Surreal Blum-Shub-Smale Machines

Lorenzo Galeotti[1,2,3]

[1] Fachbereich Mathematik, Universität Hamburg, Bundesstraße 55, 20146 Hamburg, Germany
[2] Institute for Logic, Language and Computation, Universiteit van Amsterdam, Postbus 94242, 1090 GE Amsterdam, The Netherlands
[3] Amsterdam University College, Postbus 94160, 1090 GD Amsterdam, The Netherlands

Abstract. Blum-Shub-Smale machines are a classical model of computability over the real line. In [9], Koepke and Seyfferth generalised Blum-Shub-Smale machines to a transfinite model of computability by allowing them to run for a transfinite amount of time. The model of Koepke and Seyfferth is asymmetric in the following sense: while their machines can run for a transfinite number of steps, they use real numbers rather than their transfinite analogues. In this paper we will use the surreal numbers in order to define a generalisation of Blum-Shub-Smale machines in which both time and register content are transfinite.

1 Introduction

In 1989 Blum, Shub and Smale introduced a model of computation to study computability over rings; see [1]. Of particular interest for us is the notion of computability that Blum-Shub-Smale machines (BSSM) induce over the real numbers. A BSSM for the real numbers is a register based machine in which each register contains a real number. A program for such a machine is a finite list of commands. Each command can be either a computation or branch command. The execution of a computation command allows the machine to apply a rational function to update the content of the registers. A branch command, on the other hand, leaves the content of the registers unchanged and allows the machine to apply a rational function to some register and execute a jump based on the result of this operation, i.e., to jump to a different point of the code if the result is 0 and to continue the normal execution otherwise.

In [9, 11], Koepke and Seyfferth defined the notion of *infinite time Blum-Shub-Smale machine* that is a generalised version of Blum-Shub-Smale machines which can carry out transfinite computations over the real numbers.

Infinite time Blum-Shub-Smale machines work essentially as standard BSSMs at successor times apart from the fact that, contrary to classical BSSMs, they can only apply rational functions with rational coefficients[1]. At limit stages an

[1] A stronger version of infinite time Blum-Shub-Smale machines could be obtained by allowing infinite time Blum-Shub-Smale machines to use rational functions with real coefficients, but this was not done in [11].

infinite time Blum-Shub-Smale machine computes the content of each register by taking the limit over the real line of the values that the register assumed at previous stages (if this exists); and updates the program counter to the inferior limit of its values at previous stages. The theory of infinite time Blum-Shub-Smale machine was further studied in [7].

Infinite time Blum-Shub-Smale machines provide an asymmetric generalisation of BSSMs. In particular, while infinite time Blum-Shub-Smale machines are allowed to run for arbitrary transfinite time, they are using real numbers, a set that can be very small compared to the running times. It is then natural to ask whether a symmetric notion can be defined.

The first problem in doing so is, as usual in this context, that of finding a suitable structure which one can use in place of the real line in the generalised context. As we will see, the surreal numbers, a very general number system which contains both real and ordinal numbers, will provide a natural framework to develop this generalised theory.

In this paper, we will introduce a generalised version of Blum-Shub-Smale machines based on surreal numbers and we will show some preliminary results of the theory of these machines.

2 Surreal Numbers

The surreal numbers were introduced by Conway in order to give a mathematical definition of the abstract notion of "number". In this section we will present basic results on surreal numbers; see [2, 5] for a detailed introduction. A *surreal number* is a function from an ordinal α to $\{+,-\}$, i.e., a sequence of pluses and minuses of ordinal length. We denote the class of surreal numbers by No. The *length* of a surreal number x (i.e., its domain) is denoted by $\ell(x)$.

For surreal numbers x and y, we define $x < y$ if there exists α such that $x(\beta) = y(\beta)$ for all $\beta < \alpha$, and $x(\alpha) = -$ and either $\alpha = \ell(y)$ or $y(\alpha) = +$, or $\alpha = \ell(x)$ and $y(\alpha) = +$.

In Conway's original construction, every surreal number is generated by filling some gap in the previously generated numbers. The following theorem connects this intuition to the surreal numbers as we have defined them. First, given sets of surreal numbers X and Y, we write $X < Y$ if for all $x \in X$ and $y \in Y$ we have $x < y$.

Theorem 1 (Simplicity theorem). *If L and R are two sets of surreal numbers such that $L < R$, then there is a unique surreal x of minimal length such that $L < \{x\} < R$, denoted by $[\,L|R\,]$. Furthermore, for every $x \in$ No we have $x = [\,L_x|R_x\,]$ for $L_x = \{y \in \text{No}\ ;\ x > y \wedge y \subset x\}$ and $R_x = \{y \in \text{No}\ ;\ x < y \wedge y \subset x\}$.*

Given two subsets L and R of surreal numbers such that $L < R$, we will call the pair (L, R) a *cut*. For any surreal number $x \in$ No we define the *canonical representation* of x as the cut (L_x, R_x).

Using the simplicity theorem Conway defined the field operations $+_{\text{s}}$, $\cdot_{\text{s}}$, $-_{\text{s}}$, and the multiplicative inverse over No and proved that these operations satisfy

the axioms of real closed fields; see [2, 5]. Moreover, Ehrlich proved that No is the universal class real closed field in the sense that every real closed field is isomorphic to a subfield of No; cf., e.g., [3].

3 Generalising Infinite Time BSSMs

The main goal of this paper is to introduce a notion of register machine which generalises infinite time Blum-Shub-Smale machines in order to allow them to work with arbitrary transfinite space. To do so we want to make our machines able to perform transfinite computations over surreal numbers.

A very natural approach to this problem would be to allow infinite time Blum-Shub-Smale machine registers to store surreal numbers keeping the behaviour of the machine the same. This means that at successor stages the machine will still be allowed to either branch or apply a rational function with surreal coefficients to the registers. At limit stages the machine would have then to use limits[2] over No to compute the contents of the registers. Unfortunately this approach does not work: one can show that if a totally ordered field[3] K has cofinality $\kappa > \omega$, then every non-eventually constant sequence of length $<\kappa$ diverges in K. Similarly for surreal numbers we have the following result:

Lemma 2 (Folklore). *For every ordinal α, every non-eventually constant sequence of length α of surreal numbers diverges.*

The previous result tells us that the classical notion of limit is not the right notion in the context of transfinite computability over a field. Note that the phenomenon of diverging sequences is not a special feature of the surreal numbers, but follows from the basic theory of ordered fields. Thus, any generalisation of the theory of BSSMs to a non-archimedean field would need to deal with this issue.

