Ways to Compute in Euclidean Frameworks

Jérôme Durand-Lose(✉)

Laboratoire d'Informatique Fondamentale d'Orléans, Université d'Orléans, Orléans, France
jerome.durand-lose@univ-orleans.fr

Abstract. This tutorial presents what kind of computation can be carried out inside a Euclidean space with dedicated primitives—and discrete or hybrid (continuous evolution between discrete transitions) time scales. The presented models can perform Classical (Turing, discrete) computations as well as, for some, hyper and analog computations (thanks to the continuity of space). The first half of the tutorial presents three models of computation based on respectively: ruler and compass, local constraints and emergence of polyhedra and piece-wise constant derivative. The other half concentrates on signal machines: line segments are extended and replaced on meeting. These machines are capable hyper-computation and analog computation and to solve PSPACE-problem in "constant space and time" though partial fractal generation.

Keywords: Analog computation · Computability · Fractal computation · Fractal generation · Hybrid-computation · Hyper-computation · Mondrian Automata · Piece-wise constant derivative · Ruler and compass · Signal machine · Turing computation

1 Introduction

This tutorial provides some insight on the following question: *What can be done with a Euclidean space with dedicated primitives and controls?* Space is not considered as the place to assemble gates and wires but as the substrate of computation itself. The general framework is not machines or automata but some Euclidean space where information is displayed and evolved according to some dynamics.

The approaches considered here are: constructions with ruler and compass, polyhedra emerging from local constraints, extending a sequence of line segments across polyhedral regions, extending line segments until they intersect, etc. In each case, distance, carried information, available room, encounters/collisions, etc., are elements where *spatial localization matters*.

Space is Euclidean, this means, on the one hand, that it is continuous and, on the other hand, that the underlying geometry is the one of points, lines and circles. This geometrical point of view is prevalent here as shown by the illustrations. This general framework has limitations: no differential equation, no

M.J. Patitz and M. Stannett (Eds.): UCNC 2017, LNCS 10240, pp. 8–25, 2017.
DOI: 10.1007/978-3-319-58187-3_2

algebraic geometry, etc. Outside of instantaneous "border" crossing or apparatus operation, all is straightforward and absolutely plain. The models presented here belong to a more general framework: hybrid systems with continuous (related to the nature of space and possibly time) and discrete (phase transition, collision, etc.) traits.

Continuity opens the way to *Zenon effects*: an infinite number of discrete transitions during a finite (continuous) duration (in a finite space). Many models use this capability to *hyper-compute* (solving the Halting problem and even "less computable" problems, see Syropoulos (2010)).

Like Euclidean geometry, presented models are idealized: lines have zero width, positions are exact, etc. From the physics standpoint, they are more abstract than realistic: unbounded density of information, space is Euclidean at every scale, etc.

When Turing computability is addressed, *rational* versions of the models are used: all coordinates, speeds, etc., are rational numbers. On the one hand, this often allows exact manipulation on a computer and on the other hand, it prevents oracles to be encoded in the system as a real number [for example the solution to the Halting problem as Chaitin's omega number (Calude 2002, Chap. 7)].

Each presented model is described, main results and references are provided. Proofs are omitted as well as complex results. Clues are provided as long as they remain intelligible.

This tutorial has two parts. The first part presents three computing models. The first model, the *Geometric Computation Machines* of Huckenbeck (1989, 1991), uses an automaton to activate ruler and compass and generates points, lines and circles. The second one, the *Mondrian Automata* of Jacopini and Sontacchi (1990), starts from uniform local constraints (on open balls from $\mathbb{R}^n$) on space-time diagrams ensuring causality; from these emerge polyhedra at the usual scale. The third one, the *Piece-wise Constant Derivative* of Asarin and Maler (1995); Asarin et al. (1995), partitions space into polyhedral regions corresponding to constant speeds; the orbit starting from a single point can perform infinitely many region changes during a finite portion.

The second part concentrates on one model: the signal machines of Durand-Lose (2005, 2006). After the definition of the model, a simulation of a generic Turing machine is presented. Using the continuity of both space and time, it is possible to dynamically scale down the computation and accelerate to implement a form of the Black Hole model of computation (and to hyper-compute). Fractal generation scheme can be used in order to dispatch sub-computations and to achieve *fractal computation* (allowing, e.g., to solve quantified SAT in constant space and time). This part ends by showing that the model is capable of analog computation (computing over real numbers).

This survey of computing models involving space is not comprehensive. Some models like cellular automata or tile assembling systems have their own devoted conferences (or already had been the subject of a tutorial at UCNC) and have so much literature about that each would spread over a few books; it would be pointless to present them in a few pages. Models using higher level mathematics

(differential equations, algebraic geometry, etc.) would not fit here and neither would algorithmic geometry. Many others (e.g. continuous counterparts of cellular automata like (Hagiya 2005; Takeuti 2005), use of optics to manipulate 2D-pictures (Naughton and Woods 2001; Woods and Naughton 2005)) are not addressed just because the purpose is to show the variety and specificity and not make an inventory.

This tutorial is based on the survey Durand-Lose (2016).

2 Three Models Operating on Euclidean Geometry

2.1 Ruler and Compass

This section is devoted to the work of Huckenbeck (1989, 1991) on *Geometric Computation Machines*. The primitives of these machines are the usual geometric operations that can be carried out with ruler and compass. The purpose is not to do algorithmic geometry (would it be discrete, symbolic or algebraic) but to construct in a two dimensional Euclidean space.

Each machine is an automaton (or program) equipped with a finite number of registers. There are three kinds of register: for points, for lines and for circles. The states of the automaton are used to represent both the program counter and to record the state of the computation (i.e. Unfinished, Finished and Error, the last two ones are final).

