JOURNAL OF FORMALIZED MATHEMATICS
Volume 4, Released 1992, Published 2003
Inst. of Computer Science, Univ. of Białystok

A Mathematical Model of CPU

Yatsuka Nakamura
Shinshu University
Nagano

Andrzej Trybulec
Warsaw University
Białystok

Summary. This paper is based on a previous work of the first author [13] in which a mathematical model of the computer has been presented. The model deals with random access memory, such as RASP of C. C. Elgot and A. Robinson [12], however, it allows for a more realistic modeling of real computers. This new model of computers has been named by the author (Y. Nakamura, [13]) Architecture Model for Instructions (AMI). It is more developed than previous models, both in the description of hardware (e.g., the concept of the program counter, the structure of memory) as well as in the description of instructions (instruction codes, addresses). The structure of AMI over an arbitrary collection of mathematical domains N consists of:

- a non-empty set of objects,
- the instruction counter,
- a non-empty set of objects called instruction locations,
- a non-empty set of instruction codes,
- an instruction code for halting,
- a set of instructions that are ordered pairs with the first element being an instruction code and the second a finite sequence in which members are either objects of the AMI or elements of one of the domains included in N,
- a function that assigns to every object of AMI its kind that is either *an instruction* or *an instruction location* or an element of N,
- a function that assigns to every instruction its execution that is again a function mapping states of AMI into the set of states.

By a state of AMI we mean a function that assigns to every object of AMI an element of the same kind. In this paper we develop the theory of AMI. Some properties of AMI are introduced ensuring it to have some properties of real computers:

- a von Neumann AMI, in which only addresses to instruction locations are stored in the program counter,
- data oriented, those in which instructions cannot be stored in data locations,
- halting, in which the execution of the halt instruction is the identity mapping of the states of an AMI,
- steady programmed, the condition in which the contents of the instruction locations do not change during execution,
- definite, a property in which only instructions may be stored in instruction locations.

We present an example of AMI called a Small Concrete Model which has been constructed in [13]. The Small Concrete Model has only one kind of data: integers and a set of instructions, small but sufficient to cope with integers.

MML Identifier: `AMI_1`.

WWW: `http://mizar.org/JFM/Vol4/ami_1.html`

The articles [15], [8], [17], [16], [2], [18], [5], [6], [11], [1], [9], [14], [10], [3], [4], and [7] provide the notation and terminology for this paper.

1. Preliminaries

The following four propositions are true:

(1) $\mathbb{N} \neq \mathbb{Z}$.

(2) For all sets a, b holds $1 \neq \langle a, b \rangle$.

(3) For all sets a, b holds $2 \neq \langle a, b \rangle$.

(5)[1] For all sets a, b, c, d such that $a \neq b$ holds $\prod[a \longmapsto \{c\}, b \longmapsto \{d\}] = \{[a \longmapsto c, b \longmapsto d]\}$.

Let I_1 be a set. We say that I_1 has non empty elements if and only if:

(Def. 1) $\emptyset \notin I_1$.

Let us observe that there exists a set which is non empty and has non empty elements.

Let A be a non empty set. Note that $\{A\}$ has non empty elements. Let B be a non empty set. Note that $\{A, B\}$ has non empty elements.

Let A, B be sets with non empty elements. One can check that $A \cup B$ has non empty elements.

2. General concepts

In the sequel N is a set with non empty elements.

Let N be a set. We consider AMI's over N as extensions of 1-sorted structure as systems

⟨ a carrier, an instruction counter, instruction locations, instruction codes, instructions, a object kind, a execution ⟩,

where the carrier is a set, the instruction counter is an element of the carrier, the instruction locations constitute a subset of the carrier, the instruction codes constitute a non empty set, the instructions constitute a non empty subset of $[:$the instruction codes, $(\bigcup N \cup$ the carrier$)^*:]$, the object kind is a function from the carrier into $N \cup \{$the instructions, the instruction locations$\}$, and the execution is a function from the instructions into $(\prod$the object kind$)^{\prod \text{the object kind}}$.

Let N be a set. The functor $\mathbf{AMI}_{\mathrm{t}}$ yielding a strict AMI over N is defined by the conditions (Def. 2).

