A reversible and conservative model based on rational signal machines for Black hole computation

Jérôme Durand-Lose* **

LIFO, Université d'Orléans,
B.P. 6759, F-45067 ORLÉANS Cedex 2.

Abstract. In the context of Abstract geometrical computation, it has been proved that black hole model (and SAD computers) can be implemented. To be more physic-like, it would be interesting that the construction is reversible and preserves some energy. There is already a (energy) conservative and reversible two-counter automaton simulation.
In the present paper, based on reversible and conservative stacks, reversible Turing machines are simulated. Then a shrinking construction that preserves these properties is presented. All together, a black hole model implementation that is reversible and conservative (both the shrinking structure and the universal Turing machine) is provided.

Key-words. Abstract geometrical computation; Black hole model; Energy conservation; Reversibility; Signal machine

1 Introduction

Reversibility —forward and backward determinism— is a very important issue in computer science [Bennett, 1988]. It has been studied from the mathematical point of view [Lecerf, 1963] from the 1960's and on a more machine approach from the 1970's: Turing machines [Bennett, 1973, 1989, MORITA et al., 1989], logic circuits [Fredkin and Toffoli, 1982, Morita, 1990, Merkle, 1993], two counter automata [Morita, 1996], physics-like models of computations [Toffoli, 1980], cellular automata [Toffoli, 1977, Toffoli and Margolus, 1990, Durand-Lose, 1997, 2001]. Reversibility has also been important as a step toward quantum computation.

In this article, we are also interest with the conservation of some local energy since reversibility might not preserve the expected energy. Here, conservation refers to the preservation of some quantified energy as explained below.

In the last fifteen years, people got very exited with the so-called, Black hole model of computation [Etesi and Németi, 2002, Lloyd and Ng, 2004] an

* Jerome.Durand-Lose@univ-orleans.fr

** This work was partially supported by the ANR project AGAPE, ANR-09-BLAN-0159-03.

SAD computers [Hogarth, 1994, 2004] since while being physically feasible —or at least not straightforwardly impossible— [Earman and Norton, 1993, Németi and Andréka, 2006, Németi and Dávid, 2006, Andréka et al., 2009] it refutes Church-Turing's Thesis by using accelerating or infinite-time Turing machines [Copeland, 2002, Hamkins and Lewis, 2000] as decision oracles.

Roughly, it works as follows: an observer throws a computing device into some special kind of black hole. The device has an infinite amount of time ahead of it to compute and may send a single signal that is perceptible from the border of the black hole by the observer. The key point is that it gets infinitely accelerated, so that its whole life span is in the past of an observer after a bounded duration after he threw the device. So any signal sent by the device, would be received before that time. Knowing that the time is past, the observer know whether or not the device ever send the message. If the message is sent only before halting, then the halting problem is solved in such a way.

Reversibility and black hole computation might look like independent concepts nevertheless at some level, the fabric of reality is believed to be reversible. One easy way to tackled the issue of considering the two concept together is that as soon as black hole computation is physically feasible, then at some level, it is done in a reversible way. One may contest this by saying that, not only conception works the other way round, but moreover if the machine send into a black hole has to run for an unlimited amount of time, it must have an unlimited amount of energy available or it should never consume any amount of it! Hopefully, at least theoretically, there are reversible universal machine of many kind (see above references).

For a dynamic system, to implement the black hole model means to be able to compute, to accelerate infinitely the computation on one part of the system, to allow a single signal to leave and on another part of the system, to have something to receive the signal and act upon.

In previous articles [Durand-Lose, 2005, 2006b, 2008b, 2009], we have provided a setting, *Abstract geometrical computation* and even nested black holes for both discrete and analog computations. To do this, we have show how to implement computing machinery and a folding/shrinking structure. This structure accumulates at some point and anything entangled inside undergo an infinite acceleration. The black hole effect is then achieved by letting some signal leave the structure. (Leaving the structure and computation on the side are plain and not addressed anymore.)

In the present article, we provide new constructions that are reversible based on reversible machinery and reversible structure. The reversibility of the structure does not go beyond the accumulation points so that nested black holes are not possible with the present construction.

Abstract geometrical computations was initiated as a continuous time and space counterpart of cellular automata [Durand-Lose, 2008a] similar to the approaches of Jacopini and Sontacchi [1990], Takeuti [2005], Hagiya [2005] and also as an idealization of collision computing [Adamatzky, 2002].

Abstract geometrical computation deals with dimensionless moving *signals*. The signals move inside continuous time and space (here, the real line) and rules describe what happens when they collide. Time is continuous and Zeno-like constructions provide infinitely many accelerating steps during a finite duration.

