Abstract geometrical computation: beyond the Blum, Shub and Smale model with accumulation

Jérôme Durand-Lose*

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

Abstract. Abstract geometrical computation (AGC) naturally arises as a continuous counterpart of cellular automata. It relies on signals (dimensionless points) traveling and colliding. It can carry out any (discrete) Turing-computation, but since it works with continuous time and space, some analog computing capability exists. In *Abstract Geometrical Computation and the Linear BSS Model* (CiE 2007, LNCS 4497, p. 238-247), it is shown that basic AGC has the same computing capability as the linear BSS.
Using continuous space and time, it is possible to embed accumulations in AGC: infinitely many time steps in a finite duration. This has been used to implement the black-hole model of computation (Fundamenta Informaticae 74(4), p. 491-510). It also makes it possible to multiply two variables, thus simulating the full BSS. Nevertheless a BSS uncomputable function, the square root, can also be implemented, thus proving that the computing capability of AGC with accumulation is strictly beyond the one of BSS.

Key-words. Abstract geometrical computation, Accumulations, Analog computation, BSS model, Signal machine.

1 Introduction

There is no agreed continuous/analog/$\mathbb{R}$ counterpart of the Church-Turing thesis. Relating the numerous models is crucial to understand the differences between their computing capabilities. For example, Bournez *et al.* [BH04,BCGH06] have related Moore's recursion theory on $\mathbb{R}$ [Moo96], computable analysis [Wei00] and the general purpose analog computer. The aim of the present paper is to relate two analog models of computation. One, abstract geometrical computation (AGC) deals with regular and automatic drawing on the Euclidean plane, while the second, the Blum, Shub and Smale model [BCSS98] relies on algebraic computations over $\mathbb{R}$. Bournez [Bou99] has already provided some relations between linear BSS and Piecewise constant derivative systems which also generates Euclidean drawings. In [DL07], AGC without accumulation was proved equivalent

* `http://www.univ-orleans.fr/lifo/Members/Jerome.Durand-Lose`, `Jerome.Durand-Lose@univ-orleans.fr`

to the linear BSS model (*i.e.* BSS restricted to multiplication only by constants) with an unbounded number of variables. In the present paper, the full BSS is related to AGC with accumulation.

Abstract geometrical computation (ACG) arises from the common use in cellular automata (CA) literature of Euclidean settings to explain an observed dynamics or to design a CA for a particular purpose. While CA operate in discrete time over discrete space, Euclidean geometry deals with both continuous time and space. This switch of context is justified by the scaling invariance of CA and relates to our preference and ability for thinking in continuous rather than discrete terms (for example just think how recent and complex is discrete geometry compared to the Euclidean one). Abstract geometrical computation works in a continuous setting: discrete signals/particles are dimensionless points; the local function of CA –computing the next state of a cell according to the states of neighbouring cells– is replaced by collision rules: which signals emerge from a collision of signals. Signals and rules define *signal machines* (SM).

This recent model, restricted to rational numbers, is able to carry out any (discrete) Turing-computation [DL05b]; even with the additional restriction of reversibility and conservativeness [DL06c]. With continuous time, comes Zeno effect (infinitely many discrete steps during a finite duration): not only are accumulations possible, but they can be used to decide recursively enumerable problems by using the black-hole scheme [DL05a,DL06a]. Although accumulations can easily be generated, they can hardly be foreseen [DL06b].

In the Blum, Shub and Smale model (BSS), machines compute over any ring. Roughly speaking, polynomial functions can be performed on variables as well as tests (according to some order) for branching. The dimension of the input is not bounded, a *shift* operator is provided in order to access any variable (finitely many variables are considered since only finitely many are accessed in finite time).

In [DL07], AGC (without accumulation) and linear (multiplying two variables is forbidden) BSS over $\mathbb{R}$ with an unbounded number of variables are proved equivalent. This is done through linear real number unlimited register machines (the arguments of [Nov95] for the equivalence of URM and BSS translate to the linear case).

