An introduction to a model of abstract computation: the BSS-RAM model

Christine Gaßner
Universität Greifswald, Germany (2019)

Abstract

We give a detailed definition of BSS RAM's for the sequential computation over first-order structures. These random access machines are abstract machines with an input procedure and an output procedure. For describing algorithms, all operations and relations of the underlying structure are available at unit cost. Each program of a machine is an element of a formal language whose syntax is defined by syntactic rules. The semantics of any program is dependent on a transition system that results from the interpretation of the symbols in the program by means of the underlying structure. Since any machine has its own program, it is not necessary to have universal machines in this framework, but it can be useful to have them. Therefore, we also discuss the possibility to define universal programs for a large number of mathematical structures and some consequences.

1 Introduction

For modelling the computation over algebraic structures supplemented with some relations, there are a lot of machine-oriented models such as the uniform models presented in [20, 17, 13] and the machine-independent models of abstract computation studied in generalised recursion theory (see e.g. [19]). For structures over the real numbers, we know machine-oriented models such as the real RAM's introduced on the basis of the concept presented in [23, 1], the modified real RAM model considered in [7], and the uniform Blum-Shub-Smale (BSS) machines introduced in [3]. A common feature of these machines is that each real number can be stored as a single unit during the computation. They are suitable for describing algorithms and analysing their complexity under the assumption that some operations and relations can be used as primitive operations and relations at unit cost. However, the comparability of the results is difficult since, for many models, the possibilities of modelling and computation are not sufficiently investigated. [5, 4, 17, 7, 6] can help to get an overview of the great variety of models.

For example, there are a lot of different real-RAM models. The first real RAM's were considered by Preparata and Shamos in [21]. The authors adopted the definition of the random access machine that was introduced for processing integers in [1]. In analogy with this model, many real RAM's work with an instruction for reading one input value and copying the real number to a register and an instruction for writing the content of one register on an output tape. Consequently, the evaluation of the complexity of algorithms on the basis of real RAM's refers often to an arbitrary fixed number of input values since the uniform treatment of all sequences of input values by one machine or one program is neither required nor expected. In contrast to that, the BSS machines use an input procedure that allows an exact mathematical description of the single steps of a uniform computational process for all possible finite sequences of real numbers without any restriction of the input space. It has been confirmed that the BSS model is suitable for developing a theory of computation and complexity over the real numbers. That is one reason why we continue this approach.

We present a random-access-machine model of abstract computation over a first-order structure including the uniform machines. These machines, the so-called BSS RAM's, can use every operation of the underlying structure as a primitive operation and the relations of the structure for changing their state. Our model of computation has been developed on the basis of the concept presented by Scott in [22] and the transition systems defined in [5]. It is a generalisation of the BSS model of computation over a ring and it was used, for instance, in [15, 16]. More precisely, we extended the model considered by Börger [5] in order to get a uniform model in the sense of the BSS model. This means that several types of instructions are now available for accessing memory and for using the indirect addressing techniques. However, the capabilities with respect to indirect addressing are so limited that the BSS RAM's over a finite structure (containing only a finite number of elements) have a power that is comparable to the power of the Turing machine. Over the ordered ring $(\mathbb{R}; \mathbb{R}; +, -, \cdot; \leq)$, where all real numbers can be used as constants, the BSS RAM's have the same computational power as the BSS machines over this ring.

Here, we continue the investigation in [14] and discuss the concept of universal programs and machines for several structures. We present a model of abstract computation without taking questions about a possible digitisation into consideration and we formalise the concept of *algorithm* as precisely and extensively as possible in our framework. This means that we provide a

general framework for describing sequential computations by a uniform transition model and investigating the power of machines that can perform each operation and use each relation of the underlying structures, by assumption, in the same time unit. We hope this model helps to better understand the different approaches to solve or analyse algorithmic problems, the meaning of different models of sequential computation, and the common properties and the differences of these models.

In Section 2 we give a detailed definition and description of $\mathcal{A}$-machines and BSS RAM's over any first-order structure $\mathcal{A}$ of signature σ. The discussion includes σ-programs, the use of pseudo instructions, and multi-tape machines. Section 3 presents the concept of universal σ-programs and universal BSS RAM's and deals with questions concerning the meaning of universal BSS RAM's for the completeness of decision problems.

We use the usual mathematical notations. $\mathbb{N}$, $\mathbb{N}_+$, and $\mathbb{R}$ are the sets of non-negative integers, positive integers, and real numbers, respectively. $=_{\rm df}$ means "is equal by definition". In defining sets we use the usual symbols such as $\forall, \exists, \&, \in, \max, \leq, \ldots$. $\{x \mid \phi(x)\}$ is the set of all objects x satisfying the property described by $\phi(x)$. The strings $^{(0)}$, $^{(1)}$, $^{(i)}$, $_i, \ldots$ are indices and placeholders for indices, respectively. For arithmetic expressions t such as $i+1$, $_t$ is usually the integer index resulting from the evaluation of t. $i \in \{1, \ldots, n\}$ and $i \leq n$ means that i is an integer with $1 \leq i \leq n$ and $i \geq 1$ means $i \in \mathbb{N}_+$, and so on. $f :\subseteq M \to N$ means that f is a partially defined function from M into N. We write $f : M \to N$ only if f is a totally defined function. For $f : M \to N$, let $f_i = f(i)$ for all $i \in M$ and, for $M = \mathbb{N}$ and $n \geq 1$, let $f_1, \ldots, f_n$ be the list of values assigned to the integers 1 to n by f in this order. Let $M^\infty = \bigcup_{n \geq 1} M^n$ be the set of all tuples $\vec{x} =_{\rm df} (x_1, \ldots, x_n)$ with $x_i \in M$ $(i \leq n)$ and $n \geq 1$ and let M^ω be the set of all infinite sequences $\bar{u} =_{\rm df} (u_1, u_2, \ldots)$ with $u_i \in M$ for all $i \geq 1$. For $n = 0$, let $x_1, \ldots, x_n$ and $\vec{x}$ be the empty list, and so on. For all $\vec{x} \in M^n$, $\vec{y} \in N^m$, and $\bar{u} \in N^\omega$, let $(\vec{x}\,.\,\vec{y}) = (x_1, \ldots, x_n, y_1, \ldots, y_m) \in (M \cup N)^{n+m}$ and $(\vec{x}\,.\,\bar{u}) = (x_1, \ldots, x_n, u_1, u_2, \ldots) \in (M \cup N)^\omega$. We use, for $d \geq 1$, $(\vec{x}^{(j)})_{j=1..d} =_{\rm df} (\vec{x}^{(1)}, \ldots, \vec{x}^{(d)})$ and $(\bar{u}^{(j)})_{j=1..d} =_{\rm df} (\bar{u}^{(1)}, \ldots, \bar{u}^{(d)})$ and, for all $j \geq 1$, $\vec{x}^{(j)} \in M^n$, and $\bar{u}^{(j)} \in N^\omega$, let $\vec{x}^{(j)} = (x_{j,1}, \ldots, x_{j,n})$ and $\bar{u}^{(j)} = (u_{j,1}, u_{j,2}, \ldots)$. For any $f :\subseteq M^\infty \to N$, let $f(x_1, \ldots, x_n) = f(\vec{x})$. For $g :\subseteq M \to N$ and $f :\subseteq N \to O$, $f \circ g :\subseteq M \to O$ is given by $(f \circ g)(x) = f(g(x))$ for all $x \in M$ and, thus, defined for those $x \in M$ for which both $g(x)$ and $f(g(x))$ are defined. Let $f_{g_x} = f(g(x))$. $\{\alpha_1, \ldots, \alpha_m\}^*$ $(m \geq 1)$ consists of the *empty string* Λ and all *strings* $\alpha_{i_1} \cdots \alpha_{i_l}$ with $l \geq 1$ and $i_1, \ldots, i_l \leq m$.

2 The BSS-RAM model

2.1 The idea and the background

For any algebraic structure $\mathcal{A}$, every $\mathcal{A}$-machine has its own program. It can perform each operation of the underlying structure $\mathcal{A}$ in a single step of computation. The BSS RAM's over a structure $\mathcal{A}$ are $\mathcal{A}$-machines that get as inputs arbitrary finite sequences of elements of $\mathcal{A}$. Figure 1 shows the essential processing steps for the computation of a function. The inputted values are assigned to registers by an input procedure. Afterwards, the machine executes its program defined by a finite sequence of labelled instructions until a stop criterion is satisfied. If the stop criterion is reached, then the machine halts and the computed values can be outputted by means of an output procedure. If $\mathcal{A}$ is a structure with the universe $U_{\mathcal{A}}$, we will say that a partially defined function f from $U_{\mathcal{A}}^{\infty}$ into $U_{\mathcal{A}}^{\infty}$ is computable by a BSS RAM over $\mathcal{A}$ if there is a BSS RAM over $\mathcal{A}$ that halts on input $(x_1, \ldots, x_n) \in U_{\mathcal{A}}^{\infty}$ if and only if the value of f is defined for $(x_1, \ldots, x_n)$, and then it outputs the value $f(x_1, \ldots, x_n)$.

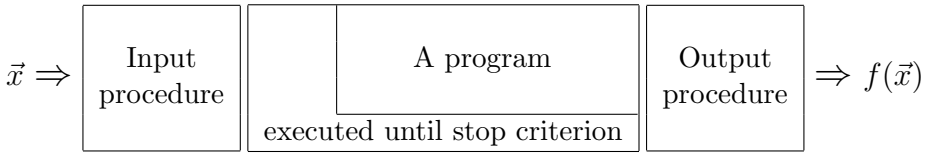

Figure 1: The BSS-RAM model

In general, we use *infinite dimensional* machines $\mathcal{M}$ over $\mathcal{A}$ that are equipped with an infinite number of registers $Z_1, Z_2, \ldots$ for storing elements of $U_{\mathcal{A}}$, a finite number of registers $I_1, I_2, \ldots, I_{k_{\mathcal{M}}}$ for storing indices in $\mathbb{N}_+$, and an auxiliary register B for storing an instruction counter, the label of the current instruction (see Figure 2). We will call $Z_1, Z_2, \ldots$ the *Z-registers*. But sometime we use *finite dimensional* machines $\mathcal{M}$ over $\mathcal{A}$ that are only equipped with a finite number of Z-registers $Z_1, Z_2, \ldots, Z_{n_{\mathcal{M}}}$ for storing the elements of $U_{\mathcal{A}}$ and a register B for the instruction counter. Such machines $\mathcal{M}$ can compute functions from $U_{\mathcal{A}}^n$ into $U_{\mathcal{A}}^m$ for some fixed $n, m \leq n_{\mathcal{M}}$.

The index registers are important if a machine uses an infinite number of Z-registers. They are used for storing the length of the input, for determining the length of the output, and in copy instructions. The *infinite dimensional machines* can copy the content of one Z-register to another Z-register by

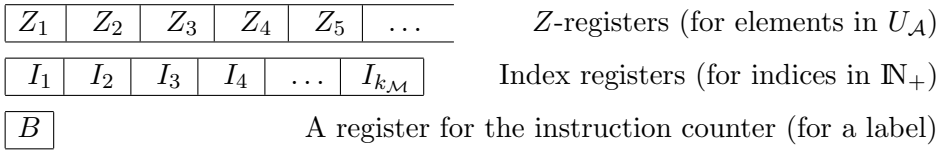

Figure 2: The registers of an $\mathcal{A}$-machine

means of indirect addressing where each address that may be used in the copy process must be stored in an index register. The content of an index register can only be changed with the help of several types of index instructions.

The underlying structure $\mathcal{A}$ can be an arbitrary first-order structure as defined e.g. in [2, p. 22]. Let $\mathcal{A} = (U; (c_i)_{i\in N_1}; (f_i)_{i\in N_2}; (r_i)_{i\in N_3})$ where U is a non-empty set of individuals and, for suitable sets N_1, N_2, and N_3, $(c_i)_{i\in N_1}$ is a family of constants $c_i \in U$, $(f_i)_{i\in N_2}$ is a family of operations f_i of arity m_i, and $(r_i)_{i\in N_3}$ is a family of relations $r_i \subseteq U^{k_i}$ of arity k_i. Let $U_\mathcal{A} = U$. The *signature of* $\mathcal{A}$ is $(|N_1|; (m_i)_{i\in N_2}; (k_i)_{i\in N_3})$. In general, any operation f_i is an everywhere defined function from $U_\mathcal{A}^{m_i}$ into $U_\mathcal{A}$. But it is also possible to permit partial functions $f_i :\subseteq U_\mathcal{A}^{m_i} \to U_\mathcal{A}$. This could be interesting if we wanted to use certain computable functions as primitive operations. Any structure $\mathcal{B} = (U_\mathcal{A}; (c_i)_{i\in L_1}; (f_i)_{i\in L_2}; (r_i)_{i\in L_3})$ with $L_1 = \{i_1, \ldots, i_{n_1}\} \subseteq N_1$, $L_2 = \{j_1, \ldots, j_{n_2}\} \subseteq N_2$, and $L_3 = \{l_1, \ldots, l_{n_3}\} \subseteq N_3$ (for some $n_1, n_2, n_3 \geq 0$) is a *reduct* of $\mathcal{A}$ and a structure of signature $\sigma =_{\mathrm{df}} (n_1; m_{j_1}, \ldots, m_{j_{n_2}}; k_{l_1}, \ldots, k_{l_{n_3}})$. For such a structure, we use also the notation $(U_\mathcal{A}; c_{i_1}, \ldots, c_{i_{n_1}}; f_{j_1}, \ldots, f_{j_{n_2}}; r_{l_1}, \ldots, r_{l_{n_3}})$. σ is a *finite signature* and a *subsignature* of the signature of $\mathcal{A}$.

Since each program is a finite string that contains only a finite number of symbols, we define a set of programs for every finite signature. Now, let $\sigma = (n_1; m_1, \ldots, m_{n_2}; k_1, \ldots, k_{n_3})$ and $n_1, n_2, n_3 \geq 0$. Let P_σ be the set of all σ-programs defined as follows. Every *σ-program* $\mathcal{P}$ has the following form.

$$1 : \mathsf{instruction}_1;\ 2 : \mathsf{instruction}_2; \ldots;\ \ell_\mathcal{P} - 1 : \mathsf{instruction}_{\ell_\mathcal{P}-1};\ \ell_\mathcal{P} : \mathsf{stop}.$$

It is a string that consists of a finite number of substrings, each of which starts with a *label* $i \in \{1, \ldots, \ell_\mathcal{P}\}$ followed by the symbol : and a string denoted by $\mathsf{instruction}_i$ and called *σ-instruction*. Here and in the following, each arithmetic expression before the symbol : such as $\ell_\mathcal{P} - 1$ stands for the integer resulting from the evaluation of this expression. For each $\mathcal{P}$, there is some $l \geq 1$ such that $\mathcal{P}$ results from the application of a syntactic rule to

each of the placeholders $\langle instruction \rangle$ in a string of the form

$$1 : \langle instruction \rangle;\ 2 : \langle instruction \rangle;\ \ldots;\ l-1 : \langle instruction \rangle;\ l : \mathsf{stop}. \quad (*)$$

Let $\ell_{\mathcal{P}} = l$. We distinguish 8 types of σ-instructions. $\langle instruction \rangle$ can be replaced by an instruction of one of the types $(1), \ldots, (7)$ given after the labels in Overview 1 where all $j, k, j_1, j_2, \ldots$ are placeholders for positive integers, each ℓ is a placeholder for a label in $\{1, \ldots, l-1\}$, ℓ_1 and ℓ_2 are placeholders for labels in $\{1, \ldots, l\}$, and every i stands for a positive integer that is less than or equal to n_1, n_2, and n_3, respectively. The names of the types are given in parentheses at the end of the lines.

