Towards a theory of infinite time Blum-Shub-Smale machines

Peter Koepke and Benjamin Seyfferth

koepke@math.uni-bonn.de
seyfferth@math.uni-bonn.de
Mathematical Institute, University of Bonn
Endenicher Allee 60, 53115 Bonn, Germany

Abstract. We introduce a generalization of Blum-Shub-Smale machines on the standard real numbers $\mathbb{R}$ that is allowed to run for a transfinite ordinal number of steps before terminating. At limit times, register contents are set to the ordinary limit of previous register contents in $\mathbb{R}$. It is shown that each such machine halts before time ω^ω or diverges. We undertake first steps towards estimating the computational strength of these new machines.

1 Introduction

In the spirit of ordinal computability - the study of classical models of computations generalized to transfinite ordinal numbers - we study a variation of the Blum-Shub-Smale (BSS) machine introduced in [1]. In contrast to established models of ordinal computability, such as Hamkins' and Lewis' infinite time Turing machines (ITTMs) [3] and Koepke's ordinal Turing machines (OTMs) [4], these machines employ real numbers in the classical continuum $\mathbb{R}$ as opposed to elements of Baire space ${}^\omega\omega$ or Cantor space ${}^\omega 2$. The topological differences matter as soon as we consider limits (see below). Variations thereof, be it in allowing infinitely many registers or changes in the limit behavior, might very well change the computational strength. In this paper, we aim for the "weakest" possible generalization of BSS machines into ordinal time. We believe that already this restricted model shows interesting properties. One could consider many variations and strengthenings, which will also affect the computational strength.

Our machines have a finite number n of *registers*, each containing a real number. Generalizations to other fields and rings are possible but shall not be of concern to this paper. The computation is steered by a finite program $P \subseteq \omega \times \{f \mid f : \mathbb{R}^n \xrightarrow{\text{rational}} \mathbb{R}^n\} \times \{0,1\} \times \omega \times \omega$, containing commands of the form (i, ϕ, j, k, l), where i is the index of the command at hand, ϕ is a rational map (with rational coefficients), and j tells us if the command represents a *computation node* or a *branch node*. In case $j = 0$, we are at a computation node, the register content $x \in \mathbb{R}^n$ is replaced by $\phi(x)$ and the next command (index $i + 1$) is carried out next. The values of k and l are ignored in this case. Otherwise, $j = 1$ and we are at a branching node. This means that the register content is left unchanged and, depending on whether $\phi(x) > 0$, the next command will be the one with index k. If on the other hand $\phi(x) \leq 0$, command number l is carried out next. We can assume the indices of a given program's commands to form an initial segment of the natural numbers and that no index appears twice. In case a command index is called for which no command in the program exists, the computation halts.

Note 1. As a minor technical detail we would like to note that, as in the original paper [1], we avoid discontinuity points of rational functions by putting decision nodes before each computation or decision node to check whether the denominator of the rational function to be evaluated is 0. If not, we continue as planned, if yes, an infinite loop is entered and the computation diverges.

So far, we have outlined a standard BSS machine with the additional restriction that the rational functions present in computation and branching nodes do not allow for arbitrary real coefficients. We add irrational coefficients in form of *parameters* later on. We now make our machines access the transfinite: In order for the machine to run for infinitely many steps, we have to *define* the register content at limit times. In the established theories of infinite time or ordinal Turing and register machines, often an inferior or superior limit is used for this purpose. Instead, we want to restrict ourselves here

to ordinary limits of sequences of real numbers. This immediately implies that there will be situations where an infinite time BSS machine will, e.g., be properly defined at any finite time but not at the first limit time ω because the register contents do not converge. We can imagine the machine to "crash" in such a case and say that for such a combination of program and input no valid computations exists. In case of converging register contents we also have to come up with a command that is carried out at a limit time. For this we will indeed use the inferior limit, i.e. the command with the least index that was used cofinally often below the limit. Note that we do not introduce a dedicated limit state.

Let us put things together in the following definition.