There are finitely many kinds of signals, called *meta-signals*. The speed of any signal does only depends on the associated meta-signal, so that similar signals are parallel. When signals meet, they are replaced by new signals according to *collision rules* that only depends on the in-coming meta-signals. The space considered here is one-dimensional, so that *space-time diagrams* are two-dimensional. They are depicted with time growing upward.

The reversibility of a signal machine corresponds to backward deterministic collision rules. Moreover we impose a strict conservative condition: the number of signal entering a collision must be equal to the one leaving. Both can be checked by going through the collision rules.

In Durand-Lose [2006a], reversible two-counter automata are used to produce a reversible machinery. In the present article, we use 1-tape Turing machines (TM) [MORITA et al., 1989]. The first issue with implementing a TM is to deal with the tape. The tape is decomposed as the word before the head, the symbol under the head and the word after. Since these words are only accessed from a single side, they can be represented and managed by *stacks*. A conservative and reversible implementation of stacks is presented.

The second step is to deal with transitions. As in cited article, they are expressed as either move or rewrite: reversibility can be checked easily and the inverse TM can be produced. In this from, they translates automatically into reversible signal machines. The signal amounting for the state remains between the two stacks. From then, any computation can be implemented within a bounded space.

The next step is to embedded it into some shrinking structure. First a reversible and conservative shrinking structure is provided by a simple geometric construction. It is then explained how the any bounded computation can be entangled inside the structure. As the structure shrinks, the computation is scaled down, thus accelerated. The structure acts like a black hole.

The paper is articulated as follows. Section 2 gathers the definitions of abstract geometrical computation. Implementing reversible Turing machines by reversible conservative signal machines is done in two part: Sect. 3 for stacks implementation and Sect. 4 for transition and putting all together. The reversible black hole emulation is also done in two parts: Sect. 5 concentrates on the shrinking structure while Sect. 6 deals with embedding a bounded space-time diagram inside it.

2 Definitions

Signals. Each *signal* is an instance of a *meta-signal*. The associated meta-signal defines its *velocity* and what happen when signals meet.

Figure 1 provides an example of a space-time diagram: time is increasing upwards and the meta-signals are indicated on the signals. Figure 2 gives the definition of the corresponding machine. Generally, over-line arrows indicate the direction of propagation of a meta-signal. For example, $\overleftarrow{\mathsf{mem}}$ and $\overrightarrow{\mathsf{mem}}$ denotes two different meta-signals; but as can be expected, they have similar semantics.

Collision rules. When a set of signals collide, they are replaced by a new set of signals according to a matching collision rule. A rule has the form:

$$\{\sigma_1, \ldots, \sigma_n\} \to \{\sigma'_1, \ldots, \sigma'_p\}$$

where all σ_i and σ'_j are meta-signals. A rule matches a set of colliding signals if its left-hand side is equal to the set of their meta-signals. Collision rules can be deduced from space-time diagram as on Fig. 1. They are also listed on Fig. 2.

Signal machine. A signal machine is defined by a set of meta-signals and a set of collision rules. An initial configuration is a set of signals placed on the real line. The evolution of a signal machine can be represented geometrically as a *space-time diagram*: space is always represented horizontally, and time vertically, growing upwards.

3 Stack implementation

3.1 Stack principle

Since we are dealing with rational numbers, it is very easy to implement an unbounded stack of natural numbers 1 to l in the following way: $\frac{1}{l+2}$ encodes the empty stack. Let σ be a rational number encoding of a stack ($0<\sigma<1$). After pushing a value v on top of a stack σ, the new stack is $\frac{v+\sigma}{l+1}$. This ensures that, as soon as the stack is not empty, $\frac{1}{l+1}<\sigma<1$ and distinct from $\frac{i}{l+1}$, $i \in \{1, 2, \ldots, l\}$. The top of the stack is $\lfloor(l+1)\sigma\rfloor$ and rest of the stack is $(l+1)\sigma - \lfloor(l+1)\sigma\rfloor$.

To implement this in a signal machine, a scale is defined (because the space is continuous and scale-less) and then the push operation is implemented. The pop corresponds to the inverse of push (but meta-signals are different) as can be seen on Fig. 1. Only the push is detailed.

The rational σ is encoded with a zero-speed signal mem. The scale is defined with zero-speed signals mark_0, mark_1,..., mark_l. They are regularly positioned, never move and defined positions 0, 1, ..., l. Thus the normal position of mem is between mark_0 and mark_1. To push v, mem is translated by v (lower part of Fig. 1). Then this position is scaled by $\frac{1}{l+1}$ (upper part of Fig. 1) to get $\frac{\sigma+v}{l+1}$. The process starts at the arrival of $\overleftarrow{\mathsf{store}_v}$.

