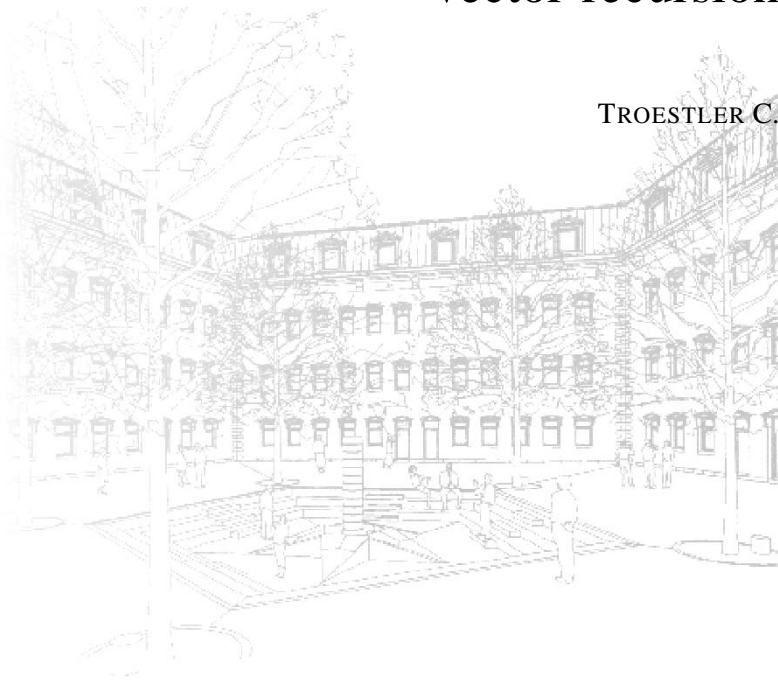


Preprint #9
June 4, 2001

Equivalence of BSS scalar- and vector-recursion

TROESTLER C.



Université de Mons-Hainaut
Institut de Mathématique

Phone: +32 65 37 35 07 — Fax: +32 65 37 33 18

Web: <http://www.umh.ac.be/math/institut>

Equivalence of BSS scalar- and vector-recursion

C. TROESTLER*

Christophe.Troestler@umh.ac.be

Université de Mons-Hainaut
Institut de Mathématique et d'Informatique
“Le Pentagone”, Avenue du Champ de Mars, 6
B-7000 Mons (Belgique)

Abstract. BSS-computable functions can be approached in two ways: from the point of view of computations performed by machines or under the angle of the theory of recursive functions. The goal of this paper is to answer negatively a basic question of the theory of BSS-recursive functions, namely “is vector-recursion stronger than scalar recursion?”

Keywords: BSS-computability, recursive functions.

*Partially supported by Esprit BRA project NeuroCOLT 27150, by Intas project 2000-447 and by a grant from the National Bank of Belgium.

Introduction

Since their creation, Turing machines have enjoyed a widespread use that is far from ceasing. They gave birth to a great deal of investigation and underlie what we nowadays mean by *computable* and *complexity*. Other equivalent approaches to computability were developed at the same time. Of particular interest for this paper is what we will call the “Kleene approach” in which computable functions (also called *recursive* functions) are described as the closure of some basic set of functions by some “generation” rules, the more important and powerful of these being recursion. Let us recall that, in this context, recursion is defined precisely as follows. Let $f_0 : \mathbb{N}^k \rightarrow \mathbb{N}^\ell$ and $F : \mathbb{N} \times \mathbb{N}^k \times \mathbb{N}^\ell \rightarrow \mathbb{N}^\ell$ ($k, \ell \leq \infty$) be two recursive functions. The recursion scheme says that the function $f : \mathbb{N} \times \mathbb{N}^k \rightarrow \mathbb{N}^\ell$ defined by

$$(1) \quad \begin{cases} f(0, x) = f_0(x) \\ f(n+1, x) = F(n, x, f(n, x)) \end{cases}$$

is recursive. We will call (1) a *vector*-recursion scheme to emphasize that ℓ may be greater than 1, that is that the recursion is performed on ℓ numbers simultaneously. A priori, vector recursion is more powerful than scalar-recursion ($\ell = 1$). However, it is shown early in the theory of recursive functions that \mathbb{N}^ℓ is isomorphic to \mathbb{N} by a computable isomorphism. As a consequence, scalar- and vector-recursion are equivalent. The proof is in fact a bit more complicated because not only we must code \mathbb{N}^ℓ in \mathbb{N} but also show that the operations on \mathbb{N}^ℓ transpose to scalar-recursive functions on the codings of \mathbb{N}^ℓ in \mathbb{N} . For details, the reader is referred to [8, p. 13].

Despite a great deal of work to extend Turing ideas, no theory became as successful and widespread as the original one. At least until Lenore

Blum, Michael Shub and Steve Smale introduced in [1] a simple yet powerful generalization of the notion of Turing machine that is today referred to as “BSS machine”. BSS machines are described briefly in [section 1](#) but for now it suffices to say that they are similar to Turing machines except that the computations are performed directly on a totally ordered ring R . With these machines on hand, L. Blum, M. Shub and S. Smale were able to define notions of computability and complexity for partial functions from R^k to R^ℓ . As in the classical case, BSS-computable functions can equivalently be defined in a “Kleene way” involving a recursion rule that read as (1) with \mathbb{N} replaced by R (see [section 1](#)).

A natural question is therefore whether vector-recursion is stronger or not than scalar-recursion for the BSS model. The answer is not as easy as before since in general R is not computably isomorphic¹ to R^ℓ . The purpose of this paper is to show that the two notions of recursion are still equivalent. Besides the fact that this result answers a basic question of the BSS-recursion theory, it has several implications, in particular in hierarchies of computable functions where it shows the equality of some complexity classes [4, 5].

The paper is organized as follows. In the next section, we will outline in an elementary way the theory of BSS-machines and BSS-computable functions. We will also take that opportunity to set our notations. [Section 2](#) is devoted to the proof of the equivalence.

¹For example when R is a real closed field whose transcendence degree over \mathbb{Q} is infinite, e.g. $R = \mathbb{R}$, the dimension of a set is invariant under computable isomorphisms and $\dim R^\ell = \ell$ (see e.g., [6]). Thus $R^\ell \not\cong R$ when $\ell \neq 1$. Let us also mention in passing a very interesting result by van den Dries [3] that roughly states that, for semi-algebraic isomorphisms, the two invariants are the dimension and the Euler characteristic.

Acknowledgments. I would like to thank Maurice Boffa for drawing my attention to this problem. I am also pleased to thank Christian Michaux for interesting discussions.

1 BSS machines and computability

In their seminal paper [1], L. Blum, M. Shub and S. Smale describe a new type of machine that can perform computations directly on a totally ordered ring R . They also show how the associated computable functions can be generated from a set of “basic functions”. The purpose of this section is to provide the reader with a sketch of that theory, emphasizing what will be of use to us. For more details, the reader is referred to [1, 2]. Our presentation will differ slightly from [1] in order to give us some more freedom in writing programs.