The available operations are:

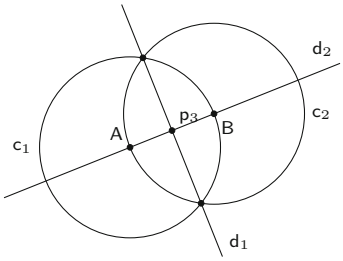
- output a value (point, line or circle),
- put in a register the intersection of two lines,
- put in a register one of the intersections of a line and a circle (optionally different from some point),
- put in a register one of the intersections of two circles (optionally different from some point),
- put in a register the line going through two points,
- put in a register the circle whose center is given (as a point) as well as its radius (as the distance between two points),
- copy a register, and
- Finished.

Intersections do not necessarily exist and neither are unique. This means that the execution of the automaton is non deterministic. Whenever an instruction cannot be carried out, the branch (of the tree of all possible executions) ends with Error.

If the whole tree of possible executions is finite, has only Finished (i.e. no Error) leafs and all its branches generate the same output, then the computation succeeds and the output is the common output (it is generated by every branch). For example, the program of Fig. 1 computes the middle of a segment (whose extremities are A and B and are the only input). Please note that there are two possible executions (where are p_1 and p_2?), but their outputs are identical.

1: $c_1 \leftarrow$ Circle (center A, radius d(A,B))
2: $c_2 \leftarrow$ Circle (center B, radius d(A,B))
3: $p_1 \leftarrow$ Intersection (c_1, c_2)
4: $p_2 \leftarrow$ Intersection (c_1, c_2) different from p_1
5: $d_1 \leftarrow$ Line (p_1, p_2)
6: $d_2 \leftarrow$ Line (A, B)
7: $p_3 \leftarrow$ Intersection (d_1, d_2)
8: Output p_3
9: Finished

(a) program



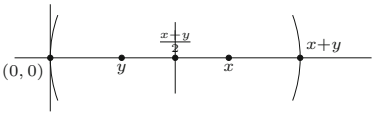
(b) construction

Fig. 1. Constructing the middle of the segment AB.

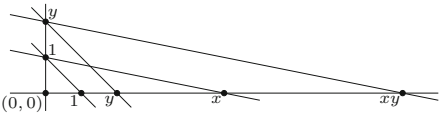
With the given primitives, it is also possible to construct the perpendicular to a line passing through a point and then the parallel.

Conditional jump instructions are like "if $p_k \in E$ go to i: otherwise to j:" where p_k is a point-register and E is a predefined set used as an oracle.

A simple case is when E contains only the origin $(0, 0)$ and points $(1, 0)$ and $(0, 1)$ are provided as constants. The functions (computable in bounded time) from an n-tuple of points to n-tuple of points are exactly the ones where the input is divided in finitely many pieces (defined as intersection of finitely many algebraic surfaces) where the coordinates of the output can be expressed with rational functions. This is related to the possibility to implement the following primitives: on the one hand, projections $(x, y) \to (x, 0)$, $(x, y) \to (y, 0)$ and reconstruction $(x, 0)\,(y, 0) \to (x, y)$ and, on the other hand, addition, multiplication and division on the x axis as on Fig. 2.



(a) addition: $(\frac{x+y}{2}, 0)$ then $(x + y, 0)$



(b) multiplication: $(0, y)$ then $(xy, 0)$

Fig. 2. Constructing from $(x, 0)$ and $(y, 0)$ with constants $(0, 0)$, $(1, 1)$ and $(0, 1)$.

This corresponds to the classical construction of numbers computable with rule and compass (Conway and Guy 1996, Chap. 7). These are also closed by square rooting. Here, the condition that each branch should generate the same output makes it impossible for root to appear (Huckenbeck 1991).

It should be noted that since the following operations can be performed: $(x, 0) \to (x + 1, 0)$, $(x, 0) \to (x - 1, 0)$, and test whether $(x, 0)$ is $(0, 0)$; an unbounded counter can be encoded with a point register. These machines can simulate any 2-counter automaton and are thus Turing-universal.

2.2 Mondrian Automata

The work of Jacopini and Sontacchi (1990) starts from a space and time modeling of reality. Hypotheses are made from which follow local constraints that brought forth the emergence of polyhedra.

In a Euclidean space of any dimension, each point is associated with a state/color. The hypothesis is made that color and local neighborhood are linked: if two points have the same color, then there is a sufficiently small (non-zero) radius where balls match. This is depicted in Fig. 3.

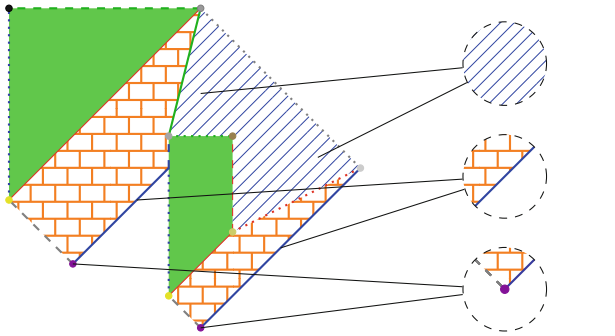


Fig. 3. Mondrian space. (Color figure online)

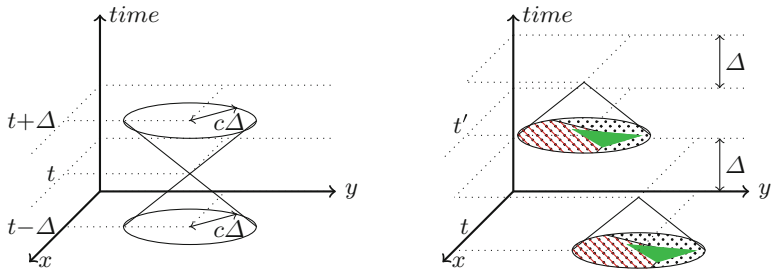
As a consequence, if there is a ball of uniform color then any point of this color is only surrounded by this color. Topologically, this means that they form an open set. Similarly, if there is a curve (of zero width) of a color then the curve must be a line segment: identical neighborhood implies a constant derivative. All the points of this color must be on parallel line segments and, following any direction, the surrounding colors should be the same. The extremities of the segments should have different colors.

More generally, each color corresponds to polyhedral regions of equal and parallel dimension. When they are restricted to their dimensions, they are open and the frontiers of lesser dimensions should be colored differently. Whereas following any other direction, adjacent colors are always the same.