4 Surreal Blum-Shub-Smale machines

A surreal Blum-Shub-Smale machine (SBSSM) is a register machine. Since, as we will see, the formal definition of SBSSMs is quite involved, let us start by giving a brief informal explanation of how they work. There are two different types of registers in our machines: *normal registers* and *Dedekind registers*. Normal registers are just registers that contain surreal numbers; as we will see, the machine can write and read normally from these registers. Dedekind registers on the other hand are a new piece of hardware. Each Dedekind register R can be thought of as to have three different components S^{L}, S^{R}, and R. The components

[2] By this we mean the notion of limit coming from the order topology over No.

[3] A totally ordered field is a field together with a total order $\leq$ such that for all x, y, and z, we have that if $x \leq y$, then $x + z \leq y + z$, and if x and y are positive, then $x \cdot y$ is positive. The cofinality of an ordered field is the least cardinal λ such that there is a sequence of length λ cofinal in the field.

S^{L} and S^{R} called left and right stack of R, respectively, can be thought of as two possibly infinite stacks of surreal numbers. The last component R of the register can be thought as a normal register whose content is automatically updated by the machine to the surreal $[S^{\mathrm{L}}|S^{\mathrm{R}}]$. Note that it could be that $S^{\mathrm{L}} \not< S^{\mathrm{R}}$; in this case we will assume that the machine crashes.

A SBSSM is just a finite set of normal and Dedekind registers. A program for such a machine will be a finite linear sequence of commands. As for BSSMs there are two types of commands:
Computation: the machine can apply a rational function to a normal register or to a Dedekind register and save the result in a register (either Dedekind or normal) or in a stack.
Branch: the machine can check if the content of a normal register or of a Dedekind register is bigger than 0 and perform a jump based on the result.

In each program we should specify two subsets of the set of normal registers; one that will contain the input of the program, and the other that will contain the output of the program.

A surreal Blum-Shub-Smale machine will behave as follows: at successor stages our machine just executes the current command and updates content of stacks, registers, and program counter accordingly. At limit stage α, the program counter is set using $\liminf$ as for infinite time Blum-Shub-Smale machines; the content of each normal register is updated as follows: if the content of the register is eventually constant with value x, then we set the value of the register to x; otherwise we set it to 0. For Dedekind registers we proceed as follows: if from some point on the content of the stacks is constant, we leave the content of the stacks, and therefore the content of the register, unchanged. If the content of the stacks is not eventually constant but from some point $\beta < \alpha$ on there is no computation instruction whose result is saved in the register, then we set the value of each stack to the union of its values from β on, and we set the content of the register accordingly. If none of the previous cases occurs, then we set the content of the register to 0 and empty the stacks.

We are now ready to give a formal definition of surreal Blum-Shub-Smale machine.

Given two polynomials $p, q \in \mathrm{No}[X_0, \ldots, X_n]$, we will call $\frac{p(X_0,\ldots,X_n)}{q(X_0,\ldots,X_n)}$ a *formal polynomial quotient* over No in $n+1$ variables.

Let $n \in \mathbb{N}$ and $F : \mathrm{No}^{n+1} \to \mathrm{No}$ be a partial class function. Then, we say that F is a *rational map* over No if there are polynomials in $n+1$ variables $p, q \in \mathrm{No}[X_0, \ldots, X_n]$ such that $F(s_0, \ldots, s_n) = \frac{p(s_0,\ldots,s_n)}{q(s_0,\ldots,s_n)}$ for each $s_0, \ldots, s_n \in \mathrm{No}$. In this case, we will say that $\frac{p(X_0,\ldots,X_n)}{q(X_0,\ldots,X_n)}$ is a formal polynomial quotient *defining* F.

Denote by $\overrightarrow{X}$ the set of finite tuples of variables of any length. Then, we will denote by $\mathrm{No}(\overrightarrow{X})$ the class of formal polynomials quotients over No in any number of variables. Given a subclass K of $\mathrm{No}(\overrightarrow{X})$ and a partial class function $F : \mathrm{No}^{m+1} \to \mathrm{No}$ with $m \in \mathbb{N}$, we will say that F is in the class K, in symbols $F \in K$, if there is a formal polynomial quotient in K defining F. Finally, given

a subclass K of No we will denote by $K(\overrightarrow{X})$ the the class of *formal polynomial quotients with coefficients* in K.

Definition 3. *Let $\mathfrak{N}$ and $\mathfrak{D}$ be two disjoint sets of natural numbers, I and O be two disjoint subsets of $\mathfrak{N}$, and K be a subclass of* No. *A $(\mathfrak{N}, \mathfrak{D}, I, O, K)$-*SBSSM program *$P$ is a finite sequence $(C_0, \ldots, C_n)$ with $n \in \mathbb{N}$ such that for every $0 \leq m \leq n$ the command C_m is of one of the following types:*

1. Computation $\mathtt{R}_i{:=}f(\mathtt{R}_{j_0}, \ldots, \mathtt{R}_{j_m})$ *were $f : \mathrm{No}^{n+1} \to \mathrm{No}$ is a map in $K(\overrightarrow{X})$ and $i \in (\mathfrak{N} \setminus I) \cup \mathfrak{D}$ and $j_0, \ldots, j_m \in \mathfrak{N} \cup \mathfrak{D}$.*
2. Stack Computation $\mathtt{Push}_d(\mathtt{R}_i, \mathtt{R}_j)$ *were $i \in \mathfrak{D}$, $j \in \mathfrak{N} \cup \mathfrak{D}$ and $d \in \{\mathrm{L}, \mathrm{R}\}$.*
3. Branch *if* $\mathtt{R}_i$ *then* j *were $i \in \mathfrak{N} \cup \mathfrak{D}$ and $j \leq n$.*

The sets $\mathfrak{N}$ and $\mathfrak{D}$ are the sets of normal and Dedekind registers of our program, respectively; and, I and O are the sets of input and output registers, respectively. When the registers are irrelevant for the argument we will omit, $\mathfrak{N}$, $\mathfrak{D}$, I, and O and call P a $K(\overrightarrow{X})$-SBSSM program.