Let R be a totally ordered ring. As usual R^k , $k \in \mathbb{N}$, stands for the cartesian product of R , k times with itself. In addition,

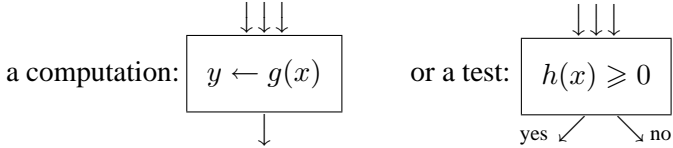
$$R^\infty := \{(x_i)_{i=1}^\infty : \text{all } x_i \in R \text{ and all but a finite number of them are null}\}.$$

We will identify² $R^k \times R^\ell$ with $R^{k+\ell}$. Let us write $\mathbb{N}_0^{\leq k} := \{1, \dots, k\}$ when k is an integer and $\mathbb{N}_0^{\leq \infty} := \mathbb{N} \setminus \{0\}$. A *machine* is the data of a finite set of variables,³ each of which belongs to some R^k with

²This identification will be made even when k and ℓ are infinite by means of the following bijection : $R^\infty \times R^\infty \rightarrow R^\infty : ((x_1, x_2, \dots), (y_1, y_2, \dots)) \mapsto (x_1, y_1, x_2, y_2, \dots)$.

³Allowing several variables, each with their own length, can be seen as working with several tapes—one per variable. Of course it is just a matter of presentation and it does not change the power of the language.

$k \in \mathbb{N} \cup \{\infty\}$, and a *flowchart* (that is, a finite directed connected graph). In addition to an *input* and an *output* node with their associated variables — through which the program receives and outputs data — the nodes of the flowchart can perform



where $x \in R^k$, $y \in R^\ell$ ($k, \ell < \infty$) are variables and $g : R^k \rightarrow R^\ell$, $h : R^k \rightarrow R$ are polynomial maps (or rational maps if R is a field). Moreover, if $x \in R^k$, $y \in R^\ell$ with $k, \ell \in \mathbb{N} \cup \{\infty\}$ and $i, j \in R$ are variables, a computational node can be of the so called fifth (or *shift*) type⁴ ‘ $y[j] \leftarrow x[i]$ ’ where $x[i]$ (resp. $y[j]$) is the i th (resp. j th) component of x (resp. y) if $i \in \mathbb{N}_0^{\leq k}$ (resp. $j \in \mathbb{N}_0^{\leq \ell}$) and is undefined otherwise (in which case the assignation $y[j] \leftarrow x[i]$ is nonterminating i.e., it goes into an infinite loop). Finally, a computational node can also be⁵ ‘ $i \leftarrow \text{length}(x)$ ’ where $i \in R$, $x \in R^k$ with $k \in \mathbb{N} \cup \{\infty\}$ and $\text{length}(x) := \max\{\iota \in \mathbb{N} : x_\iota \neq 0\}$. In order that all our programs make sense, we will assume that all the variables are implicitly initialized to 0.

In this paper we will use the same letter for a variable and for a particular instantiation of it. The context should tell the difference. For example if we say “polynomial of x ” we are referring to the variable

⁴Clearly this is only new when k or ℓ is infinite for otherwise this assignation can be done by a polynomial map.

⁵This is made explicit to clarify the requirement by L. Blum, M. Shub and S. Smale [1, p. 12] that (an upper bound for) the length of the initial data is stored in memory. But then, one can keep track of the length of any variable during the course of the computations if so one wishes. Again, this is only of interest when $k = \infty$.

x whereas if we write “we evaluate P at x ” we mean “at the number stored in x .”

From the description above, it is natural to call a *computational path* a sequence of nodes starting at the input node — which implies some data is provided through the associated variable, say $x \in R^k$ — ending at the output node — let $y \in R^\ell$ be the associated variable — and following the vertices of the graph, performing the computations or choosing the next vertice according to the outcome of the tests. So, every computational path maps some data $x \in R^k$ to a “result” $y \in R^\ell$. However, the computation starting at some $x \in R^k$ does not necessarily reach the output node — and thus does not produce a computational path —, so the map $x \mapsto y$ may not be defined for any $x \in R^k$. It is a partial map. Partial maps $R^k \circlearrowright R^\ell$ that are generated by machines in the above way are called *computable*. Here we will use the notation ‘ \circlearrowright ’ to emphasize that the map is partial and will reserve ‘ \rightarrow ’ for total maps.

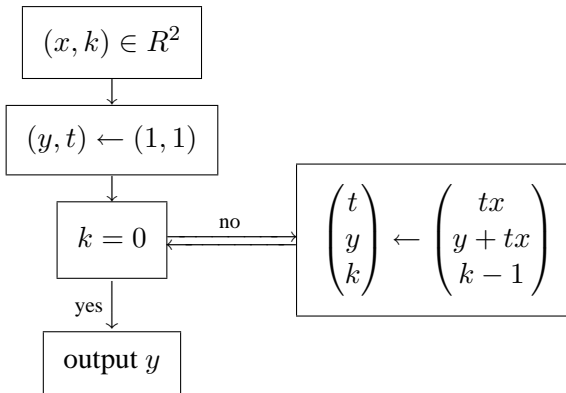


Figure 1: Machine computing f .

As an example of BSS-machine, let us consider the problem of com-

puting the Taylor expansion of $1/(1-x)$ at order k around 0. In other words, we want to compute the function $f : R \times R \rightarrow R$, whose domain is $R \times \mathbb{N}$, which is defined by

$$f(x, k) := \sum_{i=0}^k x^i.$$

The machine computing f uses the variables $(x, k) \in R^2$ as input, $y \in R$ as output and $t \in R$ as a “local” variable to the program. The flowchart⁶ of this machine is drawn **Figure 1**. It is easy to see that the computation starting with some (x, k) will stop iff $k \in \mathbb{N}$. If $k \notin \mathbb{N}$, the test ‘ $k = 0$ ’ will always be false and the program will end up in an infinite loop.

At this point it should be clear that, because $\mathbb{Z} \subset R$, BSS machines can execute all programs written for Turing machines. In particular all discrete data structures are available to us. We can even code an instance of such a discrete structure in a single integer. For example, we can say that $n \in \mathbb{N}$ represent (or even, abusively, “is”) a binary tree with vertices labelled with integers. Of course, the operations on a discrete structure that are Turing computable are also BSS-computable.

Now it is time to turn to a “Kleene vision” of BSS-computable functions. The set \mathcal{R} of partial BSS-recursive functions over R is the smaller set of partial functions containing

- (2) the polynomial maps $R^k \rightarrow R$ (or the rational maps $R^k \rightarrow R$ if R is a field) where $k < \infty$;

⁶Strictly speaking of course, we should have replaced the test ‘ $k = 0$ ’ by the two tests ‘ $k \geq 0$ ’ and ‘ $k \leq 0$ ’. We leave that modification to the reader.