With a reasonable handling of accumulations, restrictions on multiplication can be lifted thus achieving the full BSS computing capability. They are handled in the following way: there is no second (or higher) order accumulation; only finitely many signals leave any accumulation; and one signal appears where an accumulation takes place. Multiplication is embedded in the same context and the same encoding of real numbers as in [DL07], so that addition, multiplication by constants and test do not have to be implemented again. Only the basis is recalled: a real number is encoded as the distance between two signals.

Multiplication of two real numbers, x and y, is done by producing the binary extension of y and according to $\mathtt{y}_n$ adding or not $x2^n$. With integers, n is increasing from 0, with real numbers, n goes down to $-\infty$ which explains the use of an accumulation. Each iteration divides x by 2, computes the next bit of

y and updates the partial product accordingly. All the values are geometrically decreasing to zero and so are the time and space used by an iteration, so that there is an accumulation. The signal left by the accumulation is located exactly at the value of the product xy.

To prove that AGC with accumulation is, as expected, strictly more powerful than the BSS model, it is explained how to implement the square root. Each iteration only uses addition and multiplication by constants.

Since the reader might be more familiar with BSS than with ACG, more care and illustrations are given to ACG. Section 2 provides all the needed definitions. Section 3 recalls basic encoding and geometric constructions for BSS simulation. Section 4 provides the multiplication between variables. Section 5 explains how to build a signal machine able to compute the square root with an accumulation. Conclusion, remarks and perspectives are gathered in Section 6.

2 Definitions

Abstract geometrical computation. In this model, dimensionless objects are moving on the real axis. When a collision occurs they are replaced according to rules. This is defined by the following machines:

Definition 1 A *signal machine with accumulation* is defined by (M, S, R, μ_a) where M (*meta-signals*) is a finite set, S (*speeds*) a mapping from M to $\mathbb{R}$, R (*collision rules*) a partial mapping from the subsets of M of cardinality at least two into subsets of M (speeds must differ in both domain and range) and μ_a is a meta-signal, the one that comes out of any accumulation.

Each instance of a meta-signal is a *signal*. The mapping S assigns *speeds* to meta-signals. They correspond to the inverse slopes of the segments in space-time diagrams. The *collision rules*, denoted $\rho^- \rightarrow \rho^+$, define what emerge ($\rho^+$) from the collision of two or more signals (ρ^-). Since R is a mapping, signal machines are deterministic. The *extended value set*, V, is the union of M and R plus two symbols: one for void, $\oslash$, and one for accumulation ❋. A *configuration*, c, is a total mapping from $\mathbb{R}$ to V such that the set $\{ x \in \mathbb{R} \mid c(x) \neq \oslash \}$ is finite.

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. A μ_a signal may also start as the result of an accumulation. This corresponds to condition 2 in Def. 2. 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 (condition 3). Condition 4 deals with accumulations, the first line implies that sufficiently close to the accumulation, outside of the light cone, there is nothing but a μ_a signal leaving the accumulation. The second line expresses that there is indeed an accumulation (this is not formalized since it would be too ponderous).

Let S_{min} and S_{max} be the minimal and maximal speeds. The *causal past*, or *light-cone*, arriving at position x and time t, $J^-(x,t)$, is defined by all the

positions that might influence the information at (x,t) through signals, formally:

$$J^{-}(x,t) = \{\ (x',t') \mid x - S_{max}(t{-}t') \leq x' \leq x - S_{min}(t{-}t')\ \}\ .$$

Definition 2 The *space-time diagram* issued from an initial configuration c_0 and lasting for T, is a mapping c from $[0,T]$ to configurations (*i.e.* a mapping from $\mathbb{R} \times [0,T]$ to V) such that, $\forall(x,t) \in \mathbb{R} \times [0,T]$:

1. $\forall t {\in} [0,T]$, $\{\, x \in \mathbb{R} \mid c_t(x) \neq \oslash \,\}$ is finite,
2. if $c_t(x){=}\mu$ 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.:
   - $\forall t' \in (t_i, t_f)$, $c_{t'}(x + S(\mu)(t'-t)) = \mu$,
   - $t_i{=}0$ or ($c_{t_i}(x_i) = \rho^{-}{\to}\rho^{+}$ and $\mu \in \rho^{+}$) or ($c_{t_i}(x_i){=}❉$ and $\mu{=}\mu_a$) where $x_i{=}x + S(\mu)(t_i - t)$,
   - $t_f{=}T$ or ($c_{t_f}(x_f) = \rho^{-}{\to}\rho^{+}$ and $\mu \in \rho^{-}$) where $x_f{=}x + S(\mu)(t_f - t)$;
3. if $c_t(x){=}\rho^{-}{\to}\rho^{+}$ then $\exists\varepsilon$, $0{<}\varepsilon$, $\forall t' {\in} [t{-}\varepsilon, t{+}\varepsilon] \cap [0,T]$, $\forall x' {\in} [x-\varepsilon, x+\varepsilon]$,
   - $(x',t') \neq (x,t) \Rightarrow c_{t'}(x') \in \rho^{-}{\cup}\rho^{+} \cup \{\oslash\}$,
   - $\forall \mu {\in} M$, $c_{t'}(x'){=}\mu \Leftrightarrow$ or $\begin{cases} \mu \in \rho^{-} \text{ and } t' < t \text{ and } x' = x + S(\mu)(t'-t)\,, \\ \mu \in \rho^{+} \text{ and } t < t' \text{ and } x' = x + S(\mu)(t'-t)\,. \end{cases}$
4. if $c_t(x) = ❉$ then
   - $\exists \varepsilon > 0$, $\forall t' {\in} [t{-}\varepsilon, t{+}\varepsilon] \cap [0,T]$, $\forall x' {\in} [x-\varepsilon, x+\varepsilon]$,
     $(x',t') \notin J^{-}(x,t) \Rightarrow$ or $\begin{cases} c_{t'}(x) = \oslash \text{ and } x' \neq x + S(\mu_a)(t'-t) \\ c_{t'}(x) = \mu_a \text{ and } x' = x + S(\mu_a)(t'-t)\,, \end{cases}$
   - $\forall \varepsilon > 0$, there are infinitely many signals in $J^{-}(x,t)$.

On space-time diagrams, the traces of signals are line segments whose directions are defined by $(S(.),1)$ (1 is the temporal coordinate). Collisions correspond to the extremities of these segments. This definition can easily be extended to $T = \infty$. In the space-time diagrams, time increases upwards. To simplify, the same name is used for a signal throughout the computation, in fact, there is a different meta-signal for each speed. As a computing device, the input is the initial configuration and the output is the final configuration.

Blum, Shub and Smale model. (The reader is expected to be more familiar with the BSS model than with AGC, so this part is not very detailed.) A BSS machine is described by a (directed) graph. There are five kind of nodes: input, output (start and end of computation), set a variable to the value computed by a polynomial over neighboring variables, branch according to the value of a polynomial, and shift. There is exactly one input node. Outgoing edges indicate the next instruction. Each node has exactly one outgoing edge, except for output (no outgoing edge) and branch (two, the next instruction is chosen by a test of the form $0 \leq P(x_0, x_1, \ldots x_n)$) nodes. The shift operator allows to access any variable by shifting the context (this is useful since there is no indirect addressing).

3 Basic construction

Real number encoding. A Real number is encoded as the distance from a ba signal to its pairing val signal. Since signal machines are scaleless, two sca signals whose distance amounts for a scale are provided as depicted on Fig. 1. All numbers use the same scale. For the value 0, the superposition of ba and val is encoded as a single signal nul. This value is never considered in the rest of the paper; the reader is invited to check that it can be easily covered.

