

Noname manuscript No.
(will be inserted by the editor)

Abstract geometrical computation 5: embedding computable analysis

Jérôme Durand-Lose*

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

The date of receipt and acceptance will be inserted by the editor

Abstract Extended Signal machines are proven capable to compute any computable function in the understanding of recursive/computable analysis (CA), represented here with type-2 Turing machines (T2-TM) and signed binary. This relies on a mixed representation of any real number as an integer (in signed binary) plus an exact value in $(-1, 1)$. This permits to have only finitely many signals present simultaneously. Extracting a (signed) bit, improving the precision by one bit and iterating a T2-TM only involve standard signal machines.

For exact CA-computations, T2-TM have to deal with an infinite entry and to run through infinitely many iterations to produce an infinite output. This infinite duration can be provided by an infinite acceleration construction. Extracting/encoding an infinite sequence of bits is achieved as the limit of the approximation process with a careful handling of accumulations.

Key-words. Analog computation; Abstract geometrical computation; Computable analysis; Signal machine; Type-2 Turing machine.

1 Introduction

Classical computability deals with integers, finite sequences of letters, and more generally up to countably many discrete values. Does this mean that analysis and engineering are totally disconnected from computations? That physical simulation is impossible in silico? Of course not, this is done everyday.

* e-mail: <http://www.univ-orleans.fr/lifo/Members/Jerome.Durand-Lose>,
e-mail: Jerome.Durand-Lose@univ-orleans.fr

Dealing with real values is handled in various ways: fixed approximation (π is 3.14), formal manipulation (π is PI), unbounded approximation on demand (π is a program that provide extra digits on demand), interval arithmetics (π is [3.1, 3.2])... Following their definition as Cauchy sequences of rational numbers, real numbers can be encoded by infinite decreasing sequences of intervals with rational endpoints, such that the length of the intervals tends to zero. The whole infinite sequence represents exactly one real (at its intersection), but a finite prefix of the sequence is enough to get an approximation; the larger the prefix, the better the approximation. Rational endpoints allow exact coding and manipulation of each interval in classical computation, so that the infinite sequence of intervals can be written as an infinite sequence of symbols. Functions over these infinite sequences could represent functions on the reals. A Turing machine is used: the infinite entry is written on a read-once tape and the infinite output is expected on a write-once tape (extra working tapes are available). These tapes can be seen as input and output streams. To entirely process the input and generate the whole output, an infinite number of iterations is needed, but in finitely many steps an approximation of the result is generated.

This approach, now called *Computable Analysis* (CA) was initiated by Turing [1936], then Grzegorzczuk [1957] and is detailed in classical textbooks [Ko, 1991, Weihrauch, 2000]. It has been proven equivalent to a variation of Shannon's General Purpose Analog Computer [Bournez et al., 2007].

In the present paper, *Abstract Geometrical Computation* (AGC) is proven capable to carry out any CA computation with a representation of real numbers with finitely many signals. This representation is compatible with the one used to implement the original Blum, Shub and Smale (BSS) model [Durand-Lose, 2007, 2008] (a different approach to analog computing: values and primitives are freely available [Blum et al., 1989]).

Abstract geometrical computation is defined by dimensionless signals moving uniformly in an Euclidean (continuous) space in continuous time. To each signal corresponds a *meta-signal* which defines its speed. The number of meta-signals is finite. Existing signals are only modified when they collide: they are replaced by new signals according to their meta-signals and *collision rules*. A *signal machine* collects the definition of available meta-signals and collision rules.

In [Durand-Lose, 2007, 2008], to implement the BSS-model, real numbers are encoded by the distance between two parallel signals (plus two more signals to encode the scale, *i.e.* distance 1). On one side, for CA, this is not so convenient since infinite sequences are expected. On the other side, in the AGC context, infinitely many signals would either occupy the whole infinite space or produce accumulation of signals (and maybe collisions), both are problematic.

Stoking infinitely many signals in a bounded space is a very complex yet powerful feature (fore example, with the use of folding structures, exact multiplication of real numbers was achieve [Durand-Lose, 2008]). Special

care has to be taken in defining the space-time diagram near them. The only scheme used here is that any singularity reduced to a point in the space-time diagram results in one signal (always the same). More complex rules can be devised to handle accumulations —unused here— so as to emulate the (nested) black hole model of computation [Hogarth, 1994, Etesi and Némethi, 2002, Durand-Lose, 2006, 2009a].