Definition 1. *Let $n \in \omega$ be a number of registers. Let $k < n$ and let P be a $n+k$-register BSS program. The* infinite time BSS machine computation (ITBM computation) with parameters $p_1, p_2, \ldots, p_k \in \mathbb{R}$ by P *on some input $x \in \mathbb{R}^n$ is defined as the transfinite sequence $(C_t)_{t\in\theta} = (R(t), I(t))_{t\in\theta} \in {}^{\theta}(\mathbb{R}^{n+k} \times \omega)$ where:*

(a) $\theta \in \mathrm{Ord}$ *or* $\theta = \mathrm{Ord}$
(b) $R(0) = (x, p_1, p_2, \ldots, p_k)$ *and* $I(0) = 0$
(c) (computation node) *If $t < \theta$ and $I(t) = i$ let $(i, \phi, 0, k, l)$ be the command of P with index i. Then $R(t+1) = \phi(R(t))$ and $I(t+1) = i + 1$.*
(d) (branching node) *If $t < \theta$ and $I(t) = i$ let $(i, \phi, 1, k, l)$ be the command of P with index i. Then $R(t+1) = R(t)$ and if furthermore $\phi(R(t)) > 0$ then $I(t+1) = k$; if on the other hand $\phi(R(t)) \leq 0$, then $I(t+1) = l$.*
(e) If $t < \theta$ is a limit and $y = lim_{s\to t} R(s)$, then $R(t) = y$ and $I(t) = \liminf_{s\to t} I(s)$.
(f) If $\theta < \mathrm{Ord}$, then θ is a successor ordinal and $I(\theta - 1)$ calls a command index that is not in P (the machine halts*).*

We define ITBM computable functions on the reals:

Definition 2. *A function $f : \mathrm{dom} f \to Y$ where $\mathrm{dom} f, Y \subseteq \mathbb{R}$ is called* ITBM computable in parameters $p_1, p_2, \ldots, p_k$ *if there is an at least $k+1$-register ITBM program P s.t. for every $x \in \mathrm{dom} f$ the computation by P on input $(x, 0, 0, \ldots, 0, p_1, p_2, \ldots, p_k)$ halts and the final register content is of the form $(f(x), \cdot, \cdot, \ldots, \cdot)$. On input $x \notin \mathrm{dom} f$ the computation is required to diverge. We call such a function* ITBM computable *if $k = 0$, i.e. no real parameters are necessary.*

The use of one limit step enables us to compute the classical elementary functions that are defined by power series as illustrated by the following examples. While such functions as the exponential function can be computed in classical recursive analysis, they are not computable in the standard BSS model [2].

Example 1. The exponential function $e : \mathbb{R} \to \mathbb{R}, x \mapsto e^x = \sum_{k=0}^{\omega} \frac{x^k}{k!}$ is ITBM computable: Define a 5-register program that computes the desired function if $|x| < 1$. Later we shall describe the modifications necessary to work for any x.

Algorithm 1

input $R_1 = x$;
set $R_2 := R_3 := R_4 := R_5 := 1$; *(initialize)*
call loop;

loop:
if $R_2 = 0$ *then set* $R_1 := R_5$ *and halt, else continue;*
set $R_4 := \frac{R_4}{R_4+1}$; *(store* $\frac{1}{i}$, *where* i *is the current iteration)*
set $R_3 := R_3 * R_4$; *(store* $\frac{1}{i!}$*)*
set $R_2 := R_2 * R_1$; *(store* x^i*)*
set $R_5 := \frac{\frac{1}{R_2}}{R_3}$; *(store* $\frac{i!}{x^i}$*)*
call loop;

If $|x| < 1$, all register contents converge and R_5 contains the desired output at time ω, which is correctly recognized when $R_2 = 0$. We can adapt the algorithm for $|x| \geq 1$ by adding a case distinction in the beginning and, in case $|x| \geq 1$, save $\frac{1}{x^i}$ in register three. Then register three converges also at limit times. Of course, the command that updates R_5 inside the loop has to be changed accordingly.

Example 2. The sine function $sin : \mathbb{R} \to \mathbb{R}, x \mapsto \sum_{k=0}^{\omega}(-1)^k \frac{x^{2k}}{(2k)!}$ and the cosine function $cos : \mathbb{R} \to \mathbb{R}, x \mapsto \sum_{k=0}^{\omega}(-1)^k \frac{x^{(2k+1)}}{(2k+1)!}$ are ITBM computable. This is proven by the previous example and the fact that $(-1)^k$ can be recovered from $(-\frac{1}{2})^k$ and $(\frac{1}{2})^k$, both of the latter which can be convergently stored and updated in separate registers.

Note 2. Common to this type of examples is that some tricks are necessary to make all registers converge at limits. Auxiliary registers used for scratch work often do not contain converging content. If their content is bounded, however, one can simply divide the register content cofinally often by a fixed number and keep track of how often this division has occurred. Unbounded contents are best stored as their multiplicative inverse. Both approaches can be imagined as pushing the relevant data contained in a register into increasingly later places in their decimal/binary expansion. Compound limits like $\omega \cdot \omega$ are an additional problem, as scratch registers cannot be set to arbitrary values after limit times without sacrificing convergence at the compound limit. However, in every limit stage towards a compound limit the register content will be bounded if treated like above. So, in order to ensure convergence at the compound limit, these bounds themselves need to converge. See the next chapter for an example.

