Resetting Infinite Time Blum-Shub-Smale-Machines

Merlin Carl[1] and Lorenzo Galeotti[2]

[1] Europa-Universität Flensburg, 24943 Flensburg, Germany
[2] Amsterdam University College, Postbus 94160, 1090 GD Amsterdam, The Netherlands

Abstract. In this paper, we study strengthenings of Infinite Times Blum-Shub-Smale-Machines (ITBMs) that were proposed by Seyfferth in [14] and Welch in [15] obtained by modifying the behaviour of the machines at limit stages. In particular, we study Strong Infinite Times Blum-Shub-Smale-Machines (SITBMs), a variation of ITBMs where lim is substituted by lim inf in computing the content of registers at limit steps. We will provide lower bounds to the computational strength of such machines. Then, we will study the computational strength of restrictions of SITBMs whose computations have low complexity. We will provide an upper bound to the computational strength of these machines, in doing so we will strenghten a result in [15] and we will give a partial answer to a question posed by Welch in [15].

1 Introduction

In [1] Blum, Shub and Smale introduced register machines which compute over real numbers called Blum-Shub-Smale-Machines (BSSMs). A BSSM is a register machine whose registers contain real numbers and that at each step of the computation can either check the content of the registers and perform a jump based on the result, or apply a rational function to the registers. While being quite powerful, BSSMs are still bound to work for a finite amount of time. This limitation is in contrast with the fact that every real number is usually thought as to encode an infinite amount of information. It is therefore natural to ask if a transfinite version of these machines is possible. An answer to this question was first given by Koepke and Seyfferth who in [14] and [12] introduced the notion of Infinite Time Blum, Shub and Smale machine (ITBM). These machines execute classical BSSM-programs[1] but are capable of running for a transfinite amount of time. More precisely, the behaviour of ITBMs at successor stages is completely analogous to that of a normal BSSM. At limit stage an ITBM computes the content of each register using the limit operation on $\mathbb{R}$, and updates the program counter using inferior limits. Infinite Time Blum-Shub-Smale-Machines were further studied in [9].

[1] With the limitation that every rational polynomial appearing in the program has rational coefficients.

As mentioned in [7], the approach taken in extending BSSMs to ITBMs is analogous to that used by Hamkins and Lewis in [8] and by Koepke and Miller in [9] to introduce Infinite Time Turing machines (ITTMs) and Infinite Time Register machines (ITRMs), respectively. A different approach to the generalisation of BSSMs analogous to that used by Koepke in [10] to introduce Ordinal Turing Machines (OTM) was taken by the second author in [7, 6] where he introduced the notion of Surreal Blum, Shub and Smale machine (SBSSM). A complete account of all these models of computation can be found in [2].

As shown in [16, 9], because of the way in which they compute the content of registers at limit stages, ITBMs have a limited computational power, which is way below that of other notions of transfinite computability such as ITTMs, OTMs, SBSSMs, ITRMs. In this paper we consider strengthenings of the limit rule for ITBMs. We will first consider four natural modifications of the behaviour of ITBMs at limit stages. For each such modification we show that it can either be simulated by ITBMs or by machines that we will call Strong Infinite Time Register machines (SITBMs). In the first part of the paper we will provide a lower bound to the computational power of SITBMs. Then, we will restict our attention to programs whose SITBM-computation is of low complexity. We will show that Π_3-reflection is an upper bound to the computational strength of these low complexity machines, in doing so we will strengthen a result mentioned without proof by Welch in [15] and we will give a partial negative answer to the question asked by Welch of whether Π_3-reflection is an optimal upper bound to the halting times of BSSM-programs whose SITBM-computation uses only rational numbers.

Many of the arguments in this paper are inspired by those in [3] and [4].

2 ITBM Limit Rules

As we mentioned in the introduction, the value of registers of an ITBM at limit stages is computed using Cauchy limits, i.e., the content of each register at limit steps is the limit of the sequence obtained by considering the content of the register at previous stages of the computation. The machine is assumed to diverge if for at least one of the registers this limit does not exist. In this section we will consider modified versions of the limit behaviour of ITBMs.

Let $f : \alpha \to \mathbb{R}$ be an α-sequence on $\mathbb{R}$. We call $\ell \in \mathbb{R}$ a *finite limit* point of f if for all $\beta < \alpha$ and for all positive $\varepsilon \in \mathbb{R}$ there is $\beta < \gamma$ such that $|f(\gamma) - \ell| < \varepsilon$. If for all $x \in \mathbb{R}$ and for all $\beta < \alpha$ there is $\beta < \gamma < \alpha$ such that $x < f(\gamma)$ then we will say that $+\infty$ is an *infinite limit* point of f. Similarly for $-\infty$. If for all $x \in \mathbb{R}$ there is $\beta < \alpha$ such that for all $\beta < \gamma < \alpha$ we have $x < f(\gamma)$ then we will say that $+\infty$ is an *infinite strong limit* point of f. Similarly, if for all $x \in \mathbb{R}$ there is $\beta < \alpha$ such that for all $\beta < \gamma < \alpha$ we have $x > f(\gamma)$ then we will say that $-\infty$ is an *infinite strong limit* point of f.

