Reversible Simulation of Irreversible Computation ⋆

Ming Li [1]

Department of Computer Science, University of Waterloo, Waterloo, Ont. N2L 3G1, Canada. E-mail: mli@math.uwaterloo.ca

John Tromp [2]

CWI, Kruislaan 413, 1098 SJ Amsterdam, The Netherlands; Email: tromp@cwi.nl

Paul Vitányi [3]

CWI, Kruislaan 413, 1098 SJ Amsterdam, The Netherlands; Email: paulv@cwi.nl

Computer computations are generally irreversible while the laws of physics are reversible. This mismatch is penalized by among other things generating excess thermic entropy in the computation. Computing performance has improved to the extent that efficiency degrades unless all algorithms are executed reversibly, for example by a universal reversible simulation of irreversible computations. All known reversible simulations are either space hungry or time hungry. The leanest method was proposed by Bennett and can by analyzed using a simple 'reversible' pebble game. The reachable reversible simulation instantaneous descriptions (pebble configurations) of such pebble games are characterized completely. As a corollary we obtain the reversible simulation by Bennett and, moreover, show that it is a space-optimal pebble game. We also introduce irreversible steps and give a theorem on the tradeoff between the number of allowed irreversible steps and the memory gain in the pebble game. In this resource-bounded setting the limited erasing needs to be performed at precise instants during the simulation. The reversible simulation can be modified so that it is applicable also when the simulated computation time is unknown.

⋆ Part of the results appeared in [15,14] with less satisfying analysis and proofs.

[1] Supported in part by the NSERC Operating Grant OGP0046506, ITRC, a CGAT grant, and the Steacie Fellowship

[2] Partially supported by the European Union through NeuroCOLT ESPRIT Working Group Nr. 8556, and by NWO through NFI Project ALADDIN under Contract number NF 62-376

[3] Partially supported by the European Union through NeuroCOLT ESPRIT Work-

1 Introduction

Both classical and quantum physics are believed to be strictly reversible: A complete description of the microscopic state of the system uniquely determines the earlier and future states of the system—this holds not only in Newtonian mechanics but for example also for the unitary evolution of any quantum mechanical system. Currently, computations are commonly irreversible, even though the physical devices that execute them are fundamentally reversible. This contrast is only possible at the cost of efficiency loss by generating thermal entropy into the environment. With computational device technology rapidly approaching the elementary particle level it has been argued many times that this effect gains in significance to the extent that efficient operation (or operation at all) of future computers requires them to be reversible (for example, in [20,10,1,2,5,9,19,14,6]). Especially R. Landauer [10] has argued that it is only the 'logically irreversible' operations in a physical computer that necessarily dissipate energy by generating a corresponding amount of entropy for every bit of information that gets irreversibly erased; the logically reversible operations can in principle be performed dissipation-free. Reversible computers can be implemented using classical technologies [2,16,6] or quantum-mechanical technologies as in [4,17,8]; the latter quantum-mechanical computers are reversible except for the observation phases.

The traditional models used in the analysis of computation, for example Turing machines, RAMs, or circuits, allow logically irreversible operations. To reflect physical reality they must be replaced by completely reversible computational models, for example by universal simulation. Simulation of irreversible Turing machines by reversible ones goes back to Lecerf [12] and Bennett [1]. The original methods required an amount of memory proportional to the amount of computation time, since the step-by-step reproducibility of the history was achieved by remembering it during most of the computation. It was recognized later that keeping the configuration only of certain times ("checkpoints") of the original computation can reduce the memory requirement at the expense of increasing the computing time.