Definition 4. *Let $\mathfrak{N}$ and $\mathfrak{D}$ be two disjoint sets of natural numbers, the sets $I = (i_0, \ldots, i_m)$ and $O = (i_0, \ldots, i_{m'})$ be two disjoint subsets of $\mathfrak{N}$, K be a subclass of* No, *and $P = (C_0, \ldots, C_n)$ be a $(\mathfrak{N}, \mathfrak{D}, I, O, K)$-SBSSM program. Given $x \in \mathrm{No}^{m+1}$ the SBSSM* computation *of P with input x is the transfinite sequence*[4]

$$(R^{\mathrm{N}}(t), S^{\mathrm{L}}(t), S^{\mathrm{R}}(t), \mathrm{PC}(t))_{t \in \theta} \in (\mathrm{No}^{\mathfrak{N}} \times \wp(\mathrm{No})^{\mathfrak{D}} \times \wp(\mathrm{No})^{\mathfrak{D}} \times \omega)^{\theta}$$

where

1. *θ is a successor ordinal or $\theta = \mathrm{On}$;*
2. *$\mathrm{PC}(0) = 0$;*
3. *$R^{\mathrm{N}}(0)(i_j) = x(j)$ if $i_j \in I$ and $R^{\mathrm{N}}(0)(i) = 0$ otherwise;*
4. *for all $i \in \mathfrak{D}$ we have $S^{\mathrm{L}}(0)(i) = S^{\mathrm{R}}(0)(i) = \emptyset$;*
5. *if $\theta = \mathrm{On}$ then for every $t < \theta$ we have $0 \leq \mathrm{PC}(t) \leq n$. If θ is a successor ordinal $\mathrm{PC}(\theta - 1) > n$ and for every $t < \theta - 1$ we have $0 \leq \mathrm{PC}(t) \leq n$;*
6. *for all $t < \theta$ for all $j \in \mathfrak{D}$ we have $S^{\mathrm{L}}(t)(j) < S^{\mathrm{R}}(t)(j)$;*
7. *for every $t < \theta$ if $0 \leq \mathrm{PC}(t) \leq n$ and $C_{\mathrm{PC}(t)} = \mathtt{R}_i{:=}f(\mathtt{R}_{j_0}, \ldots, \mathtt{R}_{j_n})$ then $\mathrm{PC}(t+1) = \mathrm{PC}(t) + 1$ and: $R^{\mathrm{N}}(t+1)(i) = f(c(j_0), \ldots, c(j_n))$ if $i \in \mathfrak{N}$, $(S^{\mathrm{L}}(t+1)(i), S^{\mathrm{R}}(t+1)(i))$ is the canonical representation of $f(c(0), \ldots, c(n))$ if $i \in \mathfrak{D}$, where for every $m < n$ $c(m) := [S^{\mathrm{L}}(t)(j_m) | S^{\mathrm{R}}(t)(j_m)]$ if $j_m \in \mathfrak{D}$, and $c(m) := R_{j_m}$ otherwise.*
8. *for every $t < \theta$ if $0 \leq \mathrm{PC}(t) \leq n$ and $C_{\mathrm{PC}(t)} = \mathtt{Push}_d(\mathtt{R}_i, \mathtt{R}_j)$ then $\mathrm{PC}(t+1) = \mathrm{PC}(t) + 1$ and $S^d(t+1)(i) := R^{\mathrm{N}}(t)(j)$ if $j \in \mathfrak{N}$; $S^d(t+1)(i) := [S^{\mathrm{L}}(t)(j) | S^{\mathrm{R}}(t)(j)]$ if $j \in \mathfrak{D}$. The rest is left unchanged in $t+1$;*
9. *for every $t < \theta$ if $0 \leq \mathrm{PC}(t) \leq n$ and $C_{\mathrm{PC}(t)} =$ *if* $\mathtt{R}_i$ *then* j then: $\mathrm{PC}(t+1) := j$ if $i \in \mathfrak{N}$ and $R^{\mathrm{N}}(t)(i) > 0$; $\mathrm{PC}(t+1) := j$ if $i \in \mathfrak{D}$ and $[S^{\mathrm{L}}(t)(i) | S^{\mathrm{R}}(t)(i)] > 0$; $\mathrm{PC}(t+1) := \mathrm{PC}(t) + 1$ if $i \in \mathfrak{N}$ and $R^{\mathrm{N}}(t)(i) \leq 0$; and, $\mathrm{PC}(t+1) := \mathrm{PC}(t) + 1$ if $i \in \mathfrak{D}$ and $[S^{\mathrm{L}}(t)(i) | S^{\mathrm{R}}(t)(i)] \leq 0$. The rest is left unchanged in $t+1$;*

[4] By abuse of notation we write $\wp(\mathrm{No})$ for the class of subsets of No.

10. *for every $t < \theta$ if t is a limit ordinal then:* $\mathrm{PC}(t) = \liminf_{s<t} \mathrm{PC}(s)$*, for every $i \in \mathfrak{N}$ we let $R^{\mathrm{N}}(t)(i) := R^{\mathrm{N}}(t')(i)$ if there is t' such that $\forall t > t'' > t' R^{\mathrm{N}}(t')(i) = R^{\mathrm{N}}(t'')(i)$; $R^{\mathrm{N}}(t)(i) := 0$ if there is no such a t'.*
For all $i \in \mathfrak{D}$, if there are t'_L and t'_R smaller than t such that for every $t'_L < t''_L < t$ and $t'_R < t''_R < t$ we have $S^{\mathrm{L}}(t''_L)(i) = S^{\mathrm{L}}(t'_L)(i)$ and $S^{\mathrm{R}}(t''_R)(i) = S^{\mathrm{R}}L(t'_R)(i)$ we have $S^{\mathrm{L}}(t)(i) = S^{\mathrm{L}}(t'_L)(i)$ and $S^{\mathrm{R}}(t)(i) = S^{\mathrm{R}}(t'_R)(i)$. Otherwise, let $U_{t,i} := \{t'' < t \mid \forall t'' \leq t' < t (C_{\mathrm{PC}(t')} = \mathtt{R}_j := f(\mathtt{R}_{j_0}, \ldots, \mathtt{R}_{j_n}) \to i \neq j)\}$. Then $S^{\mathrm{L}}(t)(i) = \bigcup_{t' \in U_{t,i}} S^{\mathrm{L}}(t')(i)$ and $S^{\mathrm{R}}(t)(i) = \bigcup_{t' \in U_{t,i}} S^{\mathrm{R}}(t')(i)$.