Translating is very easy: $\overrightarrow{\mathsf{mem}}$ and $\overrightarrow{\mathsf{store}_v}$ are parallel. Their distance encodes σ. Their movement stops when the first one, $\overrightarrow{\mathsf{store}_v}$, reaches mark_v. The signal $\overrightarrow{\mathsf{catch}}$ is then issued to stop $\overrightarrow{\mathsf{mem}}$ as in the middle of Fig. 1. This collision is distance v away from the original position of mem. This is ensured by the definition

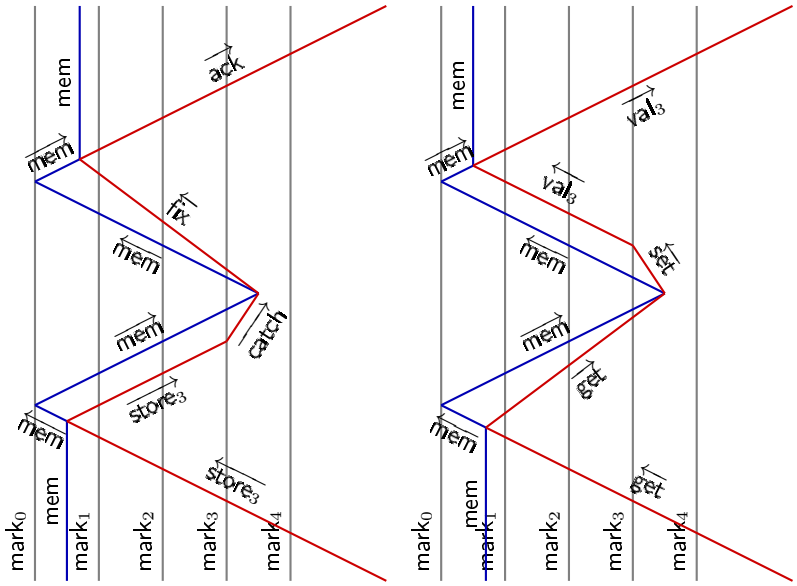


Fig. 1. Implementation the stack for $l = 4$, a push(3) and a get examples.

Meta-signal	Speed
mark_i	0
$\overleftarrow{\mathsf{mem}}$	-3
mem	0
$\overrightarrow{\mathsf{mem}}$	3
$\overleftarrow{\mathsf{store}_i}$	-3
$\overrightarrow{\mathsf{store}_i}$	3
$\overrightarrow{\mathsf{catch}}$	1
$\overleftarrow{\mathsf{fix}}$	-2
$\overrightarrow{\mathsf{ack}}$	3
$\overrightarrow{\mathsf{get}}$	2
$\overleftarrow{\mathsf{get}}$	-3
$\overleftarrow{\mathsf{set}}$	-1
$\overrightarrow{\mathsf{val}_i}$	3
$\overleftarrow{\mathsf{val}_i}$	-3