Bennett in a "Remark" in [3] compares the "checkpointing" to moves in a certain pebble game. This paper takes up this suggestion to analyze time-space and space-irreversibility tradeoffs. It completely characterizes the realizable pebble configurations of the reversible pebble games (they encode the reachable instantaneous descriptions of a Turing machine reversibly simulating an irreversible computation). As corollary we obtain Bennett's earlier [3] simula-

ing Group Nr. 8556, and by NWO through NFI Project ALADDIN under Contract number NF 62-376 and NSERC under International Scientific Exchange Award ISE0125663. Affiliations are CWI and the University of Amsterdam.

tion and a first proof that this simulation is a space-optimal pebble game. It also introduces irreversible steps and gives a theorem on the tradeoff between the number of allowed irreversible steps and the memory gain in the pebble game. For such a tradeoff the limited irreversible actions have to take place at precise times during the reversible simulation, and cannot be delayed to be executed all together at the end of the computation (as is possible in computations without time or space resource bounds). Finally, in all such reversible simulations it is assumed that the number of steps to be simulated is known in advance and used to construct the simulation (for that number of steps). We show how to reversibly simulate an irreversible computation of unknown computing time, using the same order of magnitude of simulation time.

1.1 Reversible Turing Machines

In the standard model of a Turing machine the elementary operations are rules in quadruple format (p, s, a, q) meaning that if the finite control is in state p and the machine scans tape symbol s, then the machine performs action a and subsequently the finite control enters state q. Such an action a consists of either printing a symbol s' in the tape square under scan, or moving the scanning head one tape square left, right or not at all.

Quadruples are said to *overlap in domain* if they cause the machine in the same state and scanning the same symbol to perform different actions. A *deterministic Turing machine* is defined as a Turing machine with quadruples no two of which overlap in domain.

Now consider the special format (deterministic) Turing machines using quadruples of two types: *read/write* quadruples and *move* quadruples. A read/write quadruple (p, a, b, q) causes the machine in state p scanning tape symbol a to write symbol b and enter state q. A move quadruple $(p, *, \sigma, q)$ causes the machine in state p to move its tape head by $\sigma \in \{-1, 0, +1\}$ squares and enter state q, oblivious to the particular symbol in the currently scanned tape square. (Here '-1' means 'one square left', '0' means 'no move' and '$+1$' means 'one square right'.) Quadruples are said to *overlap in range* if they cause the machine to enter the same state and either both write the same symbol or (at least) one of them moves the head. Said differently, quadruples that enter the same state overlap in range unless they write different symbols. A *reversible Turing machine* is a deterministic Turing machine with quadruples no two of which overlap in range. A k-tape reversible Turing machine uses $(2k + 2)$ tuples which, for every tape separately, select a read/write or move on that tape. Moreover, any two tuples can be restricted to some single tape where they don't overlap in range.

To show that every partial recursive function can be computed by a reversible Turing machine one can proceed as follows. Take the standard irreversible Turing machine computing that function. We modify it by adding an auxiliary storage tape called the 'history tape'. The quadruple rules are extended to 6-tuples to additionally manipulate the history tape. To be able to reversibly undo (retrace) the computation deterministically, the new 6-tuple rules have the effect that the machine keeps a record on the auxiliary history tape consisting of the sequence of quadruples executed on the original tape. Reversibly undoing a computation entails also erasing the record of its execution from the history tape. This notion of reversible computation means that only $1:1$ recursive functions can be computed. To reversibly simulate an irreversible computation from x to $f(x)$ one reversibly computes from input x to output $\langle x, f(x)\rangle$.

1.2 Reversible Programming

Reversible Turing machines or other reversible computers will require special reversible programs. One feature of such programs is that they should be executable when read from bottom to top as well as when read from top to bottom. Examples are the programs $F(\cdot)$ and $A(\cdot)$ we show in the later sections. In general, writing reversible programs will be difficult. However, given a general reversible simulation of irreversible computation, one can simply write an oldfashioned irreversible program in an irreversible programming language, and subsequently simulate it reversibly. This leads to the following:

Definition 1 *An* irreversible-to-reversible compiler *receives an irreversible program as input and reversibly compiles it to a reversible program. Subsequently, the reversible program can be executed reversibly.*

Note that there is a decisive difference between reversible circuits and reversible special purpose computers on the one hand, and reversible universal computers on the other hand. While one can design a special-purpose reversible version for every particular irreversible circuit using reversible universal gates, such a method does not yield an irreversible-to-reversible compiler that can execute any irreversible program on a fixed universal reversible computer architecture as we are interested in here.