(3) the characteristic function $\chi : R \rightarrow R$ defined by

$$\chi(x) := \begin{cases} -1 & \text{if } x < 0, \\ 0 & \text{if } x = 0, \\ 1 & \text{if } x > 0; \end{cases}$$

(4) the inclusions $R^k \hookrightarrow R^\infty : (x_1, \dots, x_k) \mapsto (x_1, \dots, x_k, 0, 0, \dots)$ and the projections $\text{pr}_k : R^\infty \rightarrow R : (x_i)_{i=1}^\infty \mapsto x_k$;

(5) the function $\sigma : R \times R \times R^\infty \rightarrow R^\infty$ with domain $\mathbb{N} \times \mathbb{N} \times R^\infty$ defined by $\sigma(n, m, x) = y = (y_i)_{i=1}^\infty$ with $y_i = x_i$ if $i \neq m$ and $y_m = x_n$;

(6) the length function $R^\infty \rightarrow R : x \mapsto \text{length}(x)$;

and closed by the rules

(7) *composition*: if $f : R^k \rightarrow R^\ell$ and $g : R^\ell \rightarrow R^m$ are recursive functions, so is $g \circ f : R^k \rightarrow R^m : x \mapsto g(f(x))$ with domain $f^{-1}(\text{Dom } g)$;

(8) *juxtaposition*: if $f : R^k \rightarrow R^\ell$ and $g : R^k \rightarrow R^m$ are recursive functions, so is $(f, g) : R^k \rightarrow R^\ell \times R^m : x \mapsto (f(x), g(x))$ with domain $\text{Dom } f \cap \text{Dom } g$;

(9) *recursion*: given two recursive maps $f_0 : R^k \rightarrow R^\ell$ and $F : R \times R^k \times R^\ell \rightarrow R^\ell$, recursion says that the following map $f : R \times R^k \rightarrow R^\ell$ is recursive where f is uniquely defined by

$$\begin{cases} f(0, x) = f_0(x) \\ f(n+1, x) = F(n, x, f(n, x)) \end{cases}$$

with domain being the set $\bigcup_{n=0}^\infty D_n \subset \mathbb{N} \times R^k$ with $D_0 := \{0\} \times \text{Dom } f_0$ and $D_{n+1} := \{(n+1, x) : (n, x, f(n, x)) \in \text{Dom } F\}$.

(10) *minimalization*: given a recursive map $F : R \times R^k \rightarrow R$ defined at least on $\mathbb{N} \times R^k$, minimalization defines a function $f : R^k \rightarrow R$

R by

$$f(x) = \mu_n F(n, x) := \min\{n \in \mathbb{N} : F(n, x) = 0\}$$

whose domain is the set of $x \in R^k$ for which there exists a $n \in \mathbb{N}$ with $F(n, x) = 0$.

It is well known [1, § 7] that the set of BSS-recursive functions and the set of BSS-computable functions coincide.

The integers k and ℓ being fixed, the set of partial maps from R^k to R^ℓ that are BSS-recursive will be written $\mathcal{R}(R^k, R^\ell)$. Because ℓ may be > 1 , (9) may be termed a vector-recursion scheme. If $\ell = 1$, (9) will be called a *scalar-recursion* scheme. The smaller set of partial functions containing (2)–(6) and closed under (7), (8), (9) with $\ell = 1$, and (10) will be denoted \mathcal{R}^1 . Also, $\mathcal{R}^1(R^k, R^\ell) := \mathcal{R}^1 \cap \mathcal{R}(R^k, R^\ell)$. When $f \in \mathcal{R}^1$, we will say that “ f is a *scalar-recursive* function.”

We now have all the ingredients to state and prove our theorem. This is the subject of the next section.

2 Proof of the equivalence

The aim of this section is to prove that $\mathcal{R}^1 = \mathcal{R}$. Of course $\mathcal{R}^1 \subset \mathcal{R}$, so we only have to show that, if $f \in \mathcal{R}$, it is also true that $f \in \mathcal{R}^1$. This is done in several steps, the more important of these is the fact that the evaluation is scalar-recursive (see lemma 5). Let us start this program with a simple generalization of the scalar-recursion scheme that we will use later on.

Lemma 1. *Let $\mathcal{N}_0 \subset \mathcal{N}$ be two Turing decidable subsets of \mathbb{N} and $\delta : \mathcal{N} \times R^k \rightarrow \mathcal{N}$, $f_0 : R \times R^k \rightarrow R$, $F : R \times R^k \times R \rightarrow R$ be*

three scalar-recursive functions. Then the map $f : R \times R^k \rightarrow R$ is scalar recursive where f is defined by

$$\begin{cases} f(\nu, x) = f_0(\nu, x) & \text{if } \nu \in \mathcal{N}_0 \\ f(\nu, x) = F(\nu, x, f(\delta(\nu, x), x)) & \text{if } \nu \in \mathcal{N} \setminus \mathcal{N}_0 \end{cases}$$

with domain being the set $\bigcup_{n=0}^{\infty} D_n \subset \mathcal{N} \times R^k$ with

- $D_0 := (\mathcal{N}_0 \times R^k) \cap \text{Dom } f_0$ and
- $D_{n+1} := \{(\nu, x) : (\nu, x) \in \text{Dom } \delta, (\delta(\nu, x), x) \in D_n \text{ and } (\nu, x, f(\delta(\nu, x), x)) \in \text{Dom } F\}$.

Let us emphasize that the above recursive definition of $\text{Dom } f$ implies that, if $(\nu, x) \in \text{Dom } f$, there is a $n \in \mathbb{N}$ such that $\Delta(n, \nu, x) := \underbrace{\delta(\cdot, x) \circ \cdots \circ \delta(\cdot, x)}_{n \text{ times}}(\nu) \in \mathcal{N}_0$.

Proof. Let us define the two functions $\varphi : R \times R \times R^k \rightarrow R$ and $\Delta : R \times R^k \times R \rightarrow R$ respectively by the recursions:

$$\begin{cases} \varphi(0, \nu, x) = f_0(\nu, x) \\ \varphi(n+1, \nu, x) = F(\nu, x, \varphi(n, \delta(\nu, x), x)) \end{cases}$$

$$\begin{cases} \Delta(0, \nu, x) = \nu \\ \Delta(n+1, \nu, x) = \delta(\Delta(n, \nu, x), x) \end{cases}$$

In virtue of (9), φ and Δ are scalar-recursive functions. Since \mathcal{N}_0 is Turing decidable, there exists a Turing-computable (hence scalar-BSS-recursive) function $g : \mathbb{N} \rightarrow \mathbb{N}$ such that: $\nu \in \mathcal{N}_0 \iff g(\nu) = 0$. Set $h(\nu, x) := \mu_n g(\Delta(n, \nu, x)) = \min\{n \in \mathbb{N} : \Delta(n, \nu, x) \in \mathcal{N}_0\}$. It is not difficult to show that

$$f(\nu, x) = \varphi(h(\nu, x), \nu, x).$$

Hence f is scalar-recursive and the proof is complete. \square

We will first prove:

Theorem 2. $\mathcal{R}(R^k, R) = \mathcal{R}^1(R^k, R)$ for all $k \in \mathbb{N} \cup \{\infty\}$.

Let $f \in \mathcal{R}$. In order to ease the explanation of the argument, we will see f as being computed by a BSS-machine M . We are first going to show that we can “get rid of the constants” of M .

Definition 3. We will say that M is a BSS-machine with coefficients in \mathbb{Z} iff every polynomial (or rational) map appearing in a computational or a test node of M has its coefficients in \mathbb{Z} .

Let $\mathcal{R}_{\mathbb{Z}}$ be the set of all functions computed by BSS-machines with coefficients in \mathbb{Z} and $\mathcal{R}_{\mathbb{Z}}(R^k, R^\ell) := \mathcal{R}_{\mathbb{Z}} \cap \mathcal{R}(R^k, R^\ell)$.

