Abstract Geometrical Computation: Turing-Computing Ability and Undecidability

Jérôme Durand-Lose

Abstract geometrical computation: Turing-computing ability and undecidability

Jérôme Durand-Lose*

Laboratoire d'Informatique Fondamentale d'Orléans, Université d'Orléans,
B.P. 6759, F-45067 ORLÉANS Cedex 2.

Abstract. In the Cellular Automata (CA) literature, discrete lines inside (discrete) space-time diagrams are often idealized as Euclidean lines in order to analyze a dynamics or to design CA for special purposes. In this article, we present a parallel analog model of computation corresponding to this idealization: dimensionless signals are moving on a continuous space in continuous time generating Euclidean lines on (continuous) space-time diagrams. Like CA, this model is parallel, synchronous, uniform in space and time, and uses local updating. The main difference is that space and time are continuous and not discrete (*i.e.* $\mathbb{R}$ instead of $\mathbb{Z}$). In this article, the model is restricted to $\mathbb{Q}$ in order to remain inside Turing-computation theory. We prove that our model can carry out any Turing-computation through two-counter automata simulation and provide some undecidability results.

Key-words: Abstract geometrical computation, Analog model of computation, Cellular automata, Geometry, Turing universality.

1 Introduction

Cellular automata (CA) form a well known and studied model of computation and simulation. Configurations are $\mathbb{Z}$-arrays of cells, the states of which belong to a finite set. Each cell can only access the states of its neighboring cells. All cells are updated iteratively and simultaneously. The main characteristics of CA are: parallelism, synchrony, uniformity and locality of updating. The trace of a computation, or the orbit starting from an initial configuration, is represented as a *space-time diagram*: a coloring of $\mathbb{Z} \times \mathbb{N}$ with states.

Discrete lines are often observed on these diagrams. They can be the key to understanding the dynamics and correspond to so-called *particles* or *signals* as in, *e.g.*, [Ila01, pp. 87–94] or [BNR91,DL98,HSC01,JSS02]. They can also be the tool to design CA for precise purposes and then named *signals* and used for, *e.g.*, prime number generation [Fis65], firing squad synchronization [Got66,VMP70,LN90,Maz96] or reversible simulation [DL97,DL00]. These discrete lines systems have also been studied on their own [DM02,MT99]. All the

* This research was done while the author was at the LIP, ÉNS Lyon, FRANCE. Jerome.Durand-Lose@lifo.univ-orleans.fr

figures in cited papers exhibit discrete lines which are explicitly refereed to –and are often idealized as Euclidean lines– for describing or designing. Many more articles could have been cited, the cited ones were randomly chosen in order to show the variety.

We want to consider this idealization: Euclidean lines on their own –not as a passing point for analysis or conception– while remaining with the main characteristics of CA. As Euclidean lines belong to Euclidean spaces and not $\mathbb{Z} \times \mathbb{N}$, the support of space and time is now $\mathbb{R} \times \mathbb{R}^+$. Configurations (at a given time or the restriction of the space-time diagram to a given time) are not mappings from $\mathbb{Z}$ to a finite set of states but partial mappings from $\mathbb{R}$ to the union of a finite set of *meta-signals* and a finite set of *collision rules*, defined for finitely many positions. The time scale is $\mathbb{R}^+$ (not $\mathbb{N}$), so that there is no such thing as a "next configuration". The following configurations are defined by the uniform movement of each signal, the speed of which is defined by its associated meta-signal. When two or more signals meet, a *collision* happens. Each collision follows a collision rule, each of which is defined by a pair of sets of meta-signals: *(incoming meta-signals, outgoing meta-signals)*. There must be at least two incoming meta-signals and all sets of incoming meta-signals must differ (which means determinism). In the configurations following a collision, incoming signals are removed and outgoing signals corresponding to the outgoing meta-signals are added.

Due to the continuous nature of space and time and the uniformity of movements, the traces of signals form Euclidean lines on the space-time diagrams, which we freely call *signals*. Each signal corresponds to a meta-signal which indicates its slope. Since there are finitely many meta-signals, there are finitely many slopes. This limitation may seem restrictive and unrealistic, even awkward as a finite quantification inside an analog model of computation. Let us notice that, first, it comes from CA: once a discrete line is identified, wherever (and whenever) the same pattern appears, the same line is expected, thus with the same slope. Second, we give two pragmatic arguments: (1) laws to define the new slopes from the previous ones in collisions are not so easy to design and pretty cumbersome to manipulate; (2) there is already much computing power (as presented in this paper and addressed in the conclusion).