1.3 Models of Reversible Simulation and Related Work

The reversible simulation in [1] of T steps of an irreversible computation from x to $f(x)$ reversibly computes from input x to output $\langle x, f(x)\rangle$ in $T' = O(T)$ time. However, since this reversible simulation at some time instant has to record the entire history of the irreversible computation, its space use increases linearly with the number of simulated steps T. That is, if the simulated irreversible computation uses S space, then for some constant $c > 1$ the simulation uses $T' \approx c + cT$ time and $S' \approx c + c(S + T)$ space. This can be an unacceptable amount of space for many practically useful computations.

In [3] another elegant simulation technique is devised reducing the auxiliary storage space. This simulation does not save the entire history of the irreversible computation but it breaks up the simulated computation into segments of about S steps and saves in a hierarchical manner *checkpoints* consisting of complete instantaneous descriptions of the simulated machine (entire tape contents, tape heads positions, state of the finite control). After a later checkpoint is reached and saved, the simulating machine reversibly undoes its intermediate computation, reversibly erasing the intermediate history and reversibly canceling the previously saved checkpoint. Subsequently, the computation is resumed from the new checkpoint onwards.

The reversible computation simulates k^n segments of length m of irreversible computation in $(2k-1)^n$ segments of length $\Theta(m+S)$ of reversible computation using $n(k-1)+1$ checkpoint registers using $\Theta(m+S)$ space each, for every k, n, m.

This way it is established that there are various tradeoffs possible in time-space in between $T' = \Theta(T)$ and $S' = \Theta(TS)$ at one extreme $(k = 1, m = T, n = 1)$ and (with the corrections of [13]) $T' = \Theta(T^{1+\epsilon}/S^{\epsilon})$ and $S' = \Theta(c(\epsilon)S(1 + \log T/S))$ with $c(\epsilon) = \epsilon 2^{1/\epsilon}$ for every $\epsilon > 0$, using always the same simulation method but with different parameters k, n where $\epsilon = \log_k(2k-1)$ and $m = \Theta(S)$. Typically, for $k = 2$ we have $\epsilon = \log 3$. Since for $T > 2^S$ the machine goes into a computational loop, we always have $S \leq \log T$. Therefore, every irreversible Turing machine using space S can be simulated by a reversible machine using space S^2 in polynomial time. Let us note that it is possible to improve the situation by reversibly simulating *only the irreversible* steps. Call a quadruple of a Turing machine *irreversible* if its range overlaps with the range of another quadruple. A step of the computation is *irreversible* if it uses an irreversible quadruple. Let the number of irreversible steps in a T step computation be denoted by I. Clearly, $I \leq T$. The simulation results hold with T in the auxiliary space use replaced by I. In particular, $S' = O(S \log I)$. In many computations, I may be much smaller than T. There arises the problem of estimating the number of irreversible steps in a computation. (More complicatedly, one could extend the notion of irreversible step to those steps which can be reversed on local information alone. In some cases this is possible even when the used quadruple itself was irreversible.)

In a preliminary version of this paper [15], two of us proposed a quantitative study of exchanges of computing resources such as time and space for number of irreversible operations which we believe will be relevant for the physics of future computation devices. We *conjectured* that *all* reversible simulations of an irreversible computation can essentially be represented as the pebble game defined below, and that consequently the lower bound of Corollary 2 applies to all reversible simulations of irreversible computations. This conjecture was refuted in [11] using a technique due to [18] to show that there exists a general reversible simulation of an irreversible computation using only order S space at the cost of using a thoroughly unrealistic simulation time exponential in S.