Another hypothesis is that there is a finite number of colors. Hence, having a common neighborhood (up to re-scaling) for each color defines all the constraints. They provide all the information on the dimensions and directions associated to each color as well as the color of the neighbors of higher dimensions.

Next step consists in adding one dimension for time and constraints for causality. This is defined by a speed of light, c and the condition that the color of a point is uniquely defined by what is inside the past cone (delimited by the speed of light). Figure 4(b) shows two portions of space at different dates

where colors are displayed similarly. The two cones based on these portions and delimited by the speed of light are thus identical.



(a) past and future cones (b) causality

Fig. 4. Cones and causality. (Color figure online)

Another argument from physical modelisation is that the system should be reversible at local scale. This implies that the same constraint is also applied with time running in the opposite direction. This corresponds to exchanging the cones (pointing to past and future) in Fig. 4(b).

Temporal constraints can also be read at the polyhedra scale. It is possible to think in terms of intersections and collisions (this kind of approach is developed in the signal machine section). At this level, simulating a Turing machine in dimension 2 (or 1 + 1 for time) would look like Fig. 8(b) except for reversibility. (Reversible signal machines can compute as proven in Durand-Lose (2012).)

2.3 Piece-Wise Constant Derivative

In this model introduced in Asarin and Maler (1995), space is partitioned into a finite number of polyhedral regions. On each region, a constant speed is defined. On Fig. 5, thick lines separate the regions and the arrows indicate the directions of speeds.

Starting from any point a trajectory is defined. When a region border is reached, movement just follows on the other side with the new speed. In Fig. 5, two trajectories are indicated. They both start on the left. The dashed trajectory changes direction twice and then goes away forever. The dotted one is wrapping itself infinitely around the intersection point of three regions.

This second trajectory is singular: it changes region infinitely often but nevertheless reaches its limit in finite time (as a convergent geometrical sum) and stops there. There are two distinct time scales: a continuous time one where the limit is reached in finite time and an infinite discrete time one (of region change events). This is a *Zeno* phenomenon/effect.

The rest of the section is restricted to rational initial points and vertices (of polyhedra). Those systems can compute considering that the input is the initial

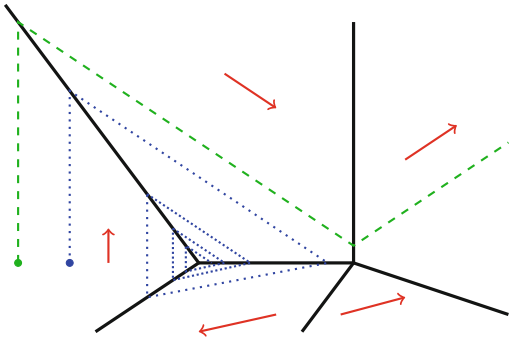
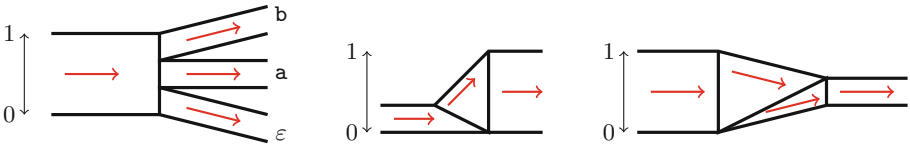


Fig. 5. Piece-wise Constant Derivative trajectories.

position (in a given zone) and that halt and result correspond to entering some other identified zone. In four dimensions, it is possible to encode the configuration of a Turing machine and operate it in the following way. The words over $\{\mathtt{a}, \mathtt{b}\}$ (left and right part of the tape) are encoded with the recursive function ψ defined by: $\psi(\varepsilon) = 1/4$, $\psi(\mathtt{a}.w) = 1/3(1 + \psi(w))$, and $\psi(\mathtt{b}.w) = 1/3(2 + \psi(w))$. Symbols can be accessed by the primitives in Fig. 6. (Scaling in Fig. 6(b) is done by changing the direction; proportions are preserved by Thales's theorem.) The tape needs two dimensions ($(0,1)^2$). The state is encoded as the part in space the trajectory is in. The trajectory loops and that each loop corresponds to a transition of the Turing machine. Building and merging the "looping pipes" use the two extra dimensions.



(a) extract the first symbol (b) re-scale (c) prefixing with a

Fig. 6. Primitives for manipulating sequences of symbols.

With a more involved proof, the Reachability problem—to decide whether a zone can be reached from a point—is thus undecidable in dimension 3 and above. Adding dimensions to the systems allows to add nested levels of Zeno effect and to climb hierarchies in the undecidable. With d dimensions, the level $d-2$ of the arithmetical hierarchy (viz. Σ_{d-2})[1] is decidable (Asarin and Maler 1995; Asarin et al. 1995). The model is even more powerful: Reachability is complete on levels of the hyper-arithmetic hierarchy[2] (Bournez 1997, 1999a,b).

[1] Σ_0 is the recursive sets, Σ_1 is recursively enumerable sets, e.g. the Halting problem.
[2] Extension of the arithmetical hierarchy to ordinal indices.

3 Signal Machines

This section is dedicated to *signal machines* (SM) (Durand-Lose 2005, 2006). This model was born as a continuous abstraction of Cellular Automata (CA) (Durand-Lose 2008). *Signals* allow to store and transmit information, to start a process, to synchronize, etc. They are the key tool in CA both for building CA and for understanding. CA dynamics are often detailed as signals interacting in collisions resulting in the generations of new signals.

CA-signals extend over one or more discrete cells whilst SM-signals are dimensionless points on a 1-dimensional Euclidean space. The main properties of CA are preserved: synchronicity (signals move at the same pace) and uniformity (the dynamics are always and everywhere the same: the speeds and interactions only rely on the nature of signals, like CA-patterns define the evolution of discrete signals). Signals have uniform movement and "draw" line segments on space-time diagrams. The nature of a signal is called a *meta-signal*.