We will denote by $\mathrm{LimP}(f) \subset \mathbb{R} \cup \{-\infty, +\infty\}$ the set of finite and infinite limit points of f, and by $\mathrm{sLimP}(f) \subset \mathbb{R} \cup \{-\infty, +\infty\}$ the set of finite limit points

and infinite strong limit points of f. Note that, if f has an infinite strong limit point, then $\mathrm{LimP}(f)$ is a singleton containing either $-\infty$ or $+\infty$.

Remark 1. If f is Cauchy with limit ℓ, then $\mathrm{sLimP}(f) = \mathrm{LimP}(f) = \{\ell\}$. The converse is not true. Indeed, fix $\rho : \omega \times \omega \to \omega$ to be any computable bijection. Let $f_n(m) = \ell + \frac{n}{m}$ for $n > 0$. Each f_n is a countable sequence which converges to ℓ. Then, setting $s(\rho(n, m)) = f_n(m)$ we see that the sequence s is such that $\mathrm{sLimP}(s) = \{\ell\}$ but is not Cauchy.

Remark 2. Note that if α is a limit ordinal, $f : \alpha \to \mathbb{R}$ is an α-sequence, $\gamma < \alpha$ is an ordinal, and g is the sequence $g(\beta) = f(\gamma + \beta)$, then $\mathrm{LimP}(f) = \mathrm{LimP}(g)$, and $\mathrm{sLimP}(f) = \mathrm{sLimP}(g)$.

Let λ be a limit ordinal and R_i be a register. Assume that for each $\alpha < \lambda$ the register R_i had content $R_i(\alpha)$ in the αth step of the computation. We consider ITBMs whose limit rule is modified as follows:

1. *Weak* ITBM (WITBM):

$$R_i(\lambda) = \begin{cases} \ell & \text{if } R_i^\lambda \text{ is Cauchy with limit } \ell; \\ 0 & \text{if } \min(\mathrm{sLimP}(R_i^\lambda)) \text{ is an infinite strong limit.} \end{cases}$$

2. *Strong* ITBM (SITBM):

$$R_i(\lambda) = \begin{cases} \min(\mathrm{LimP}(R_i^\lambda)) & \text{if } \min(\mathrm{sLimP}(R_i^\lambda)) \in \mathbb{R}; \\ 0 & \text{if } \min(\mathrm{sLimP}(R_i^\lambda)) \text{ is an infinite strong limit.} \end{cases}$$

3. *Bounded-strong* ITBM (BSITBM):

$$R_i(\lambda) = \min(\mathrm{sLimP}(R_i^\lambda)) \text{ if } \min(\mathrm{sLimP}(R_i^\lambda)) \in \mathbb{R}.$$

4. *Super-strong* ITBM (SSITBM):

$$R_i(\lambda) = \begin{cases} \min(\mathrm{LimP}(R_i^\lambda)) & \text{if } \mathrm{LimP}(R_i^\lambda) \in \mathbb{R}; \\ 0 & \text{if } \min(\mathrm{LimP}(R_i^\lambda)) \text{ is infinite.} \end{cases}$$

If $\mathrm{LimP}(R_i) = \emptyset$, $\mathrm{sLimP}(R_i) = \emptyset$, or in general if $R_i(\lambda)$ is not defined, we will assume that the machine crashes, i.e., the computation is considered divergent. In each case the rest of the machine is left unchanged. We will also consider what happens when the register contents of an SITBMs are required to belong to a restricted set of real numbers; for $X \subseteq \mathbb{R}$, an X-SITBM works like an SITBM, but the computation is undefined when a state arises in which some register content is not contained in X.

As for ITBMs all the machines we consider run BSSM-programs and work on real numbers. Therefore, given $\Gamma \in \{\mathsf{SITBM}, \mathsf{BSITBM}, \mathsf{SSITBM}\}$, the definitions of *$\Gamma$-computation*, *$\Gamma$-computable function*, *$\Gamma$-computable set*, and *$\Gamma$-clockable ordinal* are exactly the same as the correspondent notions for ITBMs, see, e.g.,

[12, Definition 1, Definition 2, Definition 3]. As noted in [12, Algorithm 4] ITBMs are capable of computing the binary representation of a real number and perform local changes to this infinite binary sequence. The same algorithms work for SITBM, and therefore for BSITBM, and SSITBM. For this reason, in the rest of the paper we will sometimes treat the content of the each register as an infinite binary sequence. Since not all the binary sequences can be represented in this way, whenever we need to treat a register as an infinite binary sequence, we will do so by representing the sequence $t : \omega \to 2$ by the real r whose binary representation is such that for every $n \in \mathbb{N}$ the $(2n+1)$st bit is $t(n)$ and all the bits in even position are 0. In this case we will call t the *binary sequence represented* by r.