If θ is a successor ordinal, we say that P halts *on x with output $y := (R^{\mathrm{N}}(\theta - 1)(i))_{i \in O}$ and write $P(x) = y$.*

In the previous definition, for each $\alpha \in \theta$ and $i \in \mathfrak{N}$, $R^{\mathrm{N}}(\alpha)(i)$ is the content of the normal register i at the αth step of the computation; similarly, $S^{\mathrm{L}}(\alpha)(i)$ and $S^{\mathrm{R}}(\alpha)(i)$ are the sets representing the left and the right stack of the Dedekind register i; moreover, $\mathrm{PC}(\alpha)$ is the value of the *program counter*. Items 2, 3, and 4 describe the initialisation of the machine. In particular, the program counter is set to 0, each normal register but the input registers are initialised to 0, the input registers are initialised to x, and each stack is emptied. Item 5 ensures that the program counter assumes correct values and that the computation *stops*. Items 7, 8, and 9 describe the semantics of the instructions according to our previous description. Finally, item 10 describes the behaviour of the machine at limit stages according to the description we gave before.

Definition 5. *Let $n, m \in \mathbb{N}$ and $F : \mathrm{No}^n \to \mathrm{No}^m$ be a (partial) class function over the surreal numbers and K a subclass of* No*. Then we say that F is $K(\overrightarrow{X})$-*SBSSM computable *iff there are $\mathfrak{N}, \mathfrak{D}, I, O \subset \mathbb{N}$ with $|I| = n$, $|O| = m$ and there is a $(\mathfrak{N}, \mathfrak{D}, I, O, K)$-SBSSM program P such that for every n-tuple x of surreal numbers we have that: if $F(x) = y$ then $P(x) = y$, and if $x \notin \mathrm{dom}(F)$ then $P(x)$ does not halt. Moreover, we say that F is* SBSSM computable *if it is $\mathrm{No}(\overrightarrow{X})$-SBSSM computable.*

As show in [7], infinite time Blum-Shub-Smale machines can only compute reals in the ω^ωth level $\mathbf{L}_{\omega^\omega}$ of constructible universe. Since $\mathbb{R}$ is a subfield of No, every constant real function is $\mathbb{R}(\overrightarrow{X})$-SBSSM computable. Therefore, our SBSSM machines are stronger than infinite time Blum-Shub-Smale machines.

Note that the hardware of our machines in principle does not allow a direct access to the sign sequence representing a surreal number, e.g., there is no instruction which allows us to read the αth sign of a surreal in the register i.

Lemma 6. *Let K be a subclass of* No *such that $\{-1, 0, 1\} \subseteq K$. Then, the following functions are $K(\overrightarrow{X})$-SBSSM computable:*

1. *The function Lim that given an ordinal number α returns 1 if α is a limit ordinal and 0 otherwise;*
2. *Gödel's pairing function $\mathfrak{g} : \mathrm{On} \times \mathrm{On} \to \mathrm{On}$;*

3. *The function* $\mathrm{sgn} : \mathrm{No} \times \mathrm{On} \to \{0, 1, 2\}$ *that for every* $\alpha \in \mathrm{On}$ *and* $s \in \mathrm{No}$ *returns* 0 *if the* $1+\alpha th$[5] *sign in the sign expansion of* s *is* $-$, 1 *if the* $1+\alpha th$ *sign in the sign expansion of* s *is* $+$ *and* 2 *if the sign expansion of* s *is shorter than* $1 + \alpha$*;*
4. *the function* $\mathrm{seg} : \mathrm{No} \times \mathrm{On} \to \mathrm{No}$ *that given a surreal* s *and an ordinal* $\alpha \in \mathrm{dom}(s)$ *returns the surreal whose sign sequence is the initial segment of* s *of length* α*.*
5. *The function* $\mathrm{cng} : \mathrm{No} \times \mathrm{On} \times \{0, 1\} \to \mathrm{No}$ *that given a surreal* $s \in \mathrm{No}$*,* $\mathrm{sgn} \in \{0, 1\}$ *and* $\alpha \in \mathrm{On}$ *such that* $\alpha < \mathrm{dom}(s)$ *returns a surreal* $s' \in \mathrm{No}$ *whose sign expansion is obtained by substituting the* $1 + \alpha th$ *sign in the expansion of* s *with* $-$ *if* $\mathrm{sgn} = 0$ *and with* $+$ *if* $\mathrm{sgn} = 1$*;*

Proof. For the first item, the algorithm is illustrated in Alg.4.

For the second item of the lemma, note that there is an algorithm that, given an ordinal γ, computes the $1+\gamma$th pair (α, β) in the ordering given by the Gödel map; see Alg.1. Now, to compute the value of the Gödel map for the pair (α, β) our algorithm can just start generating pairs of ordinals in the order given by the Gödel map using the algorithm in Alg.1 until the pair (α, β) is generated.

For the third item, it is enough to note that there is a program that can go through the surreal tree No using s as a guide. The pseudo algorithm for such a program is illustrated in Alg.2.

For fourth item, note that in Alg.2 at each step α, the register $Curr$ contains the surreal whose sign sequence is the prefix of the sign sequence of s of length α.

Finally for fifth item, note that, by using fourth item of the lemma, one can easily compute s' by using a Dedekind register. The algorithm is illustrated in Alg.3.

By interpreting 0 as $-$ and 1 as $+$, every binary sequence corresponds naturally to a surreal number. Therefore, we can represent the content of a tape of Turing machines, infinite time Turing machines (ITTMs), and ordinal Turing machines (OTMs) as a surreal number. Lemma 6 tells us that we can actually access this representation and modify it.

5 Computational power of surreal Blum-Shub-Smale machines

Now that we introduced a notion of computability over No, we shall compare our new model of computation with classical and transfinite models of computation. In this section, we will assume that the reader is familiar with the basic definitions of classical computability theory, infinite time Turing machines computability theory, and ordinal Turing machines computability theory; see, e.g., [6, 8].

[5] In this sentence $1 + \alpha$ should be read as the ordinal addition so that for $\alpha \geq \omega$ we have $1 + \alpha = \alpha$.

Given a class C and a set X we will denote by $X^{<C}$ the class of functions whose domain is in C and codomain is X. Let α be an ordinal, X be a set, and C be a class. Given a sequence $(w_\beta)_{\beta<\alpha}$ of elements in $X^{<\alpha}$, we define $[w_\beta]_{\beta<\alpha}$ to be the concatenation of the w_βs.

We start by fixing a representation of binary sequences in No. Let $\Delta : \mathrm{No} \to 2^{<\mathrm{On}}$ be such that for all $s \in \mathrm{No}$, $\Delta(s)$ is the binary sequence of length $\mathrm{dom}(s)$ obtained by substituting each $+$ in s by a 1 and each $-$ by a 0.