(Def. 2) The carrier of $(\mathbf{AMI}_{\mathrm{t}}) = \{0, 1\}$ and the instruction counter of $(\mathbf{AMI}_{\mathrm{t}}) = 0$ and the instruction locations of $(\mathbf{AMI}_{\mathrm{t}}) = \{1\}$ and the instruction codes of $(\mathbf{AMI}_{\mathrm{t}}) = \{0\}$ and the instructions of $(\mathbf{AMI}_{\mathrm{t}}) = \{\langle 0, \emptyset \rangle\}$ and the object kind of $(\mathbf{AMI}_{\mathrm{t}}) = [0 \longmapsto \{1\}, 1 \longmapsto \{\langle 0, \emptyset \rangle\}]$ and the execution of $(\mathbf{AMI}_{\mathrm{t}}) = \{\langle 0, \emptyset \rangle\} \longmapsto \mathrm{id}_{\prod[0 \longmapsto \{1\}, 1 \longmapsto \{\langle 0, \emptyset \rangle\}]}$.

Let N be a set and let S be an AMI over N. We say that S is void if and only if:

(Def. 3) The instruction locations of S are empty.

Let N be a set. Note that $\mathbf{AMI}_{\mathrm{t}}$ is non empty and non void.

Let N be a set. Note that there exists an AMI over N which is non empty and non void.

Let N be a set and let S be a non empty AMI over N. Note that the carrier of S is non empty.

Let N be a set and let S be a non void AMI over N. Observe that the instruction locations of S is non empty.

Let N be a set and let S be a non empty AMI over N. An object of S is an element of S.

Let N be a set and let S be a non empty non void AMI over N. An instruction-location of S is an element of the instruction locations of S.

Let N be a set and let S be an AMI over N. An instruction of S is an element of the instructions of S.

Let N be a set and let S be a non empty AMI over N. The functor $\mathbf{IC}_S$ yields an object of S and is defined as follows:

[1] The proposition (4) has been removed.

(Def. 5)[2] $\mathbf{IC}_S =$ the instruction counter of S.

Let N be a set, let S be a non empty AMI over N, and let o be an object of S. The functor ObjectKind(o) yields an element of $N \cup \{$the instructions of S, the instruction locations of $S\}$ and is defined as follows:

(Def. 6) ObjectKind$(o) =$ (the object kind of $S)(o)$.

Let f be a function. Observe that $\prod f$ is functional.

Let A be a set, let B be a set with non empty elements, and let f be a function from A into B. Note that $\prod f$ is non empty.

Let N be a set and let S be an AMI over N. A state of S is an element of $\prod$(the object kind of S).

Let N be a set with non empty elements, let S be a non void AMI over N, let I be an instruction of S, and let s be a state of S. The functor Exec(I,s) yielding a state of S is defined as follows:

(Def. 7) Exec$(I,s) =$ (the execution of $S)(I)(s)$.

Let us consider N, let S be a non void AMI over N, and let I be an instruction of S. We say that I is halting if and only if:

(Def. 8) For every state s of S holds Exec$(I,s) = s$.

Let us consider N and let S be a non void AMI over N. We say that S is halting if and only if:

(Def. 9) There exists an instruction I of S such that I is halting and for every instruction J of S such that J is halting holds $I = J$.

In the sequel E is a set.

The following proposition is true

(6) $\mathbf{AMI}_{\mathrm{t}}$ is halting.

Let us consider N. Note that $\mathbf{AMI}_{\mathrm{t}}$ is halting.

Let us consider N. One can verify that there exists a non void AMI over N which is halting.

Let us consider N and let S be a halting non void AMI over N. The functor $\mathbf{halt}_S$ yields an instruction of S and is defined as follows:

(Def. 10) There exists an instruction I of S such that I is halting and $\mathbf{halt}_S = I$.

Let us consider N and let S be a halting non void AMI over N. One can check that $\mathbf{halt}_S$ is halting.

Let N be a set and let I_1 be a non empty AMI over N. We say that I_1 is IC-Ins-separated if and only if:

(Def. 11) ObjectKind$(\mathbf{IC}_{(I_1)}) =$ the instruction locations of I_1.