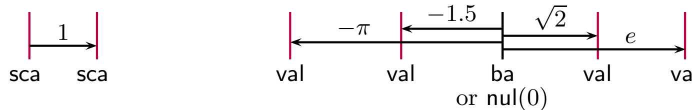
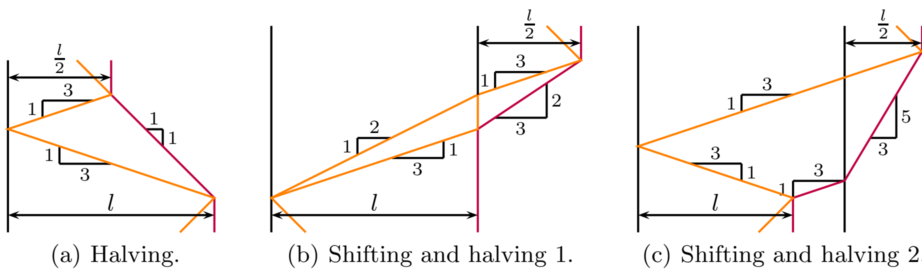


Fig. 1. Encoding: scale and positions of val for values $-\pi$, -1.5, 0, $\sqrt{2}$ and e.

Geometric constructions. The construction of the multiplication, like the addition and the multiplication by a constant, relies on some basic geometric constructions. In each picture, the slopes of the line segments are indicated. It can easily be checked that it is scale invariant and provides the desired effect. Signals with equal speeds result in parallel segments, the key to many geometric properties. Figure 2(a) shows how a distance can be halved. This is done by starting two signals. The first one is going three times slower than the second one. The time the first one crosses half the way, the second one goes one full way and half way back, so that they meet exactly in the middle. Doubling is done the other way round. Figures 2(b) and 2(c) show two ways to halve a distance while shifting it. They also work with two signals emitted that change speed or direction at some point and meet at the desired location. Generating simple shifts is done by modifying only one slope ($\frac{2}{3}$ by $\frac{1}{2}$ for Fig. 2(b) and $\frac{5}{3}$ by 1 for Fig. 2(b)).



(a) Halving. (b) Shifting and halving 1. (c) Shifting and halving 2.

Fig. 2. Basic geometric constructions.

4 Multiplication

4.1 Algorithm

The multiplication of two real numbers x and y uses the (possibly) infinite binary extension of y. This is done in two steps: normalization and then an infinite loop.

The first step starts by considering signs and special cases (*i.e.* multiplication by zero). Then, while y is greater than 2, x is multiplied by 2 and y divided by 2 (so that the product remains constant).

The second step carries out the multiplication. The binary extension $y = \mathtt{y}_0.\mathtt{y}_1\mathtt{y}_2\mathtt{y}_3\ldots$ (the initialization ensures that $0 < y < 2$) is generated bit by bit. The underlying formula is:

$$xy = \sum_{0 \leq i} \mathtt{y}_i \left(\frac{x}{2^i}\right) \quad .$$

This is computed iteratively with the updating on Table 1. The two last cases correspond to $\mathtt{y}_n = 1$ and $\mathtt{y}_n = 0$ respectively. It is started with: $p_0 = 0$ (product), $b_0 = 1$ (bit test), $x_0 = x$ and $0 < y_0 = y < 2b_0 = 2$. The following invariants are satisfied:

- $b_n = 2^{-n}$,
- $0 \leq y_n < 2b_n$,
- $x_n = x2^{-n}$, and
- $xy = p_n + \frac{x_n y_n}{b_n}$.

The last invariant is trivially true for $n = 0$ and preserved by the loop. Since $x_n \frac{y_n}{b_n} < 2x_n$ and $x_n = x2^{-n}$, from the last invariant comes that $\lim_{n\to\infty} p_n = xy$.

Table 1. Potentially infinite loop to compute the product.