Given a set $A \subseteq \omega$ we say that it is *Γ-writable* if there is a BSSM-program whose Γ-computation with no input outputs a real r such that for every n, the nth bit in the binary sequence represented by r is 1 if and only if $n \in A$. Similarly, we say that a countable ordinal α *is Γ-writable* if there is a Γ-writable set A such that $(\alpha, <) \cong (\mathbb{N}, \{(n,m) \,:\, \rho(n,m) \in A\})$.

Let P be a n-register[2] BSSM-program, $r_1, \ldots, r_n \in \mathbb{R}$ be a real numbers, and $(C_\alpha)_{\alpha<\Theta} = (R_1(\alpha) \ldots R_n(\alpha), I(\alpha))_{\alpha<\Theta}$ be the SITBM-computation of P on input $r_1, \ldots, r_n$. For every $\alpha < \Theta$ we will call C_α the *snapshot* of the execution at time α. Given an BSSM-program P and the SITBM-computation $((R_1(\alpha), \ldots, R_n(\alpha), I(\alpha)))_{\alpha<\Theta}$ of P on input $R_1(0), \ldots, R_n(0)$, for every $i \in \{1, \ldots, n\}$ and $\alpha < \Theta$ we will denote by $R_i^\alpha : \alpha \to \mathbb{R}$ the α-sequence such that $R_i^\alpha(\beta) = R_i(\beta)$ for all $\beta < \alpha$. In the rest of the paper we will omit the superscript α when it is clear from the context.

As Lemma 6, Lemma 7, and Lemma 8 show, SSITBMs, BSITBMs, and WITBMs are inessential modifications of ITBMs and SITBMs. For this reason in the rest of the paper we focus on the study of the computational strength of SITBMs. It is worth noticing that a version of SITBMs was briefly considered by Welch in [15], see, §4.

Lemma 3. *Every* ITBM*-computable function is* SITBM*-computable.*

Proof. The claim follows by Remark 1.

Remark 4. Since every ITRM program is essentially a BSSM-program, every ITRM-computable function is SITBM-computable. Moreover, the simulation can be performed in exactly the same number of steps.

By Remark 4 the result of Lemma 3 cannot be reversed.

Lemma 5. *There is a* SITBM*-computable function which is not* ITBM*-computable.*

Proof. It is enough to note that ω^ω is ITRM-computable [5, Lemma 1 & Theorem 7] and therefore SITBM-writable but not ITBM-writable.

[2] Since the number of registers is not of importance in our results, and because of Lemma 9, in the rest of the paper we will just refer to n-registers BSSM-programs as BSSM-programs.

Lemma 6. *A real function f is* ITBM*-computable iff it is* WITBM*-computable.*

Proof. Trivially every ITBM-computable function is WITBM-computable. Now, let f be any ITBM computable field isomorphism between $\mathbb{R}$ and $(0,1)$, e.g., $G : x \mapsto \frac{e^{2x}}{e^{2x}+1}$. Let f be a WITBM computable function and P be a program computing it. One can define a program P' in which every constant c is replaced by $G(c)$ and all the fields operations are replaced with the correspondent operations in $(0,1)$. Moreover, at each step of the computation the program checks if one of the registers is 0 or 1, if so the program sets the register to $G(0)$. Finally the program should compute G^{-1} of the output. It is not hard to see that the program computes f.

Lemma 7. *A function $f : \mathbb{R} \to \mathbb{R}$ is* SITBM*-computable iff it is* BSITBM*-computable.*

Proof. The proof is very similar to that of Lemma 6. Every BSITBM-computable function is SITBM-computable. Now, let f be any ITBM-computable field isomorphism between $\mathbb{R}$ and $(0,1)$, e.g., $G : x \mapsto \frac{e^{2x}}{e^{2x}+1}$. Let f be an SITBM-computable function and P be a program computing it. One can define a program P' in which every constant c is replaced by $G(c)$ and all the fields operations are replaced with the correspondent operations in $(0,1)$. For each register R_i the machine should have two auxiliary registers C_i and D_i. Each time R_i is modified the machine does the following:

If $D_i = 0$ and the new value of R_i is smaller than the old value then set $D_i = 1$ and $C_i = 0$, if $D_i = 0$ and the new value of R_i is bigger or equal to the old value then set $C_i = 1$, if $D_i = 1$ and the new value of R_i is smaller than or equal to the old value then set $C_i = 1$, if $D_i = 1$ and the new value of R_i is bigger than the old value then set $D_i = 0$ and $C_i = 0$[3].