Definition 7. *Given a partial function $f : 2^{<\mathrm{On}} \to 2^{<\mathrm{On}}$ and a class of rational functions $K(\overrightarrow{X})$ we say that f is $K(\overrightarrow{X})$-SBSSM computable if there is a $K(\overrightarrow{X})$-SBSSM program which computes the surreal function F such that $f = \Delta \circ F \circ \Delta^{-1}$.*

As we will see, if K is a subclass of No containing $\{-1, 0, 1\}$ then $K(\overrightarrow{X})$-SBSSMs are very powerful. In order to show this, we will now begin by proving their capability of simulating all the most important classical models of transfinite computation. Using Lemma 6 it is immediate to see that if K is a subclass of No such that $\{-1, 0, 1\} \subseteq K$, then every function computable by an ordinary Turing machine is $K(\overrightarrow{X})$-SBSSM computable; moreover, the classical halting problem is $K(\overrightarrow{X})$-SBSSM computable.

The following notion was introduced by Hamkins and Lewis in [6] and further studied by several authors; see, e.g., [12]. An ordinal α is *clockable* if there is an ITTM which runs on empty input for exactly α steps. We will denote by λ the supremum of the clockable ordinals.

Theorem 8. *Let K be a subclass of* No *such that $\{-1, 0, 1\} \subseteq K$. Then, every ITTM-computable function is $K(\overrightarrow{X})$-SBSSM computable. Moreover, if $\lambda \in K$, then the halting problem for ITTMs is $K(\overrightarrow{X})$-SBSSM computable.*

Proof. We will assume that our ITTM has only one tape; a similar proof works in the general case. We call a *snapshot* of an execution of an ITTM at time α a tuple $(T(\alpha), I(\alpha), H(\alpha)) \in \{0, 1\}^\omega \times \omega \times \omega$ where $T(\alpha)$ is a function representing the tape content of the ITTM at time α, $I(\alpha)$ is the state of the machine at time α, and $H(\alpha)$ is the position of the head at time α. We know that we can code $T(\alpha)$ as a sign sequence of length ω. Moreover, at the successor stages, by Lemma 6, we can modify this sequence in such a way that the result is a sign sequence in No_ω coding the ITTM tape after the operation is performed. Moreover, we know that there is a bound, λ, to the possible halting times of an ITTM. Therefore, we can code the list of the $T(\alpha)$ in the snapshots of an ITTM as a sequence of pluses and minuses length λ; hence, as a surreal number of the same length. Consider the $K(\overrightarrow{X})$-SBSSM program that uses two Dedekind registers T and S, and two normal registers I and H. The first Dedekind register is used to keep track of the tapes in the snapshots, the second Dedekind register is used to keep track of how many ITTM instructions have been executed, the register I is used to keep track of the current state of the ITTM, and the register H to keep track of the current head position.

At each step α, if S is a successor ordinal, the program first copies the last ω-many bits of T into a normal register R; then, executes the instruction I with head position[6] $(\omega \times S) + H$ on the string sequence of T writing the result in R. Then, the program computes the concatenation s_α of T and R; and pushes the canonical representation of s_α into the stacks of T. Since for all $\beta < \alpha$, the sign sequence of s_β is an initial segment of s_α, T will contain $\bigcup_{\beta\in\alpha} s_\alpha$ at limit stages.

Now, if S is a limit, the program first computes the content of R as the pointwise $\liminf$ of the snapshots in T. Note that this is computable. Indeed, suppose that the program needs to compute the $\liminf$ of the bit in position i; then it can just look sequentially at the values of the snapshots at i and if it finds a 0 at i in the αth snapshot it pushes $\alpha - 1$ into the left stack of a Dedekind register R'. Once the program has looked through all the snapshots, it will compute the $\liminf$ of the cell in position i as 0 if $R' = S$ and as 1 otherwise. Then, the program will set H to 0 and I to the special limit state and continue the normal execution. This ends the first part of the proof.

Now, assume that $\lambda \in K$. Note that the $K(\overrightarrow{X})$-SBSSM program we have just introduced can simulate the ITTM and check after the execution of every ITTM step that $S < \lambda$. If at some point the program simulates λ-many steps of the ITTM, i.e., $S \geq \lambda$, the program will just halt knowing that the ITTM can not halt.

Since, by [11, Lemma 5], ITTMs can decide the halting problem of infinite time Blum-Shub-Smale machines we get:

Corollary 9. *Let K be a subclass of* No *such that $\{-1, 0, 1\} \subseteq K$. Then, every function computable by an infinite time Blum-Shub-Smale machine is $K(\overrightarrow{X})$-SBSSM computable and the halting problem for infinite time Blum-Shub-Smale machine is $K(\overrightarrow{X})$-SBSSM computable.*

Proof. This follows from Theorems 8 and from the fact that ITTM can simulate and decide the halting problem of infinite time Blum-Shub-Smale machines; see [9, Lemma 5].

As shown in [8, Lemma 6.2], every OTM computable real is in the constructible universe $\mathbf{L}$. Therefore, if $\mathbf{V} \neq \mathbf{L}$, we have that the notions of OTM and $\mathbb{R}(\overrightarrow{X})$-SBSSM computability do not coincide. As usual we will denote by ZFC the axioms of set theory with the Axiom of Choice.

Lemma 10. *If ZFC is consistent, so is ZFC+"there is a function that is $\mathbb{R}(\overrightarrow{X})$-SBSSM computable but not OTM computable".*

Proof (Lemma 10). Let $\mathbf{V}[G]$ be the forcing extension of $\mathbf{V}$ obtained by adding a Cohen real r. Then the constant function $F : x \mapsto r$ is $\mathbb{R}(\overrightarrow{X})$-SBSSM computable. But, since from [8, Lemma 6.2] OTMs only compute elements of $\mathbf{L}$, we have that F is not OTMs computable.

[6] Once again the operations in $(\omega \times S) + H$ must be interpreted as ordinal operations.

Theorem 11. *Let K be a subclass of* No *such that* $\{-1, 0, 1\} \subseteq K$*. Then, every OTM computable partial function* $f : 2^{<\mathrm{On}} \to 2^{<\mathrm{On}}$ *is* $K(\overrightarrow{X})$*-SBSSM computable.*

Proof. We will assume that our machine has two tapes, one read-only input tape and an output tape; the general case follows.

Our program will be very similar to the one we used for ITTMs. For this reason, we will mostly focus on the differences.