Collision rules

$$\{\mathsf{mark}_0, \overleftarrow{\mathsf{mem}}\} \to \{\mathsf{mark}_0, \overrightarrow{\mathsf{mem}}\}$$
$$\{\mathsf{mark}_i, \overleftarrow{\mathsf{mem}}\} \to \{\overleftarrow{\mathsf{mem}}, \mathsf{mark}_i\}$$
$$\{\overrightarrow{\mathsf{mem}}, \mathsf{mark}_i\} \to \{\mathsf{mark}_i, \overrightarrow{\mathsf{mem}}\}$$
$$\{\mathsf{mark}_i, \overleftarrow{\mathsf{store}_v}\} \to \{\overleftarrow{\mathsf{store}_v}, \mathsf{mark}_i\}$$
$$\{\mathsf{mem}, \overleftarrow{\mathsf{store}_v}\} \to \{\overleftarrow{\mathsf{mem}}, \overrightarrow{\mathsf{store}_v}\}$$
$$i < v \quad \{\overrightarrow{\mathsf{store}_v}, \mathsf{mark}_i\} \to \{\mathsf{mark}_i, \overrightarrow{\mathsf{store}_v}\}$$
$$\{\overrightarrow{\mathsf{store}_v}, \mathsf{mark}_v\} \to \{\mathsf{mark}_v, \overrightarrow{\mathsf{catch}}\}$$
$$\{\overrightarrow{\mathsf{mem}}, \overrightarrow{\mathsf{catch}}\} \to \{\overleftarrow{\mathsf{mem}}, \overleftarrow{\mathsf{fix}}\}$$
$$\{\mathsf{mark}_i, \overleftarrow{\mathsf{fix}}\} \to \{\overleftarrow{\mathsf{fix}}, \mathsf{mark}_i\}$$
$$\{\overrightarrow{\mathsf{mem}}, \overleftarrow{\mathsf{fix}}\} \to \{\mathsf{mem}, \overrightarrow{\mathsf{ack}}\}$$
$$\{\overrightarrow{\mathsf{ack}}, \mathsf{mark}_i\} \to \{\mathsf{mark}_i, \overrightarrow{\mathsf{ack}}\}$$
$$\{\mathsf{mark}_i, \overleftarrow{\mathsf{get}}\} \to \{\overleftarrow{\mathsf{get}}, \mathsf{mark}_i\}$$
$$\{\mathsf{mem}, \overleftarrow{\mathsf{get}}\} \to \{\overleftarrow{\mathsf{mem}}, \overrightarrow{\mathsf{get}}\}$$
$$\{\overrightarrow{\mathsf{get}}, \mathsf{mark}_i\} \to \{\mathsf{mark}_i, \overrightarrow{\mathsf{get}}\}$$
$$\{\overrightarrow{\mathsf{mem}}, \overleftarrow{\mathsf{get}}\} \to \{\overleftarrow{\mathsf{mem}}, \overleftarrow{\mathsf{set}}\}$$
$$\{\mathsf{mark}_v, \overleftarrow{\mathsf{set}}\} \to \{\overleftarrow{\mathsf{val}_v}, \mathsf{mark}_v\}$$
$$i < v \quad \{\mathsf{mark}_i, \overleftarrow{\mathsf{val}_v}\} \to \{\overleftarrow{\mathsf{val}_v}, \mathsf{mark}_i\}$$
$$\{\overrightarrow{\mathsf{mem}}, \overleftarrow{\mathsf{val}_i}\} \to \{\mathsf{mem}, \overrightarrow{\mathsf{val}_i}\}$$
$$\{\overrightarrow{\mathsf{val}_v}, \mathsf{mark}_i\} \to \{\mathsf{mark}_i, \overrightarrow{\mathsf{val}_v}\}$$
with $0 < v$ and $0 < i$.

Fig. 2. Implementation the stack for $l = 4$, the signal machine.

of speeds (we leave to the reader to verify this linear equation system based on the speeds given in Fig. 2). When this point is reached, a scaling remains to be done. Scaling by $\frac{1}{l+1}$, to go from $\sigma+v$ to $\frac{\sigma+v}{l+1}$, $\overleftarrow{\mathsf{fix}}$ has to travel $(\sigma+v)-\frac{\sigma+v}{l+1}=(\sigma+v)\frac{l}{l+1}$ units, and $\overleftarrow{\mathsf{mem}}$ and $\overrightarrow{\mathsf{mem}}$ have to travel $(\sigma+v)+\frac{\sigma+v}{l+1}=(\sigma+v)\frac{l+2}{l+1}$ units. Thus if the speeds of $\overleftarrow{\mathsf{mem}}$ and $\overrightarrow{\mathsf{mem}}$ have the same absolute value then the speed of $\overleftarrow{\mathsf{fix}}$ must be $\frac{l}{l+2}$ times the one of $\overleftarrow{\mathsf{mem}}$ (notice in Fig. 2 that $-2=-3\frac{4}{4+2}$).

If the stack is empty, then backward collision between $\overleftarrow{\mathsf{mem}}$ and $\overleftarrow{\mathsf{fix}}$ happens between mark_0 and mark_1. So that there is a (backward) collision between $\overrightarrow{\mathsf{mem}}$ (regenerated from mark_0 and $\overleftarrow{\mathsf{mem}}$) and $\overrightarrow{\mathsf{catch}}$. There no such a rule, it can be defined to have, for example, mem fixed and $\overrightarrow{\mathsf{catch}}$ exiting on the left.

It is easy to check that the machine is invertible and conservative.

This stack is supposed to be accessed from the right. A mirror stack would be accessed from the left.

4 Transition

In MORITA et al. [1989], the formalism used for 1-tape Turing machines is either a write transition or a move transition. We keep this formalism with some adaptations. Let M be a reversible TM. Since it is deterministic, the set of states Q can be partitioned into: Q_M and Q_W, moving and writing states. Moving states are states that move the R/W head whatever the read symbol. Writing states rewrite the symbol under the head according to the read symbol (and the state). The transition table has two corresponding parts: δ_M and δ_W. Moreover, we suppose that there is no two consecutive writing states (this can be simplified directly in the transition table), so that a writing is followed by a move:

$$\delta_W :\subseteq Q_W \times \Gamma \longrightarrow Q_M \times \Gamma \ .$$

Since M is backward deterministic, δ_W is one-to-one.