Before presenting the results in this paper, we want to convince the reader that it is not just "one more analog model of computation". First, it does not come "out of the blue" because of its CA origin (where it is often implicitly used). Second, let's do a brief tour of analog/super-Turing models of computation[1]. To our knowledge, the closest model is the Mondrian automata of Jacopini and Sontacchi [JS90]. They also define dynamics and work on space-time diagrams which are mappings from $\mathbb{R}^n \times \mathbb{R}$ to a finite set of colors. Their diagrams should be composed of bounded finite polyhedra; we are only addressing lines –(hyper-)faces are not considered– and our diagrams may be unbounded and accumulation points may appear (they just forbid them; we address them in [DL05]). Another

[1] A wider survey can be found in [DL03, Chap. 2], unfortunately, there is no room here for more on the subject.

close model is the piecewise-constant derivative system [AM95,Bou97,Bou99]. Continuous space is partitioned into finitely many polygonal regions. The trajectory from a point is defined by a constant derivative on each region, thus an orbit is a sequence of (Euclidean) line segments. This model is very different from ours: it is sequential –there is only one "signal"– and the hyper-faces that delimit the regions are artifacts that do not exist in our model.

All the other models are based on totally different approaches. Some only define mapping over $\mathbb{R}$ like recursive analysis (type 2 Turing machines) [Wei00] or analog recursive functions [Moo96]. Many use a discrete iterative time like the BSS model [BCSS98] or analog recurrent neural networks [SS95]. The models with continuous time mostly use differential equations, finite support (variables) with uncountably many possible values [Orp94,ŠO01,PE74,Bra95]. To our knowledge, our model is the only one that is a dynamical system with continuous time and space but finitely many local values; this is also one of the few parallel ones.

In this paper, space and time are restricted to rationals (this is possible since all the operations used preserve rationality); this allows manipulation by, *e.g.*, Turing machines, thus remaining in the classical discrete computation theory, and is enough for Turing-computing capability. All intervals should be understood over $\mathbb{Q}$, not $\mathbb{R}$. Extension to $\mathbb{R}$ is automatic but only the rational case is addressed here[2].

After formally defining our model in Sect. 2, we show that any Turing-computation can be carried out through the simulation of two-counter automata in Sect. 3. The counters are encoded in unary and the instructions are going forth and back between the two groups of signals. The nature of space is used here: any finite number of signals can be enclosed in a bounded interval of $\mathbb{Q}$.

In Sect. 4, with simple reductions from the Halting problem, we prove that the following problems are undecidable: finite number of collisions, collision with a given signal, appearance of a given signal, and disappearance of all signals.

Conclusion, remarks and perspectives are gathered in Sect. 5.

2 Definitions

Our abstract geometrical computations are defined by the following machines:

Definition 1 A *signal machine* is defined by (M, S, R) where M is a finite set, S is a mapping from M to $\mathbb{Q}$, and R is a subset of $\mathcal{P}(M)\times\mathcal{P}(M)$ that corresponds to a partial mapping of the subsets of M of cardinality at least 2 to the subsets of M (both domain and range are restricted to elements of different S-images).

[2] Our personal opinion is that $\mathbb{Q}$ is indeed continuous –if not analog– even if it is countable because (1) there are no next nor previous element, (2) any interval is either empty, a singleton or infinite, and (3) its usual topology is not the discrete one.

The elements of M are called *meta-signals*. Each instance of a meta-signal is a *signal* which corresponds to a line segment in the space-time diagram. The mapping S assigns rational *speeds* to meta-signals, *i.e.* the slopes of the segments. The set R defines the *collision rules*, *i.e.* what happens when two or more signals meet. It also defines the intersections / extremities of the segments. The signal machines are deterministic because R must correspond to a mapping; if it were just a relation, then the machine would be non-deterministic.

Definition 2 A *configuration*, c, is a partial mapping from $\mathbb{Q}$ to the union of M and R such that the set $\{\, q \in \mathbb{Q} \mid c(q) \text{ is defined} \,\}$ is finite.

Let us define the dynamics:

A signal corresponding to a meta-signal μ at a position q, *i.e.* $c(q) = \mu$, is moving uniformly with constant speed $S(\mu)$. A signal must start in the initial configuration or be generated by a collision. It must end in a collision or in the last configuration. This corresponds to condition 1. in Def. 3.