Lemma 4. *If $\mathcal{R}_{\mathbb{Z}}(R^k, R) \subset \mathcal{R}^1(R^k, R)$ for all $k \in \mathbb{N} \cup \{\infty\}$, then **Theorem 2** is proven.*

Proof. Let $f \in \mathcal{R}(R^k, R)$ be computed by a machine M . Let us note g_1, \dots, g_p the polynomial (or rational) maps appearing in a computational or a test node of M . Let $a_i \in R^{r_i}$ be the coefficients of $g_i : R^{n_i} \rightarrow R^{m_i}$, $a := (a_i)_{i=1}^p \in R^{r_1} \times \dots \times R^{r_p} =: R^r$ and $\tilde{g}_i : R^{n_i} \times R^r \rightarrow R^{m_i}$ be like g_i but with its coefficients seen as variables in R^r so that $g_i = \tilde{g}_i(\cdot, a)$. Now, let \tilde{M} be the machine that possesses the same directed graph as M but with the nodes transformed in the following way:

- the variable, say $x \in R^k$, of the entry node of M is replaced with $(x, \alpha) \in R^k \times R^r$ where α is a new variable;

- each computational node ‘ $\eta \leftarrow g_i(\xi)$ ’ is changed into ‘ $\eta \leftarrow \tilde{g}_i(\xi, \alpha)$ ’;
- each test node ‘ $g_i(\xi) \geq 0$ ’ (resp. ‘ $g_i(\xi) > 0$ ’) is replaced with ‘ $\tilde{g}_i(\xi, \alpha) \geq 0$ ’ (resp. ‘ $\tilde{g}_i(\xi, \alpha) > 0$ ’);
- the other nodes are left unmodified.

Let $\tilde{f} : R^k \times R^r \rightarrow R$ be the function computed by \tilde{M} . Clearly $f(\cdot) = \tilde{f}(\cdot, a)$. Since $\mathbb{1} : R^k \rightarrow R^k : x \mapsto x$ and $p : R^k \rightarrow R^r : x \mapsto a$ are two polynomial maps,⁷ (8) implies $(\mathbb{1}, p) \in \mathcal{R}^1$. Then, from the composition rule (7), one infers $f = \tilde{f} \circ (\mathbb{1}, p) \in \mathcal{R}^1$. \square

The interesting consequence of the previous lemma is that the machines generating the functions of $\mathcal{R}_{\mathbb{Z}}(R^k, R)$ are discrete structures. Let M be such a machine. It consists of a finite set of variables and a finite directed graph whose nodes are “labelled” with polynomial (or rational) maps. But each of these polynomials is given by the name and indices of the variables on which it acts plus a finite family of integer coefficients. Nodes of the fifth type and of the type ‘ $i \leftarrow \text{length}(x)$ ’ are also determined by a finite family of integers. So we see that the whole *structure* of M can be described by a finite number of integers. Adopting a coding, we can in fact assume that M is described by a *single* integer $\mathbf{m} \in \mathbb{N}$. In the sequel, we will do various operations on the structure of a machine M (following paths, symbolically composing polynomials, ...). We will leave to the reader the proof that these operations can in fact be carried on the associated integer \mathbf{m} by a Turing machine — hence by a scalar-BSS-recursive function.

⁷If $k = \infty$, $\mathbb{1} : R^\infty \rightarrow R^\infty : x \mapsto \sigma(1, 1, x)$ and $p = (p_1, \dots, p_r)$ with $p_i : R^\infty \rightarrow R : x \mapsto x[i]$ are scalar-BSS-recursive (as $r < \infty$).

From now on, let $f \in \mathcal{R}_{\mathbb{Z}}(R^k, R)$, a machine M computing f and its corresponding integer m be fixed. We will also note $x \in R^k$ the input variable and $y \in R$ the output variable of M .

It is well known that, unfolding the loops, the computational paths can be seen on a (generally infinite) binary tree. Let us recall quickly how. If several computational nodes follow each other in the graph of the machine, we decide to group them in a single one. So, from now on, computational nodes can perform sequentially several assignments. Also, a computational node must be followed either by the output node or by a test node. On the other hand, if a test node directly follows another, we can introduce a trivial computational node ‘ $x \leftarrow \mathbb{1}(x)$ ’ between the two. Let the input node be the top node of the binary tree. According to the “normalization” described above, we can assume it is followed by a (possibly trivial) computational node. That is symbolized by an arrow. Next is a test node that gives a two-ways branching representing the two outcomes (‘yes’ or ‘no’) of the test. Each of these branches start with a computation — represented by an arrow — followed by another test — and so another two-ways branching. The binary tree then unfolds by repeating this procedure. When the output node is reached, a leave of the binary tree is created. On the other hand, if one never hits the output node, the branch continues *ad infinitum*. To illustrate that construction, we have drawn on [Figure 2](#) the beginning of the binary tree corresponding to the program of [Figure 1](#). This is the standard construction. However, here we need a special expansion of the last two type of computational nodes. For a shift node ‘ $\eta[j] \leftarrow \xi[i]$ ’ we will perform all the necessary tests to know the value of i and j and then do the corresponding assignation — see [Figure 3](#). A computation of the type ‘ $i \leftarrow \text{length}(\xi)$ ’ will also be transformed into a series of tests on the length and associated assignments — see [Figure 4](#).

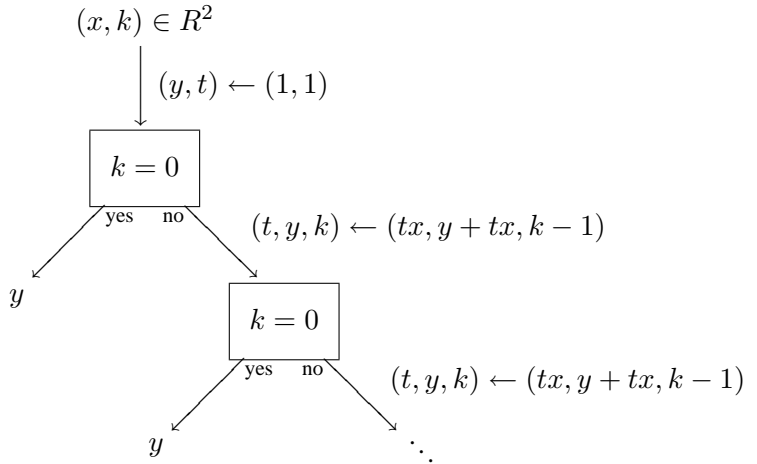


Figure 2: Tree of the machine of [Figure 1](#).

Since in general the binary tree is infinite, it cannot be coded by a single integer. However, because the tree is a mechanical unfolding of the BSS-machine, it is possible for a Turing machine to compute it up to an arbitrary depth. More precisely, a possible path of length ℓ in the tree is determined by ℓ answers ‘yes’ or ‘no’ to the ℓ first tests. If we code ‘yes’ by 1 and ‘no’ by 0, a path is just a binary word γ of length ℓ . Such a word can be coded by a single integer. Given a path γ , let $P(\gamma, z_i) \in \mathbb{Z}[x]$ denote the polynomial giving the value of the i th component of the variable $z \in R^m$ as a function of x at the ending node of γ (for $i > m$, we set $P(\gamma, z_i) = 0$). Even though $P(\gamma, z_i)$ is a polynomial of x and neither of γ nor z_i , we write it this way because we want to stress that this polynomial depends (in a computable way, as we will see) of the data γ and z_i . It is indeed a polynomial because, with the expansion above, all computation nodes