By adding extra writing states that only rewrite the symbol read, it can also be supposed, without any loss of generality that a move transition is always followed by a write one, so that:

$$\delta_M :\subseteq Q_M \longrightarrow Q_W \times \{\leftarrow, \rightarrow\} \ .$$

Since M is backward deterministic, states in Q_W appears at most once on the right side of δ_M. The function Q_W is undefined only for halting states and only the initial state may not appear on the right side (otherwise, a state from Q_W not appearing can never be reached and is removed). Directions can be associated to states in Q_W according to δ_M.

The reverse TM, M^{-1} can be generated directly from δ_W and δ_M.

Figure 3 shows how the state is encoded in-between two stacks (these are distinct, there is no common meta-signal). To distinguish the signals from each stack, exponent L and R are used. The stacks records the words before and after the head. The symbol under the head is the one about to collide with the symbol

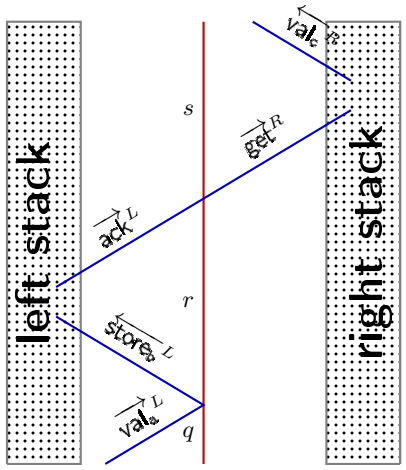


Fig. 3. Implementing the sequence of transitions of Tab. 1.

encoding the state. The side from which it comes depends only on the direction associated to the state by δ_M.

Table 1. Sequence of transitions.

Transition	Inverse transition
$\delta_M(...) = (q, \leftarrow)$	$\delta_M^{-1}(q) = (..., \rightarrow)$
$\delta_W(q, \mathtt{a}) = (r, \mathtt{b})$	$\delta_W^{-1}(r, \mathtt{b}) = (q, \mathtt{a})$
$\delta_M(r) = (s, \rightarrow)$	$\delta_M^{-1}(s) = (r, \leftarrow)$
$\delta_W(s, \mathtt{c}) = ...$	$\delta_W(...) = (s, \mathtt{c})$

The example of Fig. 3 correspond to the sequence of transitions on Tab. 1. The machine is on state q after a left move, so that the read symbol comes from the left stack. State r means a right move so that the letter written on the transition on q has to be sent to the left. On receiving the acknowledge from the left, r turns to s and send the get signal to the right stack. Reversible stacks are conceived in a symmetric way, the acknowledge and the get are symmetric. This is the same symmetry as in the expression of transition on M and M^{-1}.

Formally, there is a stack value (and associated meta-signals) for each symbol and a null-speed meta-signal for each state. The collision rules generated for writing transitions are given on Tab. 2.

From the properties of δ_W and δ_M, it is clear that the generated collision rules are one-to-one. The generated signal machine is reversible. Moreover, the construction only involves collisions of two signals that generates two signals. The machine is conservative.

Table 2. Collision rules associated to transitions.

(a) write transition $\delta_W(q, \mathtt{a}) = (r, \mathtt{b})$

	$\delta_M(r) = (., \leftarrow)$	$\delta_M(r) = (., \rightarrow)$
$\delta_M^{-1}(q) = (., \leftarrow)$	$\{q, \mathtt{a}^L\} \to \{r, \mathtt{b}^R\}$	$\{q, \mathtt{a}^L\} \to \{r, \mathtt{b}^L\}$
$\delta_M^{-1}(q) = (., \rightarrow)$	$\{q, \mathtt{a}^R\} \to \{r, \mathtt{b}^R\}$	$\{q, \mathtt{a}^R\} \to \{r, \mathtt{b}^L\}$

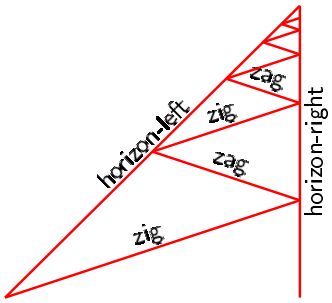
(b) move transitions

$\delta_M(r) = (s, \rightarrow)$
$\{r, \overrightarrow{\mathsf{ack}}^L\} \to \{s, \overrightarrow{\mathsf{get}}^R\}$
$\delta_M(r) = (s, \leftarrow)$
$\{r, \overleftarrow{\mathsf{ack}}^R\} \to \{s, \overleftarrow{\mathsf{get}}^L\}$

Special care has to be taken for states that are not moving for M^{-1}: halting states and possibly the initial state. For the initial state, just set the moving direction as desired. For halting states, some collisions are left undefined. They should be defined with new meta-signals that will be left unmodified by the shrinking structure and exit on the left.