A *Signal machine (SM)* is defined by a triplet (M, S, R) where M is a finite set of *meta-signals*, S is a function associating a *speed* to each meta-signal, and R is a set of collision rules. A *collision rule* associates to sets of at least two meta-signals of different speeds (*incoming*) a set of meta-signals of different speeds (*outgoing*). R is deterministic: a set appears at most once as the left (incoming) part of a rule.

In any configuration, there are finitely many signals and collisions. They are located in distinct places in space. Since a signal is completely defined by its associated meta-signal and a collision by a rule, a configuration is fully defined by associating to each point on the real axis a meta-signal, a rule or nothing.

As long as signals do not meet, each one moves uniformly; whereas as soon as two or more signals meet, a collision happens. Collisions provide a discrete time scale. Dynamics are defined using it: at any collision, incoming signals are instantly replaced by outgoing signals according to collision rules. In-between collisions, signals regularly propagate. This emphasizes the hybrid aspect of SM: continuous steps separated by discrete steps.

To find the location of a collision, a linear system of two equations in two variables has to be solved. Thus the location of any collision of signals whose speeds and initial locations are rational numbers, has to be rational. A signal machine is *rational* ($\mathbb{Q}$-SM) if all speeds are rational numbers as well as any non-void positions in any initial configuration. In any generated space-time diagram, all collisions have rational locations and the positions of signals are rational at each collision time.

Example 1 (Finding the middle). It is possible to compute the middle of two signals, i.e. to position a signal exactly there. This is illustrated in Fig. 7 where a O signal is positioned exactly half-way between two W signals (bottom of Fig. 7(c)). The meta-signals and collision rules are defined in the left Fig. 7. On the right, is depicted a space-time diagram generated from a configuration with signals of meta-signals (left to right): Sub, Add, Add, W and W.

The process is started by the arrival of a Add signal on the left. When it encounters the left W, it is transformed into A and $\overrightarrow{\mathsf{R}}$. The latter is three times faster than the former and bounces on the right W; it becomes then $\overleftarrow{\mathsf{R}}$, still three times faster (but with opposite direction). It encounters A exactly halfway between the two W. The correct positioning of this collision can be proved by computing the locations of all the intermediate collisions.

Considering the rules in Fig. 7(b), finding the middle only uses the three first ones as can be read from the diagram. The fourth one allows to generate the middle between the left W and the first O on right of it. This is started by sending another Add from the left as illustrated in the middle in Fig. 7(c).

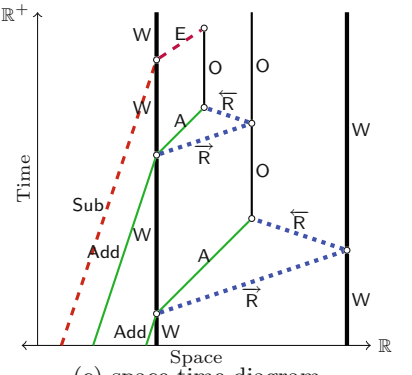
It is also possible to suppress the first O on the right of the left W. To achieve this, a Sub *order* is sent from the left. It becomes E when *passing* over W. Signal E collides and destroys the first O it encounters. This corresponds to the last two rules and the top of Fig. 7(c).

Name	Speed
Add, Sub	1/3
A, E	1
O, W	0
$\overrightarrow{\mathsf{R}}$	3
$\overleftarrow{\mathsf{R}}$	−3

(a) meta-signals

$\{\mathsf{Add}, \mathsf{W}\} \to \{\mathsf{W}, \mathsf{A}, \overrightarrow{\mathsf{R}}\}$
$\{\overrightarrow{\mathsf{R}}, \mathsf{W}\} \to \{\overleftarrow{\mathsf{R}}, \mathsf{W}\}$
$\{\mathsf{A}, \overleftarrow{\mathsf{R}}\} \to \{\mathsf{O}\}$
$\{\overrightarrow{\mathsf{R}}, \mathsf{O}\} \to \{\overleftarrow{\mathsf{R}}, \mathsf{O}\}$
$\{\mathsf{Sub}, \mathsf{W}\} \to \{\mathsf{W}, \mathsf{E}\}$
$\{\mathsf{E}, \mathsf{O}\} \to \{\}$

(b) collision rules



(c) space-time diagram

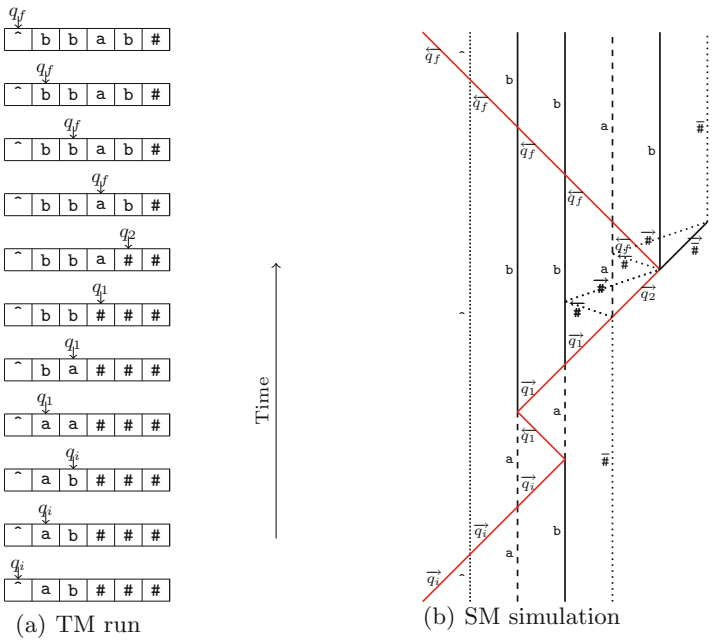
Fig. 7. Finding the middle and more.

Finding the middle is a key primitive for designing SM. For example, as shown above, it is possible to use it repeatedly to record any natural number in unary (with O's as in Fig. 7(c)) in a bounded space.