Overview 1 (σ-instructions)

Computation instructions:		
(1)	$\ell :\ Z_j := f_i^{m_i}(Z_{j_1}, \ldots, Z_{j_{m_i}})$	(F-instructions)
(2)	$\ell :\ Z_j := c_i^0$	(F_0-instructions)
Copy instructions:		
(3)	$\ell :\ Z_{I_j} := Z_{I_k}$	(C-instructions)
Branching instructions:		
(4)	$\ell :$ if $r_i^{k_i}(Z_{j_1}, \ldots, Z_{j_{k_i}})$ then goto ℓ_1 else goto ℓ_2	(T-instructions)
Index instructions:		
(5)	$\ell :$ if $I_j = I_k$ then goto ℓ_1 else goto ℓ_2	(H_{T}-instructions)
(6)	$\ell :\ I_j := 1$	(H_1-instructions)
(7)	$\ell :\ I_j := I_j + 1$	(H_{+1}-instructions)
Stop instruction:		
(8)	$l :$ stop	(S-instruction)

Purely formally, the instructions are only strings and we must distinguish between the symbols in a program as presented in Overview 1 and their interpretation determined by a structure of signature σ. This means that one symbol for a binary operation, f_i^2, can stand for the addition or for the multiplication, and so on. Strictly speaking, we always distinguish between each symbol $f_i^{m_i}$ for an m_i-ary operation in a program and its interpretation, between the symbols $r_i^{k_i}$ and the used k_i-ary relations, and between c_i^0 and the constants themselves.

If a function can be computed by means of a finite number of Z-registers, then few *copy instructions* of the form $Z_j := Z_k$ with fixed addresses j and k could be sufficient. The *σ-programs for finite dimensional machines* are restricted to instructions of the form $Z_j := f_i^{m_i}(Z_{j_1}, \ldots, Z_{j_{m_i}})$, $Z_j := c_i^0$, $Z_j := Z_k$, and if $r_i^{k_i}(Z_{j_1}, \ldots, Z_{j_{k_i}})$ then goto ℓ_1 else goto ℓ_2.

2.2 The formal definition of $\mathcal{A}$-machines

For any structure $\mathcal{A}$, a machine over $\mathcal{A}$ should be able to execute a σ-program $\mathcal{P}$ if $\mathcal{A}$ includes a reduct for interpreting all symbols of constants, operations, and relations occurring in $\mathcal{P}$ and if there is a $k \leq k_{\mathcal{M}}$ such that any index after a substring "I" in $\mathcal{P}$ is a number less than or equal to k. Let $k_{\mathcal{P}}$ be the smallest such k. Consequently, we can formally define an $\mathcal{A}$-machine as follows.

Definition 1 (Infinite dimensional $\mathcal{A}$-machine [14]) *Let a structure $\mathcal{A}$, any sets I and O, and a signature $\sigma = (n_1; m_1, \ldots, m_{n_2}; k_1, \ldots, k_{n_3})$ be given. A tuple $(U^\omega, (\mathbb{N}_+)^k, \mathcal{L}, \mathcal{P}, \mathcal{B}, \mathrm{In}, \mathrm{Out})$ consisting of*

- *a space of memory states, U^ω,*
- *a space of addresses (or indices), $(\mathbb{N}_+)^k$, with $k \geq 1$,*
- *a set of labels, $\mathcal{L} = \{1, \ldots, l\}$, with $l \geq 1$,*
- *a program $\mathcal{P} \in \mathsf{P}_\sigma$ with $\ell_{\mathcal{P}} = l$ and $k_{\mathcal{P}} \leq k$,*
- *a structure $\mathcal{B} = (U; c_1, \ldots, c_{n_1}; f_1, \ldots, f_{n_2}; r_1, \ldots, r_{n_3})$ of signature σ,*
- *an input function $\mathrm{In} : \mathsf{I} \to \{(\vec{\nu} \,.\, \bar{u}) \mid (\vec{\nu}, \bar{u}) \in (\mathbb{N}_+)^k \times U^\omega\}$,*
- *an output function $\mathrm{Out} : \{(\vec{\nu} \,.\, \bar{u}) \mid (\vec{\nu}, \bar{u}) \in (\mathbb{N}_+)^k \times U^\omega\} \to \mathsf{O}$*

is an infinite dimensional machine over $\mathcal{A}$ *(or* $\mathcal{A}$-machine*) with the machine constants $c_1, \ldots, c_{n_1}$, the input space I, and the output space O if $\mathcal{B}$ is a reduct of $\mathcal{A}$.*

Let $\mathsf{IM}_{\mathcal{A}}$ be the class of all infinite dimensional $\mathcal{A}$-machines. For any machine $\mathcal{M} = (U^\omega, (\mathbb{N}_+)^k, \mathcal{L}, \mathcal{P}, \mathcal{B}, \mathrm{In}, \mathrm{Out})$ with $\mathcal{L} = \{1, \ldots, \ell_{\mathcal{P}}\}$, let $k_{\mathcal{M}} = k$, $\mathcal{L}_{\mathcal{M}} = \mathcal{L}$, $\mathcal{P}_{\mathcal{M}} = \mathcal{P}$, $\mathcal{B}_{\mathcal{M}} = \mathcal{B}$, $\mathrm{In}_{\mathcal{M}} = \mathrm{In}$, and $\mathrm{Out}_{\mathcal{M}} = \mathrm{Out}$. Moreover, let the spaces $\mathsf{I}_{\mathcal{M}}$ and $\mathsf{O}_{\mathcal{M}}$ be the domain of $\mathrm{In}_{\mathcal{M}}$ and the codomain of $\mathrm{Out}_{\mathcal{M}}$, respectively, and $\vec{c}^{(\mathcal{M})}$ be the tuple $(c_1, \ldots, c_{n_1})$ of the constants of $\mathcal{B}_{\mathcal{M}}$.

The overall state of $\mathcal{M} \in \mathsf{IM}_{\mathcal{A}}$ results from the values assigned to the registers as presented in Figure 3. It is given by a configuration and can be changed by executing the program $\mathcal{P}_{\mathcal{M}}$.

Definition 2 (Configuration) *Any possible* configuration *of an $\mathcal{A}$-machine $\mathcal{M}$ is given by a sequence $(\ell \,.\, \vec{\nu} \,.\, \bar{u})$ where ℓ is a label in $\mathcal{L}_{\mathcal{M}}$ and $\vec{\nu} = (\nu_1, \ldots, \nu_{k_{\mathcal{M}}}) \in (\mathbb{N}_+)^{k_{\mathcal{M}}}$ and $\bar{u} \in U_{\mathcal{A}}^\omega$ hold. If $\ell = 1$, then we call $(\ell \,.\, \vec{\nu} \,.\, \bar{u})$ an* initial configuration*. If $\ell = \ell_{\mathcal{P}}$, then we call $(\ell \,.\, \vec{\nu} \,.\, \bar{u})$ a* stop configuration*.*

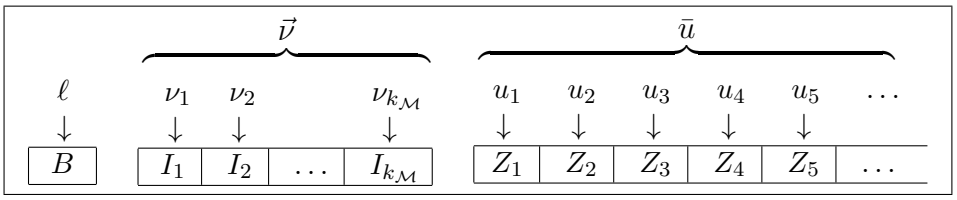

Figure 3: The assignment of values to the registers of an $\mathcal{A}$-machine $\mathcal{M}$

For any $\mathcal{M} \in \mathsf{IM}_\mathcal{A}$, let $\mathsf{S}_\mathcal{M} = \{(\ell\,.\,\vec{\nu}\,.\,\bar{u}) \mid \ell \in \mathcal{L}_\mathcal{M}\ \&\ \vec{\nu} \in (\mathbb{N}_+)^{k_\mathcal{M}}\ \&\ \bar{u} \in U_\mathcal{A}^\omega\}$ be the space of all possible configurations of $\mathcal{M}$. For any register R, let the *content of* R be the value stored in R and let it be denoted by $c(R)$. Let $\mathcal{M} \in \mathsf{IM}_\mathcal{A}$. In case that, for any Z-register Z_i of $\mathcal{M}$ (with $i \geq 1$), the current content $c(Z_i)$ of Z_i is u_i and, for any register I_j of $\mathcal{M}$ (with $j \leq k_\mathcal{M}$), the current content $c(I_j)$ of I_j is ν_j, and the current content $c(B)$ of the register B of $\mathcal{M}$ is ℓ, the configuration $(\ell\,.\,\vec{\nu}\,.\,\bar{u}) \in \mathsf{S}_\mathcal{M}$ describes the current state of $\mathcal{M}$. $\mathcal{M}$ should work similarly as a usual finite machine equipped with a finite number of Z-registers, however there are a few things one needs to make clear if one wants to know the complete current state at any given time $t \geq 1$. That is the reason why we now define a transition system $\mathcal{S}_\mathcal{M}$ for transforming one configuration in $\mathsf{S}_\mathcal{M}$ into the next configuration. For any $\mathcal{M} \in \mathsf{IM}_\mathcal{A}$, let $\mathcal{L}_\mathcal{M} = \mathcal{L}_{\mathcal{M},\mathrm{F}} \cup \mathcal{L}_{\mathcal{M},\mathrm{F}_0} \cup \mathcal{L}_{\mathcal{M},\mathrm{C}} \cup \mathcal{L}_{\mathcal{M},\mathrm{T}} \cup \mathcal{L}_{\mathcal{M},\mathrm{H}_\mathrm{T}} \cup \mathcal{L}_{\mathcal{M},\mathrm{H}_1} \cup \mathcal{L}_{\mathcal{M},\mathrm{H}_{+1}} \cup \mathcal{L}_{\mathcal{M},\mathrm{S}}$ be the union of the pairwise disjoint sets $\mathcal{L}_{\mathcal{M},\mathrm{F}}, \mathcal{L}_{\mathcal{M},\mathrm{F}_0}, \ldots$ consisting of the labels of all F-instructions in $\mathcal{P}_\mathcal{M}$, the labels of all F_0-instructions in $\mathcal{P}_\mathcal{M}$, and so on, respectively, and let $\mathcal{L}_{\mathcal{M},\mathrm{S}} = \{\ell_{\mathcal{P}_\mathcal{M}}\}$. For determining the semantics of $\mathcal{P}_\mathcal{M}$, the following family $\mathcal{F}_\mathcal{M}$ of totally defined functions for changing the labels and the values in the registers of an $\mathcal{A}$-machine $\mathcal{M} \in \mathsf{IM}_\mathcal{A}$ are introduced. The functions are determined by $k_\mathcal{M}$, $\mathcal{P}_\mathcal{M}$, and $\mathcal{B}_\mathcal{M} = (U_\mathcal{A}; c_1, \ldots, c_{n_1}; f_1, \ldots, f_{n_2}; r_1, \ldots, r_{n_3})$. For $\ell \in \mathcal{L}_\mathcal{M}$, let $\phi_{\mathcal{M},\ell}$ be a description such that the ℓ^{th} instruction of $\mathcal{P}_\mathcal{M}$ has exactly the form (including the indices) presented for this type of instructions in Overview 1 if $\phi_{\mathcal{M},\ell}$ holds. For instance, for the instruction $Z_1 := f_2^3(Z_3, Z_1, Z_2)$ labelled by 2, let $\phi_{\mathcal{M},2}$ be the expression $i = 2\ \&\ j = 1\ \&\ j_1 = 3\ \&\ j_2 = 1\ \&\ j_3 = 2$.

Definition 3 (Operations and functions in $\mathcal{F}_\mathcal{M}$) *For any $\mathcal{M} \in \mathsf{IM}_\mathcal{A}$, let the functions in $\mathcal{F}_\mathcal{M}$ be defined as follows.*

- Elementary operations $F_\ell : U_\mathcal{A}^\omega \to U_\mathcal{A}^\omega$

  $F_\ell(\bar{u}) = (u_1, \ldots, u_{j-1}, f_i(u_{j_1}, \ldots, u_{j_{m_i}}), u_{j+1}, \ldots)$ *for* $\ell \in \mathcal{L}_{\mathcal{M},\mathrm{F}}$ *if* $\phi_{\mathcal{M},\ell}$
  $F_\ell(\bar{u}) = (u_1, \ldots, u_{j-1}, c_i, u_{j+1}, \ldots)$ *for* $\ell \in \mathcal{L}_{\mathcal{M},\mathrm{F}_0}$ *if* $\phi_{\mathcal{M},\ell}$

- Copy functions $C_\ell : (\mathbb{N}_+)^{k_\mathcal{M}} \times U_\mathcal{A}^\omega \to U_\mathcal{A}^\omega$
  $C_\ell(\vec{\nu}, \bar{u}) = (u_1, \ldots, u_{\nu_j - 1}, u_{\nu_k}, u_{\nu_j + 1}, \ldots)$ *for* $\ell \in \mathcal{L}_{\mathcal{M},\mathrm{C}}$ *if* $\phi_{\mathcal{M},\ell}$
- Test functions $T_\ell : U_\mathcal{A}^\omega \to \mathcal{L}_\mathcal{M}$ *and* $T_\ell : (\mathbb{N}_+)^{k_\mathcal{M}} \to \mathcal{L}_\mathcal{M}$
  $$T_\ell(\bar{u}) = \begin{cases} \ell_1 \text{ if } (u_{j_1}, \ldots, u_{j_{k_i}}) \in r_i \\ \ell_2 \text{ if } (u_{j_1}, \ldots, u_{j_{k_i}}) \notin r_i \end{cases} \quad \text{for } \ell \in \mathcal{L}_{\mathcal{M},\mathrm{T}} \text{ if } \phi_{\mathcal{M},\ell}$$
  $$T_\ell(\vec{\nu}) = \begin{cases} \ell_1 \text{ if } \nu_j = \nu_k \\ \ell_2 \text{ if } \nu_j \neq \nu_k \end{cases} \quad \text{for } \ell \in \mathcal{L}_{\mathcal{M},\mathrm{H_T}} \text{ if } \phi_{\mathcal{M},\ell}$$
- Auxiliary functions $H_\ell : (\mathbb{N}_+)^{k_\mathcal{M}} \to (\mathbb{N}_+)^{k_\mathcal{M}}$
  $H_\ell(\vec{\nu}) = (\nu_1, \ldots, \nu_{j-1}, 1, \nu_{j+1}, \ldots, \nu_{k_\mathcal{M}})$ *for* $\ell \in \mathcal{L}_{\mathcal{M},\mathrm{H}_1}$ *if* $\phi_{\mathcal{M},\ell}$
  $H_\ell(\vec{\nu}) = (\nu_1, \ldots, \nu_{j-1}, \nu_j + 1, \nu_{j+1}, \ldots, \nu_{k_\mathcal{M}})$ *for* $\ell \in \mathcal{L}_{\mathcal{M},\mathrm{H}_{+1}}$ *if* $\phi_{\mathcal{M},\ell}$

In order to complete the concept of computation with respect to our model, we define a computational system by means of $\mathcal{F}_\mathcal{M}$ for any machine $\mathcal{M} \in \mathsf{IM}_\mathcal{A}$.