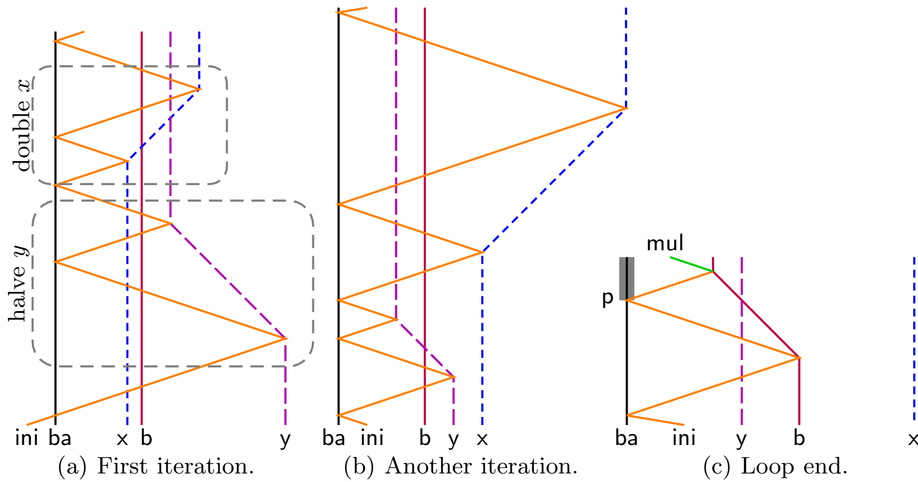
	p_{n+1}	x_{n+1}	y_{n+1}	b_{n+1}
if $y_n = 0$	stop			
else if $b_n < y_n$	$p_n + x_n$	$x_n/2$	$y_n - b_n$	$b_n/2$
else	p_n	$x_n/2$	y_n	$b_n/2$

4.2 Initialisation

With our encoding, detecting whether x or y is zero is trivial. Detecting the signs and computing the sign of the product is also very easy. The following only deals with multiplication of positive values, the other cases are generated using absolute values and symmetry if the product is negative.

The above algorithm should be started with $0 < y_0 < 2b_0 = 2$. So that there is a loop that multiplies x by 2 and divides y by 2 until y is small enough. This is illustrated on Fig. 3 (the first two space-time diagrams go one above the other).

The algorithm is sequential: signals for base (ba), bit testers (b), x (x) and y (y) are fix unless set on movement by some bouncing initialization signals (ini). All distances are from ba and are measured according to the "official" scale (not represented). The b signal stays at position 2 for testing the end of the loop (*i.e.* $y < 2$). The signal ini comes from the left. If it meets y before b, this means that the loop is finished (Fig. 3(c)). Otherwise it has to half y, to double x and to test again (figures 3(a) and 3(b)).



(a) First iteration. (b) Another iteration. (c) Loop end.

Fig. 3. Preparing the data for multiplication.

At the end of initialization (which is always achieved in finite time and finitely many collisions), the b at position 2 is set at position 1 (halved as usual). The signal p amounting for p is generated. Since p is 0 at start, it is set on ba (this corresponds to a different meta-signal). And finally, ini turns to mul that handles the second step of the multiplication. Everything is depicted on Fig. 3(c).

4.3 Main loop

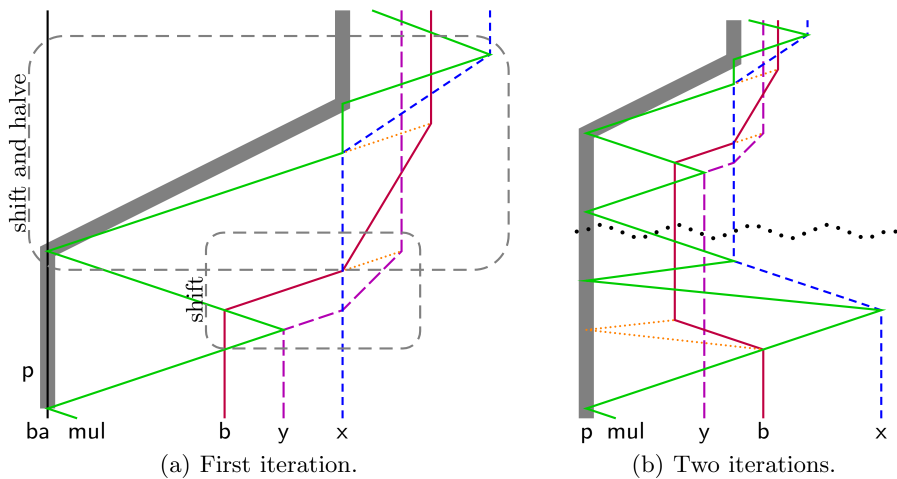
The multiplication follows the (possibly) infinite loop defined in Table 1. Basically, the things to do are additions, subtractions, divisions by 2 and tests. These are easy implementable inside signal machines. But, since the loop is infinite, a *correct* accumulation have to be generated. By correct, it is understood as at the right location and there is indeed an accumulation. The second point is not to be underestimated since, for example, if at each iteration some constant distance would have to be crossed, then there would be an infinite duration process but no accumulation.