3.1 Turing Computability

Signal machines can simulate any Turing machine (TM) as shown in Fig. 8. The evolution of the TM in Fig. 8(a) can be seen in Fig. 8(b). Vertical (null speed) signals encode each cell of the tape. Zigzagging signals indicate the position of the head and record the state of the automaton. Another interest of SM is to provide graphical traces.

The enlargement of the tape is done with the middle construction, but backwards! It is also possible to set these speeds such that the distance is halved



(a) TM run (b) SM simulation

Fig. 8. Simulating a Turing machine with a signal machine.

each time. The width of the whole tape is then bounded independently from the number of cells.

This construction works on $\mathbb{Q}$-SM that can be simulated exactly on any computer. Leaving open the definition of input, halt and output, $\mathbb{Q}$-SM have exactly the same computing power as TM. This leads to the undecidability of many problems for $\mathbb{Q}$-SM (expressible in classic context since everything is rational) like: decide whether the number of collisions is finite or decide whether a meta-signal appears or a collision rule is used.

Using various meta-signals similar to O in Fig. 7, it is possible to encode sequences of letters functioning as a stack. This can also be achieved by using positions to encode values (Durand-Lose 2006) (with irrational positions, it is even possible to encode infinite stacks). It is possible to simulate any TM with a constant number of signals and collisions involving only two signals resulting in exactly two signals (conservation of the number of signals), but moreover this remains true if rules should be injective: the rule is also defined by outgoing signals (reversibility) (Durand-Lose 2012). This simulation uses reversible universal TM (Bennett 1988; Lecerf 1963; Morita et al. 1989).

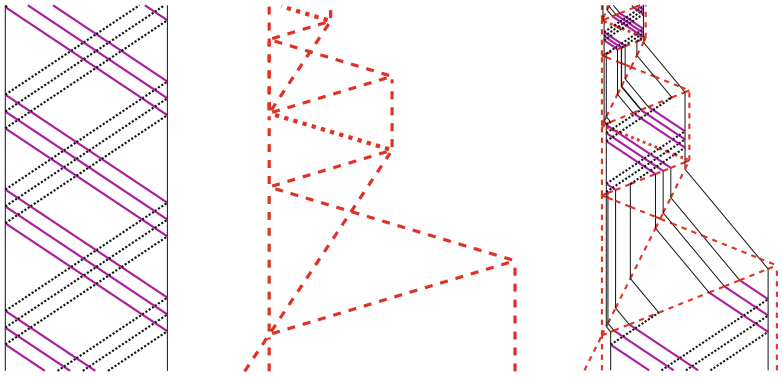
Signal machines can also be used to simulate the *Cyclic Tag Systems* introduced in Cook (2004). His work restarted the race to small universal machines, e.g. on TM (Woods and Neary 2009). The smallest Turing-universal SM known simulates any CTS and has 13 meta-signals and 21 collision rules (Durand-Lose 2011b).

3.2 Malleability of Space-Time and the Black Hole Model

The context is the continuum without scale nor origin; scaling or translating the initial produces the same space-time diagram. In particular, if all distances are halved, then so are the durations.

It is possible to dynamically re-scale a configuration and then to restart it with a construction similar to the PCD re-scaling in Fig. 6(b). To freeze a computation, a signal crosses the configuration and replaces everything it meets by parallel signals. Being parallel, there is no collision and the (relative) distances are preserved. It is unfrozen by a signal of the same velocity as the freezing one.

There is no limit to scaling. It is possible to restart the shrinking process forever as illustrated in Fig. 9. Each time the entangled original computation is activated with the same relative duration because although the activation duration is halved, since distances are halved, duration between collision is also halved. Altogether, in this finite portion of the space-time diagram, the whole infinite original space-time diagram is entangled.



(a) initial figure (b) structure (c) entanglement

Fig. 9. Iterated shrinking.

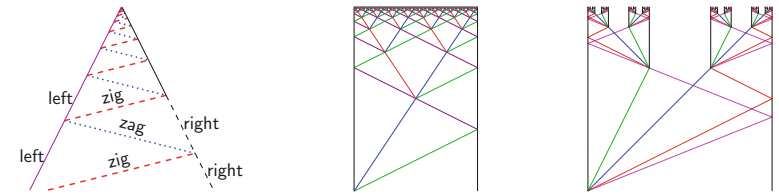
With a bounded space simulation of a TM entangled, if the computation stops, then it is in bounded time. With minor modifications, it is possible to let some signal leave the iterated shrinking in such a case. Outside of the structure, this witness of the halt could be collected but this can only happen during a bounded delay. This bound on duration can be "implemented" in the space-time diagram by a collision with another signal. If the machine does not halt, then nothing is received before that collision, whilst in case of halting, the witness is collected before. Outside of the shrinking structure, the halt is decided. SM can hyper-compute by creating a local Zeno effect (Durand-Lose 2005, 2006).

The general principle behind this construction is to have two time-lines: one is infinitely accelerated and does the computation and possibly sends some signal while the other waits for it with some timeout. This corresponds to the so-called

Black Hole model of computation (Hogarth 2004; Andréka et al. 2009; Etesi and Németi 2002). The accumulation above the space-time diagram in Fig. 9(c) corresponds to the Black Hole.

3.3 Build and Use Fractals

Many fractals can be generated using SM in a straightforward way. For example, four meta-signals are enough to build the fractal accumulation in Fig. 10(a). The space-time diagram is undefined at this accumulation *singularity*. Recursively generating middles also generate a fractal as in Fig. 10(b). By considering left and right thirds instead of halves, a classical construction of the Cantor set is generated as in Fig. 10(c). By varying the speed and the proportion, it is possible to generate sets of any fractal dimension between 0 and 1 (Senot 2013, Chap. 5).



(a) simple accumulation (b) half slicing (c) Cantor set

Fig. 10. Fractals.

In Fig. 10(b) spaces are sliced in half at each step. This can be used to provide parallelism and deal with one sub-cases on each side. For boolean formulas, it is possible to recursively slice for each and every variable. All cases are thus generated. If the variables appear in some boolean formula whose satisfiability is to be checked, then one gets a scheme to solve the satisfaction of boolean formula (SAT). This scheme can also deal with quantified variables to solve the PSPACE-complete Quantify SAT (QSAT) (Sipser 1997, Sect. 8.3).