At each step of the computation the machine should check that all the registers of the original program are not 0 or 1. If a register R_i is 0 or 1 and $C_i \neq 0$ then the machine sets R_i to $G(0)$ and continues the computation otherwise the program enters an infinite loop.

Finally, the program should compute G^{-1} of the output. It is not hard to see that the program computes f.

Finally, we note that, once more by shrinking the computation to $(0,1)$ as we did in the proof of Lemma 7, one can show that SSITBMs can be simulated by SITBMs, and therefore the two models compute exactly the same functions.

Lemma 8. *A function $f : \mathbb{R} \to \mathbb{R}$ is* SITBM*-computable iff it is* SSITBM*-computable.*

We end this section by noticing that, coding finitely many real numbers in one register one can easily build a universal SITBM machine.

Lemma 9. *There is a* BSSM*-program U which given a real number r coding a* BSSM*-program P and the values $\overline{r}$ of the input registers of P, executes P on $\overline{r}$.*

[3] This algorithm checks if the content of R_i is due to a proper limit to infinity.

3 Bounding the Computational Power of SITBMs

In this section we will provide a lower bound to the computational strength of SITBMs.

In particular we shall show that SITBMs are stronger than ITRMs. Given $X \subseteq \omega$ we will denote by $\mathcal{O}^X$ the hyperjump of X, see, e.g., [13].

In order to show that SITBMs are stronger than ITRMs we will prove that SITBMs can compute the ω iteration of the hyperjump function and much more. First, we note that as in [9, Proposition 2.8], if a function f is SITBM-computable, then iterations of f along a SITBM-clockable ordinal are also computable. Let $f : \mathbb{R} \to \mathbb{R}$ be a function, Θ be an infinite countable ordinal, and $h : \omega \to \Theta$ be a bijection, we define:

$$\begin{aligned} f_h^0 &= 0, \\ f_h^{\alpha+1} &= f(f_h^\alpha) \text{ for } \alpha + 1 \leq \Theta, \\ f_h^\lambda &= \bigoplus_{\alpha \in \lambda}^h f_h^\alpha \text{ for } \lambda \leq \Theta \text{ limit}, \end{aligned}$$

where $\bigoplus_{\alpha\in\lambda}^h f_h^\alpha$ is the real r such that if $n = \rho(i, h(\alpha))$ then the nth bit of the binary sequence represented by r is the same as the ith bit of the binary representation of f_h^α for all $i \in \mathbb{N}$ and $\alpha < \lambda$, and 0 for $\lambda < \alpha$.

Similarly to [12, Proposition 2.7], one can easily see that SITBMs can compute iterations of SITBM-computable function over an SITBM-writable ordinal.

Lemma 10 (Iteration Lemma). *Let $f : \mathbb{R} \to \mathbb{R}$ be a SITBM-computable function, and Θ be SITBM-writable. Then there is a bijection $h : \omega \to \Theta$ such that f_h^Θ is SITBM-computable.*

Proof (Sketch). We will only sketch the proof and we leave the details to the reader.

Assume that Θ is SITBM-writable. Let h be such that $h : (\mathbb{N}, \{(n, m) : \text{the } \rho(n, m)\text{th bit of the binary sequence represented by } r \text{ is } 1\}) \cong (\Theta, <)$ where r is the SITBM-writable real coding Θ. There is a machine that computes f_h^Θ.

Note that, since Θ is SITBM-writable, essentially by using the classical ITTM algorithm, given a set of natural numbers we can always compute their least upper bound (if it exists) according to the order induced by h. Therefore, we can generate the sequence $h^{-1}(0)h^{-1}(2)\ldots h^{-1}(\omega)\ldots$.

As noted in [9, Proposition 2.8] the main problem in proving this lemma is to show that we can iterate the same program infinitely many times ensuring that no register in the program will diverge because of the iteration[4]. To solve this problem we note that by Lemma 8 it is enough to show that the function is SSITBM-computable. Note that SSITBM-computations are obviously closed

[4] Note that, while the example mentioned in [9, p.42] is not problematic for SITBMs, our machines could still diverge if for example in the infinite iteration one of the registers of the program assumes values $0, 1, -1, 2, -2, 3, -3, \ldots$.

under transfinite iterations since SSITBMs never brake because of a diverging register.