The algorithm is driven by a mul signal that bounces between p and other signals. First of all, mul has to test to know which case to consider. This is done

easily: going away from ba, if b is encountered before y this means that $b_n < y_n$ otherwise $y_n < b_n$ (when they are met simultaneously, *i.e.* they are equal, this leads to the end of the loop at the next iteration).

The simpler case is when $y_n < b_n$. There is nothing to do but to halve b_n and x_n, that is halve the distance from b and x to ba. Signals p and y are left untouched. This is done as depicted on the lower part of Fig. 4(b).

The other case $b_n < y_n$ is depicted on Fig. 4(a) and on the upper part of Fig. 4(b). The addition of x_n to p_n is done by moving p to the location of x, meanwhile x is moved on the right by half the distance it has from p. The signal b is also moved on the right, at a distance from the new p that is half the previous distance. The signal y is moved to the right, at a distance from the new p that is equal to its original distance from b.



(a) First iteration. (b) Two iterations.

Fig. 4. Multiplication.

To ensure accumulation, all lengths are geometrically scaled down and all the computations take place by p and are shifted according to the moves of p. This can be seen on Fig. 4. Each iteration leads to halving the distance between ba and both x and b. Since the distance from ba to y is at most twice the distance to b, this ensures that all distances are bounded by some $\frac{M_s}{2^n}$ at the nth iteration. Clearly the duration of an iteration is bounded proportionally to the maximum distance between the signals. Thus it is also bounded by some $\frac{M_t}{2^n}$ at the nth iteration. There is no time lag between two consecutive iterations, so that there is indeed an accumulation.

5 Square root

In this section, it is briefly argued why it is possible to compute the square root with an accumulation and how. Let a be any positive real number. The construc-

tion follows the digit by digit approximation algorithm constructing a sequence b_n such that:

$$b_n^2 \leq a < \left(b_n + \frac{1}{2^n}\right)^2 \quad .$$

At each stage it should be tested whether $\left(b_n + \frac{1}{2^{n+1}}\right)^2 \leq a$. If it is the case then $b_{n+1} = b_n + \frac{1}{2^{n+1}}$ otherwise $b_{n+1} = b_n$. Using the sequences: $d_n = a - b_n^2$, $e_n = \frac{b_n}{2^n}$ and $f_n = \frac{1}{4^{n+1}}$, the test corresponds to computing the sign of

$$a - \left(b_n + \frac{1}{2^{n+1}}\right)^2 = a - b_n^2 - \frac{b_n}{2^n} - \frac{1}{4^{n+1}} = d_n - e_n - f_n \quad .$$

The updating of all these sequences is shown on Table 2 (plus another helpful sequence $g_n = \frac{1}{2^{n+1}}$). The only operations used are additions, multiplications by constants and tests. Again the size of the computing part is decreasing geometrically and computation can be done by the signal encoding the value of b_n and shifted with it.

Table 2. Infinite loop to compute the square root.

	b_{n+1}	d_{n+1}	e_{n+1}	f_{n+1}	g_{n+1}
if $d_n - e_n - f_n = 0$	stop				
else if $0 < d_n - e_n - f_n$	$b_n + g_n$	$d_n - e_n - f_n$	$e_n/2 + f_n$	$f_n/4$	$g_n/2$
else	b_n	d_n	$e_n/2$	$f_n/4$	$g_n/2$

The algorithm starts by a pretreatment that finds the initial value for n. This value might be negative (*e.g.* $n = -11$ for $a = 2^{20} + 1$).

6 Conclusion

In the present paper, AGC with accumulation is proved to be strictly more powerful than BSS. This is not very surprising because it is already known to decide in finite time any recursively enumerable problem (in the classical discrete setting). Like square root, a lot of functions should be computable with an accumulation, it would be interesting to identify them. Considering $\sqrt{2}$, an accumulation point of a rational signal machine can be irrational.