If a boolean formula contains 10 variables, then 10 levels of slicing are done. (What remains of the construction of the fractal in Fig. 10(b) is useless and dangerous since the diagram is not defined at the limit.) For example, in Fig. 11, the QSAT formula, $\exists x_1 \forall x_2 \forall x_3\ x_1 \wedge (\neg x_2 \vee x_3)$, is represented by a ray of signals encoding all its elements. Computation is organized following a complete binary tree (of depth 3). Evaluation is done on the leafs and results are aggregated on top of the space-time diagram.

A specific machine is generated for each formula. Using a more complex encoding, it is possible to use a unique machine for which the initial configuration totally encodes the formula (Duchier et al. 2012; Senot 2013, Chap. 8).

It is also possible to solve other problems on formulas: how many satisfying valuations (#SAT, #P-complete)? What is the "smallest" satisfying valuation? etc. One "just" has to change the way the variable-free formulas are evaluated

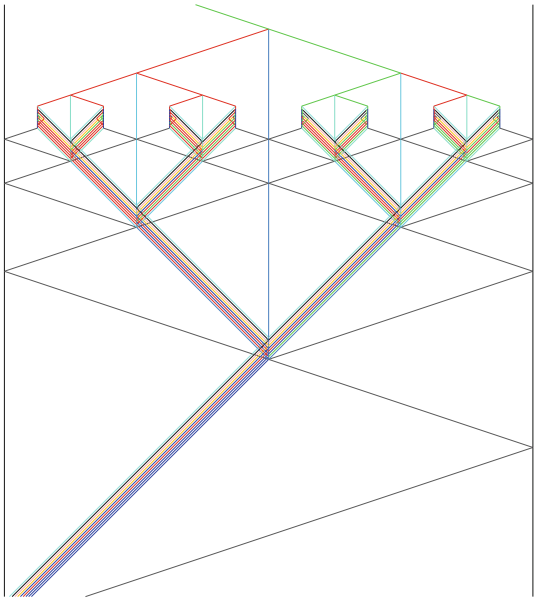


Fig. 11. Solving QSAT with fractal computation, the whole picture.

and the results are aggregated to generate an integer, a valuation, etc. This is a modular parametrization of the construction.

More levels of the fractal are generated as needed. It does not need more time or space. Altogether, there is a SM able to decide any instance of QSAT in constant space and time. Using a controlled and unfinished fractal construction to display parallel computations and then aggregate the result is called *fractal computation*.

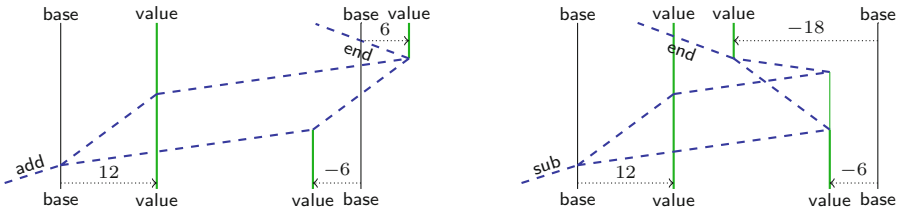
Discrete complexity measures are defined by considering space-time diagrams as directed acyclic graphs. A *direct causal link* exists between two signals if the first ends in the collision where the second starts. A *causal link* is its transitive closure. *Time complexity* is then the size of the longest sequence of signals with direct causal link between each two consecutive signals (path in the DAG). *Space complexity* is the largest number of signals without any causal link. With these definitions, complexity is quadratic in time (cubic for the generic case) but exponential in space.

Figure 10(a) shows that it is possible to build a fractal with only four different speeds. With two speeds or less, the number of collisions is finite and bounded. With three speeds, the situation is two-fold: in a $\mathbb{Q}$-SM, signals must travel on a regular mesh (without any accumulation). But accumlation might happen as soon as there is an irrational ratio between speeds or between initial positions. This can be understood by the presence of a mechanism computing the *gcd*, which only converges on rational (Becker et al. 2003).

What about the computing capability? In the rational case, the same mesh shows that usable memory is finite and bounded. Whereas in the other case, it is possible to simulate a TM using a fractal construction step to enlarge the tape (Durand-Lose 2013).

3.4 Analog Computation

This section deals with computation on real numbers. They are represented by the distance between two (parallel) signals of distinct meta-signals (base or value)—or one encoding zero (zero). Dividing by two corresponds to finding the middle. Multiplication by any constant can be done likewise. Adding two numbers can be done as in Fig. 12(a). The presence of a parallelogram proves the equality of the distances. Subtraction can be done similarly as depicted in Fig. 12(b) where the parallelogram is folded around the lower right value.



(a) adding 12 to −6 (b) subtracting 12 from −6

Fig. 12. Basic operations on reals.

Starting from a sequence of real values (like the sequence of cells of the tape of a TM), it is possible to multiply by constants and to add a value to another inside a window (bounded part of the sequence) and to move the window. These primitives could be triggered by some deterministic finite automata (or sequential program). The state of the automaton can be encoded in the meta-signals used to carry out the operations.

The automaton can be equipped with conditional transition: testing the sign of a value can be used to branch. The automaton can have an initial and final state (no more collisions then). Constants may be provided in the initial configuration.

Altogether, starting from a finite sequence of real numbers (infinity extendable on both side), it is possible to store in a cell the linear combination of values around it, branch according to sign and move inside the sequence. This corresponds to the BSS model (Blum et al. 1989, 1998) without inner multiplication. It is the linear version of it: lin-BSS (Bournez 1999b; Meer and Michaux 1997) with an unbounded number of registers.

Signal machines are capable of implementing lin-BSS. The converse is also true. The configuration of a SM can be represented by a sequence of blocks,

each one encoding the meta-signal, the distance to next plus various temporary registers. The lin-BSS machine runs through the configuration and computes the minimal time to a collision. Since speeds are constant of the SM, they become constants of the lin-BSS machine, thus everything is linear.