The SSITBM algorithm to compute the function is the following: the machine first writes Θ in one of the registers say R_1. Then, our machine starts computing $f(0)$ and saves the result in the register R_2 by copying the ith bit of $f(0)$ in position $\rho(i, h^{-1}(0))$ of the binary sequence represented by R_2. In general, the machine should compute $f(f_h^\alpha)$ and save the result in R_2 according to h^{-1} as we did for $f(0)$. Checking the correctness of the sketched algorithm is left to the reader.

Now, we can prove that SITBMs are strictly stronger than ITRMs. First note that since ITRMs can compute the hyperjump of a set, by Remark 4, so can SITBMs.

Lemma 11. *Let $X \subseteq \omega$. The hyperjump $\mathcal{O}^X$ in the oracle X is* SITBM*-computable.*

By Lemma 10 we have the following result:

Lemma 12. *Let α be* SITBM*-clockable. Then the αth iteration of the hyperjump is* SITBM*-computable.*

But since by [11, Proposition 12] ITRMs cannot compute the ω-iteration of the hyperjump we have that SITBMs are strictly stronger than ITRMs.

Corollary 13. *There is a subset of natural numbers $A \subseteq \mathbb{N}$ which is* SITBM*-computable but not* ITRM*-computable, and thus $A \notin \mathbf{L}_{\omega^{\mathrm{CK}}}$.*

We end this section by proving a looping criterion for the divergence of SITBMs which is analogous to those for ITTMs [8, Theorem 1.1], ITRMs [5, Theorem 5], and ITBMs [12, Theorem 1].

Lemma 14 (Strong Loop Lemma). *Let P be a* BSSM*-program, $(C_\alpha)_{\alpha<\Theta} = ((R_1(\alpha), \ldots, R_n(\alpha), I(\alpha)))_{\alpha<\Theta}$ be a* SITBM*-computation of P, and let $\gamma < \beta < \Theta$ be such that:*

1. *$(R_1(\gamma), \ldots, R_n(\gamma), I(\gamma)) = (R_1(\beta), \ldots, R_n(\beta), I(\beta))$;*
2. *for all $\gamma \leq \alpha \leq \beta$ we have $I(\beta) \leq I(\alpha)$ and for all $i \in \{1 \ldots n\}$ we have $R_i(\beta) \leq R_i(\alpha)$.*

Then P diverges.

Proof. Without loss of generality we will assume that $n = 1$, a similar proof works for an arbitrary number of registers. Note that since $(R_1(\gamma), I(\gamma)) = (R_1(\beta), I(\beta))$ the machine is in a loop. Let us call $L = ((R_1(\alpha), I(\alpha)))_{\gamma \leq \alpha \leq \beta}$ the *looping block of the computation*. Let δ be the smallest ordinal such that $\gamma + \delta = \beta$, we will call δ the *length* of L.

Claim 1 *If $\alpha = \gamma + \delta \times \nu$ for some $\nu > 0$ is such that $(R_1(\gamma), I(\gamma)) = (R_1(\alpha), I(\alpha))$ then for all $\mu < \delta$ we have $(R_1(\gamma + \mu), I(\gamma + \mu)) = (R_1(\alpha + \mu), I(\alpha + \mu))$, i.e., if $\alpha = \gamma + \delta \times \nu$ is such that $(R_1(\gamma), I(\gamma)) = (R_1(\alpha), I(\alpha))$ then the computation from α to $\alpha + \delta$ is the loop L.*

Proof. We prove the claim by induction on μ. If $\mu = 0$ the claim follows trivially from the hypothesis. If $\mu = \eta + 1$ then by inductive hypothesis $(R_1(\gamma+\eta), I(\gamma+\eta)) = (R_1(\alpha+\eta), I(\alpha+\eta))$. But then $(R_1(\gamma+\eta+1), I(\gamma+\eta+1)) = (R_1(\alpha+\eta+1), I(\alpha+\eta+1))$ follows from the fact that our machines are deterministic. Finally for μ limit the claim follows from the inductive hypothesis and from Remark 2.

Claim 2 *For all $\nu > 0$ we have $(R_1(\gamma), I(\gamma)) = (R_1(\gamma + \delta \times \nu), I(\gamma + \delta \times \nu))$.*

Proof. We prove the claim by induction on ν. If $\nu = 1$ the claim follows from the assumptions. For $\nu = \eta + 1$ then the claim follows from Claim 1. Assume that ν is a limit ordinal. By inductive hypothesis and by Claim 1 the computation from step γ to step $\gamma + \delta \times \nu$ consists of ν-many repetitions of the loop L. In particular this means that the snapshot $(R_1(\gamma), I(\gamma))$ appears cofinally often in the computation up to ν. Therefore $R_1(\gamma) \in \mathrm{sLimP}(R_1^{\gamma+\delta\times\nu})$. Finally, by 2 we have that $I(\gamma) = \liminf_{\alpha<\gamma+\delta\times\nu} I(\gamma) = I(\gamma + \delta \times \nu)$ and $R_1(\gamma) = \min(\mathrm{sLimP}(R_1^{\gamma+\delta\times\nu})) = R_1(\gamma + \delta \times \nu)$ as desired.