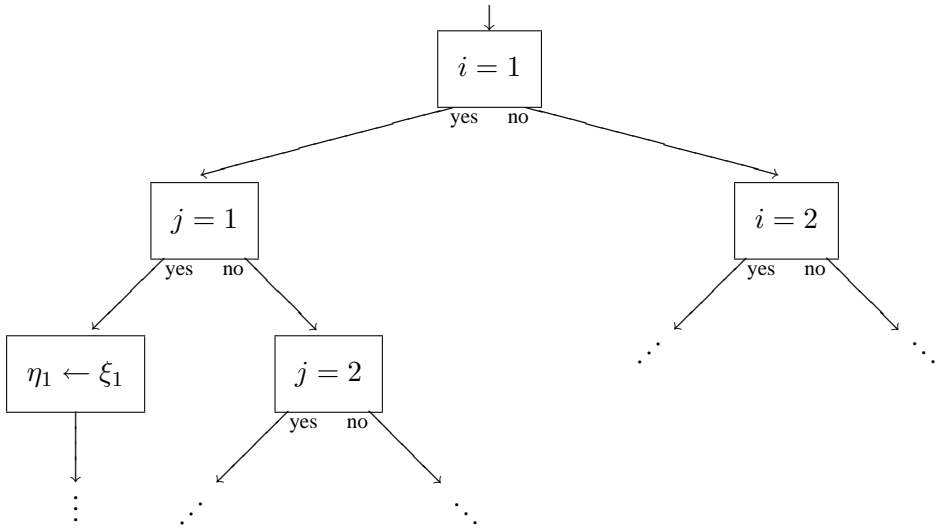


Figure 3: Tree associated to a node ' $\eta[j] \leftarrow \xi[i]$ '.

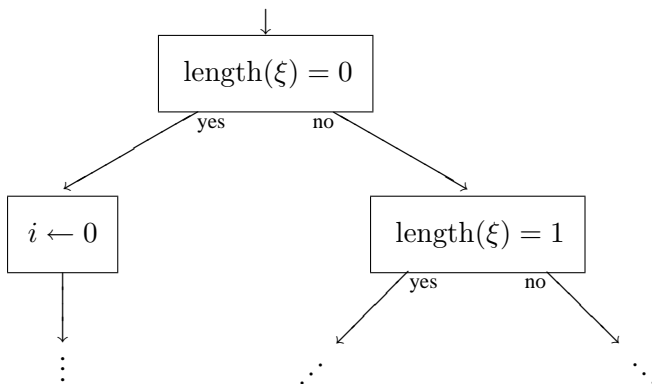


Figure 4: Expansion of ' $i \leftarrow \text{length}(\xi)$ '.

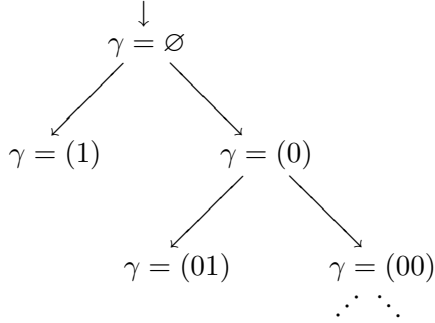


Figure 5: Paths for the tree of Figure 2.

of the tree are polynomials. In fact, it enjoys a recursive definition. If $\gamma = \emptyset$, $P(\gamma, z_i)$ is just given by the first computing node (or the implicit initialization). If $\gamma = (\gamma' e)$ where e is the last digit of the path, we just look at the last node of computation to get z as a function of other variables, say z' , and then substitute these variables thanks to $P(\gamma', z'_j)$. As a consequence $P(\gamma, z_i)$ is Turing computable.

Let us also define $T(\gamma) \in \text{Bool}(\mathbb{Z}[x])$ as the conjunction of all tests of the type ‘ $g(x) \geq 0$ ’, ‘ $g(x) > 0$ ’, and ‘ $\text{length}(g(x)) = k$ ’ that expresses that the path γ is followed by an entry x . Because $g(x)$ results from the previous computations along the path γ , it is a polynomial of x . If γ is not a path of the binary tree, $T(\gamma)$ is false. As an example, for the machine of Figure 1, $T((01)) = ‘k \neq 0 \wedge k - 1 = 0’$ whereas $T((10))$ is false (see also Figures 2 and 5). By a reasoning similar to the one for $P(\gamma, z_i)$, one shows that the formula $T(\gamma)$ is a computable function of γ .

Throughout this paper we will write $\mathbb{Z}[x]$ for the shake of simplicity

but it must be understood that we are in fact referring to a coding⁸ of $\mathbb{Z}[x]$ into \mathbb{N} and that the operations we will perform on the elements of $\mathbb{Z}[x]$ are in fact performed on the corresponding codings. The same can be said for the set $\text{Bool}(\mathbb{Z}[x])$ of boolean formulae. We shall write Γ for the set of all binary words — that we also call abusively paths — and V for the finite set containing the names of the variables of M . Again, Γ and V must be seen as (decidable) subsets of \mathbb{N} . Summing up, we can say that the two functions

$$P : \Gamma \times V \times \mathbb{N} \rightarrow \mathbb{Z}[x] : (\gamma, z, i) \mapsto P(\gamma, z_i)$$

and

$$T : \Gamma \rightarrow \text{Bool}(\mathbb{Z}[x])$$

are scalar-BSS-recursive.

We want to show that scalar recursion is enough to keep track of the path a given data $x \in R^k$ follows. The essential result is given by the lemma:

Lemma 5. *Let $k \in \mathbb{N}$ be fixed. Then the function $\text{ev} : \mathbb{Z}[x] \times R^k \subset R \times R^k \hookrightarrow R : (P, x) \mapsto \text{ev}(P, x) := P(x)$ is scalar-BSS-recursive. This is also true for $k = \infty$ if we define $\mathbb{Z}[x]$ as $\bigcup_{\ell=0}^{\infty} \mathbb{Z}[x_1, \dots, x_\ell]$.*

Proof. Let $k \in \mathbb{N} \cup \{\infty\}$ be fixed. Let us start this proof with some elementary considerations. First, because $x \mapsto x_i$ is equal to the composite function $R^k \hookrightarrow R^\infty \xrightarrow{\sigma(i,1,\cdot)} R^\infty \xrightarrow{\text{pr}_1} R$, it is easy to see that $\text{pr} : R^k \times R \hookrightarrow R$, defined by $\text{pr}(x, i) = x_i$ if $i \in \mathbb{N}_0^{\leq k}$, is scalar-BSS-recursive. Recall that, for a multi-index $\alpha \in \mathbb{N}^k$, x^α stands for

⁸By storing for example the degree and the integer coefficients of the polynomials.

$x_1^{\alpha_1} \cdots x_k^{\alpha_k}$. We will show that the map

$$\text{pow} : R^k \times \mathbb{N}^k \rightarrow R : (x, \alpha) \mapsto x^\alpha$$

is scalar-BSS-recursive. For any $\alpha \in \mathbb{N}^k \setminus \{0\}$, let $\alpha_\# := \min\{i \in \mathbb{N}_0^{\leq k} : \alpha_i \neq 0\}$ and $\delta(\alpha) := \alpha - e_{\alpha_\#}$ where $e_i \in \mathbb{N}^k$ is the vector whose all components are null except the i th one which equals 1. We also set $0_\# := 0$ and $\delta(0) := 0$. It is standard to show that the maps $\mathbb{N}^k \rightarrow \mathbb{N} : \alpha \mapsto \alpha_\#$ and $\delta : \mathbb{N}^k \rightarrow \mathbb{N}^k$ are Turing computable. Since the map pow can be defined by the recursion

$$\begin{cases} \text{pow}(x, 0) = 1, \\ \text{pow}(x, \alpha) = \text{pow}(x, \delta(\alpha)) \cdot \text{pr}(x, \alpha_\#), \quad \text{if } \alpha \in \mathbb{N}^k \setminus \{0\}, \end{cases}$$

Lemma 1 implies that it is also scalar-BSS-recursive.