In retrospect the conjecture is phrased too general: it should be restricted to *useful* simulations—using linear or slightly superlinear time and space *simultaneously*. The real question is whether there is a compiler that takes as input any irreversible algorithm A using S space and T time and produces a reversible algorithm B such that $B(x) = A(x)$ for all input x and using $T' = O(T)$ time and $S' = O(S)$ space. In the extreme cases of time and space use this is possible: If $S = \Theta(T)$ then the simulation in [1] does the trick, and if $T = \Theta(2^S)$ then the simulation of [11] works. For all other cases the pebble game analysis below has been used in [7] to show that any such simulation, if it exists, cannot relativize to oracles, or work in cases where the space bound is much less than the input length. (This is a standard method of giving evidence that the aimed-for result—here: simulation doesn't exist—is likely to be true in case the result itself is too hard to obtain.)

2 Reversible Pebbling

Let G be a linear list of nodes $\{1, 2, \ldots, T_G\}$. We define a *pebble game* on G as follows. The game proceeds in a discrete sequence of steps of a single *player*. There are n pebbles which can be put on nodes of G. At any time the set of pebbles is divided in pebbles on nodes of G and the remaining pebbles which are called *free* pebbles. At every step either an existing free pebble can be put on a node of G (and is thus removed from the free pebble pool) or be removed from a node of G (and is added to the free pebble pool). Initially G is unpebbled and there is a pool of free pebbles. The game is played according to the following rule:

Reversible Pebble Rule: If node i is occupied by a pebble, then one may either place a free pebble on node $i+1$ (if it was not occupied before), or remove the pebble from node $i+1$.

We assume an extra initial node 0 permanently occupied by an extra, fixed pebble, so that node 1 may be (un)pebbled at will. This pebble game is inspired by the method of simulating irreversible Turing Machines on reversible ones in a space efficient manner. The placement of a pebble corresponds to checkpointing the current state of the irreversible computation, while the removal of a pebble corresponds to reversibly erasing a checkpoint. Our main interest is in determining the number of pebbles k needed to pebble a given node i.

The maximum number n of pebbles which are simultaneously on G at any one time in the game gives the space complexity nS of the simulation. If one deletes a pebble not following the above rules, then this means a block of bits of size S is erased irreversibly. The limitation to Bennett's simulation is in fact space, rather than time. When space is limited, we may not have enough place to store garbage, and these garbage bits will have to be irreversibly erased. We establish a tight lower bound for *any* strategy for the pebble game in order to obtain a space-irreversibility tradeoff.

2.1 Reachable Pebble Configurations

We describe the idea of Bennett's simulation [3]. Given that some node s is pebbled, and that at least n free pebbles are available, the task of pebbling nodes $s+1, \ldots, s+2^n-1$ can be seen to reduce to the task of first pebbling nodes $s+1, \ldots, s+2^{n-1}-1$ using $n-1$ free pebbles, then placing a free pebble on node $s+2^{n-1}$, then unpebbling nodes $s+1, \ldots, s+2^{n-1}-1$ to retrieve our $n-1$ pebbles, and finally pebbling nodes $s+2^{n-1}+1, \ldots, s+2^n-1$ using these pebbles. By symmetry, an analogous reduction works for the task of unpebbling nodes $s+1, \ldots, s+2^n-1$ with n free pebbles. The following two mutually recursive procedures implement this scheme; their correctness follows by straightforward induction.

```
pebble(s, n)
{
  if (n = 0) return;
  t = s + 2^(n-1);
  pebble(s, n - 1);
  put a free pebble on node t
  unpebble(s, n - 1)
  pebble(t, n - 1);
}

unpebble(s,n)
{
  if (n = 0) return;
  t = s + 2^(n-1);
  unpebble(t, n - 1);
  pebble(s, n - 1)
```

remove the pebble from node t

unpebble($s, n-1$);

}

The difficult part is showing that this method is optimal. It turns out that characterizing the maximum node that can be pebbled with a given number of pebbles is best done by completely characterizing what pebble configurations are realizable. First we need to introduce some helpful notions.