Finally, note that Claim 2 implies that the computation of P diverges as desired.

We call a computation $L = ((R_1(\alpha), \ldots, R_n(\alpha), I(\alpha)))_{\alpha<\Theta}$ as the one in Lemma 14 a *strong loop*.

4 Low Complexity Machines

In the rest of the paper we investigate the ordinals which are clockable by SITBMs whose computations are of low complexity. In particular, we strengthen Lemma 15 which was mentioned without proof in [15][5].

Theorem 15 (Welch). *Let β be the first Π_3-reflecting ordinal. Then, for every rational* BSSM*-program P we have that the* SITBM*-computation of P with input 0 either diverges or halts before β.*

Lemma 16. *Let β be Π_3-reflecting, $\Theta > \beta$, $r \in \mathbb{R} \cap \mathbf{L}_\beta$ be a real number, and $(C_\alpha)_{\alpha\in\Theta}$ be an* SITBM*-computation with $R_i(\beta) = r$. Then R_i has value r cofinally often below β, i.e., for all $\gamma < \beta$ there is $\gamma < \alpha < \beta$ such that $R_i(\alpha) = r$. In fact, there are cofinally in β many τ such that $C_\tau = C_\beta$.*

Proof. Note that for every $\tau < \beta$ the following sentences $\phi_i := \forall r' < r \exists \alpha > \tau \forall \gamma > \alpha R_i(\gamma) > r'$, and $\psi_i := \forall r' > r \forall \alpha > \tau \exists \gamma > \alpha R_i(\gamma) < r'$ are Π_3 in $\mathbf{L}_\beta$. Moreover, they are both true in $\mathbf{L}_\beta$ since $R_i(\beta) = r$. Consequently, there are cofinally in β many τ such that both statements hold in $\mathbf{L}_\tau$. However, this

[5] In [15, p.31] Welch mentions that the first Π_3-reflecting ordinal is an upper bound to the computational strength of SITBMs. In a private communication he clarified to the authors that the machines he was referring to are actually machines which only work on rational numbers. The question of whether the first Π_3-reflecting ordinal is an upper bound to the computational strength of SITBMs is still open.

implies that $R_i(\tau) = R_i(\beta)$. Thus, for every $\tau < \beta$ there is $\tau < \alpha < \beta$ such that $R_i(\alpha) = r$.

Now let n be the maximal register index appearing in P. Then the conjunction $\bigwedge_{i \leq n}(\phi_i \wedge \psi_i)$ is still Π_3 and thus holds at cofinally often in β. Thus, the register contents at time β have appeared cofinally often before time β. Modifying P slightly to P' by storing the active program line in an additional register, we see that the same actually holds for the whole configuration.

Theorem 17. *Let β be a Π_3-reflecting ordinal. Then, for every $(\mathbf{L}_\beta \cap \mathbb{R})$-SITBM computation $(C_\alpha)_{\alpha \in \Theta}$ we have that either $\Theta = \mathrm{On}$ or $\Theta < \beta$. Moreover, if $\Theta = \mathrm{On}$ then the machine is in a strong loop.*

Proof. Assume that $\Theta > \beta$. By Lemma 16, we have that the snapshot C_β appears cofinally often before β. Now, we want to show that this means that the program is in a strong loop. Let $\delta < \beta$ be such that $C_\delta = C_\beta$. If there is no strong loop between times δ and β, there are a register index i and an ordinal γ such that $\delta + \gamma < \beta$ and $r := R_i(\delta + \gamma) < R_i(\delta)$. Note, however, that the snapshot of a computation at time $\eta + \gamma$ is determined by the snapshot at time η. Thus, for every $\tau < \beta$ such that $C_\tau = C_\beta$, we have $R_i(\tau+\gamma) = R_i(\delta+\gamma) = r$. Since β is Π_3-reflecting, we have $\tau+\gamma < \beta$ for every $\tau < \beta$. Thus, the content of R_i is equal to r cofinally often before β. Consequently, we have $r < R_i(\beta) = \liminf_{\iota<\beta} R_i(\iota) \leq r$, a contradiction. Thus, the computation is in a strong loop. The claim follows by Lemma 14.

In [15, p.31] Welch asked if the bound of Lemma 15 was optimal. The following lemma shows that this is not the case.

Lemma 18. *The supremum of the SITBM-clockable ordinals is not a Π_3-reflecting ordinal.*

Proof. It is enough to note that the sentence expressing the fact that every SITBM-computation either diverges or stops is Π_3, and can therefore be reflected below any Π_3-reflecting ordinal.