Now, let us consider the two maps

$$\begin{aligned} \mathbb{Z}[x] \times \mathbb{N}^k &\rightarrow \mathbb{Z} : (P, \alpha) \mapsto \text{coeff}(P, \alpha) \\ \mathbb{Z}[x] &\rightarrow \mathbb{N}^k : P \mapsto \text{da}(P) = \underbrace{(d(P), \dots, d(P))}_{a(P) \text{ times}}, 0, 0, \dots \end{aligned}$$

where $\text{coeff}(P, \alpha)$ is the (possibly null) coefficient of x^α in P , $d(P)$ is the greater d such that the term x_i^d appears in P for some $i \in \mathbb{N}_0^{\leq k}$, and $a(P)$ is the arity of P , that is the greater $i \in \mathbb{N}_0^{\leq k}$ for which P depends on x_i . Since computing these maps involve only some elementary operations on the structure of polynomials $P \in \mathbb{Z}[x]$, they are Turing computable. Let us also define $\lambda(P, \alpha)$ as the multi-index that is just before α for the lexicographic order and which is below $\text{da}(P)$. More precisely,

$$\begin{aligned} \lambda(P, 0) &= 0, \\ \lambda(P, \alpha) &= (d(P), \dots, d(P), \alpha_i - 1, \alpha_{i+1}, \alpha_{i+2}, \dots). \end{aligned}$$

where $i = \alpha_{\#}$. If $\alpha \in \mathbb{N}^k$ is such that $\alpha_i \in \{0, \dots, d(P)\}$ for all i , one can understand $\lambda(P, \alpha)$ as follows. One can interpret α as the number $\bar{\alpha} := \sum_{i=1}^k \alpha_i (d(P) + 1)^{i-1}$ whose digits in base $d(P) + 1$ are the α_i . Then $\lambda(P, \alpha)$ gives the digits associated with $\bar{\alpha} - 1$. For the same reasons as above, $\lambda : \mathbb{Z}[x] \times \mathbb{N}^k \rightarrow \mathbb{N}^k$ is Turing computable. Let us define $\mathcal{E} : \mathbb{N}^k \times \mathbb{Z}[x] \times R^k \rightarrow R$ by

$$\begin{cases} \mathcal{E}(0, P, x) = \text{coeff}(P, 0) \\ \mathcal{E}(\alpha, P, x) = \text{coeff}(P, \alpha) \cdot \text{pow}(x, \alpha) + \mathcal{E}(\lambda(P, \alpha), P, x). \end{cases}$$

Because of **Lemma 1**, \mathcal{E} is scalar-BSS-recursive. In order to express what the recurrence defining \mathcal{E} does, one needs to introduce the following two orders “ \leq ” and “ \preceq ” on multi-indices. First, there is the usual (partial) order:

$$\alpha \leq \beta \iff \forall i \in \mathbb{N}_0^{\leq k}, \alpha_i \leq \beta_i.$$

Second, let “ \preceq ” denote the lexicographical (total) order with the first indices getting the lower weights:

$$\begin{aligned} \alpha \preceq \beta &\iff \alpha = \beta \text{ or } \alpha \prec \beta \\ \alpha \prec \beta &\iff \exists i \in \mathbb{N}_0^{\leq k}, \alpha_i < \beta_i \text{ and } \forall j > i, \alpha_j = \beta_j. \end{aligned}$$

Given these definitions, the following four facts are easy to check: for all $P \in \mathbb{Z}[x]$,

- (11) $P = \sum_{\alpha \leq \text{da}(P)} \text{coeff}(P, \alpha) x^\alpha$,
- (12) for all $\alpha \in \mathbb{N}^k$, $\alpha \leq \text{da}(P) \implies \alpha \preceq \text{da}(P)$,
- (13) for all $\beta \prec \alpha$ in \mathbb{N}^k with $\beta \leq \text{da}(P)$, $\beta \preceq \lambda(P, \alpha) \prec \alpha$,
- (14) for all $\alpha \in \mathbb{N}^k$, $\alpha \leq \text{da}(P) \implies \lambda(P, \alpha) \leq \text{da}(P)$.

As a consequence of (13)–(14), we have

$$(15) \quad \{\beta \leq \text{da}(P) : \beta \preceq \alpha\} = \{\beta \leq \text{da}(P) : \beta \preceq \lambda(P, \alpha)\} \cup \{\alpha\}.$$

An argument by recurrence using (15) shows

$$\mathcal{E}(\alpha, P, x) = \sum_{\substack{\beta \leq \text{da}(P) \\ \beta \preceq \alpha}} \text{coeff}(P, \beta) x^\beta.$$

Thus, (11)–(12) implies $\text{ev}(P, x) = \mathcal{E}(\text{da}(P), P, x)$ is scalar-BSS-recursive and the proof is complete. \square

The previous result implies boolean formulae are also evaluable.

Lemma 6. *Let $k \in \mathbb{N} \cap \{\infty\}$ be fixed. The map $\text{ev} : \text{Bool}(\mathbb{Z}[x]) \times R^k \rightarrow \text{Bool} : (\varphi, x) \mapsto \text{ev}(\varphi, x)$, where $\text{ev}(\varphi, x) = 1$ if φ is true when evaluated with data x and 0 otherwise, is scalar-BSS-recursive.*

Proof. Let $\varphi \in \text{Bool}(\mathbb{Z}[x])$. Because it is only a mechanical transformation of φ , we can assume without lack of generality that φ has the form

$$\varphi = \bigvee_{i=1}^N \bigwedge_{j=0}^{M_i} B_{ij} \quad \text{where } B_{ij} \text{ is } \begin{cases} 'P_{ij}(x) \geq 0', \\ 'P_{ij}(x) > 0', \text{ or} \\ '\text{length}(P_{ij}(x)) = L_{ij}', \end{cases}$$

with $P_{ij} \in \mathbb{Z}[x]$, and $\varphi \mapsto N$, $(\varphi, i) \mapsto M_i$, $(\varphi, i, j) \mapsto P_{ij}$, $(\varphi, i, j) \mapsto L_{ij}$ are Turing computable. For the atom formulae B_{ij} , the evaluation is easy to define:

$$\begin{aligned} & \text{ev}(B_{ij}, x) \\ &= \begin{cases} \chi_{\geq 0}(\text{ev}(P_{ij}, x)) & \text{if } B_{ij} \text{ is } P_{ij}(x) \geq 0, \\ \chi_{> 0}(\text{ev}(P_{ij}, x)) & \text{if } B_{ij} \text{ is } P_{ij}(x) > 0, \\ \chi_{=0}(\text{length}(\text{ev}(P_{ij}, x)) - L_{ij}) & \text{if } B_{ij} \text{ is } \text{length}(P_{ij}(x)) = L_{ij}, \end{cases} \end{aligned}$$

where $\chi_{\geq 0}(t) := (-\chi^2(t) + \chi(t) + 2)/2$, $\chi_{> 0}(t) := (\chi^2(t) + \chi(t))/2$, and $\chi_{=0}(t) = \chi_{\geq 0}(t) \cdot \chi_{\geq 0}(-t)$. Thus $(\varphi, i, j, x) \mapsto \text{ev}(B_{ij}, x)$ is scalar-BSS-computable. From there it is easy to get the evaluation of φ :

$$\text{ev}(\varphi, x) = \chi_{> 0} \left(\sum_{i=1}^N \prod_{j=0}^{M_i} \text{ev}(B_{ij}, x) \right).$$

It is scalar-BSS-recursive because $(\varphi, i, x) \mapsto \prod_{j=0}^{M_i} \text{ev}(B_{ij}, x)$ can be defined by recursion on j and the same is true for the sum with respect to i . \square

With these building blocks, we will show that we can compute the path followed by an input x and then the transformation of x along that path.