Once this delay to the next collision time is known, the automaton runs through the configuration again. Distances are updated. When the distance is zero, then a collision happens. Involved signals are replaced according to rules (which are hard encoded into the automaton). If the number of signals is changed, all the values on the right are moved accordingly. When room is needed (or should be removed) shifting the values in the cells is done like for a TM.

BSS and Computable Analysis. Taking accumulations and infinitely many signals during finite duration into account allows to go further on.

For example, it is possible to extract an infinite sequence of 0 and 1 representing the binary encoding of a real number. This flow can be used to make a multiplication: each time half and add or not depending on the received bit. Inner multiplication thus becomes possible and the whole classical BSS model can be implemented (Durand-Lose 2007).

Accumulation can also be perceived as a convergent approximating process which is the foundation of recursive/computable analysis, type-2 TM (Weihrauch 2000). In this context, an input is an infinite stream of symbols representing a convergent approximation (approximation bound is known at each step) and the output is also such a stream (once something is output, it cannot be modified) with the same representation. It is possible to make an accumulation to be located according to a process generating such a stream (Durand-Lose 2009, 2011a) on a $\mathbb{Q}$-SM.

One last result about isolated accumulation on $\mathbb{Q}$-SM: not only they cannot happen everywhere (by a simple cardinality argument) but their possible locations are exactly characterized in (Durand-Lose 2011c). They can only happen at dates that correspond to *computably enumerable* (*c.e.*) real numbers (Calude 2002, Chap. 7), i.e. there is a TM that produces an increasing and convergent infinite sequence (there is no hypothesis on the quality of the approximation). The positions of isolated accumulations are exactly the differences of two such numbers. Position and date can be handled independently. This is proved by a two scales construction: an embedded TM is accelerated and stopped so that it provides the data on request in bounded time, the large scale directs the accumulation to the right spot according to the provided data.

4 Conclusion

Presented models operate inside continuous euclidean spaces. Their variety is huge as well as their computing capabilities. They bring forth a new kind of algorithmic where localization, distance, relative positions, etc., provide possibilities as well as constraints.

Unsurprisingly, the capability to compute in the Turing understanding is common. As soon as it is possible to take advantage of continuity of space and

time, analog computation and hyper-computation arise too (thus transcending the Church-Turing thesis).

This remains true only in the ideal word of the model where there is no error, nor approximation, nor limit to sub-division or to density of information. This is a limit to the realism of the models. Other arguments are the unbounded quantity of information that can be stored and retrieved in a bounded space and the absence of Heisenberg's Uncertainty principle at any scale.

This leads to wonder what would be their discrete approximation. Discrete geometry and related issues are a totally different world. Nevertheless, for signal machines, the discrete counterpart exists (CA) and there are works on exact discretization: (Besson and Durand-Lose 2016).

References

Andréka, H., Németi, I., Németi, P.: General relativistic hypercomputing and foundation of mathematics. Nat. Comput. **8**(3), 499–516 (2009). doi:10.1007/s11047-009-9114-3

Asarin, E., Maler, O.: Achilles and the Tortoise climbing up the arithmetical hierarchy. In: Thiagarajan, P.S. (ed.) FSTTCS 1995. LNCS, vol. 1026, pp. 471–483. Springer, Heidelberg (1995). doi:10.1007/3-540-60692-0_68

Asarin, E., Maler, O., Pnueli, A.: Reachability analysis of dynamical systems having piecewise-constant derivatives. Theoret. Comput. Sci. **138**(1), 35–65 (1995). doi:10.1016/0304-3975(94)00228-B

Becker, F., Chapelle, M., Durand-Lose, J., Levorato, V., Senot, M.: Abstract geometrical computation 8: small machines, accumulations & rationality (2013, submitted). http://arxiv.org/abs/1307.6468

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

Besson, T., Durand-Lose, J.: Exact discretization of 3-speed rational signal machines into cellular automata. In: Cook, M., Neary, T. (eds.) AUTOMATA 2016. LNCS, vol. 9664, pp. 63–76. Springer, Cham (2016). doi:10.1007/978-3-319-39300-1_6

Blum, L., Shub, M., Smale, S.: On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines. Bull. Am. Math. Soc. **21**(1), 1–46 (1989)

Blum, L., Cucker, F., Shub, M., Smale, S.: Complexity and Real Computation. Springer, New York (1998)

Bournez, O.: Some bounds on the computational power of piecewise constant derivative systems (extended abstract). In: Degano, P., Gorrieri, R., Marchetti-Spaccamela, A. (eds.) ICALP 1997. LNCS, vol. 1256, pp. 143–153. Springer, Heidelberg (1997). doi:10.1007/3-540-63165-8_172

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

Bournez, O.: Some bounds on the computational power of piecewise constant derivative systems. Theory Comput. Syst. **32**(1), 35–67 (1999b)

Calude, C.S.: Information and Randomness: An Algorithmic Perspective. Texts in Theoretical Computer Science. An EATCS Series, 2nd edn. Springer, Heidelberg (2002). doi:10.1007/978-3-662-04978-5. ISBN 3540434666

Conway, J.H., Guy, R.L.: The Book of Numbers. Copernicus Series. Springer, Heidelberg (1996). ISBN 9780387979939