Definition 4 (Computational system) *For any* $\mathcal{M} \in \mathsf{IM}_\mathcal{A}$, *the* computational system $\mathcal{R}_\mathcal{M} = (\mathcal{S}_\mathcal{M}, \mathrm{Input}_\mathcal{M}, \mathrm{Output}_\mathcal{M}, \mathrm{Stop}_\mathcal{M})$ *is given by*

- *the transition system* $\mathcal{S}_\mathcal{M} = (\mathsf{S}_\mathcal{M}, \to_\mathcal{M})$ *defined by*
  $$\begin{aligned} \to_\mathcal{M} = \; & \{((\ell \,.\, \vec{\nu} \,.\, \bar{u}), (\ell + 1 \,.\, \vec{\nu} \,.\, F_\ell(\bar{u}))) && \in \mathsf{S}^2_\mathcal{M} \mid \ell \in \mathcal{L}_{\mathcal{M},\mathrm{F}} \cup \mathcal{L}_{\mathcal{M},\mathrm{F}_0}\} \\ & \cup \{((\ell \,.\, \vec{\nu} \,.\, \bar{u}), (\ell + 1 \,.\, \vec{\nu} \,.\, C_\ell(\vec{\nu}, \bar{u}))) && \in \mathsf{S}^2_\mathcal{M} \mid \ell \in \mathcal{L}_{\mathcal{M},\mathrm{C}}\} \\ & \cup \{((\ell \,.\, \vec{\nu} \,.\, \bar{u}), (T_\ell(\bar{u}) \,.\, \vec{\nu} \,.\, \bar{u})) && \in \mathsf{S}^2_\mathcal{M} \mid \ell \in \mathcal{L}_{\mathcal{M},\mathrm{T}}\} \\ & \cup \{((\ell \,.\, \vec{\nu} \,.\, \bar{u}), (T_\ell(\vec{\nu}) \,.\, \vec{\nu} \,.\, \bar{u})) && \in \mathsf{S}^2_\mathcal{M} \mid \ell \in \mathcal{L}_{\mathcal{M},\mathrm{H_T}}\} \\ & \cup \{((\ell \,.\, \vec{\nu} \,.\, \bar{u}), (\ell + 1 \,.\, H_\ell(\vec{\nu}) \,.\, \bar{u})) && \in \mathsf{S}^2_\mathcal{M} \mid \ell \in \mathcal{L}_{\mathcal{M},\mathrm{H}_1} \cup \mathcal{L}_{\mathcal{M},\mathrm{H}_{+1}}\} \\ & \cup \{((\ell \,.\, \vec{\nu} \,.\, \bar{u}), (\ell \,.\, \vec{\nu} \,.\, \bar{u})) && \in \mathsf{S}^2_\mathcal{M} \mid \ell = \ell_{\mathcal{P}_\mathcal{M}}\}, \end{aligned}$$
- *the* input procedure $\mathrm{Input}_\mathcal{M} : \mathsf{I}_\mathcal{M} \to \mathsf{S}_\mathcal{M}$ *with* $\mathrm{Input}_\mathcal{M}(\mathsf{i}) = (1 \,.\, \mathrm{In}_\mathcal{M}(\mathsf{i}))$ *for all* $\mathsf{i} \in \mathsf{I}_\mathcal{M}$,
- *the* output procedure $\mathrm{Output}_\mathcal{M} : \mathsf{S}_\mathcal{M} \to \mathsf{O}_\mathcal{M}$ *with* $\mathrm{Output}_\mathcal{M}(\ell \,.\, \vec{\nu} \,.\, \bar{u}) = \mathrm{Out}_\mathcal{M}(\vec{\nu} \,.\, \bar{u})$ *for all* $(\ell \,.\, \vec{\nu} \,.\, \bar{u}) \in \mathsf{S}_\mathcal{M}$,
- *the* stop criterion $\mathrm{Stop}_\mathcal{M}(\ell \,.\, \vec{\nu} \,.\, \bar{u}) = 0$ *satisfied if* $\ell = \ell_{\mathcal{P}_\mathcal{M}}$.

The relation $\to_\mathcal{M}$ determines the transition rules $\mathsf{conf}_1 \to_\mathcal{M} \mathsf{conf}_2$ that are permitted for all $(\mathsf{conf}_1, \mathsf{conf}_2) \in \mathsf{S}^2_\mathcal{M}$ if $(\mathsf{conf}_1, \mathsf{conf}_2) \in \to_\mathcal{M}$. Each instruction causes the change of a configuration by applying a transition rule.

$$(\ell_0 + 1 \,.\, \vec{\nu} \,.\, \underbrace{(u_1, \ldots, u_{j-1}, f_i(u_{j_1}, \ldots, u_{j_{m_i}}), u_{j+1}, \ldots)}_{F_{\ell_0}(\bar{u})}) \quad \text{if } \ell_0 \in \mathcal{L}_{\mathcal{M},\mathrm{F}}$$

$$(\ell_0 + 1 \,.\, \underbrace{(\nu_1, \ldots, \nu_{j-1}, 1, \nu_{j+1}, \ldots, \nu_{k_{\mathcal{M}}})}_{H_{\ell_0}(\vec{\nu})} \,.\, \bar{u}) \quad \text{if } \ell_0 \in \mathcal{L}_{\mathcal{M},\mathrm{H}_1}$$

Figure 4: Examples for a possible successor of the configuration $(\ell_0 \,.\, \vec{\nu} \,.\, \bar{u})$

For instance, the instruction $Z_{I_j} := Z_{I_k}$ labelled by ℓ_0 implies the transformation of $(\ell_0 \,.\, \vec{\nu} \,.\, \bar{u})$ into $(\ell_0 + 1 \,.\, \vec{\nu} \,.\, (u_1, \ldots, u_{\nu_j - 1}, u_{\nu_k}, u_{\nu_j + 1}, \ldots))$ for all $\bar{u} = (u_1, u_2, \ldots) \in U_{\mathcal{A}}^{\infty}$ (for further examples of transformations see Figure 4).

For any $\mathcal{M} \in \mathsf{IM}_{\mathcal{A}}$ and each input $\mathsf{i} \in \mathsf{I}_{\mathcal{M}}$, the system $(\mathsf{S}_{\mathcal{M}}, \to_{\mathcal{M}})$ can be used to generate finite sequences $(\ell_t \,.\, \vec{\nu}^{(t)} \,.\, \bar{u}^{(t)})_{1..s}$ of configurations or an infinite sequence of configurations $(\ell_t \,.\, \vec{\nu}^{(t)} \,.\, \bar{u}^{(t)})$ where we have $(\ell_1 \,.\, \vec{\nu}^{(1)} \,.\, \bar{u}^{(1)}) = \mathrm{Input}_{\mathcal{M}}(\mathsf{i})$ and $(\ell_t \,.\, \vec{\nu}^{(t)} \,.\, \bar{u}^{(t)}) \to_{\mathcal{M}} (\ell_{t+1} \,.\, \vec{\nu}^{(t+1)} \,.\, \bar{u}^{(t+1)})$ and $\ell_t \neq \ell_{\mathcal{P}_{\mathcal{M}}}$ hold for all $t < s$ and $t \geq 1$, respectively. We call the longest such sequences *computation paths*. Informally, we say that $\mathcal{M}$ *goes through a computation path* that is given by the sequence of instructions executed in this order by $\mathcal{M}$ and that $\mathcal{M}$ does it *in exactly t_0 steps* if this path is a sequence of $t_0 + 1$ instructions. We say that $\mathcal{M}$ *halts* on $\mathsf{i} \in \mathsf{I}_{\mathcal{M}}$ if the corresponding maximal sequence contains the S-instruction labelled by $\ell_{\mathcal{P}_{\mathcal{M}}}$. Only in this case, the machine outputs an element of $\mathsf{O}_{\mathcal{M}}$.

Since $\to_{\mathcal{M}}$ is a unique mapping, it is possible to write $(\to_{\mathcal{M}})(\mathsf{conf}_1) = \mathsf{conf}_2$ and $(\to_{\mathcal{M}})^1(\mathsf{conf}_1) = \mathsf{conf}_2$ instead of $\mathsf{conf}_1 \to_{\mathcal{M}} \mathsf{conf}_2$ and to use the composition $(\to_{\mathcal{M}})^{t+1}$ defined by $(\to_{\mathcal{M}})^{t+1}(\mathsf{conf}) = (\to_{\mathcal{M}})((\to_{\mathcal{M}})^t(\mathsf{conf}))$ for all $\mathsf{conf} \in \mathsf{S}_{\mathcal{M}}$ and $t \geq 1$. Moreover, let $(\to_{\mathcal{M}})_{\mathrm{Stop}_{\mathcal{M}}} :\subseteq \mathsf{S}_{\mathcal{M}} \to \mathsf{S}_{\mathcal{M}}$ be a function where $(\to_{\mathcal{M}})_{\mathrm{Stop}_{\mathcal{M}}}(\mathsf{conf}) = (\to_{\mathcal{M}})^{t_0}(\mathsf{conf})$ holds if the set $h_{\mathsf{conf}} = \{t \mid \mathrm{Stop}_{\mathcal{M}}((\to_{\mathcal{M}})^t(\mathsf{conf})) = 0\}$ is not empty and $t_0 = \min h_{\mathsf{conf}}$ holds and otherwise $(\to_{\mathcal{M}})_{\mathrm{Stop}_{\mathcal{M}}}(\mathsf{conf})$ is not defined. Thus, for any finite computation path $(\ell_t \,.\, \vec{\nu}^{(t)} \,.\, \bar{u}^{(t)})_{1..s}$, we get $(\to_{\mathcal{M}})_{\mathrm{Stop}_{\mathcal{M}}}(\ell_1 \,.\, \vec{\nu}^{(1)} \,.\, \bar{u}^{(1)}) = (\ell_s \,.\, \vec{\nu}^{(s)} \,.\, \bar{u}^{(s)})$.

Definition 5 (Result function) *For any $\mathcal{A}$-machine $\mathcal{M}$, the partial function* $\mathrm{Res}_{\mathcal{M}} :\subseteq \mathsf{I}_{\mathcal{M}} \to \mathsf{O}_{\mathcal{M}}$*, defined by*

$$\mathrm{Res}_{\mathcal{M}}(\mathsf{i}) = \mathrm{Output}_{\mathcal{M}} \circ (\to_{\mathcal{M}})_{\mathrm{Stop}_{\mathcal{M}}} \circ \mathrm{Input}_{\mathcal{M}}(\mathsf{i})$$

for all $\mathsf{i} \in \mathsf{I}_{\mathcal{M}}$, is called the result function of $\mathcal{R}_{\mathcal{M}}$ *and* computable over $\mathcal{A}$.

We say that $\mathcal{M}$ *computes* $\mathrm{Res}_{\mathcal{M}}$ and $\mathcal{M}$ *executes the program $\mathcal{P}_{\mathcal{M}}$ on input* i in order to compute $\mathrm{Res}_{\mathcal{M}}(\mathsf{i})$. For any $\mathcal{A}$-machine $\mathcal{M}$, the input function

$\mathrm{In}_{\mathcal{M}}$ and the output function $\mathrm{Out}_{\mathcal{M}}$ can be defined in several ways where the input space and the output space can, for instance, be $U_{\mathcal{A}}^n$ for some $n \geq 1$, $U_{\mathcal{A}}^\infty$, $(U_{\mathcal{A}}^{\omega,u})_{\mathrm{fin}} =_{\mathrm{df}} \{(u_1, u_2, \ldots) \in U_{\mathcal{A}}^\omega \mid (\exists j \in \mathbb{N}_+)(u = u_j = u_{j+1} = \cdots)\}$ for some $u \in U_{\mathcal{A}}$, and $U_{\mathcal{A}}^\omega$, respectively, or the result of a combination of these or other sets. The following machine with the input space $\mathbb{R}^\infty$ is a variant of the BSS machine over $\mathbb{R}^{\geq 0} =_{\mathrm{df}} (\mathbb{R}; \mathbb{R}; +, -, \cdot, /; \{r \mid r \geq 0\})$ that essentially corresponds to the real Turing machines described in [11, pp. 455–456].

Example 1 (BSS $\mathbb{R}^{\geq 0}$-machine) *Let $\mathcal{P}$ be an $(n_1; 2, 2, 2, 2; 1)$-program each of whose instructions can be an* F*-instruction, an* F_0*-instruction, an index instruction of the type (6) or (7) for the two index register I_1 and I_2, the* C*-instruction $Z_{I_2} := Z_{I_1}$, or a* T*-instruction with the condition $r_1^1(Z_1)$ for evaluating $c(Z_1) \geq 0$. Then, the following tuple is a BSS $\mathbb{R}^{\geq 0}$-machine.*

$$\begin{array}{l} \mathcal{M}_{\mathbb{R}^{\geq 0}}^{\mathrm{BSS}} = (\mathbb{R}^\omega, (\mathbb{N}_+)^2, \mathcal{L}, \mathcal{P}, \mathbb{R}^{\geq 0}, \mathrm{In}, \mathrm{Out}), \qquad (x_1, \ldots, x_n) \in \mathbb{R}^\infty \\ \mathrm{In}(x_1, \ldots, x_n) = (1, 1, x_1, n, x_2, 0, x_3, \ldots, 0, x_n, 0, 0, \ldots) \\ \mathrm{Out}(\nu_1, \nu_2, u_1, u_2, u_3, \ldots) = (u_1, u_2, u_3, \ldots) \end{array}$$

By [3], every BSS machine is a graph such that, for any input, the computational process of this machine can be represented by a computation path as usual in flow charts where the nodes correspond to the labelled instructions of some BSS $\mathbb{R}^{\geq 0}$-machine.