A collision corresponds to a collision rule $\rho = (\rho^-, \rho^+)$, also noted as $\rho^- \to \rho^+$. All, and only, signals corresponding to the meta-signals in ρ^- (resp. ρ^+) must end (resp. start) in this collision. No other signal should be present. This corresponds to condition 2. in Def. 3.

Definition 3 The *space-time diagram*, or orbit, issued from an initial configuration c_0 and lasting for T[3], is a mapping c from $[0, T]$ to configurations (*i.e.* a partial mapping from $\mathbb{Q} \times [0, T]$ to $M \cup R$) such that, $\forall (q, t) \in \mathbb{Q} \times [0, T]$:

1. if $c_t(q) = \mu$ then $\exists t_i, t_f \in [0, T]$ with $t_i < t < t_f$ or $0 = t_i = t < t_f$ or $t_i < t = t_f = T$ s.t.:
   - $\forall t' \in (t_i, t_f)$, $c_{t'}(q + S(\mu)(t' - t)) = \mu$,
   - $t_i = 0$ or $c_{t_i}(q_i) \in R$ and $\mu \in (c_{t_i}(q_i))^+$ where $q_i = q + S(\mu)(t_i - t)$,
   - $t_f = T$ or $c_{t_f}(q_f) \in R$ and $\mu \in (c_{t_f}(q_f))^-$ where $q_f = q + S(\mu)(t_f - t)$;
2. if $c_t(q) = \rho^- \to \rho^+ \in R$ then $\exists \varepsilon$, $0 < \varepsilon$, $\forall t' \in [0, T]$,
   - if $t - \varepsilon < t' < t$ then $\forall \mu \in \rho^-$, $c_t(q + S(\mu)(t' - t)) = \mu$,
   - if $t < t' < t + \varepsilon$ then $\forall \mu \in \rho^+$, $c_t(q + S(\mu)(t' - t)) = \mu$,
   - $\exists \alpha$, $0 < \alpha$, $c_{[t-\varepsilon, t)}([q - \alpha, q + \alpha]) = \rho^-$ and $c_{(t, t+\varepsilon]}([q - \alpha, q + \alpha]) = \rho^+$.

The traces of signals represent line segments whose directions are defined by $(S(.), 1)$ (1 is the temporal coordinate). Collisions correspond to the extremities of these segments. Examples of space-time diagrams are provided by the various figures. Time is always increasing upwards.

3 Turing-computation capability

We prove the Turing-computation power of our model by simulating any two-counter automaton (a finite automaton couple with two counters, A and B). Each automaton can only *add/subtract* 1 to a counter and *branch if* a counter

[3] This definition can easily be extended to the case where $T = \infty$.

is *non-zero*. It can be described with a six-operations (the three aforementioned ones for each of the two counters) assembly language with branching labels as on the left part of Fig. 6 (see [Min67] for more on two-counter automata).

Definition 4 A *two-counter automaton* has two non-negative integer counters (`A` and `B`). Its action is defined by a sequence of labeled instructions; the only instructions available are:

- `A++` : add 1 to the value of `A`,
- `B++` : add 1 to the value of `B`,
- `A--` : if `A` is not at 0 then subtract 1 from its value,
- `B--` : if `B` is not at 0 then subtract 1 from its value,
- `A !=0` m : if `A` is not at 0 then the next instruction is the one labeled m,
- `B !=0` m : if `B` is not at 0 then the next instruction is the one labeled m.

The simulation is carried out with both counters unary encoded. Configurations are formed by, left to right: a left end-marking signal, a signals amounting for the value of A, one signal encoding the current instruction, b signals for B and a right end-marking signal. This is depicted on Fig. 1.

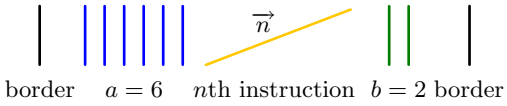

Fig. 1. Configuration encoding.

The only active signal (no collision ever happens without this signal being present) is the middle one; this signal both carries out the operation and encodes the line number. It goes forth and back between the signals encoding A and B. The end-markers are needed when the value is zero; in such a case, there would be no other signal on the side and the active signal would drift away infinitely. These end-markers also provide a simple way to test for non-zero: an end marker is met if and only if the value of the corresponding counter is zero.