Cook, M.: Universality in elementary cellular automata. Complex Syst. **15**, 1–40 (2004)
Duchier, D., Durand-Lose, J., Senot, M.: Computing in the fractal cloud: modular generic solvers for SAT and Q-SAT variants. In: Agrawal, M., Cooper, S.B., Li, A. (eds.) TAMC 2012. LNCS, vol. 7287, pp. 435–447. Springer, Heidelberg (2012). doi:10.1007/978-3-642-29952-0_42. http://arxiv.org/abs/1105.3454
Durand-Lose, J.: Abstract geometrical computation for black hole computation. In: Margenstern, M. (ed.) MCU 2004. LNCS, vol. 3354, pp. 176–187. Springer, Heidelberg (2005). doi:10.1007/978-3-540-31834-7_14
Durand-Lose, J.: Abstract geometrical computation 1: embedding black hole computations with rational numbers. Fund. Inf. **74**(4), 491–510 (2006)
Durand-Lose, J.: Abstract geometrical computation and the linear blum, shub and smale model. In: Cooper, S.B., Löwe, B., Sorbi, A. (eds.) CiE 2007. LNCS, vol. 4497, pp. 238–247. Springer, Heidelberg (2007). doi:10.1007/978-3-540-73001-9_25
Durand-Lose, J.: The signal point of view: from cellular automata to signal machines. In: Durand, B. (ed.) Journees Automates cellulaires (JAC 2008), pp. 238–249 (2008)
Durand-Lose, J.: Abstract geometrical computation and computable analysis. In: Calude, C.S., Costa, J.F., Dershowitz, N., Freire, E., Rozenberg, G. (eds.) UC 2009. LNCS, vol. 5715, pp. 158–167. Springer, Heidelberg (2009). doi:10.1007/978-3-642-03745-0_20
Durand-Lose, J.: Abstract geometrical computation 5: embedding computable analysis. Nat. Comput. **10**(4), 1261–1273 (2011a). doi:10.1007/s11047-010-9229-6. Special issue on Unconv. Comp. 2009
Durand-Lose, J.: Abstract geometrical computation 4: small Turing universal signal machines. Theoret. Comput. Sci. **412**, 57–67 (2011b). doi:10.1016/.tcs.2010.07.013
Durand-Lose, J.: Geometrical accumulations and computably enumerable real numbers. In: Calude, C.S., Kari, J., Petre, I., Rozenberg, G. (eds.) UC 2011. LNCS, vol. 6714, pp. 101–112. Springer, Heidelberg (2011c). doi:10.1007/978-3-642-21341-0_15
Durand-Lose, J.: Abstract geometrical computation 6: a reversible, conservative and rational based model for black hole computation. Int. J. Unconv. Comput. **8**(1), 33–46 (2012)
Durand-Lose, J.: Irrationality is needed to compute with signal machines with only three speeds. In: Bonizzoni, P., Brattka, V., Löwe, B. (eds.) CiE 2013. LNCS, vol. 7921, pp. 108–119. Springer, Heidelberg (2013). doi:10.1007/978-3-642-39053-1_12. http://cie2013.disco.unimib.it/. Invited talk for special session Computation in nature
Durand-Lose, J.: Computing in perfect euclidean frameworks. In: Adamatzky, A. (ed.) Advances in Unconventional Computing. ECC, vol. 22, pp. 141–163. Springer, Cham (2017). doi:10.1007/978-3-319-33924-5_6
Etesi, G., Németi, I.: Non-turing computations via Malament-Hogarth space-times. Int. J. Theoret. Phys. **41**(2), 341–370 (2002). http://arxiv.org/abs/gr-qc/0104023
Hagiya, M.: Discrete state transition systems on continuous space-time: a theoretical model for amorphous computing. In: Calude, C.S., Dinneen, M.J., Păun, G., Pérez-Jímenez, M.J., Rozenberg, G. (eds.) UC 2005. LNCS, vol. 3699, pp. 117–129. Springer, Heidelberg (2005). doi:10.1007/11560319_12
Hogarth, M.L.: Deciding arithmetic using SAD computers. Br. J. Philos. Sci. **55**, 681–691 (2004)
Huckenbeck, U.: Euclidian geometry in terms of automata theory. Theoret. Comput. Sci. **68**(1), 71–87 (1989). doi:10.1016/0304-3975(89)90120-5
Huckenbeck, U.: A result about the power of geometric oracle machines. Theoret. Comput. Sci. **88**(2), 231–251 (1991). doi:10.1016/0304-3975(91)90375-C

Jacopini, G., Sontacchi, G.: Reversible parallel computation: an evolving space-model. Theoret. Comput. Sci. **73**(1), 1–46 (1990). doi:10.1016/0304-3975(90)90160-J

Lecerf, Y.: 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)

Meer, K., Michaux, C.: A survey on real structural complexity theory. Bull. Belg. Math. Soc. **4**, 113–148 (1997)

Morita, K., Shirasaki, A., Gono, Y.: A 1-tape 2-symbol reversible Turing machine. Trans. IEICE **E72**(3), 223–228 (1989)

Naughton, T.J., Woods, D.: On the computational power of a continuous-space optical model of computation. In: Margenstern, M., Rogozhin, Y. (eds.) MCU 2001. LNCS, vol. 2055, pp. 288–299. Springer, Heidelberg (2001). doi:10.1007/3-540-45132-3_20

Senot, M.: Modèle géométrique de calcul: fractales et barrières de complexité. Thèse de doctorat, Université d'Orléans, June 2013. https://tel.archives-ouvertes.fr/tel-00870600

Sipser, M.: Introduction to the Theory of Computation. PWS Publishing Co., Boston (1997). ISBN 0-534-944728-X

Syropoulos, A.: Hypercomputation. Springer, Heidelberg (2010)

Takeuti, I.: Transition systems over continuous time-space. Electron. Notes Theoret. Comput. Sci. **120**, 173–186 (2005). doi:10.1016/j.entcs.2004.06.043

Weihrauch, K.: Introduction to computable analysis. Texts in Theoretical Computer Science. Springer, Berlin (2000)

Woods, D., Naughton, T.J.: An optical model of computation. Theoret. Comput. Sci. **334**(1–3), 227–258 (2005). doi:10.1016/j.tcs.2004.07.001

Woods, D., Neary, T.: The complexity of small universal Turing machines: a survey. Theoret. Comput. Sci. **410**(4–5), 443–450 (2009). doi:10.1016/j.tcs.2008.09.051

http://www.springer.com/978-3-319-58186-6

Unconventional Computation and Natural Computation
16th International Conference, UCNC 2017, Fayetteville, AR, USA, June 5-9, 2017, Proceedings
Patitz, M.J.; Stannett, M. (Eds.)
2017, XVIII, 221 p. 89 illus., Softcover
ISBN: 978-3-319-58186-6