2.3 Subprograms and pseudo instructions

Here, any *σ-subprogram* is a substring of a σ-program (a macro) that has the form $\mathsf{instruction}_{\ell_1}$; $\ell_1{+}1 : \mathsf{instruction}_{\ell_1+1}; \ldots ; \ell_2 : \mathsf{instruction}_{\ell_2}$ where every $\mathsf{instruction}_\ell$ $(\ell_1 \leq \ell \leq \ell_2)$ denotes a σ-instruction. Thus, any $\mathcal{P} \in \mathsf{P}_\sigma$ can be represented by certain strings of the form

$$1 : \mathsf{subprogram}_1;\ \ell_2 : \mathsf{subprogram}_2; \ldots ;\ \ell_s : \mathsf{subprogram}_s;\ \ell_{\mathcal{P}} : \mathsf{stop}.$$

with strings $\mathsf{subprogram}_m$ $(1 \leq m \leq s)$ denoting σ-subprograms and resulting from replacing the placeholder $\langle \mathit{subprogram} \rangle$ with the help of syntactic rules given in Table 1 (for suitable $\ell \geq 1$). Any pair (A_1, A_2) given in one row of Table 1 means that the placeholder A_1 given in the first column can be replaced by the string A_2 given in the second column by the application of the rule $A_1 \to A_2$. In the replacing process, all placeholders marked by angle brackets $\langle$ and $\rangle$ must be replaced step-by-step. Accordingly, all

A_1	A_2
$\langle condition \rangle$	$I_j = I_k$
$\langle condition \rangle$	$r_i^{k_i}(Z_{j_1}, \ldots, Z_{j_{k_i}})$
$\langle condition \rangle$	$\langle condition \rangle$ & $\langle condition \rangle$
$\langle condition \rangle$	$\langle condition \rangle$ or $\langle condition \rangle$
$\langle instructions \rangle$	$\langle instruction \rangle$
$\langle instructions \rangle$	$\langle pseudo\_instr \rangle$
$\langle instructions \rangle$	$\langle instructions \rangle$; ℓ: $\langle instructions \rangle$
$\langle pseudo\_instr \rangle$	goto ℓ
$\langle pseudo\_instr \rangle$	if $\langle condition \rangle$ then $\langle subprogram \rangle$
$\langle pseudo\_instr \rangle$	if $\langle condition \rangle$ then $\langle subprogram \rangle$ else $\langle subprogram \rangle$
$\langle subprogram \rangle$	$\langle instruction \rangle$
$\langle subprogram \rangle$	$\{\langle instructions \rangle\}$

Table 1: Syntactic rules $A_1 \to A_2$

placeholders $\langle pseudo\_instr \rangle$ and $\langle condition \rangle$ may also be replaced by new pseudo instructions and new conditions $\mathsf{condition}_j$, respectively, introduced later. In this way we get new strings for denoting subprograms. For example, any resulting string "ℓ_1 : $\mathsf{if\ condition}_j\ \mathsf{then\ subprogram}_m$; ℓ_2 :" corresponds to "ℓ_1 : $\mathsf{if\ condition}_j\ \mathsf{then\ goto}\ \ell_1{+}1\ \mathsf{else\ goto}\ \ell_2$; $\ell_1{+}1$: $\mathsf{subprogram}_m$; ℓ_2 :" where $\mathsf{subprogram}_m$ can stand, e. g., for "$\mathsf{instruction}_{\ell_1+1}$; $\ell_1{+}2$: $\mathsf{instruction}_{\ell_1+2}$; ...; $\ell_1{+}s$: $\mathsf{instruction}_{\ell_1+s}$" if, for the arithmetic expressions, $\ell_2 = \ell_1{+}s{+}1$ holds. In the following, some of the labels before an instruction that are not used in branching instructions (and the symbol : after these labels) are omitted. Summarizing, we can say that an application of a syntactic rule is only permitted if the resulting string does not contain two (pseudo) instructions with the same label and the result of all applications is — after a possible renumbering of all instructions and the corresponding renaming of the labels — a σ-program. The braces can be omitted if no ambiguity is possible. Therefore, the braces in $\{\langle instructions \rangle\}$ are omitted if $\langle instructions \rangle$ is replaced by a simple pseudo instruction without an if part (such as some of the pseudo instructions given e.g. in Overview 2).

2.4 Meaning of the space of indices and properties

The BSS machines introduced in [3] work with only two index registers and they have been defined for a large class of commutative rings that includes

the ring over the real numbers and integers, respectively. Our generalisation of this kind of machines leads to the concept of $\mathcal{A}$-machines. Accordingly, all Z-registers have an address such that their contents can definitely be read by means of index registers. But, we clearly distinguish between the values used as addresses of storage locations and the elements of the underlying structure $\mathcal{A}$. We want to have the possibility to store some additional information such as the length of the input regardless of whether the encoding of the information by the inputted individuals or other elements of the structure is possible. Therefore, the space of addresses contains, in general, tuples of more than two components. The addresses belong to a second structure, to the Peano structure $\mathcal{A}_{\mathbb{N}} = (\mathbb{N}_+; 1; succ; =)$ where $succ(n) = n + 1$ for all $n \geq 1$. Consequently, any $\mathcal{A}$-machine is able to use the functions and the relations of both structures $\mathcal{A}$ and $\mathcal{A}_{\mathbb{N}}$ and to compute several arithmetic functions $f : \mathbb{N}_+^s \to \mathbb{N}_+$ by manipulating indices. With respect to processing indices, we allow only the interpretation of the symbols as usual in arithmetic. Therefore, we do not distinguish between the arithmetic functions and relations and their symbols in arithmetic expressions occurring in pseudo instructions and comparisons if they refer to indices or index registers. Hence, the semantics is easy to understand without explaining all details and the meaning of each single pseudo instruction. Accordingly, we use the following pseudo instructions and the like and further conditions A_2 in syntactic rules of the form $\langle condition \rangle \to A_2$. In Overview 2, let $\nu, \mu \geq 1$ or, in the third line, let $\mu - \nu > 0$ and $\mu \geq \nu \geq 0$, respectively, and, in the fourth line, let $\nu + \mu \geq 1$. Any expression $t + 0$ may be replaced by t.

Overview 2 (Pseudo instructions)

$I_j := \nu,$	$I_j := I_j + \nu,$	if $I_j > \nu$ then $I_j := I_j - \nu$
$I_j := I_k + \nu,$	$I_j := I_k + I_m,$	$Z_{I_j} := Z_{I_k+I_m}, \quad Z_{I_j+I_k} := Z_{I_l+I_m}$
$I_j := \mu - \nu,$		$(Z_{I_j+\nu}, \ldots, Z_{I_j+\mu}) := (Z_{I_k}, \ldots, Z_{I_k+\mu-\nu})$
$(Z_{I_j+\nu}, \ldots, Z_{I_k+\nu}) := (Z_{I_j+\mu}, \ldots, Z_{I_k+\mu})$		
		(permitted if $c(I_j) \leq c(I_k)$ is ensured)
$(Z_1, Z_{\mu+1}, \ldots, Z_{\nu\cdot\mu+1}) := (Z_1, Z_2, \ldots, Z_{\nu+1}),$		$(Z_{I_j}, Z_{I_j+1}) := (c_{i_1}, c_{i_2})$
Further pseudo conditions: $(Z_{I_j}, Z_{I_j+1}) = (c_{i_1}, c_{i_2})$		
$I_j = \nu$, $I_j > \nu$, $I_j \geq \nu$, $I_j \leq \mu - \nu$, $I_j \geq I_k$, $I_j = I_k + \nu$, $I_j > I_k + \nu$		

For $\nu = 1$, the third pseudo instruction can be realised as follows where K_1 and K_2 are new index registers. if $I_j > 1$ then goto ℓ_1 else goto ℓ_4; $\ell_1 : K_1 := 1$; $K_2 := 1$; $\ell_2 : K_1 := K_1 + 1$; if $K_1 = I_j$ then goto ℓ_3 else $K_2 := K_2 + 1$; goto

ℓ_2; $\ell_3 : I_j := K_2$; $\ell_4 : \ldots$ The fourth last pseudo instruction corresponds to $I_m := I_j + \nu$; if $I_k > I_m$ then $\{Z_{I_m} := Z_{I_k}$; $K_1 := 1$; goto $\ell_1\}$ else $\{K_1 := \mu - \nu$; goto $\ell_2\}$; ℓ_1 : if $K_1 \leq \mu - \nu$ then $\{Z_{I_m+K_1} := Z_{I_k+K_1}$; $K_1 := K_1 + 1$; goto $\ell_1\}$ else goto ℓ_3; ℓ_2 : if $K_1 \geq 1$ then $\{Z_{I_m+K_1} := Z_{I_k+K_1}$; $K_1 := K_1 - 1$; goto $\ell_2\}$; $Z_{I_m} := Z_{I_k}$; $\ell_3 : \ldots$.

In any program, every Z_i is a variable to which values can be assigned. However, informally we can associate $Z_1, Z_2, \ldots$ with places for storing objects. These places are boxes, cells or so-called registers. Since each of the Z-registers has a number (an address), this corresponds to the idea that the Z-registers form a tape infinite to the right and that any value on the tape can be read by a head whose position is stored in one index register.

Let $\mathcal{A}_0 = (\{0,1\}; 0, 1; ; =)$. Then, any simple 1-tape Turing machine M (for the definition see, e. g., [8, p. 159 in the German version]) computing a function $f :\subseteq \{0,1\}^* \to \{0,1\}^*$ corresponds to the following Turing $\mathcal{A}_0$-machine $\mathcal{M}^{\mathrm{T}}(M)$ with the input space $\mathsf{I}_{\mathcal{M}^{\mathrm{T}}(M)} = \{0,1\}^*$. Since every positive integer has a unique binary representation, it is also possible that M computes a function $f :\subseteq \mathbb{N}_+ \to \mathbb{N}_+$. For this purpose, we define a Turing $\mathcal{A}_0$-machine $\mathcal{M}^{\mathrm{T}}_{\mathbb{N}}(M)$ with $\mathsf{I}_{\mathcal{M}^{\mathrm{T}}_{\mathbb{N}}(M)} = \mathbb{N}_+$ and new input and output functions, $\mathrm{In}_{\mathcal{M}^{\mathrm{T}}_{\mathbb{N}}(M)}$ and $\mathrm{Out}_{\mathcal{M}^{\mathrm{T}}_{\mathbb{N}}(M)}$, derived from $\mathrm{In}_{\mathcal{M}^{\mathrm{T}}(M)}$ and $\mathrm{Out}_{\mathcal{M}^{\mathrm{T}}(M)}$.

Example 2 (Turing $\mathcal{A}_0$-machines) *Let M work with the symbols $0, 1, \square$ and a finite number of states in $Q = \{q_1, \ldots, q_{i_0}\}$ for some $i_0 \geq 1$. Here, the symbols 0 and 1 are encoded by $(0,0)$ and $(0,1)$, respectively, and the symbol $\square$ by $(1,1)$. The content of the relevant cells $\tau_{-s_1}, \ldots, \tau_{s_2}$ of M (that do not contain the symbol $\square$) is stored in the Z-registers $Z_1, \ldots, Z_{2(s_1+s_2+1)}$, the head position s is given by the index register I_2 such that $c(I_2) = 2(s_1 + s + 1) - 1$. $c(I_1) = 2(s_1 + s_2 + 1)$ is the address of the last relevant Z-register whereas the first relevant cell τ_{-s_1} corresponds to Z_1 and Z_2. Any state $q_i \in Q$ is represented by one label $\ell_i \in \mathcal{L}_{\mathcal{M}^{\mathrm{T}}(M)}$. The program $\mathcal{P} = \mathcal{P}_{\mathcal{M}^{\mathrm{T}}(M)}$ contains, for any label ℓ_i ($i \leq i_0$), a pseudo instruction of the following form with $(c_{i_1}, c_{i_2}) \in \{(0,1), (0,1), (1,1)\}$ and $j \leq i_0$.*

$$\boxed{\ell_i : \text{ if } (Z_{I_2}, Z_{I_2+1}) = (c_{i_1}, c_{i_2}) \text{ then } \{\mathsf{subprogram}_{i,i_1,i_2}; \text{ goto } \ell_j\}}$$

The Turing machine M can only change the content of a cell τ_s or make a move to a neighbour cell τ_{s-1} or τ_{s+1} in dependence on the current state and the content of τ_s. Hence, any subprogram $\mathsf{subprogram}_{i,i_1,i_2}$ is one of the following pseudo instructions. (a) means to write the symbol encoded by (c_{i_1}, c_{i_2}). (b) means to go to the right. (c) means to go to the left.

(a)	$(Z_{I_2}, Z_{I_2+1}) := (c_{i_1}, c_{i_2})$
(b)	**if** $I_1 = I_2 + 1$ **then** $I_1 := I_1 + 2$; $I_2 := I_2 + 2$
(c)	**if** $I_2 > 2$ **then** $I_2 := I_2 - 2$ **else** $\{I_1 := I_1 + 2;$ $(Z_{I_2+2}, \ldots, Z_{I_1+2}) := (Z_{I_2}, \ldots, Z_{I_1})$; $(Z_{I_2}, Z_{I_2+1}) := (1,1)\}$

Without loss of generality, let Q contain only one final state. Therefore, let the final state be represented by $\ell_{\mathcal{P}} \in \mathcal{L}_{\mathcal{M}^{\mathrm{T}}(M)}$. Formally, we describe $\mathcal{M}^{\mathrm{T}}(M)$ as follows. $\mathcal{L}$ contains all ℓ_i for $i \le i_0$ and the labels used in the subprograms. $k > 2$ is also dependent on the subprograms. The integers $\lambda_0 = \min\{\nu \mid (2\nu - 1 \ge \nu_2 \;\&\; (u_{2\nu-1}, u_{2\nu}) \ne (1,1)) \text{ or } \nu = \frac{\nu_1}{2}\}$ and $\lambda_1 = \min\{\nu \mid \nu \ge \lambda_0 \;\&\; (u_{2\nu+1}, u_{2\nu+2}) = (1,1)\}$ give, if possible, the positions for an output without the symbol $\Box$.

$$\mathcal{M}^{\mathrm{T}}(M) = (\{0,1\}^\omega, (\mathbb{N}_+)^k, \mathcal{L}, \mathcal{P}, \mathcal{A}_0, \mathrm{In}_*, \mathrm{Out}_*), \qquad x_1 \cdots x_n \in \{0,1\}^*$$

$$\mathrm{In}_*(x_1 \cdots x_n) = ((2n, 1, \ldots, 1)\,.\,(0, x_1, 0, x_2, \ldots, 0, x_n, 1, 1, 1, 1, \ldots))$$

$$\mathrm{Out}_*(\nu_1, \ldots, \nu_k, u_1, u_2, u_3, \ldots) = \Lambda \qquad \textit{if } u_{2\lambda_0 - 1} = 1$$

$$\mathrm{Out}_*(\nu_1, \ldots, \nu_k, u_1, u_2, u_3, \ldots) = u_{2\lambda_0} u_{2\lambda_0+2} \cdots u_{2\lambda_1} \qquad \textit{otherwise}$$

Let $\mathrm{bin} : \mathbb{N}_+ \to \{0,1\}^*$ *be defined by* $\mathrm{bin}(\sum_{i=1}^n x_i \cdot 2^{i-1}) = x_n \cdots x_1$ *for all* $\vec{x} \in \{0,1\}^\infty$ *with* $x_n = 1$ *and let* bin^{-1} *be the partial inverse of* bin.

$$\mathcal{M}^{\mathrm{T}}_{\mathbb{N}}(M) = (\{0,1\}^\omega, (\mathbb{N}_+)^k, \mathcal{L}, \mathcal{P}, \mathcal{A}_0, \mathrm{In}_{\mathbb{N}}, \mathrm{Out}_{\mathbb{N}}), \qquad m \in \mathbb{N}_+$$

$$\mathrm{In}_{\mathbb{N}}(m) = \mathrm{In}_* \circ \mathrm{bin}(m)$$

$$\mathrm{Out}_{\mathbb{N}}(\nu_1, \ldots, \nu_k, u_1, u_2, u_3, \ldots) = \mathrm{bin}^{-1} \circ \mathrm{Out}_*(\nu_1, \ldots, \nu_k, u_1, u_2, u_3, \ldots)$$

2.5 BSS RAM's over $\mathcal{A}$

One of the main characteristics of the infinite dimensional $\mathcal{A}$-machines is that any $\mathcal{M} \in \mathsf{IM}_{\mathcal{A}}$ has its own program. This means that the code of the program $\mathcal{P}_{\mathcal{M}}$ does not need to be part of the input of $\mathcal{M}$ and it is not necessary to have a universal algorithm for executing a program whose code is only given as input. Moreover, $\mathcal{A}$-machines have features suitable for modelling the object-oriented programming where provided data structures, the so-called classes, including the operations for processing the data, the so-called methods, can be used without knowing the implementation of the methods. Besides these properties, a further key feature of BSS RAM's over $\mathcal{A}$ should be the ability to process all tuples of elements of $U_{\mathcal{A}}$ uniformly.