In a given pebble configuration with f free pebbles, a placed pebble is called *available* if there is another pebble at most 2^f positions to its left (0 being the leftmost node). According to the above procedures, an available pebble can be removed with the use of the free pebbles. For convenience we imagine this as a single big step in our game.

Call a pebble configuration *weakly solvable* if there is a way of repeatedly removing an available pebble until all are free. Note that such configurations are necessarily realizable, since the removal process can be run in reverse to recreate the original configuration. Call a pebble configuration *strongly solvable* if all ways of repeatedly removing an available pebble lead to all being free. Obviously any strongly solvable configuration is also weakly solvable.

The starting configuration is obviously both weakly and strongly solvable. How does the single rule of the game affect solvability? Clearly, adding a pebble to a weakly solvable configuation yields another weakly solvable configuation, while removing a pebble from a strongly solvable configuation yields another strongly solvable configuation. It is not clear if removing a pebble from a weakly solvable configuation yields another one. If such is the case then we may conclude that all realizable configurations are weakly solvable and hence the two classes coincide. This is exactly what the next theorem shows.

Theorem 1 *Every weakly solvable configuration is strongly solvable.*

Proof. Let f be the number of free pebbles in a weakly solvable configuration. Number the placed pebbles $f, f+1, \ldots, n-1$ according to their order of removal. It is given that, for all i, pebble i has a higher-numbered pebble at most 2^i positions to its left (number the fixed pebble at 0 infinity). We know that pebble f is available. Suppose a pebble g with $g > f$ is also available—so there must be a pebble at most 2^f positions to its left. It suffices to show that if pebble g is removed first, then pebbles $f, f+1, \ldots, g-1$ are still available when their turn comes. Suppose pebble j finds pebble g at most 2^j places to its left (otherwise j will still be available after g's removal for sure). Then after removal of pebbles $g, f, f+1, \ldots, j-1$, it will still find a higher-numbered pebble at most $2^j + 2^f + 2^f + 2^{f+1} + \cdots + 2^{j-1} \leq 2^{j+1}$ places to its left, thus making it available given the extra now free pebble g. $\square$

Corollary 1 *A configuration with f free pebbles is realizable if and only if its placed pebbles can be numbered $f, f+1, \ldots, n-1$ such that pebble i has a higher-numbered pebble at most 2^i positions to its left.*

Corollary 2 *The maximum reachable node with n pebbles is $\sum_{i=0}^{n-1} 2^i = 2^n - 1$.*

Moreover, if pebble(s, n) takes $t(n)$ steps we find $t(0) = 1$ and $t(n) = 3t(n-1) + 1 = (3^{n+1} - 1)/2$. That is, the number of steps T'_G of a winning play of a pebble game of size $T_G = 2^n - 1$ is $T'_G \approx 1.53^n$, that is, $T'_G \approx T_G^{\log 3}$.

2.2 Tradeoffs

The simulation given in [3] follows the rules of the pebble game of length $T_G = 2^n - 1$ with n pebbles above. A winning strategy for a game of length T_G using n pebbles corresponds with reversibly simulating T_G segments of S steps of an irreversible computation using S space such that the reversible simulator uses $T' \approx ST'_G \approx ST_G^{\log 3}$ steps and total space $S' = nS$. The space S' corresponds to the maximal number of pebbles on G at any time during the game. The placement or removal of a pebble in the game corresponds to the reversible copying or reversible cancelation of a 'checkpoint' consisting of the entire instantaneous description of size S (work tape contents, location of heads, state of finite control) of the simulated irreversible machine. The total time $T_G S$ used by the irreversible computation is broken up in segments of size S so that the reversible copying and canceling of a checkpoints takes about the same number of steps as the computation segments in between checkpoints. [4]

We can now formulate a tradeoff between space used by a polynomial time reversible computation and irreversible erasures. First we show that allowing a limited amount of erasure in an otherwise reversible computation means that we can get by with less work space. Therefore, we define an m-*erasure* pebble game as the pebble game above but with the additional rule

- In at most m steps the player can remove a pebble from any node $i > 1$ without node $i-1$ being pebbled at the time.