Corollary 19. *Let α be an ordinal. The supremum of the ordinals clockable by an $(\mathbf{L}_\alpha \cap \mathbb{R})$-SITBM is strictly smaller than the first Π_3-reflecting ordinal above α.*

A BSSM-program *slow* if its SITBM-computation $(C_\beta)_{\beta \in \Theta}$ is such that for every $\omega \leq \beta \leq \Theta$ we have $C_\beta \in \mathbf{L}_\beta$.

Corollary 20. *For every α the supremum of the ordinals that are $(\mathbf{L}_\alpha \cap \mathbb{R})$-SITBM-clockable by a slow program is smaller than or equal to the first Π_3-reflecting ordinal.*

Let P be a BSSM-program whose SITBM-computation $(C_\beta)_{\beta \in \mathrm{On}}$ diverges. We will call the *strong looping time* of P the least ordinal α such that there is $\beta < \alpha$ and $(C_\gamma)_{\beta \leq \gamma \leq \alpha}$ is a strong loop. Analogously to what was proved in [5, Theorem 5] for ITRM, one can prove that strong loops characterise divergent SITBM-computation.

Lemma 21. *Every program P whose* SITBM*-computation $(C_\beta)_{\beta\in\mathrm{On}}$ diverges has a strong looping time.*

Proof. Take $\beta = \omega_1$ in Theorem 17[6]. Then, every divergent $(\mathbb{L}_{\omega_1} \cap \mathbb{R}^{\mathbf{L}})$-SITBM-computation must have a strong loop. Finally, it is enough to note that since $\mathbb{L}_{\omega_1} \cap \mathbb{R} = \mathbb{R}^{\mathbf{L}}$ we have that $(\mathbb{L}_{\omega_1} \cap \mathbb{R}^{\mathbf{L}})$-SITBM-computation and SITBM-computations coincide.

The following is an analogue of the observation in [8] that the supremum Σ of the looping times for ITTMs is not admissible.

Lemma 22. *The supremum γ of the strong looping times is not Π_2-reflecting thus in particular not admissible.*

Proof. Suppose otherwise. Let U be the universal program introduced above. Since U does not halt, it will loop. By definition of γ, the first loop of U will be finished exactly at time γ. Thus, there is $\iota < \gamma$ such that the configurations of U at times ι and γ agree and are component-wise (weakly) majorized by all configurations occuring in between. Let ϕ denote the statement "for all $\xi > \iota$, the configuration at time ξ is component-wise greater than or equal to that at time ι"; clearly, ϕ is Π_2.

Now let i be the index of a register used in U, and let r be its content at time ι (and thus at time γ). Now let ψ_i be the statement "for all rational $q > r$ and all τ, there is $\zeta > \tau$ such that, at time ζ, the content of R_i is $< q$", which is again Π_2.

Now, the conjunction $\tilde{\phi}$ of ϕ and all the ψ_i is a Π_2-formula expressing that U strongly loops. Since γ is Π_2-reflecting by assumption, there is $\tau < \gamma$ such that $L_\tau \models \tilde{\phi}$. But then, U already loops at time τ, contradicting the definition of γ.

We end this section by showing that the model of computation obtained by restricting SITBMs to reals in BSSM-programs is strictly weaker than the unrestricted one.

Lemma 23. *Let β be such that there is an* SITBM*-computable code for $\mathbf{L}_\beta$. There is a* BSSM*-program P whose* SITBM*-computation decides the halting problem for $(\mathbf{L}_\beta \cap \mathbb{R})$-*SITBM*s, i.e., given the code c for a* BSSM*-program, P halts with output 0 if the $(\mathbf{L}_\beta \cap \mathbb{R})$-*SITBM*-computation of the program coded by c with input 0 halts, and P halts with output 1 if the $(\mathbf{L}_\beta \cap \mathbb{R})$-*SITBM*-computation of the program coded by c with input 0 diverges. (If c codes a program that does not produce an $(\mathbf{L}_\beta \cap \mathbb{R})$-*SITBM*-computation, we make no statement about the behaviour of P, although it is easily possible to arrange P to, e.g., diverge or halt with output 2 in this case.)*

[6] It is a classical result of set theory that ω_1 is a Π_3-reflecting ordinal. Indeed, via a classical hull construction and an application of the Condensation Lemma one can easily find an $\alpha \in \omega_1$ such that $\mathbf{L}_\alpha \prec \mathbf{L}_{\omega_1}$.

Proof. Let U be the universal SITBM machine of Lemma 9. We will describe the program P and leave the precise implementation to the reader. We start by computing a real number b coding $\mathbf{L}_\beta$.