They are two kinds of meta-signals: five for the counters and end-markers and the ones generated for the code as depicted on Fig. 2. The meta-signals of the first kind are: border, a and b of speed 0 used to mark the borders and to encode respectively A and B in unary, and aMv and bMv of speed $1/2$ and $-1/2$ used to increment A and B. The use of bMv is explained in the presentation of `B++` (Fig. 5). For the second kind, each line n of the program is converted to $\overrightarrow{n}$ and $\overleftarrow{n}$ of speed 1 and -1 (except for the test B $\neq 0$ which generates $\overrightarrow{n}$, $\overleftarrow{n}_Y$, and $\overleftarrow{n}_N$). The program is encoded in the collision rules.

The full transformation of a program into a signal machine is given in Fig. 3. We only detail the action of the collision rules generated for `B !=0` and `B++` instructions (at line n). All other instructions are carried out in a similar way.

Fig. 2. The two kinds of meta-signals.

Instruction	Rightwards	Leftwards
n `A++`	$\{\overrightarrow{n}, \mathsf{border}\} \to \{\overleftarrow{n}, \mathsf{border}\}$ $\{\overrightarrow{n}, \mathsf{b}\} \to \{\overleftarrow{n}, \mathsf{b}\}$ $\{\overrightarrow{n}, \mathsf{bMv}\} \to \{\overleftarrow{n}, \mathsf{b}\}$	$\{\mathsf{border}, \overleftarrow{n}\} \to \{\mathsf{border}, \mathsf{aMv}, \overrightarrow{n+1}\}$ $\{\mathsf{a}, \overleftarrow{n}\} \to \{\mathsf{a}, \mathsf{aMv}, \overrightarrow{n+1}\}$ $\{\mathsf{aMv}, \overleftarrow{n}\} \to \{\mathsf{a}, \mathsf{aMv}, \overrightarrow{n+1}\}$
n `B++`	$\{\overrightarrow{n}, \mathsf{border}\} \to \{\overleftarrow{n}, \mathsf{bMv}, \mathsf{border}\}$ $\{\overrightarrow{n}, \mathsf{b}\} \to \{\overleftarrow{n}, \mathsf{bMv}, \mathsf{b}\}$ $\{\overrightarrow{n}, \mathsf{bMv}\} \to \{\overleftarrow{n}, \mathsf{bMv}, \mathsf{b}\}$	$\{\mathsf{border}, \overleftarrow{n}\} \to \{\mathsf{border}, \overrightarrow{n+1}\}$ $\{\mathsf{a}, \overleftarrow{n}\} \to \{\mathsf{a}, \overrightarrow{n+1}\}$ $\{\mathsf{aMv}, \overleftarrow{n}\} \to \{\mathsf{a}, \overrightarrow{n+1}\}$
n `A--`	$\{\overrightarrow{n}, \mathsf{border}\} \to \{\overleftarrow{n}, \mathsf{border}\}$ $\{\overrightarrow{n}, \mathsf{b}\} \to \{\overleftarrow{n}, \mathsf{b}\}$ $\{\overrightarrow{n}, \mathsf{bMv}\} \to \{\overleftarrow{n}, \mathsf{b}\}$	$\{\mathsf{border}, \overleftarrow{n}\} \to \{\mathsf{border}, \overrightarrow{n+1}\}$ $\{\mathsf{a}, \overleftarrow{n}\} \to \{\overrightarrow{n+1}\}$ $\{\mathsf{aMv}, \overleftarrow{n}\} \to \{\overrightarrow{n+1}\}$
n `B--`	$\{\overrightarrow{n}, \mathsf{border}\} \to \{\overleftarrow{n}, \mathsf{border}\}$ $\{\overrightarrow{n}, \mathsf{b}\} \to \{\overleftarrow{n}\}$ $\{\overrightarrow{n}, \mathsf{bMv}\} \to \{\overleftarrow{n}\}$	$\{\mathsf{border}, \overleftarrow{n}\} \to \{\mathsf{border}, \overrightarrow{n+1}\}$ $\{\mathsf{a}, \overleftarrow{n}\} \to \{\mathsf{a}, \overrightarrow{n+1}\}$ $\{\mathsf{aMv}, \overleftarrow{n}\} \to \{\mathsf{a}, \overrightarrow{n+1}\}$
n `A !=0` m	$\{\overrightarrow{n}, \mathsf{border}\} \to \{\overleftarrow{n}, \mathsf{border}\}$ $\{\overrightarrow{n}, \mathsf{b}\} \to \{\overleftarrow{n}, \mathsf{b}\}$ $\{\overrightarrow{n}, \mathsf{bMv}\} \to \{\overleftarrow{n}, \mathsf{b}\}$	$\{\mathsf{border}, \overleftarrow{n}\} \to \{\mathsf{border}, \overrightarrow{n+1}\}$ $\{\mathsf{a}, \overleftarrow{n}\} \to \{\mathsf{a}, \overrightarrow{m}\}$ $\{\mathsf{aMv}, \overleftarrow{n}\} \to \{\mathsf{a}, \overrightarrow{m}\}$
n `B !=0` m	$\{\overrightarrow{n}, \mathsf{border}\} \to \{\overleftarrow{n}_N, \mathsf{border}\}$ $\{\overrightarrow{n}, \mathsf{b}\} \to \{\overleftarrow{n}_Y, \mathsf{b}\}$ $\{\overrightarrow{n}, \mathsf{bMv}\} \to \{\overleftarrow{n}_Y, \mathsf{b}\}$	$\{\mathsf{border}, \overleftarrow{n}_N\} \to \{\mathsf{border}, \overrightarrow{n+1}\}$ $\{\mathsf{a}, \overleftarrow{n}_N\} \to \{\mathsf{a}, \overrightarrow{n+1}\}$ $\{\mathsf{aMv}, \overleftarrow{n}_N\} \to \{\mathsf{a}, \overrightarrow{n+1}\}$ $\{\mathsf{border}, \overleftarrow{n}_Y\} \to \{\mathsf{border}, \overrightarrow{m}\}$ $\{\mathsf{a}, \overleftarrow{n}_Y\} \to \{\mathsf{a}, \overrightarrow{m}\}$ $\{\mathsf{aMv}, \overleftarrow{n}_Y\} \to \{\mathsf{a}, \overrightarrow{m}\}$
	$\{\overrightarrow{\mathsf{stop}}, \mathsf{border}\} \to \{\overleftarrow{\mathsf{stop}}, \mathsf{border}\}$ $\{\overrightarrow{\mathsf{stop}}, \mathsf{b}\} \to \{\overleftarrow{\mathsf{stop}}, \mathsf{b}\}$ $\{\overrightarrow{\mathsf{stop}}, \mathsf{bMv}\} \to \{\overleftarrow{\mathsf{stop}}, \mathsf{b}\}$	$\{\mathsf{border}, \overleftarrow{\mathsf{stop}}\} \to \{\mathsf{border}\}$ $\{\mathsf{a}, \overleftarrow{\mathsf{stop}}\} \to \{\mathsf{a}\}$ $\{\mathsf{aMv}, \overleftarrow{\mathsf{stop}}\} \to \{\mathsf{a}\}$