An m-erasure pebble game corresponds with an otherwise reversible computation using mS irreversible bit erasures, where S is the space used by the irreversible computation being simulated.

[4] If we are to account for the permanent pebble on node 0, we get that the simulation uses $n+1$ pebbles for a pebble game with n pebbles of length $T_G + 1$. The simulation uses $n + 1 = S'/S$ pebbles for a simulated number of $S(T_G + 1)$ steps of the irreversible computation.

Lemma 1 *There is a winning strategy with $n+2$ pebbles and $m-1$ erasures for pebble games G with $T_G = m2^n$, for all $m \geq 1$.*

Proof. The strategy is to use 2 pebbles as springboards that are alternately placed 2^n in front of each other using the remaining n pebbles to bridge the distance. The most backward springboard can be erased from its old position once all n pebbles are cleared from the space between it and the front springboard. We give the precise procedure in self-explanatory pseudo PASCAL using the procedures given in section 2.1.

Procedure $A(n, m, G)$:
for $i := 0, 1, 2, \ldots, m-1$:
    pebble($i2^n, n$);
    put springboard on node $(i+1)2^n$;
    unpebble($i2^n, n$);
    if $i < m-1$ erase springboard on node $i2^n$;

The simulation time T'_G is $T'_G \approx 2m \cdot 3^{n-1} + 2 \approx 2m(T_G/m)^{\log 3} = 2m^{1-\log 3} T_G^{\log 3}$ for $T_G = m2^{n-1}$. □

Theorem 2 (Space-Irreversibility Trade-off) *(i) Pebble games G of size $2^n - 1$ can be won using n pebbles but not using $n-1$ pebbles.*

(ii) If G is a pebble game with a winning strategy using n pebbles without erasures, then there is also a winning strategy for G using E erasures and $n - \log(E+1)$ pebbles (for E is an odd integer at least 1).

Proof. (i) By Corollary 2.

(ii) By (i), $T_G = 2^n - 1$ is the maximum length of a pebble game G for which there is a winning strategy using n pebbles and no erasures. By Lemma 1, we can pebble a game G of length $T_G = m2^{n-\log m} = 2^n$ using $n + 1 - \log m$ pebbles and $2m - 1$ erasures. □

We analyze the consequences of Theorem 2. It is convenient to consider the special sequence of values $E := 2^{k+2} - 1$ for $k := 0, 1, \ldots$. Let G be Bennett's pebble game of Lemma 2 of length $T_G = 2^n - 1$. It can be won using n pebbles without erasures, or using $n - k$ pebbles plus $2^{k+2} - 1$ erasures (which gives a gain over not erasing as in Lemma 2 only for $k \geq 1$), but not using $n-1$ pebbles.

Therefore, we can exchange space use for irreversible erasures. Such a tradeoff can be used to reduce the space requirements of the reversible simulation. The correspondence between the erasure pebble game and the otherwise reversible computations using irreversible erasures is that if the pebble game uses $n-k$ pebbles and $2^{k+2}-1$ erasures, then the otherwise reversible computation uses $(n-k)S$ space and erases $(2^{k+2}-1)S$ bits irreversibly.

Therefore, a reversible simulation according to the pebble game of every irreversible computation of length $T=(2^n-1)S$ can be done using nS space using $(T/S)^{\log 3}S$ time, but is impossible using $(n-1)S$ space. It can also be performed using $(n-k)S$ space, $(2^{k+2}-1)S$ irreversible bit erasures and $2^{(k+1)(1-\log 3)+1}(T/S)^{\log 3}S$ time. In the extreme case we use no space to store the history and erase about $4T$ bits. This corresponds to the fact that an irreversible computation may overwrite its scanned symbol irreversibly at each step.

Definition 2 Consider a simulation according to the pebble game using S' storage space and T' time which reversibly computes $y=\langle x,f(x)\rangle$ from x in order to simulate an irreversible computation using S storage space and T time which computes $f(x)$ from x. The *irreversible simulation cost* $B^{S'}(x,y)$ of the simulation is the number of irreversibly erased bits in the simulation (with the parameters S,T,T' understood).