To avoid handling infinitely many signals, a mixed representation is used. A real number x is encoded by $n + \varepsilon$ where n is an integer and $-1 < \varepsilon < 1$; n is encoded in signed binary and ε as a distance between two signals (a pair of signals somewhere in the configuration amounts for distance 1). The encoding used for CA implementation is signed binary with symbols $\bar{1}$, 0 , 1 plus a decimal point. To go back and forth between the encodings, n is encoded as a sequence of signed bits (and thus does not have to be taken care of) and the first approximation is $(n-1, n+1)$. Then the approximation goes by steps: the interval (a, b) is replaced by $(a, \frac{a+b}{2})$, $(\frac{3a+b}{4}, \frac{a+3b}{4})$ and $(\frac{a+b}{2}, b)$ that corresponds to $\bar{1}$, 0 and 1 respectively. (Intervals overlap and the infinite sequence is not unique.) The geometric construction to generate from the decimal part an infinite sequence corresponding to the real number relies on five signals amounting for values $\frac{-1}{2^n}$, $\frac{-1}{2^{n+1}}$, 0 , $\frac{1}{2^{n+1}}$ and $\frac{1}{2^n}$. The position of ε gives the signed bit, ε is reduced while the five-signal structure is scaled down.

The presentation of the implementation of (classical) Turing machines (TM) is only illustrated; it is quite straightforward and already done in [Durand-Lose, 2009a, 2010]. A Type-2 TM needs infinitely many iterations to complete a computation. To get the results in finite duration, TM and input are embedded into a folding structure as in [Durand-Lose, 2006, 2009a].

In Durand-Lose [2009b], a *convoy* is defined and used to bring the infinite output to a structure that bring forth better and better approximations and at the limit the exact number. This is not needed here anymore: the shrinking structure is itself embedded into another shrinking structure that accumulates at the exact location of the output. These shrinking structures are called, respectively, *inner* and *outer*. The inner one stops each time a signed bit is output. On receiving this bit, the outer structure makes one shrinking operation on the left, central or right half depending on the value of the bit. Then some acknowledgement signal is sent to resume the inner structure shrinking and the T2-TM computation.

Another way to express this is to say that the TM is accelerated until a signed bit is output then it is stopped until the bit is proceeded, then the process starts again. The computation providing the bits is embedded inside the structure that accumulates on the exact output. (The inner structure been stopped and translated, it accumulates at the same location).

Definitions of AGC and CA are gathered in Section 2. Section 3 concentrates on approximation, *i.e.* the mixed representation as well as how to get bits, and the simulation of type-2 Turing machines. Section 4 deals with exact computation in finite duration. Section 5 concludes this paper.

2 Definitions

2.1 Abstract geometrical computation

A *signal machine* (SM) is defined by (M, S, R) where M (*meta-signals*) is a finite set, S (*speeds*) a mapping from M to \mathbb{R} , and R (*collision rules*) a function from the subsets of M of cardinality at least two into subsets of M (all these sets are composed of meta-signals of distinct speeds). A signal machine is *extended* if some meta-signal μ_* is distinguished to be used as the result of an isolated accumulation.

Each instance of a meta-signal is a *signal* located on the real axis. The mapping S assigns *speeds* to signals. A *collision rule*, $\rho^- \rightarrow \rho^+$, defines what emerges ($\rho^+ \subseteq M$) from the collision of two or more signals ($\rho^- \subseteq M$). Since R is a function, SM are deterministic. The *extended value set*, V , is the union of M and R plus two special values: \emptyset for void, $*$ for isolated accumulation. A (*finite*) *configuration*, c , is a mapping from \mathbb{R} to $M \cup R \cup \{\emptyset\}$ such that the set $\{x \in \mathbb{R} \mid c(x) \neq \emptyset\}$ is finite. An *extended (finite) configuration*, is a mapping from \mathbb{R} to V such that the set $\{x \in \mathbb{R} \mid c(x) \neq \emptyset, *\}$ is finite.

A (resp. extended) *space-time diagram* is a mapping from an interval of \mathbb{R} (representing the time) into (resp. extended) configurations. A signal corresponding to a meta-signal μ at a position x , i.e. $c(x) = \mu$, is moving uniformly with constant speed $S(\mu)$. A signal must start (resp. end) in the initial (resp. final) configuration or in a collision. At a $\rho^- \rightarrow \rho^+$ collision, signals corresponding to the meta-signals in ρ^- (resp. ρ^+) must end (resp. start) and no other signal should be present. There is a $*$ if and only if collisions are accumulating and it is isolated in the configuration (i.e. at the time it exists). A $*$ immediately turns into a (regular) μ_* signal, so that a μ_* signal can also result from an accumulation.

Space-time diagrams are represented with time increasing upward. The traces of signals are line segments whose directions are defined by $(S(\cdot), 1)$ (1 is the temporal coordinate) so that the speed is the inverse of the slope. Collisions correspond to the common extremities of these segments.