Fig. 3. Translation of instructions.

In the space-time diagram of figures 4 and 5, we suppose that instructions at lines $n-1$ and $n+1$ are not doing anything, except in the right cases where the previous one is `B++`, so that there is some bMv to set in position.

The `B !=0` instruction is carried out very simply: the active signal goes right. If it encounters border (counter `B` is zero) then it comes back as $\overleftarrow{n}_N$. Otherwise (counter `B` is not zero), it encounters b or bMv and comes back as $\overleftarrow{n}_Y$. On the left it meets border or a and turns to the next instruction ($n+1$) if it was $\overleftarrow{n}_N$ or the indicated jump (m) if it was $\overleftarrow{n}_Y$. This is all summed up in Fig. 4.

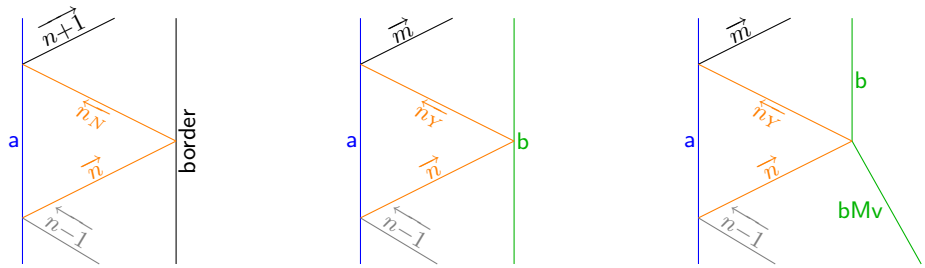

Fig. 4. Implementation of n `B!=0` m.