If the irreversible simulated computation from x to $f(x)$ uses T steps, then for $S'=nS$ and $n=\log(T/S)$ we have above treated the most space parsimonious simulation which yields $B^{S'}(x,y)=0$, with $y=\langle x,f(x)\rangle$.

Corollary 3 (Space-Irreversibility Trade-off) *Simulating a $T=(2^n-1)S$ step irreversible computation from x to $f(x)$ using S space by a computation from x to $y=\langle x,f(x)\rangle$, the irreversible simulation cost satisfies:*

(i) $B^{(n-k)S}(x,y)\leq B^{nS}(x,y)+(2^{k+2}-1)S$, *for* $n\geq k\geq 1$.

(ii) $B^{(n-1)S}(x,y)>B^{nS}(x,y)$, *for* $n\geq 1$.

For the most space parsimonious simulation with $n=\log(T/S)$ this means that

$$B^{S(\log(T/S)-k)}(x,y)\leq B^{S\log(T/S)}(x,y)+(2^{k+2}-1)S.$$

2.3 Local Irreversible Actions

Suppose we have an otherwise reversible computation containing local irreversible actions. In [14] it is shown that we can always simulate such a computation with an otherwise reversible computation with all irreversibly provided bits provided at the beginning of the computation, and all irreversibly erased bits erased at the end of the computation. This is when we are in the situation when there are no a priori bounds on the resources in time or space consumed by the computation.

However, in the case above where there are very tight bounds on the space used by the computation, we found in Lemma 1 a method where at the cost of limited erasing, precisely controlled with respect to its spacing in the computation time, we could save on the auxiliary space use. By Corollary 2 it is *impossible* in our pebble game to shift these erasures to the end of the computation, since if we do, then the same auxiliary space is still needed at precise times spaced during the simulation time.

Quantum computing is a particular form of reversible computation. Apart from classical irreversible erasures, quantum computing has a nonclassical form of irreversibility, namely the irreversible observations. An *irreversible observation* makes the superposition of the quantum state of the computer collapse from the original state space to a subspace thereof, where the probability amplitudes of constituent elements of the new superposition are renormalized. It is well-known and observed in some papers [17], that we can replace all observations during the quantum computation by a composition of observations at the end of the computation. One wonders if this non-classical type of irreversibility constituted by irreversible observation of quantum states also is constrained to strictly local instants during the computation by restrictions on time or space resources. This seems to be the case in the $\sqrt{n}$ data item queries unstructured database search algorithm of Grover [8]. There, we have to observe and renormalize at precise time instants during the computation to achieve the improvement of $O(\sqrt{n})$ data item queries in the quantum algorithm over the classically required $\Omega(n)$ queries.

2.4 Reversible Simulation of Unknown Computing Time

In the previous analysis we have tacitly assumed that the reversible simulator knows in advance the number of steps T taken by the irreversible computation to be simulated. Indeed, the exhibited programs $F(\cdot)$ and $A(\cdot)$ have parameters I_k and G involving T. In this context one can distinguish on-line computations and off-line computations to be simulated. On-line computations are computations which interact with the outside environment and in principle keep running forever. An example is the operating system of a computer. Off-line computations are computations which compute a definite function from an input (argument) to an output (value). For example, given as input a positive integer number, compute as output all its prime factors. For every input such an algorithm will have a definite running time. A similar problem is choosing optimal parameters m, n as in Section 1.3 without knowing T and space S.