Proof of theorem 2. Let $f \in \mathcal{R}(R^k, R)$. As said before, **Lemma 4** implies one can assume that f is computed by a machine M with coefficients in \mathbb{Z} . Let P and T be defined as before for M . Let also $\Gamma_M \subset \Gamma$ be the set of computational paths, that is paths $\gamma \in \Gamma$ that are ending at a leaf of the binary tree. Since it basically suffices to follow the path (which is of finite length) on the binary tree expanded from the flowchart of M to see whether or not it ends at a leaf, Γ_M is a Turing decidable subset of Γ . Consequently there exists a Turing computable bijection $\mathbb{N} \rightarrow \Gamma_M : i \mapsto \gamma_i$. Let us set

$$I(x) := \mu_i \text{ev}(T(\gamma_i), x) - 1 \quad \text{and} \quad \gamma(x) := \gamma_{I(x)}.$$

Clearly, $I : R^k \circlearrowright \mathbb{N}$ and $\gamma : R^k \circlearrowright \Gamma_M$ are scalar-BSS-recursive. Because an input $x \in R^k$ belongs to $\text{Dom } f$ iff it follows a computational path, we have that $\text{Dom } f = \text{Dom } I = \text{Dom } \gamma$. In fact, $\gamma(x)$ is the computational path followed by x (if such one exists). If $y \in R$ is

the output variable of M , the result of a computation starting with x is given by how y depends on x along the path $\gamma(x)$. That is given by $P(\gamma(x), y) \in \mathbb{Z}[x]$. Thus

$$f(x) = \text{ev}(P(\gamma(x), y), x)$$

wich implies that f is scalar-BSS-recursive. □

Corollary 7. $\mathcal{R}(R^k, R^\ell) = \mathcal{R}^1(R^k, R^\ell)$ for any $k \in \mathbb{N} \cup \{\infty\}$ and any finite ℓ .

Proof. Let $f \in \mathcal{R}(R^k, R^\ell)$. One can write $f = (f_1, \dots, f_\ell)$. Since $f_i = \text{pr}_i \circ f \in \mathcal{R}(R^k, R)$, the preceding theorem shows that $f_i \in \mathcal{R}^1(R^k, R)$. Then the **juxtaposition rule (8)** says that $f \in \mathcal{R}^1(R^k, R^\ell)$. □

For functions valued in R^∞ , the theorem does not hold unless we allow scalar recursive functions to be build using an additional operation. Let us describe it.

(16) *vectorization* : If $F : R \times R^k \multimap R$ and $L : R^k \multimap \mathbb{N} \subset R$ are recursive functions such that

$$D := \{x \in R^k : \forall n \in \mathbb{N}_0, (n, x) \in \text{Dom } F\} \subset \text{Dom } L,$$

$$\forall x \in D, \forall n > L(x), F(n, x) = 0,$$

then the function $\text{vect } F : R^k \multimap R$ is recursive, where $\text{vect } F$ is defined by

$$(\text{vect } F)(x) = (F(n, x))_{n=1}^\infty$$

with domain $\text{Dom}(\text{vect } F) = D$.

Theorem 8. Let $k \in \mathbb{N} \cup \{\infty\}$. The set $\mathcal{R}(R^k, R^\infty)$ coincide with $\mathcal{R}^1(R^k, R^\infty)$ if and only if we allow *rule (16)* to be used to construct \mathcal{R}^1 .

Proof. Necessity. We need to show that (16) is a rule that is valid in \mathcal{R} . Let us start by showing that the function

$$\text{put} : R \times R \times R^\infty \rightarrow R^\infty : (n, \xi, x) \mapsto (y_i)_{i=1}^\infty$$

where $y_i = x_i$ if $i \neq n$ and $y_n = \xi$, with domain $\mathbb{N} \times R \times R^\infty$, is a BSS-recursive function. It results from the fact that it is the following composition of functions

$$\begin{array}{ccccccc} \mathbb{N} \times R \times R^\infty & \xrightarrow{1 \times \mathbb{1}} & \mathbb{N} \times \mathbb{N} \times R^\infty & \xrightarrow{\sigma} & R^\infty & \xrightarrow{\text{shift}} & R^\infty \\ (n, \xi, x) & \longmapsto & (1, n, (\xi, x)) & \longmapsto & (\xi, \text{put}(n, \xi, x)) & \longmapsto & \text{put}(n, \xi, x) \end{array}$$

where, for the first map, 1 denote the constant map with value 1 and we have used $R \times R^\infty \cong R^\infty$, and the last map is the left shift defined by $\text{shift}(x) := s(\text{length}(x) + 1, x)$ with $s : \mathbb{N} \times R^\infty \rightarrow R^\infty$ being given by

$$\begin{cases} s(0, x) = x, \\ s(n + 1, x) = \sigma(n + 2, n + 1, s(n, x)). \end{cases}$$

Now, let $f : \mathbb{N} \times R^\infty \rightarrow R^\infty$ be defined by the recursion

$$\begin{cases} f(0, x) = 0, \\ f(n + 1, x) = \text{put}(x, F(n, x), f(n, x)). \end{cases}$$

Then, $(\text{vect } F)(x) = f(L(x), x)$ and we are done.

Sufficiency. This part comes from the fact that we are able to get an upper bound on the length of the result without actually computing it.

More precisely, let $f \in \mathcal{R}(R^k, R^\infty)$. It is easy to see that **Lemma 4** extends to this case and so one may assume that the machine M computing f has its coefficients in \mathbb{Z} . Let us call $x \in R^k$ the input variable and $y \in R^\infty$ the output one. Let also P and T be defined as above. In the same way as in the proof of **theorem 2**, we can define the map $R^k \circlearrowright \Gamma_M : x \mapsto \gamma(x)$ that gives the path associated with an input $x \in R^k$. Then the following map

$$\varphi : \mathbb{N} \times R^k \circlearrowright R : (n, x) \mapsto \text{ev}(P(\gamma(x), y_n), x)$$

is scalar-BSS-recursive. Moreover $\text{Dom } \varphi = \mathbb{N} \times \text{Dom } \gamma = \mathbb{N} \times \text{Dom } f$. On the other hand, given a path $\gamma \in \Gamma_M$, let $L(\gamma)$ be the maximum $n \in \mathbb{N}$ such that y_n depends on x . The map $\Gamma_M \rightarrow \mathbb{N} : \gamma \mapsto L(\gamma)$ is Turing-computable because it suffices to symbolically compose the polynomials giving y as a function of x along the path γ to see which is the greater component of y that is involved. Clearly, one has the following property:

$$\forall x \in \text{Dom } f, \forall n > L(\gamma(x)), \quad \varphi(n, x) = 0.$$

We can therefore use vectorization on φ . Since

$$f = \text{vect } \varphi,$$

the map f is scalar-recursive and the proof is complete. \square

To conclude this paper, we would like to show that the use of the vectorization rule is necessary in the sense that, without it, one can compute all the components of f but not pack them into a single vector. In other words, if we denote $\mathcal{R}_{\text{vect}}^1$ the set of scalar-BSS-recursive functions build using vectorization, we claim that

$$\mathcal{R}^1(R^k, R^\infty) \subsetneq \mathcal{R}(R^k, R^\infty) = \mathcal{R}_{\text{vect}}^1(R^k, R^\infty).$$

This will result from the following property of functions in \mathcal{R}^1 .