The main difference is that, while for ITTM we can just save the sequence of tape snapshots, for OTM we cannot simply do that because the tape has class length. The problem can be solved by padding. Given a binary sequence $b := [b_\beta]_{\beta\in\alpha}$ where $b_\beta \in \{-, +\}$ for each $\beta < \alpha$, let b^p be the sequence obtained by concatenating the sequence $[+b_\beta+]_{\beta\in\alpha}$ with the sequence $--$. We call b^p the padding of b. With this operation, we can now save the initial meaningful part of the OTM tape in a register.

The program has four Dedekind registers T, S, H_i, I_i, and two normal registers H and I. As for ITTMs, the Dedekind register T is used to keep track of the tapes in the snapshots; the Dedekind register S is used to keep track of how many OTM instructions have been executed; the register I is used to keep track of the current state of the OTM; and the register H to keep track of the current head position. Note that, since at limit stages the head position and the state of the machine need to be set to the lim inf of their previous contents, we added the Dedekind registers H_i and I_i to keep track of the histories of H and I, respectively.

The registers T, H_i and I_i are really the main difference between this program and the one we used to simulate ITTM. At each stage, T will contain the concatenation of the paddings of the previous configurations of the OTM tape. Note that the sequence $--$ works as a delimiter between one snapshot and the next one. Also, since we cannot save all the OTM tape, each time we will just record the initial segment of the OTM tape of length S, i.e., the maximum portion we could have modified.

If $S := \alpha + 1$, the program first copies the last snapshot in T to a normal register s_α removing the padding. At this point, the program can just simulate one step of OTM and then compute the padding s^p_α of s_α, and push the standard representation of s^p_α in T.

Now, the program will take the content of H_i, and will compute the surreal number h_α whose sign sequence is H_i followed by H minuses and one plus. Then, the program will push the canonical representation of h_α into the stacks of H_i. Similarly for I, the program will take the content of I_i, and will compute the surreal number i_α whose sign sequence is I_i followed by I_i minuses and one plus. Then, the program will push the canonical representation of i_α into the stacks of I_i.

Again, note that, as for ITTMs, at limit stages T, H_i and I_i will contain the concatenation of the padded snapshots of the tape, H and I, respectively.

If S is a limit ordinal, with a bit of overhead due to padding, the program can compute the pointwise lim inf of the tape. It is not hard to see that this

operation is a minor modification of the one used for ITTMs. Note that, in this case, not all the bits will be present in every snapshot; if we want to compute the ith bit of the limit snapshot we will have to start computing the lim inf from the ith snapshot in T. The rest is essentially the same as what we did for ITTM case. Then, the program will compute the content of I using Alg.5; and, using H_i and I_i, it can compute the lim inf of H only considering the stages where I was the current state. Then, the program can proceed exactly as in the successor case.

As we have seen so far, if K is a subclass of No such that $\{-1, 0, 1\} \subseteq K$ then $K(\overrightarrow{X})$-SBSSMs are at least as powerful as OTMs. It turns out that, if $K = \{-1, 0, 1\}$, the two models of computation are actually equivalent; see Theorem 14.

As shown in [4], via representations it is possible to use OTMs to induce a notion of computability over surreal numbers. We will take the same approach here.

Let $\delta_{\mathrm{No}} : 2^{<\mathrm{On}} \to \mathrm{No}$ be the function that maps each surreal number to a binary sequence as follows: $\delta_{\mathrm{No}}(p) = q$ iff p is a binary sequence of length $2 \times \ell(q) + 2$, such that $p = [w_\alpha]_{\alpha \in \ell(q)+1}$ where: $w_\alpha := 00$ if $\alpha \in \mathrm{dom}(q)$ and $q(\alpha) = -$ and, $w_\alpha := 11$ if $\alpha \in \mathrm{dom}(q)$ and $q(\alpha) = +$, and $w_{\ell(q)} := 01$.

To avoid unnecessary complications, in the following we will only deal with unary surreal functions.

Definition 12. *Given a partial function $F : \mathrm{No} \to \mathrm{No}$, we say that F is* OTM computable *if there is an OTM program that computes the function G such that $F = \delta_{\mathrm{No}} \circ G \circ \delta_{\mathrm{No}}^{-1}$.*

Note that the function δ_{No} is essentially[7] an extension to the class of surreal numbers of the function $\delta_{\mathbb{Q}_\kappa}$ introduced in [4]. From this fact, and from the fact that, as shown in [4, Lemma 9 & 10], OTMs are capable of computing surreal operations and convert back and forth from cut representation to sign sequences, it is easy to see that OTMs and $\{-1, 0, 1\}(\overrightarrow{X})$-SBSSM have the same computational strength.

Theorem 13. *Let K be a subclass of OTM computable elements of* No*, i.e., such that for every $s \in K$ the sequence $\delta_{\mathrm{No}}^{-1}(s)$ is computable by an OTM with no input. Then, every $K(\overrightarrow{X})$-SBSSM computable function is OTM computable. In particular, every $\{-1, 0, 1\}(\overrightarrow{X})$-computable function is OTM computable.*

Proof. It is enough to note that rational functions and the behaviour of Dedekind registers can be simulated by an OTM. Moreover, note that our representation

[7] The class function δ_{No} is not literally an extension of $\delta_{\mathbb{Q}_\kappa}$ just because in [4] we assumed $\mathrm{dom}(\delta_{\mathbb{Q}_\kappa}) \subset 2^\kappa$ rather than $\mathrm{dom}(\delta_{\mathbb{Q}_\kappa}) \subset 2^{<\kappa}$. This does not make much of a difference in our algorithms as far as we have a marker for the end of the code of the sign sequence (i.e., the last two bits in the definition of δ_{No}).

function for surreals is the extension of $\delta_{\mathbb{Q}_\kappa}$ introduced in [4] to the all class of surreals.

Using the algorithms that we have introduced in [4, Lemma 9] and [4, Lemma 10] to compute the field operations over $\mathbb{Q}_\kappa$, one can see that rational functions with computable coefficients can be computed. Moreover, again by using the algorithms in [4, Lemma 10] that convert $\delta_{\mathbb{Q}_\kappa}$ into $\delta_{Cut_{\mathbb{Q}_\kappa}}$ and vice versa, it is easy to see that each Dedekind register can be simulated. Therefore, since by assumptions the (codes for) the elements of K are computable, every $K(\overrightarrow{X})$-SBSSM computable function is OTM computable.