If the computation is stopped before the accumulation happens, then an approximation is generated. Computable analysis relies on the idea of an infinite approximating sequence both for representing real numbers and for computing (type-2 Turing machine needs an infinite number of iterations to compute a function on real numbers). The next step would be to relate these two models. One problem would be to miniaturize and to ensure the generation of an accumulation. Another one is that computable analysis only provides continuous functions, while in ACG, there is, for example, the sign function which is clearly not continuous. On the other side, Moore's recursion theory allows non continuous functions (even the characteristic function of rational numbers).

There might be many accumulations to simulate BSS, but none is of order two. Another issue is to consider nth order accumulation and connect with infinite Turing machines and ordinals [Ham07].

References

[BCGH06] Olivier Bournez, Manuel L. Campagnolo, Daniel S. Graça, and Emmanuel Hainry. The general purpose analog computer and computable analysis are two equivalent paradigms of analog computation. In J.-Y. Cai, S. B. Cooper, and A. Li, editors, *Theory and Appliacations of Models of Computations (TAMC '06), Beijing, China*, number 3959 in LNCS, pages 631–643. Springer, 2006.

[BCSS98] Lenore Blum, Felipe Cucker, Michael Shub, and Steve Smale. *Complexity and real computation*. Springer, New York, 1998.

[BH04] Olivier Bournez and Emmanuel Hainry. An analog characterization of elementarily computable functions over the real numbers. In J. Diaz, J. Karhumäki, A. Lepisto, and D. T. Sannella, editors, *31st International Colloqiuim on Automata, Languages and Programming (ICALP '04), Turku, Finland*, number 3142 in LNCS, pages 269–280. Springer, Jul 2004.

[Bou99] Olivier Bournez. Some bounds on the computational power of piecewise constant derivative systems. *Theory of Computing Systems*, 32(1):35–67, 1999.

[DL05a] Jérôme Durand-Lose. Abstract geometrical computation for black hole computation (extended abstract). In M. Margenstern, editor, *Machines, Computations, and Universality (MCU '04)*, number 3354 in LNCS, pages 176–187. Springer, 2005.

[DL05b] Jérôme Durand-Lose. Abstract geometrical computation: Turing-computing ability and undecidability. In B. S. Cooper, B. Löwe, and L. Torenvliet, editors, *New Computational Paradigms, 1st Conference on Computability in Europe (CiE '05)*, number 3526 in LNCS, pages 106–116. Springer, 2005.

[DL06a] Jérôme Durand-Lose. Abstract geometrical computation 1: embedding black hole computations with rational numbers. *Fundamenta Informaticae*, 74(4):491–510, 2006.

[DL06b] Jérôme Durand-Lose. Forcasting black holes in abstract geometrical computation is highly unpredictable. In J.-Y. Cai, S. B. Cooper, and A. Li, editors, *Theory and Appliacations of Models of Computations (TAMC '06)*, number 3959 in LNCS, pages 644–653. Springer, 2006.

[DL06c] Jérôme Durand-Lose. Reversible conservative rational abstract geometrical computation is turing-universal. In A. Beckmann and J. V. Tucker, editors, *Logical Approaches to Computational Barriers, 2nd Conference on Computability in Europe (CiE '06)*, number 3988 in LNCS, pages 163–172. Springer, 2006.

[DL07] Jérôme Durand-Lose. Abstract geometrical computation and the linear Blum, Shub and Smale model. In S. Cooper, B. Löwe, and A. Sorbi, editors, *Computation and Logic in the Real World 3rd Conference on Computability in Europe (CiE '07)*, number 4497 in LNCS, pages 238–247. Springer, 2007.

[Ham07] Joel David Hamkins. A survey of infinite time turing machines. In J. Durand-Lose and M. Margenstern, editors, *Machines, Computations and Universality (MCA '07)*, number 4664 in LNCS, pages 62–71. Springer, 2007.

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

[Nov95] Erich Novak. The real number model in numerical analysis. *J. Complex.*, 11(1):57–73, 1995.

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