Lemma 9. *If $f : R^k \multimap R^\ell$ belongs to \mathcal{R}^1 , there exists a finite number of recursive functions $a_1, \dots, a_N : R^k \multimap R$ such that*

$$(17) \quad \text{Dom } f \subset \bigcap_{n=1}^N \text{Dom } a_n,$$

$$(18) \quad \forall x \in \text{Dom } f, \forall j \in \mathbb{N}_0^{\leq \ell}, \\ f_j(x) \in \{x_i : i \in \mathbb{N}_0^{\leq k}\} \cup \{a_1(x), \dots, a_N(x)\}$$

To ease the notations, we will set, for all $z \in R^m$, $\mathcal{S}(z) := \{z_i : i \in \mathbb{N}_0^{\leq m}\}$. With this, we can write (18) as

$$\forall x \in \text{Dom } f, \quad \mathcal{S}(f(x)) \subset \mathcal{S}(x) \cup \{a_1(x), \dots, a_N(x)\}.$$

Of course, that property is only of interest if $\ell = \infty$ for otherwise it is always satisfied (take $a_n = f_n$ for $n \leq \ell =: N$).

Proof. We must see that the basic functions (2)–(6) satisfy that property and that it is preserved by (7)–(10). As just said, if the target of the function is not R^∞ , the property automatically holds. So (2), (3), the projections of (4), (6), (9) with $\ell = 1$, and (10) need no further argument. Let us examine the other cases.

- For the inclusion $i_k : R^k \multimap R^\infty$ of (4), they trivially satisfy the above property because $\mathcal{S}(i_k(x)) \subset \mathcal{S}(x)$.
- For the map σ , it is also obvious as $\mathcal{S}(\sigma(n, m, x)) \subset \mathcal{S}(x)$.
- For the composition, suppose $f : R^k \multimap R^\ell$ and $g : R^\ell \multimap R^m$ are such that $\mathcal{S}(f(x)) \subset \mathcal{S}(x) \cup \{a_1(x), \dots, a_N(x)\}$ and $\mathcal{S}(g(y)) \subset \mathcal{S}(y) \cup \{b_1(y), \dots, b_M(y)\}$ where $a_i : R^k \multimap R$ and

$b_j : R^\ell \circlearrowright R$ are recursive. Then

$$\begin{aligned} \mathcal{S}((g \circ f)(x)) &\subset \mathcal{S}(f(x)) \cup \{b_1 \circ f(x), \dots, b_M \circ f(x)\} \\ &\subset \mathcal{S}(x) \cup \{a_1(x), \dots, a_N(x), \\ &\quad b_1 \circ f(x), \dots, b_M \circ f(x)\}. \end{aligned}$$

- It remains to deal with the juxtaposition rule. Let $f : R^k \circlearrowright R^\ell$ and $g : R^k \circlearrowright R^m$ be such that $\mathcal{S}(f(x)) \subset \mathcal{S}(x) \cup \{a_1(x), \dots, a_N(x)\}$ and $\mathcal{S}(g(x)) \subset \mathcal{S}(x) \cup \{b_1(x), \dots, b_M(x)\}$. Then

$$\begin{aligned} \mathcal{S}((f, g)(x)) &= \mathcal{S}(f(x)) \cup \mathcal{S}(g(x)) \\ &\subset \mathcal{S}(x) \cup \{a_1(x), \dots, a_N(x), b_1(x), \dots, b_M(x)\} \end{aligned}$$

□

Using this Lemma, we will show that $\mathcal{R}^1(R, R^\infty) \subsetneq \mathcal{R}(R, R^\infty)$. Let us consider the map

$$f : R \circlearrowright R^\infty : x \mapsto (1, 2, 3, \dots, x, 0, 0, \dots) \text{ if } x \in \mathbb{N}.$$

It is recursive because $f = \text{vect } F$ with $F(n, x) = n \chi_{\geq 0}(x - n) \text{ int}(x)$ and $\text{int}(x) = 1$ if $x \in \mathbb{N}$ and is undefined otherwise. However, $\mathcal{S}(f(x))$ can possess an arbitrary large number of elements so it cannot satisfy (18).

References

- [1] Lenore BLUM, Mike SHUB, Steve SMALE, On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines, *Bulletin of the A.M.S.*, Vol. **21**, N. 1, July 1989, 1–46.

- [2] Lenore BLUM, Felipe CUCKER, Michael SHUB, Steve SMALE, *Complexity and Real Computation*, Springer-Verlag New York, Inc., 1998.
- [3] Lou VAN DEN DRIES, Tame topology and o-minimal structures, *LMS Lecture Note Series* **248** CUP (1998), in particular p. 69–77 and p. 132.
- [4] Jean-Sylvestre GAKWAYA, Christian MICHAUX, Universal function for BSS-Grzegorzcyk hierarchy of relations, *technical report in NeuroCOLT series*, Royal Holloway (2000), 37 pages.
- [5] Jean-Sylvestre GAKWAYA, *Extensions de la Hiérarchie de Grzegorzcyk dans le modèle de calculabilité de Blum, Shub et Smale*, Thèse de doctorat, Université de Mons-Hainaut, 1999.
- [6] Christian MICHAUX, Christophe TROESTLER, Isomorphism theorem for BSS recursively enumerable sets over real closed fields, *Theoretical Computer Science*, **231** (2000), 253–273.
- [7] Michael RABIN, Computable Algebra, General Theory and Theory of computable fields, *Trans. Amer. Math. Soc.*, **95** (1960), 341–360.
- [8] H. E. ROSE, *Subrecursion, Functions and Hierarchies*, School of Mathematics, University of Bristol, Clarendon Press, Oxford, 1984.

Recent preprints

- [1] Maurice Boffa's 60th Birthday Workshop, March 23, 2000.
- [2] Catherine FINET, *Perturbed minimization principles in partially ordered Banach spaces*, June 29, 2000.
- [3] Arnaud MAES, Corinne CERF, *A family of brunnian links based on Edwards' construction of Venn diagrams*, July 15, 2000.
- [4] Catherine FINET & Lucas QUARTA & Christophe TROESTLER, *Vector-valued Variational Principles*, January 19, 2000.
- [5] Gilles GODEFROY, *Montons les degrés*, 8 mars 2001.
- [6] Paul VAN PRAAG, *Quaternions as reflexive skew fields*, 22 mars 2001.
- [7] Maurice BOFFA, *Théorie des ensembles et dualité*, 17 avril 2001.
- [8] Paul VAN PRAAG, Pedro Nuñez, Simon Stevin, *et le plus grand commun diviseur des polynômes*, 17 avril 2001.

You can find more informations as well as download the preprints of the *Institut de Mathématique* on the web site: <http://www.umh.ac.be/math/preprints/>. Printed copies are available upon request by writing to:

Institut de Mathématique
Université de Mons-Hainaut
« Le Pentagone », 6 av. du champ de Mars
7000 Mons, Belgique