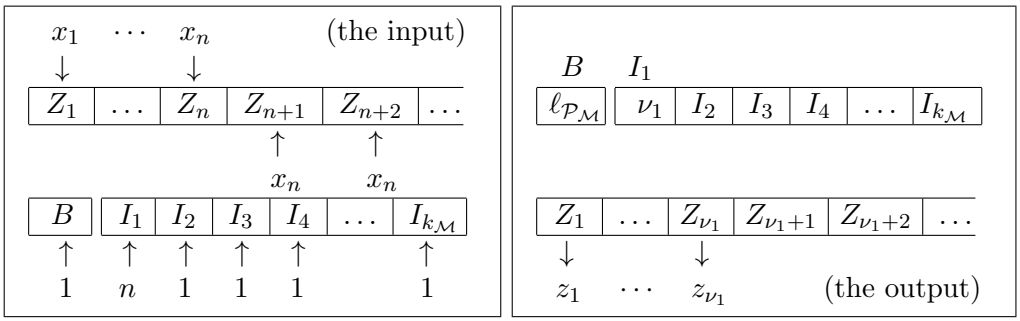

Figure 5: The input procedure and the output procedure for $\mathcal{M} \in \mathsf{M}_\mathcal{A}$

Definition 6 (BSS RAM) *For any structure $\mathcal{A}$, any $\mathcal{M} \in \mathsf{IM}_\mathcal{A}$ with the spaces $\mathsf{I}_\mathcal{M} = \mathsf{O}_\mathcal{M} = U_\mathcal{A}^\infty$ is called* BSS RAM over $\mathcal{A}$ *if* $\mathrm{In}_\mathcal{M}$ *and* $\mathrm{Out}_\mathcal{M}$ *are defined for all $(x_1, \ldots, x_n) \in U_\mathcal{A}^\infty$ and $((\nu_1, \ldots, \nu_{k_\mathcal{M}}), (u_1, u_2, \ldots)) \in \mathbb{N}_+^{k_\mathcal{M}} \times U_\mathcal{A}^\omega$ as follows (see also Figure 5).*

$$\mathrm{In}_\mathcal{M}(x_1, \ldots, x_n) = (\underbrace{n, 1, \ldots, 1}_{k_\mathcal{M} \text{ indices}}, x_1, \ldots, x_n, x_n, x_n, \ldots)$$
$$\mathrm{Out}_\mathcal{M}(\nu_1, \ldots, \nu_{k_\mathcal{M}}, u_1, u_2, u_3, \ldots) = (u_1, \ldots, u_{\nu_1})$$

Let $\mathsf{M}_\mathcal{A}$ be the class of all BSS RAM's over $\mathcal{A}$. Consequently, for any $\mathcal{M} \in \mathsf{M}_\mathcal{A}$ and each input $(x_1, \ldots, x_n) \in U_\mathcal{A}^\infty$, the input procedure $\mathrm{Input}_\mathcal{M}$ of $\mathcal{R}_\mathcal{M}$ provides the initial configuration $(1\,.\,(n, 1, \ldots, 1)\,.\,(x_1, \ldots, x_n, x_n, x_n, \ldots))$. Here, to simplify matters, the registers $Z_{n+1}, Z_{n+2}, \ldots$ get the value x_n. Therefore, for the input $(x_1, \ldots, x_n)$, $\mathcal{M}$ uses only the subspace $(U_\mathcal{A}^{\omega, x_n})_{\mathrm{fin}}$ of the space $U_\mathcal{A}^\omega$. Note that the concept could also be changed such that the initialisation of Z_j for $j > n$ would be done during the computation before Z_j should be used. Moreover, a constant c_i could be assigned to $Z_{n+1}, Z_{n+2}, \ldots$ if $\mathcal{A}$ is a structure with a constant and c_i is one of the constants of $\mathcal{A}$. If the machine halts, then the values $c(Z_1), \ldots, c(Z_{c(I_1)})$ are outputted. The following example helps to understand that there are differences between the BSS RAM's and the real RAM's working with a read instruction and a write instruction. It remains unclear how the following functions could be computed by means of real RAM's of the latter form.

Example 3 (Functions computable over $\mathbb{R}^{\geq 0}$) *Since the register I_1 of any BSS RAM contains the length of any input at the beginning, for $i \in \{1, 2, 3\}$, there are BSS RAM's $\mathcal{M}_i$ over $\mathbb{R}^{\geq 0}$ computing the functions $f_i : \mathbb{R}^\infty \to \mathbb{R}^\infty$ everywhere defined by $f_1(x_1, \ldots, x_n) = \sum_{i=1}^n x_i$, $f_2(x_1, \ldots, x_n) =$*

x_n, *and* $f_3(x_1, \ldots, x_n) = n$, *respectively, for any* $n \geq 1$ *and all* $(x_1, \ldots, x_n) \in \mathbb{R}^\infty$. *More precisely, we have* $\mathrm{Res}_{\mathcal{M}_i} = f_i$ *for each* $i \in \{1, 2, 3\}$.

2.6 Multi-tape machines and the multi-tape mode

Now, we consider machines that work with a finite number of tapes. For any $d \geq 1$, a *d-tape $\mathcal{A}$-machine* is a tuple $((U_\mathcal{A}^\omega)^d, \mathbb{N}_+^{\kappa_1} \times \cdots \times \mathbb{N}_+^{\kappa_d}, \mathcal{L}, \mathcal{P}, \mathcal{B}, \mathrm{In}, \mathrm{Out})$ that can be defined analogously to the usual $\mathcal{A}$-machines in $\mathsf{IM}_\mathcal{A}$. It is equipped, for all $j \leq d$, with the Z-registers $Z_{j,1}, Z_{j,2}, \ldots$ forming the j^{th} tape and a finite number of index registers $I_{j,1}, \ldots, I_{j,\kappa_j}$. For $\ell \in \mathcal{L}$, $(\vec{\nu}^{(j)})_{j=1..d} \in \mathbb{N}_+^{\kappa_1} \times \cdots \times \mathbb{N}_+^{\kappa_d}$, and $(\bar{u}^{(j)})_{j=1..d} \in (U_\mathcal{A}^\omega)^d$, the tuple $(\ell, (\vec{\nu}^{(j)} \,.\, \bar{u}^{(j)})_{j=1..d})$ is a *configuration of this machine*. The overall state of such a machine $\mathcal{M}$ is described by $(\ell, (\vec{\nu}^{(j)} \,.\, \bar{u}^{(j)})_{j=1..d})$ if ℓ is the label $c(B)$ stored in the register B of $\mathcal{M}$, every component $\nu_{j,k}$ of $\vec{\nu}^{(j)}$ is the content $c(I_{j,k})$ of the index register $I_{j,k}$ of $\mathcal{M}$, and the component $u_{j,k}$ of $\bar{u}^{(j)}$ is the content $c(Z_{j,k})$ of the Z-register $Z_{j,k}$ of $\mathcal{M}$. Let $\mathsf{IM}_\mathcal{A}^{(d)}$ be the class of all d-tape $\mathcal{A}$-machines. For signature $\sigma = (n_1; m_1, \ldots, m_{n_2}; k_1, \ldots, k_{n_3})$, any *$d$-tape σ-program* results from replacing each placeholder $\langle \textit{instruction} \rangle$ in a string of the form (*) by one of the *d-tape σ-instructions* whose form is given in Overview 3 where i is a positive integer less than or equal to n_1, n_2, and n_3, respectively, $d_0, d_1, d_2, \ldots$ stand for positive integers $\leq d$, and all $j, k, j_1, j_2, \ldots$ are placeholders for positive integers. Let $\mathsf{P}_\sigma^{(d)}$ be the set of all d-tape σ-programs.

Overview 3 (d-tape σ-instructions)

$Z_{d_0,j} := f_i^{m_i}(Z_{d_1,j_1}, \ldots, Z_{d_{m_i},j_{m_i}})$, $\quad Z_{d_0,j} := c_i^0$, $\quad Z_{d_1,I_{d_1,j}} := Z_{d_2,I_{d_2,k}}$
if $r_i^{k_i}(Z_{d_1,j_1}, \ldots, Z_{d_{k_i},j_{k_i}})$ then goto ℓ_1 else goto ℓ_2,
if $I_{d_1,j} = I_{d_2,k}$ then goto ℓ_1 else goto ℓ_2, $\quad I_{d_1,j} := 1$, $\quad I_{d_1,j} := I_{d_1,j} + 1$

Each execution of an instruction in $\mathcal{P}_\mathcal{M} \in \mathsf{P}_\sigma^{(d)}$ by an $\mathcal{M} \in \mathsf{IM}_\mathcal{A}^{(d)}$ can cause the change of the current configuration. For instance, $\ell_0 : \; Z_{1,I_{1,j}} := Z_{2,I_{2,k}}$ means $(\ell_0, (\vec{\nu}^{(j)} \,.\, \bar{u}^{(j)})_{j=1..d}) \to_\mathcal{M} (\ell_0 + 1, (\vec{\nu}^{(j)} \,.\, \bar{w}^{(j)})_{j=1..d})$ where $\bar{w}^{(j)} = \bar{u}^{(j)}$ for $j \in \{2, \ldots, d\}$ and $\bar{w}^{(1)} = (u_{1,1}, \ldots, u_{1,\nu_{1,j-1}}, u_{2,\nu_{2,k}}, u_{1,\nu_{1,j+1}}, u_{1,\nu_{1,j+2}}, \ldots)$. Accordingly, let the result function be defined.

Any machine $\mathcal{M} \in \mathsf{IM}_\mathcal{A}^{(d)}$ is a *d-tape BSS RAM* if, for any input $\vec{x} = (x_1, \ldots, x_n)$, $\mathrm{In}_\mathcal{M}(\vec{x}) = (\vec{\nu}^{(j)} \,.\, \bar{u}^{(j)})_{j=1..d}$ is determined by $\vec{\nu}^{(1)} = (n, 1, \ldots, 1)$ and $\bar{u}^{(1)} = (x_1, \ldots, x_n, x_n, \ldots)$ and by $\vec{\nu}^{(j)} = (1, \ldots, 1)$ and $\bar{u}^{(j)} = (x_n, x_n, \ldots)$ for all $j \in \{2, \ldots, d\}$ and $\mathrm{Out}_\mathcal{M}$ is defined by $\mathrm{Out}_\mathcal{M}((\vec{\nu}^{(j)} \,.\, \bar{u}^{(j)})_{j=1..d}) = (u_{1,1}, u_{1,2}, \ldots, u_{1,\nu_{1,1}})$. Let $\mathsf{M}_\mathcal{A}^{(d)}$ be the class of all d-tape BSS RAM's.

Since the Z-registers of the first tape of a machine in $\mathsf{M}_{\mathcal{A}}^{(d)}$ get the input values and provide the output in the same way as the registers of a machine in $\mathsf{M}_{\mathcal{A}}$, we have $\mathsf{M}_{\mathcal{A}}^{(1)} = \mathsf{M}_{\mathcal{A}}$. On the other hand, any program of a machine $\mathcal{M} \in \mathsf{IM}_{\mathcal{A}}^{(d)}$ can be simulated by a machine $\mathcal{N}_{\mathcal{M}} \in \mathsf{IM}_{\mathcal{A}}^{(1)}$. Let $\mathcal{M}$ be any machine in $\mathsf{IM}_{\mathcal{A}}^{(d)}$ and, for all $j \le d$, let $I_{j,1}, \ldots, I_{j,\kappa_j}$ be the index registers of $\mathcal{M}$ allowing the access to the j^{th} tape $Z_{j,1}, Z_{j,2}, \ldots$ of $\mathcal{M}$. Then, let $\mathcal{N}_{\mathcal{M}}$ be a machine in $\mathsf{IM}_{\mathcal{A}}^{(1)}$ with $k_{\mathcal{N}_{\mathcal{M}}} = \kappa_1 + \cdots + \kappa_d$ working in *d-tape mode* with d tracks by using the j^{th} *track* $Z_j, Z_{d+j}, Z_{2d+j}, \ldots$ instead of the j^{th} tape $Z_{j,1}, Z_{j,2}, \ldots$ of $\mathcal{M}$. If $\mathrm{Input}_{\mathcal{M}}(\mathsf{i}) = (1, (\vec{\nu}^{(j)} \,.\, \bar{u}^{(j)})_{j=1..d})$ holds, then let $\mathrm{In}_{\mathcal{N}_{\mathcal{M}}}(\mathsf{i}) = ((\vec{\nu}^{(1)} \,.\, \ldots \,.\, \vec{\nu}^{(d)}) \,.\, \bar{u})$ with $(u_j, u_{d+j}, u_{2d+j}, \ldots) = \bar{u}^{(j)}$ for all $j \le d$. However, if $\mathcal{M}$ and $\mathcal{N}_{\mathcal{M}}$ should be BSS RAM's, then, after the input, let $\mathcal{N}_{\mathcal{M}}$ arrange its input $\vec{x} \in U_{\mathcal{A}}^{\infty}$ on the first track $Z_1, Z_{d+1}, Z_{2d+1}, \ldots$ and execute $I_1 := (I_1 - 1) \cdot d + 1$ before the simulation starts with a suitable configuration.