2 Clockable ordinals

Since Hamkins' and Lewis' paper on ITTMs [3], determining those ordinals that appear as halting times on empty inputs has proved to be important for the study of machine models. Since our machines do not halt at limit times, we are interested in machines that run for some limit number α many steps and halt in the next step:

Definition 3. *An ordinal α is called* ITBM clockable *if there is an $n \in \omega$ and an n-register ITBM program that halts on input $0 \in \mathbb{R}^n$ in exactly $\alpha + 1$ many steps.*

The algorithms above prove that ω is clockable, but let us establish this anew with an algorithm that uses only one register.

Lemma 1. *The first limit ordinal ω is ITBM clockable.*

Proof (by algorithm).

Algorithm 2

set $R_1 := 1$*;*
call loop*;*

loop :
if $R_1 = 0$ *then halt else continue;*
set $R_1 := \frac{R_1}{2}$*;*
call loop*;*

We can clock $\omega \cdot n$ by having n loops in separate lines of code, where loop1 calls loop2 and loop i calls loop $i + 1$ instead of the halting command, each time resetting R_1 to 1. Instead, we could also use a separate register to perform a countdown from n. When trying to extend this approach trivially to clock $\omega \cdot \omega$, we run into a problem that is connected to the fact that $\omega \cdot \omega$ is a compound limit, i.e. a limit of limits: At time $\omega \cdot \omega$, R_1 will have cofinally often been set from 0 to 1, so convergence or R_1 fails. While this is easily fixed as seen below, it hints at the limitations imposed by the strict limit rule and why the supremum of ITBM clockable ordinals might be quite low in the ordinals.

Lemma 2. *The first compound limit ordinal $\omega \cdot \omega$ is ITBM clockable.*

Proof (by algorithm).

Algorithm 3

set $R_1 := 1$*;*
set $R_2 := 1$*;*
call loop*;*

loop*;*
if $R_2 = 0$ *then halt else continue;*
if $R_1 = 0$ *then set* $R_2 := \frac{R_2}{2}$ *and set* $R_1 := R_2$ *and continue, else continue;*
set $R_1 := \frac{R_1}{2}$*;*
call loop*;*

In this algorithm, R_1 is halved repeatedly to detect limits. Once a limit time is reached ($R_1 = 0$), R_2 is halved and R_1 is reset not to 1 but to the current value of R_2. Once R_2 hits 0, we have found the compound limit $\omega \cdot \omega$. At every limit, every register content converges.

As before, it is easy to clock finite multiples of the form $\omega \cdot \omega \cdot n$. Also, if we extend the algorithm to use extra registers $R_3, R_4, \ldots, R_n$ in the same manner as we extended the ω-algorithm with R_2, we can in fact clock an finite power ω^n. However, this is as far as we can get, as ω^ω turns out to be the supremum of the ITBM clockable ordinals.

First observe that at any limit time, the register contents are a fixed point for every command that has been carried out cofinally often:

Lemma 3. *Let $(C_t)_{t<\theta}$ be an ITBM computation by some program P and let $\alpha < \theta$ be a limit ordinal. Then the first command in the computation that alters the register contents after time α has not been carried out cofinally often below α.*

Proof. Let $(c, \phi, 0, \cdot, \cdot) \in P$ be a computation node which is called cofinally often below α. Let c be called at some time $\beta > \alpha$, where for all $\alpha < \gamma < \beta$ the register content has not been changed yet, i.e. $R(\alpha) = R(\gamma) = R(\beta)$. Since ϕ may be assumed as locally continuous (cf. Note 1), we get:

$$\begin{aligned} R(\beta + 1) &= \phi(R(\beta)) \\ &= \phi(R(\alpha)) \\ &= \phi(\lim_{t \to \alpha} R(t)) \\ &= \lim_{t \to \alpha \wedge I(t) = c} \phi(R(t) \\ &= \lim_{t \to \alpha \wedge I(t) = c} R(t+1) \\ &= R(\alpha) \end{aligned}$$

So the first computation node that changes the register content after time α cannot have been called cofinally below α.

Theorem 1. *Let P be a program with k computation nodes. Then in any computation $(C_t)_{t<\theta}$ according to P, the register contents stabilize before ω^{k+1}.*

Proof. If $k = 0$ then the computation halts after finitely many steps or diverges since the program contains only finitely many branch nodes: The computation may run through these nodes in finite sequence or in an infinite loop. Every branch node may trigger halting depending on its rational function evaluated on the unchanged input. After finitely many steps, every node in the sequence or loop has been visited once. If the computation didn't halt up to this point, the program will go on forever as the register content is never changed.