2.2 Computable Analysis

A *type-2 Turing machine* (T2-TM) is a regular Turing machine (TM) such that the entry is an infinite sequence of symbols written on a read-only tape. The output is also expected to be an infinite sequence of symbols written on a write once tape. The TM has an extra work tape on which it can freely read and write. It needs infinitely many iterations to read a whole entry and write the whole result. These read-once and write-once tapes are treated as input and output streams. After finitely many iterations, a finite part of the entry is read and a finite part of the result is generated; they represent approximation of the entry and the result. Since the output is write-once, anything written never changes and it converges to the infinite output according to the prefix topology.

To link this machinery to analysis, a representation of real numbers by infinite sequences should be provided. The larger the prefix of the representation is read, the more should be known on the encoded real number x , ultimately x should be perfectly known and distinguished from any other real number. The standard representation of a real number x is by any decreasing sequence of intervals with rational ends such that their intersection reduces to $\{x\}$. The infinite sequence is then just the self-delimiting concatenation of the naming of the intervals. In the present paper, an equivalent representation is used.

Let $\Sigma = \{\bullet, \bar{1}, 0, 1\}$, the *signed binary representation*, $\rho_{\text{sb}} : \subseteq \Sigma^\omega \longrightarrow \mathbb{R}$, (from [Weihrauch, 2000, Def. 7.2.4 p.206]) is defined only for infinite sequences with exactly one dot (\bullet) by:

$$w_0 \bullet d_1 d_2 d_3 \dots d_n \dots \mapsto \nu_{\text{sb}}(w_0) + \sum_{1 \leq i} \frac{d_i}{2^i}$$

where $w_0 \in \{\bar{1}, 0, 1\}^*$, $d_i \in \{\bar{1}, 0, 1\}$ and ν_{sb} is a naming of natural integers signed in base 2 ($\bar{1}$ stands for -1).

Each real number is represented by infinitely many sequences. Computing from different representations of x might produce different sequences, but since CA defines functions over \mathbb{R} , these should represent the same real number y .

3 Approximation

This section deals with finite run of type-2 Turing machines. After defining the mixed representation of real numbers and the extraction of signed bits, T2-TM implementation is presented.

3.1 Mixed representation and decoding

Each real number can be decomposed as $n + \varepsilon$ where n is an integer and ε belongs to $(-1, 1)$. The representation used in AGC is in two parts. The integer n is encoded as a sequence of signals in $\{\bar{1}, 0, 1\}$ followed by a \bullet signal. This is a direct signed binary representation. The real number ε is represented in AGC, by the distance between two signals (there exist two signals somewhere representing the scale, *i.e.* their distance is 1).

The aim is now to extract an infinite sequence of signed bits (plus a \bullet) encoding the real. Each *query* for a signed bit is started by receiving a *get* signal. The integral part is treated very simply: the first signal encountered get transformed into $\bar{1}_R$, 0_R or 1_R . The dot is treated similarly. The integral part is not addressed anymore in this paper.

The two signals to represent ε are *zero* and *e*. More signals are present: *one*, *half*, *half* and *one* at position respectively -1 , $-\frac{1}{2}$, $\frac{1}{2}$ and 1 (position 0 is the one of zero). Signal *e* is between the two one signals, it might be

superposed with half or zero, in such a case a different meta-signal is used (these special cases are omitted).

Signed bits are extracted one after the other and companion signals are prepared for next extraction: one, half, half and one are scaled by one half and e is translated in order to correspond to the remainder. The structure on Fig. 1 is used for this. On this space-time diagram, the signal amounting for ε , e, is missing. Its presence only superposes signals and collisions.

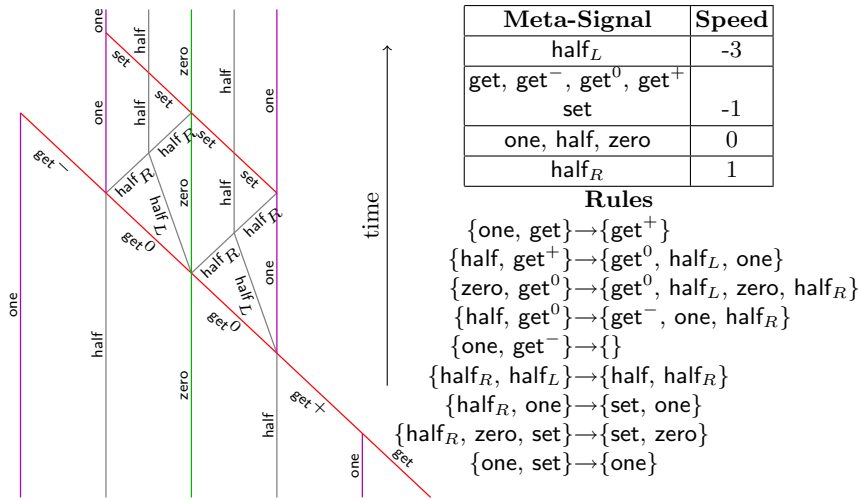


Fig. 1 Empty structure for extracting bits

The Figure 1 presents all the basic ingredients in AGC. A lot of insights on this first construction is provided in order to illustrate how AGC works. In the following, meta-signals are not listed and collisions rules are not listed, all has been implanted and the reader should be able to name signals, deduce rules and prove the correction of the constructions.