The `B++` instruction is carried out by issuing a moving b signal (*i.e.* bMv) that will be set in position by the next return of the active signal as depicted on Fig. 5. The active signal is faster than bMv so that it can fix bMv before bMv reaches the left side. Since space is continuous there is always room for more signal in the middle. Each instruction sets the moving bMv (or aMv), if any.

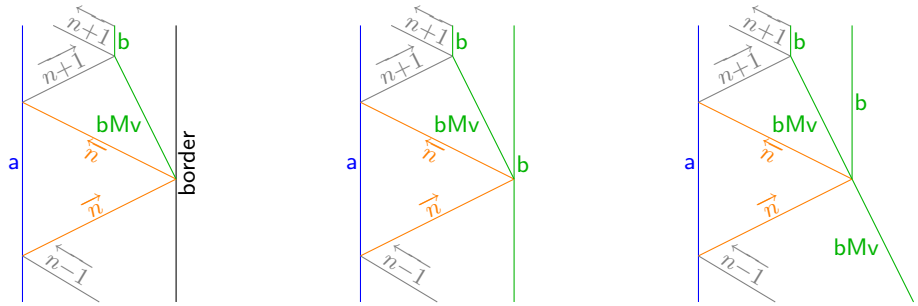

Fig. 5. Implementation of `B++`.

The instruction `A++` is carried out similarly. The instructions `A--` and `B--` are just erasing exactly one corresponding signal, if any. The non-zero test for `A` is done by simply noticing that the left border is met only if it is zero; branching is done on the collision on the left side.

Figure 6 provides three space-time diagrams associated to different initial values. The pictures are strained vertically in order to fit.

A two-counter automaton stops when it should execute the instruction right after the last one, *i.e.*, when executing instruction $n_0{+}1$ (n_0 is the number of instructions). We call the instruction $\overrightarrow{n_0{+}1}$ ($\overleftarrow{n_0{+}1}$) $\overrightarrow{\mathsf{stop}}$ and $\overleftarrow{\mathsf{stop}}$. These signals fix any moving aMv or bMv as indicated at the bottom of Fig. 3.

This way, any two-counter automaton can be simulated by a signal machine. Signal machines thus form a model of computation which has at least Turing-computing capability.

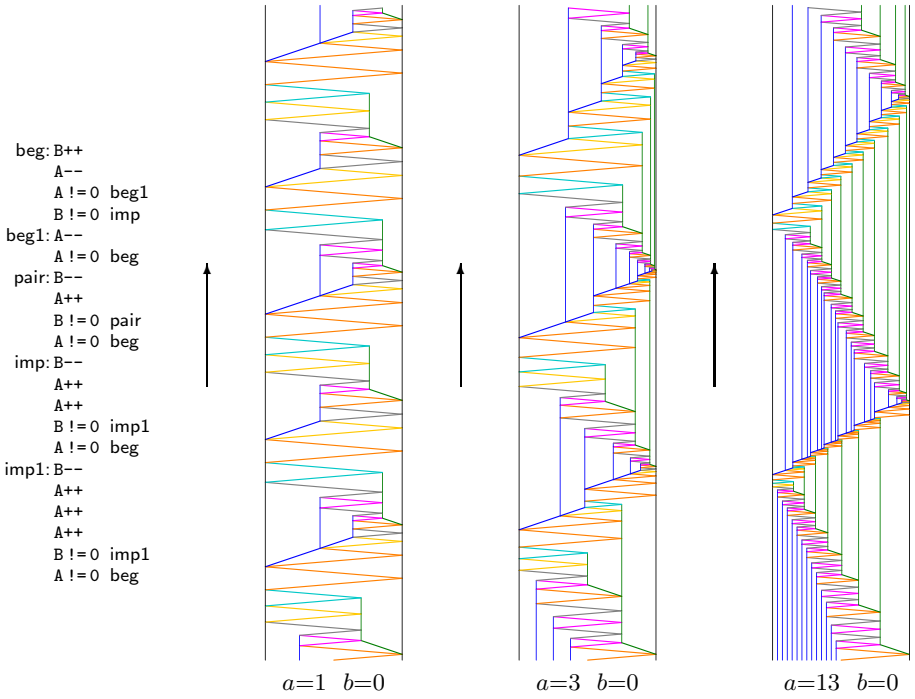

Fig. 6. A two-counter automaton and its simulations for three different initial values.