Given a code c for a BSSM-program P' as input P starts executing U on c. The program P will use the register R to keep track of the snapshots that appeared in the computation. At each step of the computation P first checks if the simulation halted, in which case halts with output 0. If the current snapshot C of the simulation of P' is not an halting snapshot, P will use b to find a natural number n that codes C in the sense of b.

Once P finds the code for the snapshot it will check if the nth bit of binary sequence represented by the real in register R is one, in which case the P halts with output 1. If not then for every $i \in \mathbb{N}$ the program P checks if the ith bit of the binary sequence represented by the register register R is 1 in this case computes the snapshot coded by i and checks that every element of the snapshot is smaller than or equal to the corresponding element in the snapshot coded by n. If this is the case then P sets the $(2n+1)$st bit of R to 1 and continues. If not then machine erases R, set the $(2n+1)$st bit of R to 1 and continues.

One can check that the P does compute the halting problem restricted to $(\mathbf{L}_\beta \cap \mathbb{R})$-ITBMs.

As usual a subset A of real numbers is called SITBM-*semi decidable* if there is a BSSM-program P such that for every $r \in A$ the SITBM-computation of P with input r halts, and for every $r \notin A$ the SITBM-computation of P with input r diverges. Moreover, if P can be chosen in such a way that for every $r \in A$ the $\mathbb{Q}$-SITBM-computation of P with input r halts, and for every $r \notin A$ the $\mathbb{Q}$-SITBM-computation of P with input r diverges. Then we will say that A is *rationally* SITBM-*semi decidable*.

Corollary 24. *Every rationally* SITBM-*semi decidable set is* SITBM-*computable.*

Corollary 25. *There is a* SITBM-*semi decidable set which is not rationally* SITBM-*semi decidable.*

References

[1] Blum, L., Shub, M., Smale, S.: 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**, 1–46 (1989)

[2] Carl, M.: Ordinal Computability. An Introduction to Infinitary Machines. De Gruyter (2019)

[3] Carl, M.: Space and time complexity for Infinite Time Turing Machines (2019), arXiv:1905.06832

[4] Carl, M.: Taming Koepke's zoo II: Register machines (2019), arXiv:1907.09513

[5] Carl, M., Fischbach, T., Koepke, P., Miller, R., Nasfi, M., Weckbecker, G.: The basic theory of infinite time register machines. Archive for Mathematical Logic **49**(2), 249 – 273 (2010)

[6] Galeotti, L.: The theory of generalised real numbers and other topics in logic. Ph.D. thesis, Universität Hamburg (2019)

[7] Galeotti, L.: Surreal blum-shub-smale machines. In: Manea, F., Martin, B., Paulusma, D., Primiero, G. (eds.) Computing with Foresight and Industry. pp. 13–24. Springer International Publishing, Cham (2019)
[8] Hamkins, J.D., Lewis, A.: Infinite time Turing machines. Journal of Symbolic Logic **65**, 567–604 (2000). https://doi.org/10.2307/2586556
[9] Koepke, P., Morozov, A.S.: The computational power of infinite time Blum-Shub-Smale machines. Algebra and Logic **56**(1), 37–62 (2017)
[10] Koepke, P., Seyfferth, B.: Ordinal machines and admissible recursion theory. Annals of Pure and Applied Logic **160**(3), 310 – 318 (2009)
[11] Koepke, P.: Ordinal computability. In: Ambos-Spies, K., Löwe, B., Merkle, W. (eds.) Mathematical Theory and Computational Practice. pp. 280–289. Springer Berlin Heidelberg, Berlin, Heidelberg (2009)
[12] Koepke, P., Seyfferth, B.: Towards a theory of infinite time Blum-Shub-Smale Machines. In: Cooper, S.B., Dawar, A., Löwe, B. (eds.) How the World Computes: Turing Centenary Conference and 8th Conference on Computability in Europe, CiE 2012, Cambridge, UK, June 18-23, 2012. Proceedings. vol. 7318, pp. 405–415. Springer (2012)
[13] Sacks, G.E.: Higher Recursion Theory. Perspectives in Logic, Cambridge University Press (2017). https://doi.org/10.1017/9781316717301
[14] Seyfferth, B.: Three Models of Ordinal Computability. Ph.D. thesis, Rheinische Friedrich-Wilhelms-Universität Bonn (2013)
[15] Welch, P.D.: Turing's legacy: developments from Turing's ideas in logic. Lecture Notes in Logic (42), 493–529 (2014)
[16] Welch, P.D.: Discrete transfinite computation. In: Sommaruga, G., Strahm, T. (eds.) Turing's Revolution: the impact of his ideas about computability. pp. 161–185. Springer Verlag, Basel (2016)