On this figure, it can first be proved that starting with one, half, zero, half and one at position $-4x$, $2x$, 0 , $2x$ and $4x$ (x is any positive real number) and get coming from the right. At the end there should only be present one, half, zero, half and one at position $-2x$, x , 0 , x and $2x$. Looking at the speeds and collision rules, everything happens according to the picture. Since the new one signals replace the old half, they are directly correctly positioned. Let's take time 0 at the collision half with get⁺. The collision of zero and get⁰ happens at $(0, 2x)$ since get⁰ has speed -1 . Computing the intersection of two lines, the (right) collision of half_R and half_L happens at $(x, 3x)$.

The three cases of bit extraction are represented on Fig. 2 where the signals handling ε , e is drawn with dashed lines. If $-1 < \varepsilon < -\frac{1}{2^n}$ then the extracted bit is $\bar{1}$ (signal $\bar{1}_L$ exits on the left) and ε is replaced by $\varepsilon + \frac{1}{2^n}$ (e is shifted by the distance from one to half). If $-\frac{1}{2^n} < \varepsilon < \frac{1}{2^n}$ then

the extracted bit is 0 (signal 0_L) and ε is kept (e is not shifted). Except for the geometric construction, the last case is symmetric to the first one. Arbitrarily, if $-\frac{1}{2} < x < 0$, then the extracted bit is 0 although it could have been $\bar{1}$ (similarly for $0 < x < \frac{1}{2}$).

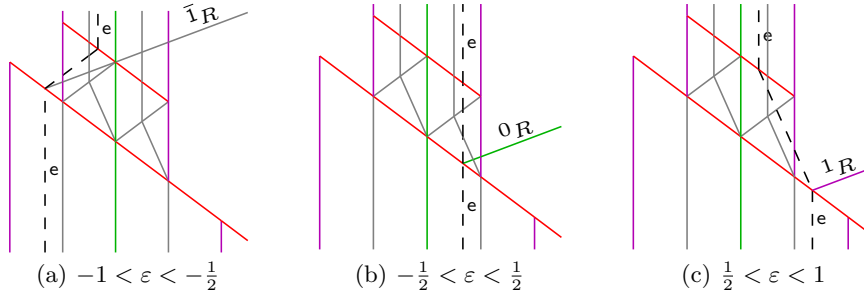


Fig. 2 Extracting a single signed bit.

For bit extraction, *get* comes from the right and is subsequently transformed into *get*⁺ (on meeting the first one) then *get*⁰ (on the first half) then *get*⁻ (on the second half). According to what signal meets *e*, 1, 0 or $\bar{1}$ is send back. If *e* is between the two half (Fig. 2(b)), it remains in position which is correct. If *e* is on the left between one and half (Fig. 2(a)), then it is set parallel to half_R (they have the same speed). It get set into position by *set*, since *set* and *get*⁻ are parallel, the displacement of *e* is exactly the same as the one of half_R from the initial half to the zero where *set* is generated. If *e* is on the right between half and one (Fig. 2(c)), the construction is basically the same.

All the meta-signals and rules are not detailed in the paper. The corrections of the various constructions rely on simple geometry. All have been checked (and implemented in Java to generate the pictures).

3.2 Computable Analysis

A type-2 Turing machine is nothing but a regular TM that never halts together with two special purpose tapes: one initially totally filled — entry — and one for writing once — result. On the entry tape, the head cannot change the read symbol nor move left (*i.e.* back to the start). On the result tape, the head either rewrites the blank symbol or moves right. In the transition table, transitions that read on the entry can be clearly identified. Reading is replaced by sending a *get* signal to the mixed structure and waiting for the signed bit. The same is done with writing, in this case, signals encoding signed bits are sent on the side but no acknowledge/returned value is expected (in this subsection).

Implementing a TM in AGC has already been done in Durand-Lose [2009a]. Read and write operations are carried out by signals sent on the side. Reading sent a *get* to the left and wait for an answer. Since the T2-TM is running for infinitely many iterations, the tape might be extended infinitely, car is taken that the space used for the tape remains bounded. Since space is continuous, it is pretty easy to put less and less space between signals amounting for the cells of the tape. Indeed, a geometric decrease is used so that each signal is two third of the distance from the last than the distance from the last to the previous one. This is illustrated on Fig. 3 with the first iterations of a TM and its implementation.