So, $\{-1, 0, 1\}(\overrightarrow{X})$-SBSSM have the same computational power as OTMs. Note that, if we enlarge the class of rational functions our machine is allowed to use, we obtain progressively stronger models of computations. Moreover, it is easy to see that the class of coefficients allowed in the class of rational functions acts as a set of parameters on the OTMs side.

Theorem 14. *Let K be a subclass of* No. *Then a partial function $F : \mathrm{No} \to \mathrm{No}$ is $K(\overrightarrow{X})$-SBSSM computable iff it is computable by an OTM with parameters in K.*

Proof. For the right to left direction, note that each element of K is $K(\overrightarrow{X})$-SBSSM computable. Therefore, by using the algorithm in the proof of Theorem 11 we have that, if F is OTM computable with parameters in K, then it is $K(\overrightarrow{X})$-SBSSM computable. For the other direction, note that, as we have just showed in Theorem 13, surreal operations and operations of SBSSM which involve computable coefficients are computable. Therefore, it is enough to input to the OTM the coefficients of the rational functions involved in the $K(\overrightarrow{X})$-SBSSM algorithm in order to make the OTM capable of computing F. Therefore, $F : \mathrm{No} \to \mathrm{No}$ will be OTM computable with parameters in K as desired.

Corollary 15. *Every partial function $F : \mathrm{No} \to \mathrm{No}$ which is a set is* $\mathrm{No}(\overrightarrow{X})$*-SBSSM computable.*

Proof. Note that F is a sequence of pairs of surreal numbers $\{(s^\ell_\beta, s^r_\beta) \mid \beta \in \alpha\}$ for some $\alpha \in \mathrm{On}$. Consider the function $G := \{(\Delta(s^\ell_\beta), \Delta(s^r_\beta)) \mid \beta \in \alpha\}$. As usual, using some padding bits, we can code each pair in G as a binary sequence. Then, by using the Gödel function $\mathfrak{g}$ we can code G as a binary sequence. Therefore, using Δ again, G can be coded a surreal number s.

Now, given a surreal s', our program can just go through the coding of G using the functions in Lemma 6 looking for a pair of the form (s', s''). Then, the program will return s'' in case of success or will diverge otherwise.

In [10], Ethan Lewis defines a notion of computability based on OTMs which allows for infinite programs. We will call these machines *infinite program machines* (IPMs). In [10], Lewis shows that IPMs are equivalent to OTMs with parameters in 2^{On}. Therefore, Theorem 14 tells us that $\mathrm{No}(\overrightarrow{X})$-SBSSM are a register model for IPMs.

We end this paper by introducing halting sets and universal programs for our new model of computation. Using classical coding techniques, given a class of rational functions $K(\overrightarrow{X})$, every $K(\overrightarrow{X})$-SBSSM program can be coded as one (possibly infinite) binary sequence, i.e., a surreal number.

Given two natural numbers n and m, and a subclass K of surreal numbers we will denote by $\mathfrak{P}_K^{n,m}$ the class of $(\mathfrak{N}, \mathfrak{D}, I, O, K)$-SBSSM programs with $|I| = n$, $|O| = m$.

Let K be a class of the surreal numbers. We define the following class[8]: $H_K^{n,m} := \{(p, s) \in \mathrm{No} \mid \ p \text{ is a } K(\overrightarrow{X})\text{-SBSSM program in } \mathfrak{P}_K^{n,m} \text{ halting on } s\}$.

As usual, we say that a set of surreal numbers is decidable if its characteristic function is computable.

If we assume that K contains $\{-1, 0, 1\}$ we can use the code of a program, together with the fact that OTMs can simulate $K(\overrightarrow{X})$-SBSSM and can be simulated by $K(\overrightarrow{X})$-SBSSM, to define a universal SBSSM program.

Let $\mathfrak{N}$ and $\mathfrak{D}$ be two disjoint sets of natural numbers, I and O be two disjoint subsets of $\mathfrak{N}$, and K be a class of surreal numbers. A $(\mathfrak{N}, \mathfrak{D}, I, O, K)$-SBSSM program P with $|I| = n + 1$ and $|O| = m$ is called *universal* if for every code p' of a program in $\mathfrak{P}_K^{n,m}$ and for every $x \in \mathrm{No}^n$ we have $P(p', x) = P(x)$.

A straightforward generalisation of the classical arguments shows the following results:

Theorem 16. *Let K be a subclass of* No *containing* $\{-1, 0, 1\}$. *Moroever, let $\mathfrak{N}, \mathfrak{D}, I, O \subset \mathbb{N}$ be such that: $\mathfrak{N}$ and $\mathfrak{D}$ are disjoint; and, I and O are disjoint subsets of $\mathfrak{N}$. Then, there is a universal $(\mathfrak{N}, \mathfrak{D}, I, O, K)$-SBSSM program.*

Proof. It is enough to note that any reasonable coding of a $K(\overrightarrow{X})$-SBSSM program can be computably translated into a code for an OTM computing the same function. In particular, note that all the functions in $K(\overrightarrow{X})$ used in the program will be in the code and all the parameters will also be encoded. So our universal program can start by taking the code of the program and converting it into an OTM simulation of the corresponding SBSSM program with the coefficients of the rational functions in $K(\overrightarrow{X})$-SBSSM program as parameters as in Theorem 11. Then, by using the algorithms in Lemma 6, the universal program will compute $\delta_{\mathrm{No}}^{-1}$ of the input of the $K(\overrightarrow{X})$-SBSSM program; and, as in Theorem 11, it will simulate the OTM obtained with the translation on the $\delta_{\mathrm{No}}^{-1}$ of the input. Finally, the universal program will, again by using the algorithms in Lemma 6, translate back the output of the OTM using δ_{No}.

Corollary 17. *Let K be a subclass of* No *containing* $\{-1, 0, 1\}$. *Then $H_K^{1,1}$ is not $K(\overrightarrow{X})$-SBSSM computable.*

Proof. Assume that $H_K^{1,1}$ is computable. Then, there is a program P that computes it. Now, consider the program P' that converges on x only if $(x, x) \notin H_K^{1,1}$.

[8] Note that, if K is a set $H_K^{n,m}$ is also a set.

This program is computable by Theorem 16 and by the assumptions. Now, let p' be a code for P'. We have that, $P'(p')$ converges if and only if $(p', p') \notin H_K$ diverges if and only if $P'(p')$ diverges.