5 Structure

In this section we briefly present a shrinking structure as depicted on Fig. 4. It is fractal and accumulates to a single point.



Meta-signal	Speed
horizon-left	1
horizon-right	0
zig	3
zag	−3

Rules

{ zig, horizon-right } → { zag, horizon-right }
{ horizon-left, zag } → { horizon-left, zig }

Fig. 4. Shrinking structure.

As can be seen, 4 meta-signals and 2 collision rules are enough to generate an accumulation. Accumulation is an important and common phenomenon in AGC.

6 Embedding

Embedding into the structure is more involving. The scheme to do this is presented. There is no room for details.

The gray zone on Fig. 5 is any computation that is spatially bounded. On the right, it is displayed embedded inside the structure. The whole infinite gray

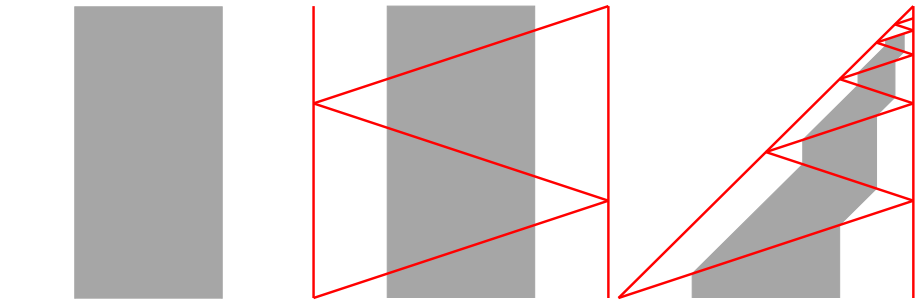


Fig. 5. Shrinking a bounded computation.

computation should be inserted inside a bounded duration. This is done through unending down-scaling presented below.

The gray zone alternate between triangular zones. In the triangular zone pointing right, the gray zone is bend. In the triangular zone pointing left, the grey zone is unbend.

Moreover, by construction, each triangle with the same orientation is half the size of the previous one with the same orientation. By ensuring that each time, the space-time diagram is also scaled down by two, then all the triangle, rescaled to scale 1 have the same size and composed the gray computation as on Fig. 5(b).

6.1 Shrinking step

Various geometric modifications are easy to produce by just modifying the speed or the initial configuration: for example, if the initial configuration is scaled by one half, so is the space-time diagram. Another example is to take a signal machine and multiply by two all speed. The same initial configuration generates a space-time diagram which is scaled by one half on the temporal axis only.

Transformations made on speed reflect on the diagram since it is actually *drawn* by the signal. A more complex construction is to consider two non parallel axis and use one as a frontier and the other for scaling as depicted on Fig. 6.

On Fig. 6, the thick lines are used as frontiers between the various zones. The directions of two frontiers are the ones used for scaling. The incoming signal at the bottom get decomposed, considered as a vector, as a sum of two vectors according to the two axis. If there were no modification, it would have follow the dotted path. But since it scaled by one half on one direction, it follows the dashed path. Passing through the second frontier, the modification is on the component corresponding to the other direction. So that, altogether, it is scaled by one half in both directions, thus it regains its original speed but the space-time diagram is scaled by half!

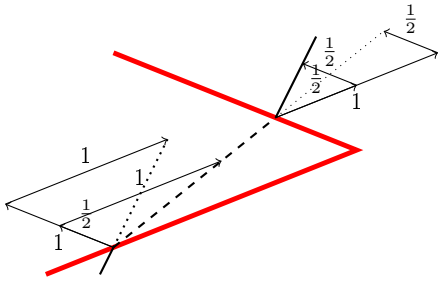


Fig. 6. Scaling by one half in two steps.

The fact that the coordinate on the direction of the axis is not modified is very important since it ensures that the space-time diagram connect exactly on the frontier.

This is implemented by adding two meta-signals for the frontiers and one extra meta-signal, μ', one for each original one, μ. The speed of μ' is computed by a simple formula. Collision rules have to be added in order to replace μ by μ' and *vice-versa*.