Figure 1 provides the definition of a type-2 Turing machine that is used to exemplify the construction. The *regular* transitions are as usual. The other two (reading a bit of the entry or writing a bit of the result) are done without moving the head nor changing the value under the head. The table reads as follows: $q_2!0$ means that the next state is q_2 and 0 is written on the result tape and $?1:q_i \dots ?1:q_2$ means that a symbol should be queried if it is $\bar{1}$ then the next state is $q_i \dots$ if 1 is read then the next state is q_2 .

Table 1 Transition table of the type-2 Turing machine.

state \ symbol	$\bar{1}$	0	1	#
q_i	$q_i,0,\rightarrow$	$q_i,\bar{1},\rightarrow$	$q_i,1,\rightarrow$	$?1:q_1 ?0:q_2 ?1:q_3$
q_1	-	-	-	$q_i!\bar{1}$
q_2	-	-	-	$q_i!0$
q_3	-	-	-	$q_i!1$

This T2-TM goes right, inverting $\bar{1}$ and 0 and then queries the entry and output the same bit on and on. The simulation of the TM on Fig. 3(b) has three regular transitions then it enters a cycle of read in and write out. On Fig. 3(c), on the left is the structure holding the entry and on the right the simulation of the machine.

While the entry is queried, the state is encoded together with the letter on the tape as indicated on the upper part of Fig. 3(b). This way the position of the head is also indicated by the only such meta-signal. The write out signals are the ones leaving on the right. Each time a symbol is queried or send to the result, the signal encoding the state bounced on the signal directly on the right; this separated transitions which is necessary when symbols are outputted directly one after the other. The first transitions are displayed on Fig. 3(b).

These signals might go accumulating if the head is moving right forever. But in a proper CA-computation, it is not the case: there must be infinitely many readings and each one needs signals to go forth and back from the tape to the mixed structure. Since this distance is bounded away from zero, each reading requires a minimal time. There is no accumulation whatsoever in a proper CA-computation. A bounded space is used by the T2-TM simulation

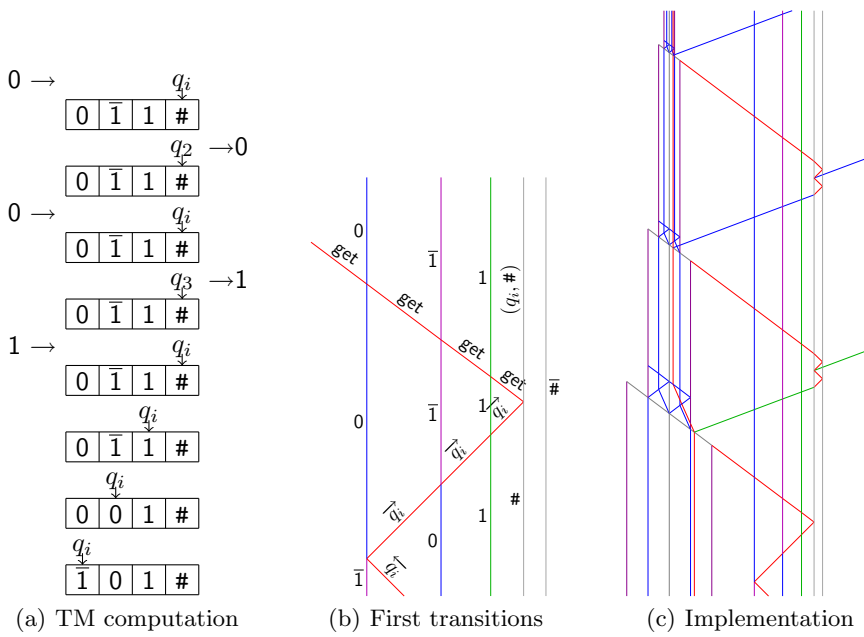


Fig. 3 Simulation of a type-2 Turing machine.

and the encoded real number. There is always finitely many signals in any configuration.

In case of multiple entries (for example for an addition), it is easy to set one after the other on the right and to have a distinct get for each entry.

4 Exact computation

4.1 Folding and extended signal machines

Except for the output signals, the T2-TM simulation and the entry use a bounded portion of space during the whole computation. There exist constructions to fold into bounded space and time a spatially bounded infinite time computation. These can be used to fold the computation without interfering with the output signals (by running through the collision rules and cancelling any action that the folding structure would have on them, they become insensible to the folding). Inside the structure, the computation is scaled down, generating an infinite acceleration (used to emulate the black hole in Durand-Lose [2006, 2009a]). In finite time, the whole infinite output is generated and exits the structure as displayed on Fig. 9.