4 Undecidability results

Straight from the possibility to simulate Turing Machine comes a few undecidability results. The proofs are not detailed, they are only sketched as they correspond to classical reduction from the Halting problem.

In all the following problems, a signal machine is *rational* iff all its speeds are rationals and all initials positions are rational. A direct recurrence shows that all collisions take place at rational locations. This is needed for data to be addressed in classical recursion theory.

[Finite number of collisions]
Instance A rational signal machine, and an initial configuration.
Question Does the computation of the machine on the initial configuration stop?

The undecidability comes from the ability of rational signal machine to simulate any two-counter automaton in a way that there is finitely many collisions iff the simulation stops.

[Appearance of a given meta-signal]
Instance A rational signal machine, an initial configuration, and a meta-signal.
Question Does the computation of the machine on the initial configuration ever generates a signal of this meta-signal?

The undecidability comes from the ability of rational signal machine to simulate any two-counter automaton in a way that a signal $\overrightarrow{\mathsf{stop}}$ appears iff the simulation stops.

[Collision with a given signal]
Instance A rational signal machine, an initial configuration, and a signal in the initial configuration.
Question Is there any collision involving the given signal on the computation of the machine on the initial configuration?

The undecidability comes from a slight modification of the machine: add a second border signal on the right and make $\overrightarrow{\mathsf{stop}}$ pass the first one and collide with the second. This second border takes part in a collision if and only if $\overrightarrow{\mathsf{stop}}$ appears.

[Disappearance of all signals]
Instance A rational signal machine, and an initial configuration.
Question Does the computation of the machine on the initial configuration stop on an empty configuration?

The undecidability also comes from the simulation: it is easy to modify it in such that $\overrightarrow{\mathsf{stop}}$ and $\overleftarrow{\mathsf{stop}}$ erase everything.

5 Conclusion

As long as the model is restricted to rationals, there are finitely many signals present at any instant and there is no accumulation, the model is Turing-universal and can be simulated by any Turing machine and is thus Turing-equivalent. If the "finite number of signals" condition is lifted, but signals are isolated, then this is a super-Turing model of computation following the "Infinity principle" of [EW03]. The analog power really appears when one of the remaining two constraints is lifted.

Allowing real values for speeds or positions is simple. These real values can be used as oracles and thus provide computing ability that goes beyond Turing-computation.

With a careful "treatment of accumulations", it is possible to access infinite Turing computation or computation on ordinals [HL00,Ham02]. Our model then becomes somehow similar to the black hole model [EN93,EN02] as done in [DL05].

References

[Ada02] A. Adamatzky, editor. *Collision based computing*. Springer, 2002.
[AM95] E. Asarin and O. Maler. Achilles and the Tortoise climbing up the arithmetical hierarchy. In *FSTTCS '95*, number 1026 in LNCS, pp. 471–483, 1995.
[BCSS98] L. Blum, F. Cucker, M. Shub, and S. Smale. *Complexity and real computation*. Springer, New York, 1998.
[BNR91] N. Boccara, J. Nasser, and M. Roger. Particle-like structures and interactions in spatio-temporal patterns generated by one-dimensional deterministic cellular automaton rules. *Phys. Rev. A*, 44(2):866–875, 1991.

[Bou97] O. Bournez. Some bounds on the computational power of piecewise constant derivative systems. In *ICALP '97*, number 1256 in LNCS, pp. 143–153, 1997.

[Bou99] O. Bournez. Achilles and the Tortoise climbing up the hyper-arithmetical hierarchy. *Theoret. Comp. Sci.*, 210(1):21–71, 1999.

[Bra95] M. S. Branicky. Universal computation and other capabilities of hybrid and continuous dynamical systems. *Theoret. Comp. Sci.*, 138(1):67–100, 1995.

[DL97] J. Durand-Lose. Intrinsic universality of a 1-dimensional reversible cellular automaton. In *STACS '97*, number 1200 in LNCS, pp. 439–450. Springer, 1997.

[DL98] J. Durand-Lose. Parallel transient time of one-dimensional sand pile. *Theoret. Comp. Sci.*, 205(1–2):183–193, 1998.

[DL00] J. Durand-Lose. Reversible space-time simulation of cellular automata. *Theoret. Comp. Sci.*, 246(1–2):117–129, 2000.