It might append that a *grey* collision happen exactly on the frontier. Then new collisions are rules are added with each time a frontier in and out. Signals that are above the frontier are replaced by their bent versions (and *vice-versa* for the other frontier).

The zone above and the zone below should not meet. In our construction, this is ensured because the TM simulation is bounded.

Finally, a computation and its shrinking version are displayed on Fig. 7.

Bibliography

Andrew Adamatzky, editor. *Collision based computing*. Springer, 2002.

Hajnal Andréka, István Németi, and Péter Németi. General relativistic hypercomputing and foundation of mathematics. *Nat. Comput.*, 8(3):499–516, 2009.

Charles H. Bennett. Logical reversibility of computation. *IBM J. Res. Dev.*, 6:525–532, 1973.

Charles H. Bennett. Notes on the history of reversible computation. *IBM J. Res. Dev.*, 32(1):16–23, 1988.

Charles H. Bennett. Time/space trade-offs for reversible computation. *SIAM Journal on Computing*, 18(4):766–776, August 1989. ISSN 0097-5397 (print), 1095-7111 (electronic).

Jack B. Copeland. Accelerating Turing machines. *Minds & Machines*, 12(2):281–300, 2002.

Jérôme Durand-Lose. Intrinsic universality of a 1-dimensional reversible cellular automaton. In *STACS '97*, number 1200 in LNCS, pages 439–450. Springer, 1997. doi: 10.1007/BFb0023479.

Jérôme Durand-Lose. Representing reversible cellular automata with reversible block cellular automata. In Robert Cori, Jacques Mazoyer, Michel Mor-

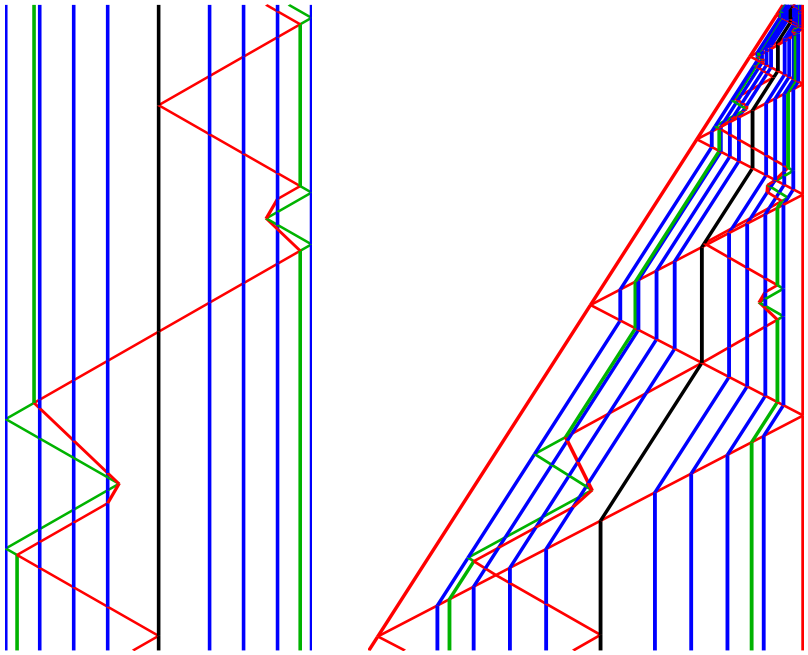


Fig. 7. All together.

van, and Rémy Mosseri, editors, *Discrete Models: Combinatorics, Computation, and Geometry, DM-CCG '01*, volume AA of *Discrete Mathematics and Theoretical Computer Science Proceedings*, pages 145–154, 2001. URL http://dmtcs.loria.fr/volumes/abstracts/dmAA0110.abs.html.

Jérôme Durand-Lose. Abstract geometrical computation for black hole computation (extended abstract). In M. Margenstern, editor, *Machines, Computations, and Universality (MCU '04)*, number 3354 in LNCS, pages 176–187. Springer, 2005. doi: 978-3-540-25261-0. URL http://lita.sciences.univ-metz.fr/ mcu2004/.

Jérôme Durand-Lose. Reversible conservative rational abstract geometrical computation is Turing-universal. In Arnold Beckmann and John V. Tucker, editors, *Logical Approaches to Computational Barriers, 2nd Conf. Computability in Europe (CiE '06)*, number 3988 in LNCS, pages 163–172. Springer, 2006a. doi: 10.1007/11780342_18.