Inside the folding structure, the Turing machine has an infinite time ahead of it. The structure *preserves ratios* in the understanding that there are infinitely many pieces at different scales that reform the original space-time diagram after rescaling, unbending and translating. Even though the

mixed representation undergoes multiple rescaling, it still generates the same infinite sequence, and the T2-TM the same result.

The problem is then to deal with the output: infinitely many signals are generated in a bounded space, so that accumulation of signals have to be considered. In Durand-Lose [2009b], this was considered and *convoy* were introduced. We provide a construction avoiding that. The shrinking structure is embedded inside another one that will accumulate. (From now these structures are respectively designed as *outer* and *inner*.) The signals outputted from the inner are used to drive the outer structures so that it accumulates at the right place.

4.2 Inner structure

This first structure accelerates (shrinking) until a signal is eventually outputted. The is then frozen and only remains null speed signals (until they are unfrozen).

First the T2-TM is modified in order that writing is freezing the computation. Instead of the bouncing signal, the state and the read symbol are encoded together, waiting for some writing acknowledgement. This functions exactly as in the read case above, except that it is not restarted by a read bit signal but by a *write-done* signal. So that when a bit is outputted, the signals encoding the T2-TM are all parallel. This is described in Fig. 4(a).

The real-number representation and T2-TM simulation are embedded in the inner shrinking structure as displayed on Fig. 4(b). As presented alone on Fig. 4(c), this structure is defined by 5 meta-signals: ${}_i\text{border}_{le}$, ${}_i\text{scale}_{hi}$, ${}_i\text{scale}_{lo}$, ${}_i\text{back}$ and ${}_i\text{border}_{ri}$ and three collision rules. (The prefix ${}_i$ denotes meta-signals from the inner structure). An extra meta-signal, ${}_i\text{start}$, is used to start shrinking from just ${}_i\text{border}_{le}$ and ${}_i\text{border}_{ri}$.

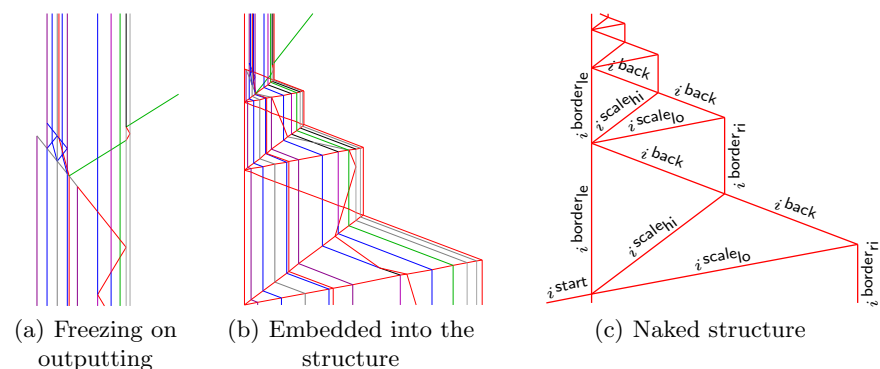


Fig. 4 Inner structure.

In the inner shrinking, the computation is displayed bent in triangles delimited ${}_i\text{scale}_{lo}$, ${}_i\text{scale}_{hi}$ and ${}_i\text{back}$ and unbent (but scaled!) everywhere else. Please note that the computation is going on from one piece to the next and remains between ${}_i\text{border}_{le}$ on the left and a succession of ${}_i\text{border}_{ri}$ and ${}_i\text{back}$ on the right. The bending is done by replacing signals by *bent signals*. Crossing ${}_i\text{scale}_{lo}$, each signal is replaced by a signal such that the directions (*i.e.* speeds) undergo a linear transformation corresponding to factor 1 on ${}_i\text{scale}_{lo}$ (ensuring correct transition) and factor $\frac{1}{2}$ on ${}_i\text{scale}_{hi}$. Crossing ${}_i\text{scale}_{hi}$, a (bent) signal is replaced by its unbent version. This result in applying the linear transformation corresponding to the same directions but exchanged factors. After being unbent, the combined linear transformation is factor $\frac{1}{2}$ on both directions. The scaling is thus achieved.