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] Conway, J.H.: On Numbers and Games. A K Peters & CRC Press (2000)

[3] Ehrlich, P.: An alternative construction of Conway's ordered field No. Algebra Universalis **25**(1), 7–16 (1988)

[4] Galeotti, L., Nobrega, H.: Towards computable analysis on the generalised real line. In: Kari, J., Manea, F., Petre, I. (eds.) Unveiling Dynamics and Complexity: 13th Conference on Computability in Europe, CiE 2017, Turku, Finland, June 12–16, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10307, pp. 246–257. Springer (2017)

[5] Gonshor, H.: An Introduction to the Theory of Surreal Numbers, London Mathematical Society Lecture Note Series, vol. 110. Cambridge University Press (1986)

[6] Hamkins, J.D., Lewis, A.: Infinite time Turing machines. Journal of Symbolic Logic **65**, 567–604 (2000). https://doi.org/10.2307/2586556

[7] Koepke, P., Morozov, A.S.: The computational power of infinite time Blum-Shub-Smale machines. Algebra and Logic **56**(1), 37–62 (2017)

[8] Koepke, P.: Turing computations on ordinals. Bulletin of Symbolic Logic **11**(3), 377–397 (2005)

[9] 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)

[10] Lewis, E.: Computation with Infinite Programs. Master's thesis, ILLC Master of Logic Thesis Series MoL-2018-14, Universiteit van Amsterdam (2018)

[11] Seyfferth, B.: Three Models of Ordinal Computability. Ph.D. thesis, Rheinische Friedrich-Wilhelms-Universität Bonn (2013)

[12] Welch, P.D.: The length of infinite time turing machine computations. Bulletin of the London Mathematical Society **32**(2), 129–136 (2000)

Pseudo Algorithms

Algorithm 1: Gödel Map $G(\alpha)$

Input: Input in R_0
Output: Output in R_1 and R_2
Data: Dedekind Registers: Max, $Alpha$, $Beta$, $Count$

```
1  0 → Beta
2  0 → Alpha
3  Push_L(Beta, Max − 1)
4  if Count = R_0 then
5  |  R_1 := Alpha
6  |  R_2 := Beta
7  |  Stop
8  if Alpha < Max then
9  |  Push_L(Count, Count)
10 |  if Count = R_0 then
11 |  |  R_1 := Alpha
12 |  |  R_2 := Beta
13 |  |  Stop
14 |  Push_L(Alpha, Alpha)
15 |  Jump to 8
16 0 → Beta
17 if Beta < Max then
18 |  if Count = R_0 then
19 |  |  R_1 := Alpha
20 |  |  R_2 := Beta
21 |  |  Stop
22 |  Push_L(Beta, Beta)
23 |  Push_L(Count, Count)
24 |  Jump to 17
25 if Count = R_0 then
26 |  R_1 := Alpha
27 |  R_2 := Beta
28 |  Stop
29 Push_L(Max, Max)
30 Push_L(Count, Count)
31 Jump to 1
```

Algorithm 2: Sign Sequence $\mathrm{sgn}(s, \alpha)$

Input: Input in R_1, R_2
Output: Output in R_0
Data: Dedekind registers: $Step$, $Curr$

if $Curr = R_1$ **then**
  $R_0 := 2$
  Stop
if $R_1 < Curr$ **then**
  $R_0 := 0$
  $Push_R(Curr, Curr)$
if $Curr < R_1$ **then**
  $R_0 := 1$
  $Push_L(Curr, Curr)$
$Push_L(Step, Step)$
if $Step < R_2$ **then**
  GoTo 1

Algorithm 3: Bit Change $\mathrm{cng}(s, \alpha, \mathrm{sgn})$

Input: Input in R_1, R_2, R_3
Output: Output in R_0
Data: Dedekind registers: $Step$, $Curr$

$R_0 := Curr$
if $R_2 = Step$ **then**
  if $R_3 = 0$ **then**
    $Push_R(Curr, Curr)$
  else
    $Push_L(Curr, Curr)$
if $\mathrm{sgn}(R_1, Step) = 0 \wedge R_2 \neq Step$ **then**
  $Push_R(Curr, Curr)$
if $\mathrm{sgn}(R_1, Step) = 1 \wedge R_2 \neq Step$ **then**
  $Push_L(Curr, Curr)$
$Push_L(Step, Step)$
if $\mathrm{sgn}(R_1, Step) \neq 2$ **then**
  GoTo 1

Algorithm 4: Limit Ordinal $Lim(\alpha)$

Input: Input in R_1
Output: Output in R_0
Data: Dedekind registers: $Step$

```
1  if R_1 = Step then
2  |  R_0 := 1
3  |  Stop;
4  if R_1 = Step + 1 then
5  |  R_0 := 0
6  |  Stop;
7  Push_L(Step, Step)
8  Jump 1
```

Algorithm 5: $Liminf$ Subroutine

Input: Input in H_i
Data: Dedekind registers: Inf, Aus, $Step$, $Step_2$, Lim, $Zero$

```
1  0 → Step
2  0 → Inf
3  0 → Zero
4  if Zero < Step_2 ∧ sgn(H_i, Step) ≠⊥ then
5  |  if sgn(H_i, Step) = + then
6  |  |  Push_L(Zero, Zero)
7  |  Push_L(Step, Step)
8  |  Jump 4
9  if Pluses(H_i) = Step_2 then
10 |  Stop
11 if sgn(H_i, Step) = − then
12 |  Push_L(Aus, Aus)
13 |  Push_L(Step, Step)
14 |  Jump 11
15 if sgn(H_i, Step) = + then
16 |  Push_L(Step, Step)
17 |  Push_R(Inf, Aus + 1)
18 |  Aus := 0
19 |  Jump 11
20 Push_L(Lim, Inf − 1)
21 Push_L(Step_2, Step_2)
22 Jump 1
```

Algorithm 6: *Pluses* Subroutine

Input: Input in H_i
Data: Dedekind registers: $Plus$, $Step$

```
1 if sgn(H_i, Step) ≠⊥ then
2     if sgn(H_i, Step) = + then
3         Push_L(Plus, Plus)
4     Push_L(Step, Step)
5     Jump 1
```

Algorithm 7: *CanonicalRep* Subroutine

Input: Input in R_1
Data: Dedekind registers: $Step$, H

```
1 if sgn(R_1, Step) ≠⊥ then
2     if sgn(R_1, Step) = + then
3         Push_L(H, seg(R_1, Step))
4     if sgn(R_1, Step) = − then
5         Push_R(H, seg(R_1, Step))
6     Push_L(Step, Step)
7     Jump 1
```