Jérôme Durand-Lose. Abstract geometrical computation 1: Embedding black hole computations with rational numbers. *Fund. Inf.*, 74(4):491–510, 2006b.

Jérôme Durand-Lose. The signal point of view: from cellular automata to signal machines. In Bruno Durand, editor, *Journées Automates cellulaires (JAC '08)*, pages 238–249, 2008a. URL http://www.lif.univ-mrs.fr/jac/.

Jérôme Durand-Lose. Black hole computation: implementation with signal machines. In C. S. Calude and J. F. Costa, editors, *International Workshop Physics and Computation, Wien, Austria, Agust 25-28*, Research Report CDMTCS-327, pages 136–158, 2008b.

Jérôme Durand-Lose. Abstract geometrical computation 3: Black holes for classical and analog computing. *Nat. Comput.*, 8(3):455–472, 2009. doi: 10.1007/s11047-009-

9117-0.

John Earman and John D. Norton. Forever is a day: supertasks in Pitowsky and Malament-Hogarth spacetimes. *Philos. Sci.*, 60(1):22–42, 1993.

Gábor Etesi and Istvan Németi. Non-Turing computations via Malament-Hogarth space-times. *Int. J. Theor. Phys.*, 41(2):341–370, 2002. gr-qc/0104023.

Edward F. Fredkin and Tommaso Toffoli. Conservative logic. *Int. J. Theor. Phys.*, 21, 3/4:219–253, 1982.

Masami Hagiya. Discrete state transition systems on continuous space-time: A theoretical model for amorphous computing. In Cristian Calude, Michael J. Dinneen, Gheorghe Paun, Mario J. Pérez-Jiménez, and Grzegorz Rozenberg, editors, *Unconventional Computation, 4th International Conference, UC '05, Sevilla, Spain, October 3-7, 2005, Proceedings*, volume 3699 of *LNCS*, pages 117–129. Springer, 2005.

Joel David Hamkins and Andy Lewis. Infinite time Turing machines. *J. Symb. Log.*, 65(2):567–604, 2000. arXiv:math.LO/9808093.

Mark L. Hogarth. Non-Turing computers and non-Turing computability. In *Biennial Meeting of the Philosophy of Science Association*, pages 126–138, 1994.

Mark L. Hogarth. Deciding arithmetic using SAD computers. *Brit. J. Philos. Sci.*, 55: 681–691, 2004.

Giuseppe Jacopini and Giovanna Sontacchi. Reversible parallel computation: an evolving space-model. *Theoret. Comp. Sci.*, 73(1):1–46, 1990.

Yves Lecerf. Machines de Turing réversibles. Récursive insolubilité en $n \in \mathbb{N}$ de l'équation $u = \theta^n u$, où θ est un isomorphisme de codes. *Comptes rendus des séances de l'académie des sciences*, 257:2597–2600, 1963.

Seth Lloyd and Jack Ng, Y. Black hole computers. *Scientific American*, 291(5):31–39, November 2004.

Ralph C. Merkle. Reversible electronic logic using switches. *Nanotechnology*, 4(1): 20–41, 1993.

Kenichi Morita. A simple construction method of a reversible finite automaton out of Fredkin gates, and its related problem. *Transactions of the IEICE*, E 73(6):978–984, June 1990.

Kenichi Morita. Universality of a reversible two-counter machine. *Theoret. Comp. Sci.*, 168(2):303–320, 1996.

Kenichi MORITA, Akihiko Shirasaki, and Yoshifumi Gono. A 1-tape 2-symbol reversible Turing machine. *Transactions of the IEICE*, E 72(3):223–228, March 1989.

István Németi and Hajnal Andréka. Can general relativistic computers break the Turing barrier? In Arnold Beckmann, Ulrich Berger, Benedikt Löwe, and John V. Tucker, editors, *Logical Approaches to Computational Barriers, 2nd Conf. on Computability in Europe, CiE '06*, volume 3988 of *LNCS*, pages 398–412. Springer, 2006. doi: 10.1007/11780342_42.

István Németi and Gyula Dávid. Relativistic computers and the Turing barrier. *Appl. Math. Comput.*, 178(1):118–142, 2006. doi: 10.1016/j.amc.2005.09.075.

Izumi Takeuti. Transition systems over continuous time-space. *Electr. Notes Theor. Comput. Sci.*, 120:173–186, 2005.

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

Tommaso Toffoli. Reversible computing. In *ICALP*, volume 85 of *LNCS*, pages 632–644. Springer, 1980.

Tommaso Toffoli and Norman Margolus. Invertible cellular automata: a review. *Phys. D*, 45:229–253, 1990.