To stop the inner structure and be able to restart it, it suffices to remove all but ${}_i\text{border}_{le}$ and ${}_i\text{border}_{ri}$. This is done by changing the collision rules so that the signal corresponding to a symbol output is not only unaffected by the structure but also affects it. If it is generated in the unbent zone, it crosses unaffected by the bending, turn ${}_i\text{back}$ to ${}_i\text{back}^F$ and/or ${}_i\text{border}_{ri}$ to ${}_i\text{border}_{ri}^F$. As can be seen on Fig. 5, each case results in only two motionless signals: ${}_i\text{border}_{le}$ on the left and ${}_i\text{border}_{ri}$ or ${}_i\text{border}_{ri}^F$ on the right. Rules are changes so that the output signal is always generated unbent (this explains why it is not the bent version on the right diagram of Fig. 5).

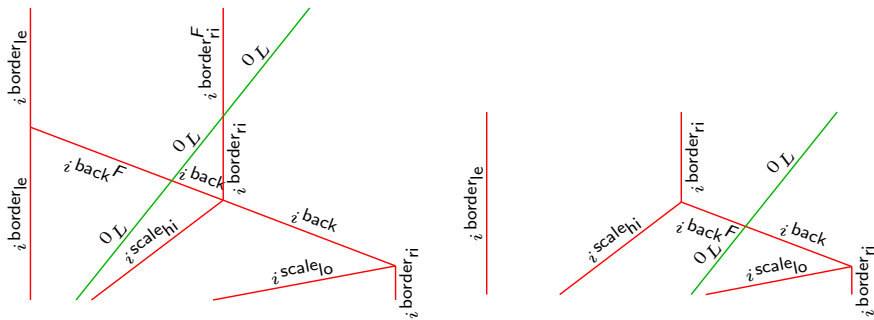


Fig. 5 Stopping the inner structure.

In any case, the simulation of the T2-TM is scaled and represented with unbent signals only. The order in which the inner structure and the T2-TM computation stop does not matter. They are both stopped before the output signal reaches the outer structure.

It is crucial that the computation as well as the inner folding could be resumed: the T2-TM computation is infinite and the shrinking ensure that the next output symbol comes in bounded time.

The awaking is done in two times. One symbol, ${}_o\text{write-done}$ —generated by the outer structure— crosses the configuration from right to left to restart the T2-TM computation. This signal reaches the left end of the outer struc-

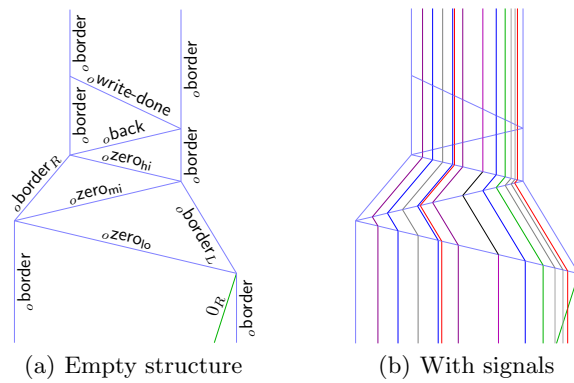


Fig. 7 Outer structure shrinking step for 0.

If 1 is received, all is described on Fig. 8. The triangle plus_{lo} , plus_{hi} and back is used both to scale and to relocate.

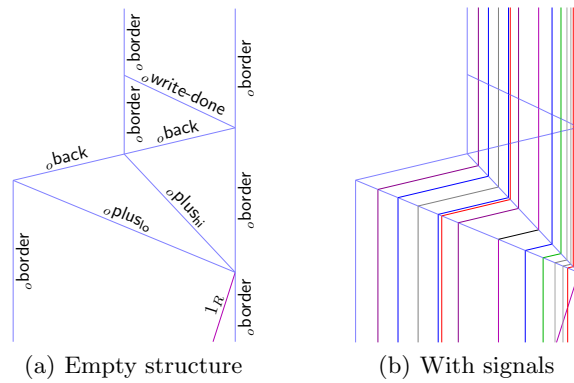


Fig. 8 Outer structure shrinking step for 1.

Finally, Fig. 9 depicts a whole run. It becomes rapidly hard to distinguished the inner part due to the cumulative shrinking effects. Nevertheless, the output signals can be distinguished, the sequence starts with 1, 0, 0, and 1.

The result might be an infinite constant sequences of 1. This would provide an ϵ equals to 1. This is very easy to test and to deal with.

5 Conclusion

The time for the accumulation is not only finite but bounded: the duration can be decomposed in accumulation time for inner structure, outer structure

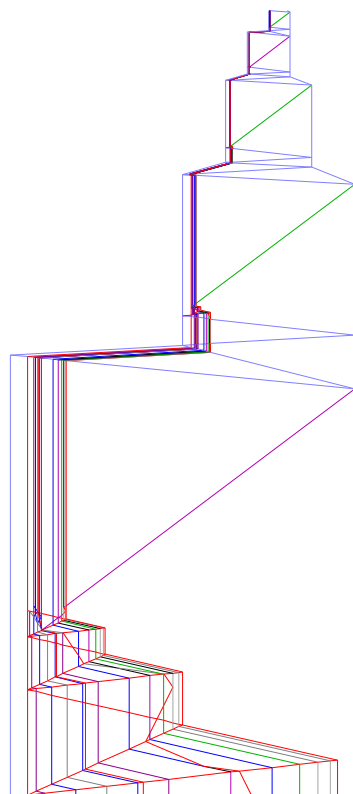


Fig. 9 The whole picture.

and interactions between the two structures. Each one is bounded by a geometrical series.