Proposition 1 (Simulation of d-tape $\mathcal{A}$-machines) *For any structure $\mathcal{A}$ and all $t \ge 1$, the work of any $\mathcal{M} \in \mathsf{IM}_{\mathcal{A}}^{(d)}$ can be done by a machine $\mathcal{N}_{\mathcal{M}} \in \mathsf{IM}_{\mathcal{A}}^{(1)}$ where t steps of $\mathcal{M}$ can be realised by $\mathcal{N}_{\mathcal{M}}$ within dt steps.*

Proof. Let $\mathcal{P}_{\mathcal{M}}$ contain only instructions given in Overview 3. To get the corresponding program $\mathcal{P}_{\mathcal{N}_{\mathcal{M}}}$, we replace every variable $Z_{j,i}$ $(j \le d, i \ge 1)$ in $\mathcal{P}_{\mathcal{M}}$ by the variable $Z_{(i-1)d+j}$ so that we have to consider the register blocks $Z_{(i-1)d+1}, \ldots, Z_{id}$ of $\mathcal{N}_{\mathcal{M}}$ instead of the registers $Z_{1,i}, \ldots, Z_{d,i}$. The variables for index registers $I_1, \ldots, I_{\kappa_1+\cdots+\kappa_d}$ are substituted for $I_{1,1}, \ldots, I_{1,\kappa_1}$, $\ldots$ $I_{d,1}, \ldots, I_{d,\kappa_d}$ in this order. Consequently, in all copy instructions in $\mathcal{P}_{\mathcal{M}}$, the indices of Z-registers including the index register variables must be replaced as follows. $Z_{I_{\kappa_1+\cdots+\kappa_{j-1}+i}}$ must be substituted for $Z_{j,I_{j,i}}$. Moreover, we replace any index instruction $I_{j,i} := 1$ in $\mathcal{P}_{\mathcal{M}}$ by $I_{\kappa_1+\cdots+\kappa_{j-1}+i} := j$ and $I_{j,i} := I_{j,i} + 1$ by $I_{\kappa_1+\cdots+\kappa_{j-1}+i} := I_{\kappa_1+\cdots+\kappa_{j-1}+i} + d$. Then, the execution of each of these pseudo instructions in $\mathcal{P}_{\mathcal{N}_{\mathcal{M}}}$ can be realised within d transformation steps defined by $\to_{\mathcal{N}_{\mathcal{M}}}$. Any other instruction of $\mathcal{P}_{\mathcal{M}}$ can be simulated in one step. □

3 Universal machines and consequences

3.1 Universal programs and machines: the definitions

We want to transfer the classical theorems about the existence of a universal partial recursive function (see e.g. the Enumeration Theorem in [24, p. 18])

and the existence of a universal Turing machine (see e.g. [9, Theorem IV, p. 22]) and generalise a result about the existence of a universal BSS machine presented by a flow chart in Figure 15 in [3, p. 35]. The following definition also includes the case considered for a similar model in [17, p. 39]. Let a and b be two arbitrary objects and let k be an integer such that it is possible to encode the characters in any string $\mathcal{P} \in \mathsf{P}_\sigma$ (including all symbols such as r_{21}^3 after their replacement, e. g., by r_21_3) by tuples in $\{a,b\}^k$. Then, for any σ-program $\mathcal{P}$, let $\mathrm{code}_{(a,b)}(\mathcal{P}) \in \{a,b\}^\infty$ result from the concatenation of an a and the sequence of the codes of all characters in $\mathcal{P}$ in this order. Let $\mathrm{code}^*(\mathcal{P})$ be the string $s_1 \cdots s_m \in \{0,1\}^*$ if $\mathrm{code}_{(1,0)}(\mathcal{P}) = (s_1, \ldots, s_m)$ holds and let $\mathrm{code}_{\mathbb{N}}(\mathcal{P}) = \mathrm{bin}^{-1}(\mathrm{code}^*(\mathcal{P}))$. Let $\mathcal{A}$ be a structure. For any $\mathcal{M} \in \mathsf{IM}_\mathcal{A}$ with the constants in $\vec{c}^{(\mathcal{M})} = (c_{j_1}, \ldots, c_{j_{n_1}})$, let $\mathrm{code}_{(a,b)}(\mathcal{M}) = (\mathrm{code}_{(a,b)}(\mathcal{P}_\mathcal{M}) \,.\, \vec{a}^{\,(\mathcal{M},a)})$ where $\vec{a}^{\,(\mathcal{M},a)} = (a_1, \ldots, a_{\ell_{\mathcal{P}_\mathcal{M}}})$ is defined as follows. For any $\ell \le \ell_{\mathcal{P}_\mathcal{M}}$, let a_ℓ be the i^{th} component c_{j_i} in $\vec{c}^{(\mathcal{M})}$ if the ℓ^{th} instruction of $\mathcal{P}_\mathcal{M}$ is the instruction $Z_j := c_i^0$ for some j and otherwise let $a_\ell = a$.

Definition 7 (Universal machines of type 1 for BSS RAM's) *Let a and b be two constants of $\mathcal{A}$. We say that $\mathcal{M}_0 \in \mathsf{M}_\mathcal{A}$ is a* universal BSS RAM *over $\mathcal{A}$ if $\mathrm{Res}_{\mathcal{M}_0}(\mathrm{code}_{(a,b)}(\mathcal{M}) \,.\, \vec{x}) = \mathrm{Res}_\mathcal{M}(\vec{x})$ holds for all $\vec{x} \in U_\mathcal{A}^\infty$ and any $\mathcal{M} \in \mathsf{M}_\mathcal{A}$.*

The following definition includes also the definition of a further variant of a universal BSS RAM for several structures. It is in particular suitable for structures of finite signature without constants. For this purpose, let $p_1, p_2, \ldots$ be the sequence of all prime numbers with $p_i \le p_{i+j}$ for all $i, j \ge 1$. For any $\mathcal{A}$-machine $\mathcal{M}_0$, let $\mathsf{ireg} : \mathbb{N}_+^\infty \to (\mathbb{N}_+^{k_{\mathcal{M}_0}} \cap (\mathbb{N}_+ \times \{1\} \times \cdots \times \{1\}))$ be any bijective function such that this function as well as its partial inverse can be computed by BSS RAM's over $\mathcal{A}_\mathbb{N}$. Moreover, for any $k \ge 1$, let the function $\mathsf{end}_{\mathcal{M}_0,k} : \mathsf{S}_{\mathcal{M}_0} \to \{(\vec{\nu} \,.\, \bar{u}) \mid (\vec{\nu}, \bar{u}) \in \mathbb{N}_+^k \times U_\mathcal{A}^\omega\}$ be defined, for all $(\ell \,.\, \vec{\mu} \,.\, \bar{u}) \in \mathsf{S}_{\mathcal{M}_0}$, by $\mathsf{end}_{\mathcal{M}_0,k}(\ell \,.\, \vec{\mu} \,.\, \bar{u}) = (\vec{\nu} \,.\, \bar{u})$ where $\vec{\nu} = (\mu_1, \nu_2, \ldots, \nu_k)$ holds if $\mu_2 = 2^{\nu_2} \cdots p_k^{\nu_k}$ and otherwise $\vec{\nu} = (\mu_1, 1, \ldots, 1) \in \mathbb{N}_+^k$ holds.

Definition 8 (Universal programs and machines of type 2) *Let $\mathcal{M}_0$ be an $\mathcal{A}$-machine. We say that $\mathcal{P}_{\mathcal{M}_0}$ is* $\mathsf{IM}_\mathcal{A}$-universal *(or* universal*) if, for all $\mathcal{M} \in \mathsf{IM}_\mathcal{A}$ and $\mathsf{i} \in \mathsf{I}_\mathcal{M}$,*

$$\mathrm{Res}_\mathcal{M}(\mathsf{i}) = \mathrm{Out}_\mathcal{M} \circ \mathsf{end}_{\mathcal{M}_0,k_\mathcal{M}} \circ (\to_{\mathcal{M}_0})_{\mathrm{Stop}_{\mathcal{M}_0}}(\mathsf{init}_{\mathcal{M}_0}(\mathcal{M}, \mathsf{i}))$$

holds for the initial configuration $\mathsf{init}_{\mathcal{M}_0}(\mathcal{M}, \mathsf{i}) \in \mathsf{S}_{\mathcal{M}_0}$ that is given by

$$\mathsf{init}_{\mathcal{M}_0}(\mathcal{M}, \mathsf{i}) = (1 \,.\, \mathsf{ireg}(\vec{\nu} \,.\, \mathrm{code}_\mathbb{N}(\mathcal{P}_\mathcal{M})) \,.\, \vec{a}^{\,(\mathcal{M},u_1)} \,.\, \bar{u})$$

Type	e_1	e_2	e_3	e_4	e_5	e_6	e_7	$\cdots$	$e_{\dots}$	$\mu_{\mathcal{P},\ell}$
(1)	1	i			j	j_1	j_2		j_{m_i}	$2^1\cdot 3^i\cdot 11^j\cdot 13^{j_1}\cdot 17^{j_2}\cdots p_{j_{m_i}+5}^{j_{m_i}}$
(2)	2				j					$2^2\cdot 11^j$
(3)	3				j	k				$2^3\cdot 11^j\cdot 13^k$
(4)	4	i	ℓ_1	ℓ_2		j_1	j_2		j_{k_i}	$2^4\cdot 3^i\cdot 5^{\ell_1}\cdot 7^{\ell_2}\cdot 13^{j_1}\cdot 17^{j_2}\cdots p_{j_{k_i}+5}^{j_{k_i}}$
(5)	5		ℓ_1	ℓ_2	j	k				$2^5\cdot 5^{\ell_1}\cdot 7^{\ell_2}\cdot 11^j\cdot 13^k$
(6)	6				j					$2^6\cdot 11^j$
(7)	7				j					$2^7\cdot 11^j$
(8)	8									1

Table 2: The codes $\mu_{\mathcal{P},\ell}$ for the instruction with label ℓ in $\mathcal{P}$

and $(\vec{\nu}\,.\,\bar{u}) = \mathrm{In}_{\mathcal{M}}(\mathsf{i})$. $\mathcal{M}_0$ *is called (*$\mathsf{M}_{\mathcal{A}}$*-)*universal BSS RAM *if* $\mathcal{M}_0 \in \mathsf{M}_{\mathcal{A}}$.

If $\mathsf{I}_{\mathcal{M}_0} = \{(\mathcal{M},\mathsf{i}) \mid \mathcal{M} \in \mathsf{IM}_{\mathcal{A}}\ \&\ \mathsf{i} \in \mathsf{I}_{\mathcal{M}}\}$, then let $\mathrm{Input}_{\mathcal{M}_0} = \mathsf{init}_{\mathcal{M}_0}$. If we consider only BSS RAM's, then the function ireg is determined by a binary function. This means that there is some $\mathsf{idx} : \mathbb{N}_+^2 \to \mathbb{N}_+$ such that $\mathsf{ireg}(\vec{\nu}\,.\,\mu) = (\mathsf{idx}(\nu_1,\mu),1,\dots,1)$ holds for all $\vec{\nu} = (\nu_1,\dots,\nu_k) \in \mathbb{N}_+^\infty$.

In the following, we want to consider universal $\mathcal{A}$-machines that are able to simulate the execution of every $\mathcal{M} \in \mathsf{M}_{\mathcal{A}}$ instruction-by-instruction. Let $\sigma = (n_1; m_1,\dots,m_{n_2}; k_1,\dots,k_{n_3})$ and $\mathcal{P}_{\mathcal{M}} \in \mathsf{P}_\sigma$. Since we want to make the important information about the instructions of $\mathcal{P}_{\mathcal{M}}$ available for easy access, we use strings $\alpha_{e_1}\cdots\alpha_{e_{\max\{m_1,\dots,m_{n_2},k_1,\dots,k_{n_3}\}}} \in \{\alpha_0,\dots,\alpha_{h_{\mathcal{P}_{\mathcal{M}}}}\}^*$ for some $h_{\mathcal{P}_{\mathcal{M}}} \geq 1$ and Gödel numberings (for the definition see e. g. [18, p. 183]) for storing the information about a single instruction labelled by ℓ in a Gödel number $\mu_{\mathcal{P}_{\mathcal{M}},\ell}$. These numbers are dependent on the type of instructions and $\phi_{\mathcal{P}_{\mathcal{M}},\ell} =_{\mathrm{df}} \phi_{\mathcal{M},\ell}$ (used also in Definition 3) for $\ell \leq \ell_{\mathcal{P}_{\mathcal{M}}}$. Table 2 shows the details. Let $e_s = 0$ if there is no other information. Moreover, let a be any element of $\mathcal{A}$. For any $\mathcal{P} \in \mathsf{P}_\sigma$, let the program information be given by $\mu_\ell = \mu_{\mathcal{P},\ell}$ for $\ell \leq \ell_{\mathcal{P}}$ and

$$\mathrm{code}_a(\mathcal{P}) = (a,\dots,a) \in \{a\}^{\nu_{\mathcal{P}}} \subseteq \{a\}^\infty \text{ with } \nu_{\mathcal{P}} = 2^{\mu_1}3^{\mu_2}\cdots p_{\ell_{\mathcal{P}}-1}^{\mu_{\ell_{\mathcal{P}}-1}} p_{\ell_{\mathcal{P}}} \in \mathbb{N}_+.$$

For any $\mathcal{M} \in \mathsf{IM}_{\mathcal{A}}$, let $\ell_{\mathcal{M}} = \ell_{\mathcal{P}_{\mathcal{M}}}$ and $\nu_{\mathcal{M}} = \nu_{\mathcal{P}_{\mathcal{M}}}$. Consequently, every machine $\mathcal{M} \in \mathsf{IM}_{\mathcal{A}}$ could be encoded by $\vec{a}^{(\mathcal{M},a)}$ and a tuple $(a,\dots,a)$ whose length is dependent on the Gödel number $\nu_{\mathcal{M}} = |\mathrm{code}_a(\mathcal{P}_{\mathcal{M}})|$ and the length of the input. However, in order to complement any possible input

$(x_1, \ldots, x_n) \in U_{\mathcal{A}}^{\infty}$ of an $\mathcal{M} \in \mathsf{M}_{\mathcal{A}}$ with a code of $\mathcal{M}$ that is computable from the input by some machine in $\mathsf{M}_{\mathcal{A}}$, we take $\vec{a}^{\,(\mathcal{M},x_1)}$ and a suitable tuple in $\{x_n\}^{\infty}$. For computing the total length of this tuple we use the Cantor pairing function $cantor : \mathbb{N}^2 \to \mathbb{N}$ defined by $cantor(\mu_1, \mu_2) = \frac{1}{2}((\mu_1+\mu_2)^2 + 3\mu_1+\mu_2)$. Now, let $\mathrm{code}_{n,a}(\mathcal{P}_{\mathcal{M}}) = (a, \ldots, a) \in \{a\}^{cantor(n,\nu_{\mathcal{M}})-\ell_{\mathcal{M}}-n}$ for $n \geq 1$ and any possible $a \in U_{\mathcal{A}}$. In this way, for all $\vec{x} \in U_{\mathcal{A}}^n$ and $\mathcal{M} \in \mathsf{M}_{\mathcal{A}}$, we get an input

$$(\vec{a}^{\,(\mathcal{M},x_1)} \,.\, \vec{x} \,.\, \mathrm{code}_{n,x_n}(\mathcal{P}_{\mathcal{M}})) \in U_{\mathcal{A}}^{cantor(n,\nu_{\mathcal{M}})} \qquad (**)$$

for a universe BSS RAM $\mathcal{M}_0$ of type 2 such that we have

$$\mathsf{init}_{\mathcal{M}_0}(\mathcal{M}, \vec{x}) = (1 \,.\, (cantor(n, \nu_{\mathcal{M}}), 1, \ldots, 1) \,.\, (\vec{a}^{\,(\mathcal{M},x_1)} \,.\, \vec{x} \,.\, (x_n, x_n, \ldots)). \quad (***)$$

3.2 Simulation of BSS RAM's by universal machines

Here, we will describe the work of a universal BSS RAM $\mathcal{M}_0 \in \mathsf{M}_{\mathcal{A}}$ that can simulate any machine $\mathcal{M} \in \mathsf{M}_{\mathcal{A}}$ by means of three tracks after assigning its own input to the first track in case that the input has the form (**). For explaining the algorithm we only construct a BSS RAM $\mathcal{M}_0^{(3)} \in \mathsf{M}_{\mathcal{A}}^{(3)}$ that is able to simulate any $\mathcal{M} \in \mathsf{M}_{\mathcal{A}}$ by a program that is also useful for other investigations. To simplify matters, we use further pseudo instructions. The strings in square brackets are optional.

Overview 4 (Pseudo instructions for d tapes)

$Z_{1,I_{1,k}} := f_i^{m_i}(Z_{1,I_{1,k+1}}, \ldots, Z_{1,I_{1,k+m_i}})$
$(Z_{d_1,[I_{d_1,j}+]1}, \ldots, Z_{d_1,[I_{d_1,j}+]I_{1,m}}) := (Z_{1,[I_{1,k}+]1}, \ldots, Z_{1,[I_{1,k}+]I_{1,m}})$
Further pseudo conditions: $r_i^{k_i}(Z_{1,I_{1,k+1}}, \ldots, Z_{1,I_{1,k+k_i}})$

In Overview 5, J_i, J_k, J_l, and J_m stand for $I_{1,i}$, $I_{1,k}$, $I_{1,l}$, and $I_{1,m}$. The operator div denotes the integer division, and $\nu|\mu$ means that ν is a divisor of μ. Moreover, let $cantor_1(\mu) = \mu_1$ and $cantor_2(\mu) = \mu_2$ if $\mu = cantor(\mu_1, \mu_2)$.

