bits to store the length of the input and some further bits to represent the counter. Since the number of initial configurations counted is at most 2^n, this counting incurs an additive space cost of at most $\log n + n$, which is within $O(S(n))$.

In combination with Savitch's theorem, our result implies the reversible simulation of nondeterministic space by Crescenzi and Papadimitriou:

Corollary 3.4 *[CrPa95] Any language in NSPACE($S(n)$) is accepted by a reversible TM using $O(S^2(n))$ space.* ■

4 Discussion

In this paper we showed determinism to coincide with reversibility for space. Meanwhile, the relativization results of Frank and Ammer suggest that deterministic computation cannot reversibly be simulated within the same time and space bounds if the time is neither linear nor exponential in the space [FrAm97].

It is interesting to compare our result with the equivalence of deterministic time and reversible time, which is simply shown by storing the whole computational history on a separate write-only tape [Le63, Be89]. It is remarkable that this is the very same construction which proves the equivalence of nondeterministic time and symmetric[5] time [LePa82]. This duality of the pairs nondeterminism versus symmetry and determinism versus reversibility is tied to the question of whether transitions can be regarded as directed or as undirected: This makes no difference if the indegree of every configuration is at most one.

The duality mentioned above and our new results point to the question of the relationship between nondeterministic space and symmetric space. In this case however, some recent results like the inclusion of symmetric logspace in parity logspace, SC^2, or DSPACE($\log^{4/3} n$) [KaWi93, Ni92, ArTaWiZh97] suggest that the computational power of nondeterministic space and symmetric space differ.

We mention in passing that in the case of time bounded auxiliary pushdown automata the situation is the opposite. While nondeterminism and symmetry coincide

[5]A nondeterministic machine is symmetric if each transition can be done in both directions: forward and backward.

for these devices, there seems to be no obvious way to simulate determinism in a reversible way [AlLa97].

Finally, we would like to mention that a further obvious consequence of our result is the observation that there are no space-bounded one-way functions.

Acknowledgments Special thanks go to Michael Frank, who pointed out that the reversible constructibility condition imposed on $S(n)$ in a former version of Theorem 3.3 was not required. Thanks also go to Paul Vitanyi, who made us aware of the infinite reverse computations present in some of our simulations, and who suggested alternatives. We also thank Richard Beigel, and the anonymous Complexity Theory conference referees, for their helpful comments.

References

[AlLa97] E. Allender and K.-J. Lange, Symmetry coincides with nondeterminism for time bounded auxiliary pushdown automata, *In preparation* 1997.

[ArTaWiZh97] R. Armoni, A. Ta-Shma, A. Wigderson and S. Zhou, $SL \subseteq L^{\frac{4}{3}}$, *Proc. of the 29th ACM Symp. on Theory of Computing* (1997).

[Be73] C. Bennett, Logical reversibility of computation, *IBM. J. Res. Develop.* **17** (1973), pp. 525–532.

[Be89] C. Bennett, Time/space trade-offs for reversible computation, *SIAM J. Comput.* **81**:4 (1989), pp. 766–776.

[Br95] G. Brassard, A quantum jump in Computer Science, in *Computer Science Today,* ed. J. van Leeuwen, Lecture Notes in Computer Science, Volume 1000 (special anniversary volume), Springer-Verlag (1995), pp. 1–14.

[Co85] S.A. Cook, A taxonomy of problems with fast parallel algorithms, *Information and Control* **64** (1985), pp. 2–22.

[CoMc87] S.A. Cook and P. McKenzie, Problems complete for deterministic logarithmic space, *J. Algorithms* **8** (1987), pp. 385–394.

[CrPa95] P.L. Crescenzi and C.H. Papadimitrou, Reversible simulation of space-bounded computations, *Theoret. Comput. Sci.* **143** (1995), 159 – 165.

[De85] D. Deutsch, Quantum theory, the Church-Turing principle and the universal quantum computer, *Proc. Royal Soc. London,* Series A **400** (1985), pp. 97–117.

[Fey96] R. Feynman, *Feynman lectures on computation*, Addison-Wesley (1996).

[FrAm97] M.P. Frank and M.J. Ammer, *Separations of reversible and irreversible space-time complexity classes*, manuscript, 1997.

[HoUl79] J.E. Hopcroft and J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation,* Addison-Wesley (1979).

[KaWi93] M. Karchmer and A. Wigderson, On span programs, *Proc. 8th IEEE Structure in Complexity Theory Conference* (1993), pp. 102–111.

[La61] R. Landauer, Irreversibility and heat generation in the computing process, *IBM J. Res. Develop.* **5** (1961), pp. 183–191.

[Le63] Y. LECERF, Machines de Turing réversibles. Insolubilité récursive en $n \in N$ de l'équation $u = \theta^n$, où θ est un "isomorphisme de codes", *Comptes Rendus* **257** (1963), pp. 2597–2600.

[LeSh90] R. LEVINE AND A. SHERMAN, A note on Bennett's time-space trade-off for reversible computation, *SIAM J. Comput.* **19**:4 (1990), pp. 673–677.

[LePa82] P. LEWIS AND C. PAPADIMITRIOU, Symmetric space-bounded computation, *Theoret. Comput. Sci.* **19** (1982), 161–187.

[LiVi96a] M. LI AND P. VITANYI, Reversible simulation of irreversible computation, *Proc. 11th IEEE Conference on Computational Complexity* (1996), pp. 301–306.

[LiVi96b] M. LI AND P. VITANYI, Reversibility and adiabatic computation: trading time and space for energy, *Proc. Royal Soc. London,* Series A **452** (1996), pp. 769–789.

[Ne66] J. VON NEUMANN, *Theory of self-reproducing automata,* A.W. Burks, Ed., Univ. Illinois Press, Urbana (1966).

[Ni92] N. NISAN, $RL \subseteq SL$, *Proc. 24th ACM Symp. on Theory of Computing* (1992), pp. 619–623.

[Sh94] P. SHOR, Algorithms for quantum computation: Discrete log and factoring, *Proc. 35th IEEE Symp. Foundations of Comp. Sci.* (1994), pp. 124–134.

[Si80] MICHAEL SIPSER, Halting Space-Bounded Computations, *Theoretical Computer Science 10* (1990),pp. 335–338.