Theorem 1 *With a proper handling of accumulations, it is possible to compute any function of computable analysis in bounded time. There are always finitely many signals in each configuration.*

There is no hidden oracle. Although speeds may be any real and thus encode information, only a few rational values are used. And apart from mixed representation of ε , the distance between signals are also proportional with rational ratio.

Not only does the mixed representation use finitely many signals, but is also almost directly reusable to do analog computation in the Blum-Shub-Smale understanding (as presented in [Durand-Lose, 2007, 2008]). This leads to consider even more powerful analog computational system since CA and BSS are not comparable, e.g. analytic machines [Chadzelek and Hotz, 1999].

It should also be possible to use higher order accumulation to hyper-compute as is done for the BSS model in Durand-Lose [2009a] and to do some real hyper-computing Ziegler [2007].

References

- L. Blum, M. Shub, and S. Smale. On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines. *Bull. Amer. Math. Soc.*, 21(1):1–46, 1989.
- O. Bournez, M. L. Campagnolo, D. S. Graça, and E. Hainry. Polynomial differential equations compute all real computable functions on computable compact intervals. *Journal of Complexity*, 23(3):317–335, 2007.
- T. Chadzelek and G. Hotz. Analytic machines. *Theoret. Comp. Sci.*, 219(1-2): 151–167, 1999. doi: 10.1016/S0304-3975(98)00287-4.
- J. Durand-Lose. Abstract geometrical computation 1: Embedding black hole computations with rational numbers. *Fund. Inf.*, 74(4):491–510, 2006.
- J. Durand-Lose. Abstract geometrical computation and the linear Blum, Shub and Smale model. In S. B. Cooper, B. Löwe, and A. Sorbi, editors, *Computation and Logic in the Real World, 3rd Conf. Computability in Europe (CiE '07)*, number 4497 in LNCS, pages 238–247. Springer, 2007. doi: 10.1007/978-3-540-73001-9_25.
- J. Durand-Lose. Abstract geometrical computation with accumulations: Beyond the Blum, Shub and Smale model. In A. Beckmann, C. Dimitracopoulos, and B. Löwe, editors, *Logic and Theory of Algorithms, 4th Conf. Computability in Europe (CiE '08) (abstracts and extended abstracts of unpublished papers)*, pages 107–116. University of Athens, 2008.
- J. Durand-Lose. Abstract geometrical computation 3: Black holes for classical and analog computing. *Nat. Comput.*, 8(3):455–572, 2009a. doi: 10.1007/s11047-009-9117-0.
- J. Durand-Lose. Abstract geometrical computation and computable analysis. In J. F. Costa and N. Dershowitz, editors, *International Conference on Unconventional Computation 2009 (UC '09)*, number 5715 in LNCS, pages 158–167. Springer, 2009b. doi: 10.1007/978-3-642-03745-0_20.
- J. Durand-Lose. Abstract geometrical computation 4: small Turing universal signal machines. Submitted to TCS, special issue on Complexity of Simple Programs, 2010.
- G. Etesi and I. Németi. Non-Turing computations via Malament-Hogarth spacetimes. *Int. J. Theor. Phys.*, 41(2):341–370, 2002. gr-qc/0104023.
- A. Grzegorzcyk. On the definitions of computable real continuous functions. *Fund. Math.*, 44:61–77, 1957.
- M. L. Hogarth. Non-Turing computers and non-Turing computability. In *Biennial Meeting of the Philosophy of Science Association*, pages 126–138, 1994.
- K.-I. Ko. *Computational Complexity of Real Functions*. Birkhäuser, 1991.
- A. M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42(2):230–265, 1936.
- K. Weihrauch. *Introduction to computable analysis*. Texts in Theoretical computer science. Springer, Berlin, 2000.
- M. Ziegler. (Short) Survey of real hypercomputation. In S. B. Cooper, B. Löwe, and A. Sorbi, editors, *Computation and Logic in the Real World, 3rd Conf. Computability in Europe, CiE '07*, volume 4497 of LNCS, pages 809–824. Springer, 2007. doi: http://dx.doi.org/10.1007/978-3-540-73001-9_86.