So let the hypothesis hold for k. Let P be a program with $(k+1)$-many computation nodes. Suppose the register contents change after ω^{k+2}. The first computation node c responsible for a new register content is not used cofinally often below ω^{k+2}. Let α be the supremum over the times $< \omega^{k+2}$ when c was carried out nontrivially. We can view $(C_t)_{\alpha<t<\theta}$ as the a computation by $P \setminus \{c\}$ on input $R(\alpha)$. Inductively, the register content of this computation stabilizes in ω^{k+1}-many steps. Since $\alpha + \omega^{k+1} < \omega^{k+2}$ this means that the original computation stabilizes before ω^{k+2}. But a computation that stabilizes before a limit can never change its register content again.

Once the register contents have stabilized, an ITBM computation may run for an additional finitely many steps before halting, or diverges. So we get:

Theorem 2. *The supremum of ITBM clockable ordinals is ω^ω.*

The above argument is in fact independent of the input:

Corollary 1. *Every ITBM computation halts before ω^ω-many steps or diverges.*

Corollary 2. *If an ITBM computation diverges, the register content at time ω^ω is not changed any more and can be considered the* pseudo-output *of the diverging computation.*

3 Connections to other models of computation

From a computability perspective, reals provide ample possibilities as codes for complex objects. We can code and decode into reals with our ITBMs by interpreting the binary expansion of a real in the interval $[0, 1]$ as an element of the Cantor space ${}^\omega 2$, i.e. an ω-long sequence of 0's and 1's. The binary expansion of a real is not necessarily unique, so two binary strings representing the same real will appear to our machines as equivalent.

Let us give an algorithm to retrieve the n-th binary digit of an $x \in [0, 1]$.

Algorithm 4

input $R_1 = x$*;*
input $R_2 = n$*;*
set $R_3 = 1$*;* $(= 2^0)$
set $R_4 = 0$*; (the current approximation to x)*
call loop*;*

loop*:*
set $R_2 := R_2 - 1$*; (= #remaining_ iterations)*
if $R_2 := 0$ *then call* lastloop *else continue; if* $R_4 + R_3 > x$ *then continue (do not add* R_3 *to the approximation* R_4*)*
else set $R_4 := R_4 + R_3$ *and continue; (add* R_3 *to the approximation)*
set $R_3 := \frac{R_3}{2}$*;* $(= 2^{\#iterations})$
call loop*;*

lastloop*:*
if $R_4 + R_3 > x$ *then set* $R_1 := 0$ *and halt else set* $R_1 := 1$ *and halt;*

With this algorithm we can also do local changes to the binary bits of x in the fashion of a Turing machine. So finite Turing computations can obviously be implemented on BSS/ITBMs. Also, the halting problem for Turing machines becomes ITBM computable:

Lemma 4. *The classical halting problem is ITBM computable.*

Proof. Since standard Turing computations are ITBM computable, we can generate Turing programs successively. So, in iteration n carry out the first n Turing programs for n many steps on empty input, using a dedicated simulation register. In step n, use only the n-th and later binary digits for the Turing simulation, so at time ω, this register will have converged to 0 (cf. Note 2). Once the algorithm finds that some program (say, the i-th) halts on its input, change the i-th binary digit of an initially zero output register to 1. Since there is a finite time when all halting computations of programs with index $< n$ will have halted, this register converges to the halting problem.

So our machines have more computing strength than Turing machines or classical BSS machines. Also, the type-2 Turing machines of computable analysis (see [5]) can easily be simulated by ITBMs: The output tape of a type-2 Turing machine, when modelled as an ITBM register, converges by definition. Input tapes do not change their content and the finitely many work tapes can be made convergent like in Note 2.

Using above coding, ITBMs can also operate on functions. A continuous function $f : \mathbb{R} \to \mathbb{R}$ may be input to an ITBM as its restriction to the rationals $f^{\mathbb{Q}} = f \restriction \mathbb{Q} : \mathbb{Q} \to \mathbb{R}$ coded into a real $x_f \in [0, 1]$. This requires a fixed enumeration of rational numbers $q : \omega \to \mathbb{Q}$ which may be chosen as ITBM computable and an ITBM computable bijective pairing function $\langle \cdot, \cdot \rangle : \omega \times \omega \to \omega$. Then $f^{\mathbb{Q}}(x) = y$ may be expressed as "the $\langle q^{-1}(x), q^{-1}(y)\rangle$-th binary digit of p_f is 1" (and 0 in the case of $f^{\mathbb{Q}}(x) \neq y$). By computing a convergent sequence of rationals for a given real (nested intervals) we can compute the function value at this real. This takes $\omega \cdot \omega$-many steps: Produce, for all $n < \omega$, the approximations of x up to n binary digits in a similar way to Algorithm 4 as the desired sequence of rationals converging to x. For every such approximation, the function value needs to be decoded from p_f. This takes ω-many steps each and can be done with bounded scratch registers. So, with the tricks described in Note 2, all registers can be arranged to converge at limits.