[DL03] J. Durand-Lose. *Calculer géométriquement sur le plan – machines à signaux*. Habilitation à diriger des recherches, École Doctorale STIC, Université de Nice-Sophia Antipolis, 2003.

[DL05] J. Durand-Lose. Abstract geometrical computation for black hole computation (extended abstract). In M. Margenstern, editor, *Universal Machines and Computations (MCU '04)*, number 3354 in LNCS, pp. 175–186. Springer, 2005.

[DM02] M. Delorme and J. Mazoyer. Signals on cellular automata. in [Ada02], pp. 234–275, 2002.

[EN93] J. Earman and J. D. Norton. Forever is a day: supertasks in Pitowsky and Malament-Hogarth spacetimes. *Philosophy of Science*, 60(1):22–42, 1993.

[EN02] G. Etesi and I. Nemeti. Non-Turing computations via Malament-Hogarth space-times. *Int. J. Theor. Phys.*, 41(2):341–370, 2002. gr-qc/0104023.

[EW03] E. Eberbach and P. Wegner. Beyond Turing machines. *Bull. EATCS*, 81:279–304, 2003.

[Fis65] P. C. Fischer. Generation of primes by a one-dimensional real-time iterative array. *J. ACM*, 12(3):388–394, 1965.

[Got66] E. Goto. Ōtomaton ni kansuru pazuru [Puzzles on automata]. In T. Kitagawa, editor, *Jōhōkagaku eno michi [The Road to information science]*, pp. 67–92. Kyoristu Shuppan Publishing Co., Tokyo, 1966.

[Ham02] J. D. Hamkins. Infinite time Turing machines: Supertask computation. *Minds and Machines*, 12(4):521–539, 2002. math.LO/0212047.

[HL00] J. D. Hamkins and A. Lewis. Infinite time turing machines. *J. Symb. Log.*, 65(2):567–604, 2000. *arXiv:math.LO/9808093*.

[HSC01] W. Hordijk, C. R. Shalizi, and J. P. Crutchfield. An upper bound on the products of particle interactions in cellular automata. *Phys. D*, 154:240–258, 2001.

[Ila01] A. Ilachinski. *Cellular automata –a discrete universe*. World Scientific, 2001.

[JS90] G. Jacopini and G. Sontacchi. Reversible parallel computation: an evolving space-model. *Theoret. Comp. Sci.*, 73(1):1–46, 1990.

[JSS02] M. H. Jakubowsky, K. Steiglitz, and R. Squier. Computing with solitons: a review and prospectus. in [Ada02], pp. 277–297, 2002.

[LN90] K. Lindgren and M. G. Nordahl. Universal computation in simple one-dimensional cellular automata. *Complex Systems*, 4:299–318, 1990.

[Maz96] J. Mazoyer. On optimal solutions to the Firing squad synchronization problem. *Theoret. Comp. Sci.*, 168(2):367–404, 1996.

[Min67] M. Minsky. *Finite and infinite machines*. Prentice Hall, 1967.

[Moo96] C. Moore. Recursion theory on the reals and continuous-time computation. *Theoret. Comp. Sci.*, 162(1):23–44, 1996.

[MT99] J. Mazoyer and V. Terrier. Signals in one-dimensional cellular automata. *Theoret. Comp. Sci.*, 217(1):53–80, 1999.

[Orp94] P. Orponen. A survey of continuous-time computation theory. In D.-Z. Du and K.-J. Ko, editors, *Advances in Algorithms, languages and complexity*, pp. 209–224. Kluwer Academic Publisher, 1994.

[PE74] M. B. Pour-El. Abstract computability and its relation to the general purpose analog computer (some connections between logic, differential equations and analog computers). *Trans. Amer. Math. Soc.*, 199:1–28, 1974.

[ŠO01] J. Šíma and P. Orponen. Computing with continuous-time Liapunov systems. In *STOC '01*, pp. 722–731. ACM Press, 2001.

[SS95] H. T. Siegelmann and E. D. Sontag. On the computational power of neural nets. *J. Comput. System Sci.*, 50(1):132–150, 1995.

[VMP70] V. I. Varshavsky, V. B. Marakhovsky, and V. A. Peschansky. Synchronization of interacting automata. *Math. System Theory*, 4(3):212–230, 1970.

[Wei00] K. Weihrauch. *Introduction to computable analysis*. Texts in Theoretical computer science. Springer, Berlin, 2000.