Overview 5 (Pseudo instructions for decoding numbers)

$J_i := cantor_1(J_k)$, $J_i := cantor_2(J_k)$, $J_i := p_m^{J_k}$, $J_i := J_k \cdot p_{J_m}$	
$J_i := \max\{s \mid p_m^s \mid J_k\}$	(permitted if $p_m \mid c(J_k)$ is ensured)
$J_i := [J_l+]\max\{s \mid p_{J_m}^s \mid J_k\}$	(permitted if $p_{c(J_m)} \mid c(J_k)$ is ensured)
$J_i := \max\{m \mid p_m \mid J_k\}$	(permitted if $c(J_k) > 1$ is ensured)
$J_i := J_i \text{ div } p_{J_m}^{J_l-J_k-1}$	(permitted if $c(J_i) \geq p_{c(J_m)}^{c(J_l)-c(J_k)-1}$ is ensured)
Further pseudo conditions: $p_m \mid J_k$, $p_{J_m}^s \mid J_k$	

If $d_1 \leq 3$, then the instructions in Overview 4 can be executed by a 3-tape machine. All values that are necessary for executing the first pseudo instruction and checking the new condition are firstly copied on a second tape where the new values are computed and the tests are performed. For this purpose, a finite number of instructions of the form $I_{2,1} := 1, I_{2,2} := 2, \ldots, Z_{2,I_{2,1}} := Z_{1,I_{1,k+1}}, Z_{2,I_{2,2}} := Z_{1,I_{1,k+2}}, \ldots, Z_{2,1} := f_i^{m_i}(Z_{2,1}, \ldots, Z_{2,m_i})$, $Z_{1,I_{1,k}} := Z_{2,I_{2,1}}$, and if $r_i^{k_i}(Z_{2,1}, \ldots, Z_{2,k_i})$ then goto ℓ_1 else goto ℓ_2, respectively, can be used. As known from the classical recursion theory, the pseudo instructions listed in Overview 5 can be used for evaluating Gödel numbers, and they can be replaced by subprograms consisting only of index instructions of the types (5) to (7). For example, since $J_k|J_i$ holds if and only if $J_i \geq J_k$ & $(J_i \text{ div } J_k)\cdot J_k = J_i$ holds, the pseudo instruction if $J_k|J_i$ then goto ℓ_1 else goto ℓ_2 can be performed as follows. One goes to ℓ_2 if $c(J_i) < c(J_k)$. Otherwise one takes a new index register J_l, executes $J_l := J_k$, repeats the loop where $J_l := J_l + 1$ is executed $c(J_k)$ times while $c(J_l) < c(J_i)$, but stops immediately if $c(J_l) = c(J_i)$. If $c(J_l) = c(J_k)$ or $J_l := J_l + 1$ can be executed $c(J_k)$ times in the last loop, then the execution can be continued with the instruction labelled by ℓ_1 and, otherwise, with the instruction labelled by ℓ_2. The number of the loops where the addition $c(J_l)+c(J_k)$ is completely executed is $(c(J_i) \text{ div } c(J_k))-1$. Thus, we can also use the pseudo instruction $J_j := J_i \text{ div } J_k$ that allows to compute $c(J_j) = \max\{s \in \mathbb{N}_+ \mid (\exists s_0 \in \mathbb{N})(0 \leq s_0 < s \;\&\; s \cdot c(J_k) + s_0 = c(J_i))\}$ if $c(J_i) \geq c(J_k)$. $c(J_k) = p_m$ can be computed by a program as follows where the index register K_1 is used for storing the index of the next searched prime number after $p_{c(K_1)-1}$ (if $c(K_1) - 1 \geq 1$), the index register K_2 is used for storing the integers greater than $p_{c(K_1)-1}$ and checking whether the stored number $c(K_2)$ is the next prime number, and the index register K_3 is used for storing a possible non-trivial factor of $c(K_2)$. $\ell_1 : K_1 := 1$; $K_2 := 2$; $\ell_2 :$ if $K_1 = m$ then goto ℓ_7; $\ell_3 : K_1 := K_1 + 1$; $\ell_4 : K_2 := K_2 + 1$; $K_3 := 1$; $\ell_5 : K_3 := K_3 + 1$; if $K_2 = K_3$ then goto ℓ_2; $\ell_6 :$ if $K_3|K_2$ then goto ℓ_4 else goto ℓ_5; $\ell_7 : J_k := K_2$.

Now, we assume that corresponding to (**) the registers $Z_{1,1}, Z_{1,2}, \ldots$ of the first tape $\mathcal{M}_0^{(3)}$ contain the values $a_1, \ldots, a_{\ell_\mathcal{M}}, x_1, \ldots, x_n, x_n, x_n, \ldots$ for $(a_1, \ldots, a_{\ell_\mathcal{M}}) = \vec{a}^{(\mathcal{M},x_1)}$ before $\mathcal{M}_0^{(3)}$ executes the program $\mathcal{P}_0^{(3)}$ given in Figure 6. Let $\mathcal{P}_{\mathcal{M}_0^{(3)}} = \mathcal{P}_0^{(3)}$. In $\mathcal{P}_0^{(3)}$, the index registers $I_{1,1}, \ldots, I_{1,2m_0+15}$ with $m_0 = \max\{m_1, \ldots, m_{n_2}, k_1, \ldots, k_{n_3}\}$ are denoted by N_0, N, C, L, L_P, V, S, $E_1, \ldots, E_{m_0+5}, J_0, \ldots, J_{m_0}$, H_1, and H_2. $c(N_0)$ is firstly the length n_0 of the input of $\mathcal{M}_0^{(3)}$ and later the length of the output $\mathcal{M}_0^{(3)}$. $c(N)$ is firstly

$1:$ $N := cantor_1(N_0)$; $C := cantor_2(N_0)$;
$L_P := \max\{m \mid p_m | C\}$; $(Z_{3,1}, \ldots, Z_{3,L_P}) := (Z_{1,1}, \ldots, Z_{1,L_P})$;
$(Z_{1,N_0+1}, \ldots, Z_{1,N_0+N}) := (Z_{1,L_P+1}, \ldots, Z_{1,L_P+N})$; $V := p_1^N$; $L := 1$;

$\ell_1:$ if $L = L_P$ then goto ℓ_7 else goto ℓ_2;

$\ell_2:$ $S := \max\{s \mid p_L^s | C\}$;
if $p_1 | S$ then $E_1 := \max\{e \mid p_1^e | S\}$; $\ldots$;
if $p_{m_0+5} | S$ then $E_{m_0+5} := \max\{e \mid p_{m_0+5}^e | S\}$;
if $E_1 = 1$ or $E_1 = 2$ then $J_0 := N_0 + E_5$;
if $E_1 = 1$ or $E_1 = 4$ then $\{J_1 := N_0 + E_6; \ldots; J_{m_0} := N_0 + E_{m_0+5}\}$;
if $E_1 = 3$ or $E_1 = 5$ then $\{H_1 := \mathrm{comp}_{E_5}; H_2 := \mathrm{comp}_{E_6}\}$;
if $E_1 = 6$ or $E_1 = 7$ then $H_1 := \mathrm{comp}_{E_5}$;

$\ell_3:$ if $E_1 = 1$ then goto $\tilde{\ell}_1$; $\ldots$; if $E_1 = 7$ then goto $\tilde{\ell}_7$;

$\tilde{\ell}_1:$ if $E_2 = 1$ then $Z_{1,J_0} := f_1^{m_1}(Z_{1,J_1}, \ldots, Z_{1,J_{m_1}})$; $\ldots$;
if $E_2 = n_2$ then $Z_{1,J_0} := f_{n_2}^{m_{n_2}}(Z_{1,J_1}, \ldots, Z_{1,J_{m_{n_2}}})$; goto ℓ_4;

$\tilde{\ell}_2:$ $Z_{1,J_0} := Z_{3,L}$; goto ℓ_4;

$\tilde{\ell}_3:$ $Z_{1,H_1} := Z_{1,H_2}$; goto ℓ_4;

$\tilde{\ell}_4:$ if $E_2 = 1$ & $r_1^{k_1}(Z_{1,J_1}, \ldots, Z_{1,J_{k_1}})$ then goto ℓ_5 else goto ℓ_6; $\ldots$;
if $E_2 = n_3$ & $r_{n_3}^{k_{n_3}}(Z_{1,J_1}, \ldots, Z_{1,J_{k_{n_3}}})$ then goto ℓ_5 else goto ℓ_6;

$\tilde{\ell}_5:$ if $H_1 = H_2$ then goto ℓ_5 else goto ℓ_6;

$\tilde{\ell}_6:$ if $H_1 > N_0 + 1$ then $V := V \mathrm{\ div\ } p_{E_5}^{H_1 - N_0 - 1}$; goto ℓ_4;

$\tilde{\ell}_7:$ $V := V \cdot p_{E_5}$;

$\ell_4:$ $L := L + 1$; goto ℓ_1;

$\ell_5:$ $L := E_3$; goto ℓ_1;

$\ell_6:$ $L := E_4$; goto ℓ_1;

$\ell_7:$ $N := \max\{j \mid p_1^j | V\}$; $H_1 := 1$;

$\ell_8:$ if $N \geq H_1$ then $\{Z_{1,H_1} := Z_{1,N_0+H_1}; H_1 := H_1 + 1;$ goto $\ell_8\}$; $N_0 := N$;

$\ell_9:$ stop.

Subprogram $H_\nu := \mathrm{comp}_{E_\mu}$
if $p_{E_\mu} | V$ then $H_\nu := N_0 + \max\{s \mid p_{E_\mu}^s | V\}$ else $\{V := V \cdot p_{E_\mu}; H_\nu := N_0 + 1\}$

Figure 6: The program $\mathcal{P}_0^{(3)}$ of the universal machine $\mathcal{M}_0^{(3)}$

the length n of the input of $\mathcal{M}$. C is used for storing the Gödel number $\nu_{\mathcal{M}}$, and we use $c(L_P) = \ell_{\mathcal{M}}$. Since any machine can contain only a finite number of index registers, $\mathcal{M}_0^{(3)}$ cannot store all values $c(I_1), \ldots, c(I_{k_{\mathcal{M}}})$ for every possible $k_{\mathcal{M}} \geq 1$ in different index registers. Thus, we use the possibility to store all values of the registers $I_1, \ldots, I_{k_{\mathcal{M}}}$ of $\mathcal{M}$ in one index register of $\mathcal{M}_0^{(3)}$. V contains the product $p_{s_1}^{c(I_{s_1})} \cdots p_{s_\mu}^{c(I_{s_\mu})}$ for the registers $I_{s_1}, \ldots, I_{s_\mu}$ $(s_1, \ldots, s_\mu \leq k_{\mathcal{M}})$ of $\mathcal{M}$ having already been considered during the simulation of $\mathcal{M}$ by $\mathcal{M}_0^{(3)}$ until that time. L and S contain the label and the code, respectively, of the current instruction of $\mathcal{P}_{\mathcal{M}}$, $E_1, E_2, \ldots$ contain the current values $e_1, e_2, \ldots$. $J_0, J_1, \ldots$ contain the values $n_0 + e_5, n_0 + e_6, \ldots$. Since the Z-registers $Z_{1,n_0+1}, Z_{1,n_0+2}, \ldots$ are used for storing all values $c(Z_1), c(Z_2), \ldots$ of the Z-registers of $\mathcal{M}$ during the simulation, the universal machine $\mathcal{M}_0^{(3)}$ assigns the start values $x_1, \ldots, x_n$ to its registers $Z_{1,n_0+1}, \ldots, Z_{1,n_0+n+1}$ and the start values $a_1, \ldots, a_{\ell_{\mathcal{M}}}$ to the registers $Z_{3,1}, \ldots, Z_{3,\ell_{\mathcal{M}}}$ by the subprogram of $\mathcal{P}_0^{(3)}$ that is labelled by 1 at the beginning. $Z_{1,n_0+n+1}, Z_{1,n_0+n+2}, \ldots$ have got the value x_n by the input procedure of $\mathcal{M}_0^{(3)}$. For any $e_1 \leq 7$, $\mathcal{P}_0^{(3)}$ contains a subprogram labelled by $\tilde{\ell}_{e_1}$ for simulating the instructions of type (e_1). In addition to the pseudo instructions listed in Overview 5, the subprograms contain 3-tape pseudo instructions that can be introduced in analogy with the pseudo instructions given in Overview 2 and the like. Since $\mathcal{P}_0^{(3)}$ can also contain only a finite number of strings denoting indices of Z-registers, the F-instructions $Z_j := f_i^{m_i}(Z_{j_1}, \ldots, Z_{j_{m_i}})$ cannot in general be simulated by executing F-instructions of the form $Z_{1,n_0+j} := f_i^{m_i}(Z_{1,n_0+j_1}, \ldots, Z_{1,n_0+j_{m_i}})$. The parameters $n_0 + j, n_0 + j_1, \ldots, n_0 + j_{m_i}$ may vary depending on the machines that should be simulated. Therefore, the index registers $J_0, J_1, \ldots$ get these values before, for $k = m_0 + 13$, the first pseudo instruction given in Overview 4 or a branching instruction with a pseudo condition will be executed. Moreover, further pseudo instructions given in Overview 5 are used before some registers of the second tape $Z_{2,1}, Z_{2,2}, \ldots$ can be used by $\mathcal{M}_0^{(3)}$ for the simulation of an F-instruction or a T-instruction by $\mathcal{M}$. To search an information in $c(C) = |\mathrm{code}_a(\mathcal{P}_{\mathcal{M}})|$ necessary for the simulation of the considered instruction of $\mathcal{M}$, the machine $\mathcal{M}_0^{(3)}$ uses L for storing the current label. The instruction labelled by ℓ_2 in Figure 6 means to search the code of the instruction with label $c(L)$. During computing the code $c(S)$ of the instruction with the current label $c(L)$ from $c(C)$, $p_{c(L)}$ is the corresponding prime number. The subprogram labelled by ℓ_2 in Figure 6 also allows to compute the values $c(E_1) = e_1, c(E_2) = e_2, \ldots$ (as given in Table 2) for the

current instruction with label $c(L)$. The index register E_1 stands for the type of the current instruction and $E_2, \ldots, E_4$ for the used indices i and the labels ℓ_1 and ℓ_2, respectively, in the current instruction of $\mathcal{P}_\mathcal{M}$. Then, depending on the type of the instruction, further values $e_5, e_6, \ldots$ can be made available before simulating the current instruction. The subprograms labelled by ℓ_4, ℓ_5, and ℓ_6 in Figure 6 are intended for determining the label of the next relevant instruction in $\mathcal{P}_\mathcal{M}$. The subprograms labelled by ℓ_7 and ℓ_8 are used for preparing the output of $\mathcal{M}_0^{(3)}$. The subprogram $H_\nu := \mathrm{comp}_{E_\mu}$ allows to compute the values $c(I_{c(E_5)})$ and $c(I_{c(E_6)})$ of the relevant index registers of $\mathcal{M}$ from $c(V)$.