There is a well-known simple device (used in detail in [3]) to remove this dependency for batch computations without increasing the simulation time (and space) too much. Suppose we want to simulate a computation with unknown computation time T. Then we simulate t steps of the computation with t running through the sequence of values $2, 2^2, 2^3, \ldots$ For every value t takes on we reversibly simulate the first t steps of the irreversible computation. If $T > t$ then the computation is not finished at the end of this simulation. Subsequently we reversibly undo the computation until the initial state is reached again, set $t := 2t$ and reversibly simulate again. This way we continue until $t \geq T$ at which bound the computation finishes. The total time spent in this simulation is

$$T' \leq 2 \sum_{i=1}^{\lceil \log T \rceil} 2^{i \log 3} \leq 2(4T)^{\log 3}.$$

This is the canonical case. With these figures, just like the original simulation, by suitable choice of parameter k we can obtain $T' = \Theta(T^{1+\epsilon}/S^{\epsilon})$. for every constant $\epsilon > 0$.

Acknowledgment

We thank Wim van Dam for pointing out an (harmless) error in the original proof (in [14]) of Lemma 1, and Tom Toffoli and the referees for useful comments.

References

[1] C.H. Bennett. Logical reversibility of computation. *IBM J. Res. Develop.*, 17:525–532, 1973.

[2] C.H. Bennett. The thermodynamics of computation—a review. *Int. J. Theoret. Phys.*, 21(1982), 905-940.

[3] C.H. Bennett. Time-space tradeoffs for reversible computation. *SIAM J. Comput.*, 18(1989), 766-776.

[4] D. Deutsch, Quantum theory, the Church-Turing principle and the universal quantum computer. *Proc. Royal Society London.* Vol. A400(1985), 97-117.

[5] E. Fredkin and T. Toffoli. Conservative logic. *Int. J. Theoret. Phys.*, 21(1982),219-253.

[6] M. Frank, T. Knight, and N. Margolus, Reversibility in optimally scalable computer architectures, Manuscript, MIT-LCS, 1997: http://www.ai.mit.edu/~mpf/publications.html.

[7] M.P. Frank and M.J. Ammer, Separations of reversible and irreversible space-time complexity classes, Submitted to the *13th IEEE Computational Complexity Conference*; http://www.ai.mit.edu/~mpf/rc/memos/M06_oracle.html.

[8] L.K. Grover, A fast quantum mechanical algorithm for database search, *Proc. 28th ACM Symp. Theory of Computing*, 1996, 212-219.

[9] R.W. Keyes, *IBM J. Res. Dev.*, 32(1988), 24-28.

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

[11] K.J. Lange, P. McKenzie, and A. Tapp, Reversible space equals deterministic space, *Proc. 12th IEEE Computational Complexity Conference*, IEEE Comp. Soc. Press, 1997.

[12] Y. Lecerf, Machines de Turing réversibles. Récursive insolubilité en $n \in N$ de l'équation $u = \theta^n$, où θ est un "isomorphisme de codes", *Comptes Rendus*, 257(1963), 2597-2600.

[13] R.Y. Levine and A.T. Sherman, A note on Bennett's time-space tradeoff for reversible computation, *SIAM J. Comput.*, 19:4(1990), 673-677.

[14] M. Li and P.M.B. Vitányi, Reversibility and adiabatic computation: trading time and space for energy, *Proc. Royal Society of London, Series A*, 452(1996), 769-789.

[15] M. Li and P.M.B. Vitányi, Reversible simulation of irreversible computation, *Proc. 11th IEEE Computational Complexity Conference*, IEEE Comp. Soc. Press, 1996, 301-306.

[16] R.C. Merkle, Reversible electronic logic using switches, *Nanotechnology*, 4(1993), 21-40.

[17] P.W. Shor, Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM Journal on Computing, 26:5(1997), 1484–1509.

[18] M. Sipser, Halting space-bounded computations, *Theoretical Computer Science*, 10(1980), 335-338.

[19] P.M.B. Vitányi, Physics and the New Computation, Proc. 20th Int. Symp. Math. Foundations of Computer Science, MFCS'95, *Lecture Notes in Computer Science, Vol 969*, Springer-Verlag, Heidelberg, 1995, 106-128.

[20] J. von Neumann. *Theory of Self-Reproducing Automata.* A.W. Burks, Ed., Univ. Illinois Press, Urbana, 1966.