Example 3. The derivative of a differentiable function is ITBM computable.

Proof. Given a point x, have an ITBM evaluate the differential quotient in x using only $f(x)$ itself and rational approximations to $f(x)$.

An upper bound on strength of ITBMs is given by the strength of ITTMs:

Lemma 5. *Let P be an n-register BSS program. There is an ITTM program Q and a map $f : \mathbb{R}^n \to {}^{\omega}2$ s.t. for every $x \in \mathbb{R}$ the ITTM computation by Q on input $f(x)$ halts and returns the information that either no computation by P on x exists, or that P halts on x with output $y \in {}^{\omega}2$, or that the P diverges on x with pseudo-output $y \in {}^{\omega}2$, where $f^{-1}(y)$ is the final register content of the ITBM computation by P on x.*

Proof. We assume that the ITTM we are working with has a finite number of read-write tapes, which can be accomplished of interlacing these tapes onto the single scratch tape in the definition of ITTMs in [3]. Due to Corollary 2, we know that after time ω^ω also a diverging ITBM computation does not change its register content anymore. Since it is easy for an ITTM to construct a well order of ω of length ω^ω on one tape, it is possible for an ITTM to code the complete ITBM computation $(R_t, I_t)_{t<\omega^\omega}$ up to time ω^ω on the output tape.

So let us begin with defining a map $f' : \mathbb{R} \to {}^{\omega}2$. First, imagine the value $f'(x)$ to be contained in three elements of ${}^{\omega}2$ (i.e. three Turing tapes) where the first tape contains only a 0 or 1 in the first cell to specify the sign of x, the second contains the maximal exponent n s.t. $2^n \leq x$, and the third contains the binary expansion of $\frac{x}{2^n}$, ignoring the decimal (binary) point and normalized in the following way: Binary expansions that end on an infinite trail of 1s are replaced by the unique binary expansion of the same number that ends on a trail of 0s. Then, use an ITTM computable pairing function to interlace these three tapes into one. The function f' can easily be extended to a function $f : \mathbb{R}^n \to {}^{\omega}2$.

It is easy to see that an ITTM is perfectly capable - albeit with an enormous time consumption - of computing addition, subtraction, multiplication and division on elements of ${}^{\omega}2$ and of normalizing the result in the way described above. So, given the input, the program Q will start carrying out the sequence of ITBM commands in P and writing the results (and the program instructions used) one

after another in the respective cells of the output tape. At limit times, the output tape contains all the information of the previous times, so it is easy for the Q to check whether everything converges and compute the limit if one exists. If not, output that there is no valid computation by P on x. If yes, Q will continue its simulation of P up to time ω^ω. At time ω^ω, it can replace the output tape content with $f(y)$ where y is P's output or pseudo-output.

3.1 Open questions

1. Details in the definition of ITBMs might have strong influence on the strength of the resulting machines. Other (stronger) variants may have interesting properties as well.
2. When allowing parameters, it appears that functions higher up in the Baire hierarchy of iterated pointwise limits of continuous functions might be computable by our machines.
3. With previous models of ordinal computations, tie ins to descriptive set theory and constructibility theory proved to be very insightful. We believe this to be applicable to ITBMs as well.

References

1. Lenore Blum, Mike Shub, and Steve Smale. On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines. *Bulletin of the American Mathematical Society*, 21(1):1–46, 1989.
2. Vasco Brattka. The emperors new recursiveness: The epigraph of the exponential funciton in two models of computability. In Masami Ito and Teruo Imaoka, editors, *Words, languages and combinatorics III*, pages 63–72. World Scientific Publishing.
3. Joel D. Hamkins and Andy Lewis. Infinite time Turing machines. *The Journal of Symbolic Logic*, 65(2):567–604, 2000.
4. Peter Koepke. Turing computations on ordinals. *The Bulletin of Symbolic Logic*, 11:377–397, 2005.
5. Klaus Weihrauch. *Computable analysis. An Introduction.* Springer-Verlag, Berlin, Heidelberg, 2000.