Consequently, by Proposition 1, we have shown the following theorem.

Theorem 1 *For any structure $\mathcal{A}$ with a finite number of operations and relations, there exists an $\mathsf{M}_\mathcal{A}$-universal machine of type 2.*

3.3 Complete problems

The here considered *decision problems* are algorithmic problems completely defined by a decision question for which we want to get the answer yes or the answer no only. For instance, for any structure $\mathcal{A}$ and $\mathsf{I} = \mathsf{M}_\mathcal{A} \times U_\mathcal{A}^\infty$, the halting problem $\mathrm{HP}_\mathcal{A}^t = \{(\mathcal{M}, \vec{x}) \in \mathsf{I} \mid \mathcal{M}(\vec{x}) \downarrow^t\}$ where $\mathcal{M}(\vec{x}) \downarrow^t$ stands for the fact that $\mathcal{M}$ halts (stops) on input $\vec{x}$ after the execution of t transformation steps is a decision problem.

Let $\mathcal{A}$ be a structure of signature $(|N_1|; m_1, \ldots, m_{n_2}; k_1, \ldots, k_{n_3})$. Then, $\mathrm{HP}_\mathcal{A}^t$ is decidable by a machine $\mathcal{M}_0 \in \mathsf{IM}_\mathcal{A}$ with $\mathsf{I}_{\mathcal{M}_0} = \mathsf{I}$, $\mathsf{O}_{\mathcal{M}_0} = \{\mathsf{yes}, \mathsf{no}\}$, $\mathrm{Input}_{\mathcal{M}_0} = \mathsf{init}_{\mathcal{M}_0}$ (as defined by (***)), and the function $\mathrm{Out}_{\mathcal{M}_0}$ defined, for all $(\vec{\nu}, \bar{u}) \in \mathbb{N}^{k_{\mathcal{M}_0}} \times U_\mathcal{A}^\omega$, by $\mathrm{Out}_{\mathcal{M}_0}(\vec{\nu} \,.\, \bar{u}) = \mathsf{yes}$ if $\nu_1 = 1$ and otherwise by $\mathrm{Out}_{\mathcal{M}_0}(\vec{\nu} \,.\, \bar{u}) = \mathsf{no}$. Starting from a configuration given by (***), the execution of $\mathcal{M}$ on $\vec{x}$ can be simulated by $\mathcal{M}_0$ with the help of an $\mathsf{IM}_\mathcal{A}$-universal program derived from $\mathcal{P}_0^{(3)}$ where the simulated steps are simultaneously counted by means of an additional index register. After simulating t steps, $\mathcal{M}_0$ can output yes if the S-instruction of $\mathcal{M}$ is reached and otherwise no. Thus, the corresponding result function $\mathrm{Res}_{\mathcal{M}_0}$ is totally defined on $\mathsf{I}_{\mathcal{M}_0}$.

With respect to BSS RAM's over $\mathcal{A}$, any *decision problem* is a subset $P \subseteq U_\mathcal{A}^\infty$ connected with the question whether $\vec{x} \in U_\mathcal{A}^\infty$ is in P. If $\mathcal{A}$ has at least two constants, denoted here by a and b, we want to assume that a stands for the answer yes and the second constant b stands for the answer no. Under this condition, an $\mathcal{M} \in \mathsf{M}_\mathcal{A}$ *decides* the problem P if the computed function is

the characteristic function $\chi_P : U_{\mathcal{A}}^{\infty} \to \{a, b\}$ such that $\mathrm{Res}_{\mathcal{M}}(\vec{x}) = \chi_P(\vec{x}) = a$ holds for all $\vec{x} \in P$ and $\mathrm{Res}_{\mathcal{M}}(\vec{x}) = \chi_P(\vec{x}) = b$ holds for all $\vec{x} \in U_{\mathcal{A}}^{\infty} \setminus P$. In more general terms, we can define the notion decidable *by a BSS RAM* as follows. A set $P \subseteq U_{\mathcal{A}}^{\infty}$ is a *halting set* (*over* $\mathcal{A}$) if it is the domain of definition of the result function $\mathrm{Res}_{\mathcal{M}}$ for some $\mathcal{M} \in \mathsf{M}_{\mathcal{A}}$. A problem $P \subseteq U_{\mathcal{A}}^{\infty}$ is *semi-decidable* (*over* $\mathcal{A}$) if it is the halting set of a BSS RAM. Consequently, a semi-decidable set $P \subseteq U_{\mathcal{A}}^{\infty}$ does not have to be recursively enumerable by a function $f : U_{\mathcal{A}}^{\infty} \to U_{\mathcal{A}}^{\infty}$ with the properties that f is computable over $\mathcal{A}$, P is the image of f, and $f(\vec{x}) = f(\vec{y})$ holds for all $\vec{x} \in U_{\mathcal{A}}^{n}$ and $\vec{y} \in U_{\mathcal{A}}^{m}$ if $n = m$. $P \subseteq U_{\mathcal{A}}^{\infty}$ is *decidable* (*over* $\mathcal{A}$) if P and its complement $U_{\mathcal{A}}^{\infty} \setminus P$ are semi-decidable over $\mathcal{A}$ and, thus, halting sets of BSS RAM's.

In [16], two hierarchies of decision problems over algebraic structures are defined such that both definitions coincide with the usual descriptions of the arithmetical hierarchy over $(\mathbb{N}; \mathbb{N}; +; =)$ and the definitions given in [10]. For any $Q \subseteq U_{\mathcal{A}}^{\infty}$, let $\mathsf{M}_{\mathcal{A}}^{Q}$ be the class of all oracle BSS RAM's over $\mathcal{A}$ that are able to execute, additionally to the instructions of the types $(1), \ldots, (8)$, all oracle instructions of the form $\mathsf{if}\ (Z_1, \ldots, Z_{I_1}) \in \mathcal{O}\ \mathsf{then\ goto}\ \ell_1\ \mathsf{else\ goto}\ \ell_2$ by evaluating the query $(c(Z_1), \ldots, c(Z_{c(I_1)}) \in Q?$. Then, the first hierarchy consists of the class $\mathcal{A}\text{-}\Sigma_0^0 = \mathrm{DEC}_{\mathcal{A}}$ of all problems decidable by a machine in $\mathsf{M}_{\mathcal{A}}$ and the classes $\mathcal{A}\text{-}\Sigma_n^0 = \bigcup_{Q \in \mathcal{A}\text{-}\Sigma_{n-1}^0} \mathrm{SDEC}_{\mathcal{A}}^{Q}$ of all problems semi-decidable by a machine in $\mathsf{M}_{\mathcal{A}}^{Q}$ for some $Q \in \mathcal{A}\text{-}\Sigma_{n-1}^0$. Whereas in [16], the halting problems are considered only for structures with two constants a and b that are effectively distinguishable, we can now define further *halting problems* as follows, where the oracle instructions are encoded by $2^9 \cdot 5^{\ell_1} \cdot 7^{\ell_2}$ and $\mathcal{M}(\vec{x}) \downarrow$ means $\mathcal{M}(\vec{x}) \downarrow^t$ for some $t \geq 1$.

$$\mathbb{H}_{\mathcal{A}}^{Q} =_{\mathrm{df}} \{(\vec{a}^{\,(\mathcal{M},x_1)} \,.\, \vec{x} \,.\, \mathrm{code}_{n,x_n}(\mathcal{P}_{\mathcal{M}})) \mid \mathcal{M} \in \mathsf{M}_{\mathcal{A}}^{Q} \ \&\ \vec{x} \in U_{\mathcal{A}}^{\infty} \ \&\ \mathcal{M}(\vec{x}) \downarrow\}$$

If $(|N_1|; m_1, \ldots, m_{n_2}; k_1, \ldots, k_{n_3})$ is the signature of $\mathcal{A}$, then let $\mathcal{M}_0^{Q} \in \mathsf{M}_{\mathcal{A}}^{Q}$ be the universal oracle machine defined analogously to the universal BSS RAM considered in Section 3.2. If the set of constants of $\mathcal{A}$ is decidable, then $\mathsf{I} =_{\mathrm{df}} \{(\vec{a}^{\,(\mathcal{M},x_1)} \,.\, \vec{x} \,.\, \mathrm{code}_{n,x_n}(\mathcal{P}_{\mathcal{M}})) \mid (\mathcal{M}, \vec{x}) \in \mathsf{M}_{\mathcal{A}}^{Q} \times U_{\mathcal{A}}^{\infty}\}$ is decidable and $\mathbb{H}_{\mathcal{A}}^{Q} \subseteq U_{\mathcal{A}}^{\infty}$ is semi-decidable by a machine in $\mathsf{M}_{\mathcal{A}}^{Q}$ that uses $\mathcal{P}_{\mathcal{M}_0^{Q}}$ for all inputs in I and does not halt for all inputs in $U_{\mathcal{A}}^{\infty} \setminus \mathsf{I}$. By generalising Theorem 1 we get the following result.

Theorem 2 *For any structure $\mathcal{A}$ with a decidable set of constants and a finite number of operations and relations, the Halting problem $\mathbb{H}_{\mathcal{A}}^{Q}$ is semi-decidable by a universal oracle machine in $\mathsf{M}_{\mathcal{A}}^{Q}$.*

The decision problems $\mathbb{H}_{\mathcal{A}}^{(n)} =_{\mathrm{df}} \mathbb{H}_{\mathcal{A}}^{\mathbb{H}_{\mathcal{A}}^{(n-1)}}$ (for $n > 0$) form a sequence of halting problems resulting from the so-called jumps where $\mathbb{H}_{\mathcal{A}}^{(0)} =_{\mathrm{df}} \emptyset$ and each $\mathbb{H}_{\mathcal{A}}^{(n)}$ is semi-decidable by a machine in $\mathsf{M}_{\mathcal{A}}^{\mathbb{H}_{\mathcal{A}}^{(n-1)}}$. Let $P \subseteq U_{\mathcal{A}}^{\infty}$ and $Q \subseteq U_{\mathcal{A}}^{\infty}$ be given. Then, by analogy with the classical case (cf. [24], p. 50 and p. 19), we say that P is *Turing reducible to* Q (denoted by $P \preceq_{\mathcal{A},T} Q$) if P is decidable by a machine in $\mathsf{M}_{\mathcal{A}}^{Q}$ and P *is one-one reducible to* Q (denoted by $P \preceq_{\mathcal{A},1} Q$) if there is an $\mathcal{M} \in \mathsf{M}_{\mathcal{A}}$ computing a total and injective function $\mathrm{Res}_{\mathcal{M}}$ such that, for all $\vec{x} \in U_{\mathcal{A}}^{\infty}$, $\mathrm{Res}_{\mathcal{M}}(\vec{x}) \in Q$ holds if and only if $\vec{x} \in P$ holds. $\mathbb{H}_{\mathcal{A}}^{(n)}$ is called *complete* in $\mathcal{A}\text{-}\Sigma_n^0$ since any problem of this class is one-one-reducible to it. This means that $\mathcal{A}\text{-}\Sigma_n^0 = \{P \subseteq U_{\mathcal{A}}^{\infty} \mid P \preceq_{\mathcal{A},1} \mathbb{H}_{\mathcal{A}}^{(n)}\}$ and $(\mathcal{A}\text{-}\Sigma_n^0) \cap \{U_{\mathcal{A}}^{\infty} \setminus P \mid P \in \mathcal{A}\text{-}\Sigma_n^0\} = \{P \subseteq U_{\mathcal{A}}^{\infty} \mid P \preceq_{\mathcal{A},T} \mathbb{H}_{\mathcal{A}}^{(n-1)}\}$ hold.

The described universal oracle $\mathcal{A}$-machines allow to extend the results from [16]. Note that the discussion about the existence of universal BSS RAM's is also very helpful for characterising complexity classes by complete problems, in particular, for structures without constants and for groups (see e.g. [12]).

References

[1] Aho, A. V., J. E. Hopcroft, and J. D. Ullman: *The design and analysis of computer algorithms*. Addison-Wesley (1974).

[2] Asser, G.: *Einführung in die mathematische Logik, Teil 2, Prädikatenkalkül der ersten Stufe*. BSB B. G. Teubner Verlagsgesellschaft (1975).

[3] Blum, L., M. Shub, and S. Smale: *On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines*. Bulletin of the Amer. Math. Soc. 21 (1989), 1–46.

[4] Blum, L., F. Cucker, M. Shub, and S. Smale: *Complexity and real computation*. Springer (1998).

[5] Börger, E.: *Berechenbarkeit, Komplexität, Logik*. Vieweg (1992). In English: *Computability, complexity, logic*. Elsevier (1889).

[6] Bournez, O. and A. Pouly: *A survey on analog models of computation*. arXiv:1805.05729 (2018).

[7] Brattka, V., and P. Hertling: *Feasible real random access machines*. Journal of complexity 14 (1998), 490–526.

[8] Hopcroft, J. E., and J. D. Ullman: *Einführung in die Automatentheorie, formale Sprachen und Komplexitätstheorie*. Addison-Wesley (1993). In En-

glish: *Introduction to automata theory, languages, and computation.* Addison-Wesley (1979).

[9] Rogers, H.: *Theory of recursive functions and effective computability.* McGraw-Hill (1967).

[10] Cucker, F.: *The arithmetical hierarchy over the reals.* Journal of Logic and Computation 2 (3) (1992), 375–395.

[11] Cucker, F., and A. Torrecillas: *Two P-complete problems in the theory of the reals.* Journal of Complexity 8 (1992), 454–466.

[12] Gaßner, C.: *Computation over groups.* In: Arnold Beckmann, Costas Dimitracopoulos, and Benedikt Löwe (eds.): Logic and Theory of Algorithms (2008), 147–156.

[13] Gaßner, C.: *Oracles and relativizations of the P =? NP question for several structures.* Journal of Universal Computer Science 15 (6) (2009), 1186–1205.

[14] Gaßner, C.: *Gelöste und offene P-NP-Probleme über verschiedenen Strukturen.* Habilitationsschrift, Greifswald (2011).

[15] Gaßner, C.: *Strong Turing degrees for additive BSS RAM's.* Logical Methods in Computer Science 9 (4:25) (2013), 1–18.

[16] Gaßner, C.: *Computation over algebraic structures and a classification of undecidable problems.* Mathematical Structures in Computer Science 27 (8) (2017) 1386–1413.

[17] Hemmerling, A.: *Computability of string functions over algebraic structures.* Mathematical Logic Quarterly 44 (1998), 1–44.

[18] Kondakow, N. I.: *Wörterbuch der Logik.* Erhard Albrecht, Günter Asser (eds.); Bibliographisches Institut, Leipzig (1983).

[19] Moschovakis, Y. N.: *Abstract first order computability. I.* Transactions of the American Mathematical Society 138 (1969), 427–464.

[20] Poizat, B.: *Les petits cailloux.* Aléas (1995).

[21] Preparata, F. P., and M. I. Shamos: *Computational geometry: an introduction.* Springer (1985).

[22] Scott, D.: *Some definitional suggestions for automata theory.* Journal of Computer and System Sciences 1 (1967), 187–212.

[23] Shepherdson, J. C., and H. E. Sturgis: *Computability of recursive functions.* Journal of the Association for Computing Machinery 10 (1963), 217–255.

[24] Soare, R. I.: *Recursively enumerable sets and degrees: a study of computable functions and computably generated sets.* Springer (1987).