Let N be a set and let I_1 be an AMI over N. We say that I_1 is data-oriented if and only if:

(Def. 12) (The object kind of $I_1)^{-1}(\{$the instructions of $I_1\}) \subseteq$ the instruction locations of I_1.

Let N be a set with non empty elements and let I_1 be a non empty non void AMI over N. We say that I_1 is steady-programmed if and only if:

(Def. 13) For every state s of I_1 and for every instruction i of I_1 and for every instruction-location l of I_1 holds $(\mathrm{Exec}(i,s))(l) = s(l)$.

Let N be a set and let I_1 be a non empty non void AMI over N. We say that I_1 is definite if and only if:

(Def. 14) For every instruction-location l of I_1 holds ObjectKind$(l) =$ the instructions of I_1.

Next we state several propositions:

[2] The definition (Def. 4) has been removed.

(7) $\mathbf{AMI}_{\mathrm{t}}$ is IC-Ins-separated.

(8) $\mathbf{AMI}_{\mathrm{t}}$ is data-oriented.

(9) For all states s_1, s_2 of $\mathbf{AMI}_{\mathrm{t}}$ holds $s_1 = s_2$.

(10) $\mathbf{AMI}_{\mathrm{t}}$ is steady-programmed.

(11) $\mathbf{AMI}_{\mathrm{t}}$ is definite.

Let E be a set. One can verify that $\mathbf{AMI}_{\mathrm{t}}$ is data-oriented.

Let E be a set. Observe that $\mathbf{AMI}_{\mathrm{t}}$ is IC-Ins-separated and definite.

Let N be a set with non empty elements. Observe that $\mathbf{AMI}_{\mathrm{t}}$ is steady-programmed.

Let E be a set. One can verify that there exists an AMI over E which is data-oriented and strict.

Let M be a set. Note that there exists a non empty non void AMI over M which is IC-Ins-separated, data-oriented, definite, and strict.

Let us consider N. One can check that there exists a non empty non void AMI over N which is IC-Ins-separated, data-oriented, halting, steady-programmed, definite, and strict.

Let N be a set with non empty elements, let S be an IC-Ins-separated non empty non void AMI over N, and let s be a state of S. The functor $\mathbf{IC}_s$ yields an instruction-location of S and is defined by:

(Def. 15) $\mathbf{IC}_s = s(\mathbf{IC}_S)$.

3. Preliminaries

We adopt the following convention: x, y, z, A, B denote sets, f, g, h denote functions, and i, j, k denote natural numbers.

The following propositions are true:

(13)[3] For every function f holds $\pi_1(\mathrm{dom}\, f \times \mathrm{rng}\, f)^\circ f = \mathrm{dom}\, f$.

(14) If $f \approx g$ and $\langle x, y\rangle \in f$ and $\langle x, z\rangle \in g$, then $y = z$.

(15) Suppose for every x such that $x \in A$ holds x is a function and for all functions f, g such that $f \in A$ and $g \in A$ holds $f \approx g$. Then $\bigcup A$ is a function.

(16) If $\mathrm{dom}\, f \subseteq A \cup B$, then $f{\restriction}A +\cdot f{\restriction}B = f$.

(18)[4] For all sets x_1, x_2, y_1, y_2 holds $[x_1 \longmapsto y_1, x_2 \longmapsto y_2] = (x_1 \dot{\longmapsto} y_1) +\cdot (x_2 \dot{\longmapsto} y_2)$.

(19) For all x, y holds $x \dot{\longmapsto} y = \{\langle x, y\rangle\}$.

(20) For all sets a, b, c holds $[a \longmapsto b, a \longmapsto c] = a \dot{\longmapsto} c$.

(21) For every function f holds $\mathrm{dom}\, f$ is finite iff f is finite.

(22) If $x \in \prod f$, then x is a function.

4. Superproducts

Let f be a function. The functor $\prod^{\cdot} f$ yields a set and is defined by:

(Def. 16) $x \in \prod^{\cdot} f$ iff there exists g such that $x = g$ and $\mathrm{dom}\, g \subseteq \mathrm{dom}\, f$ and for every x such that $x \in \mathrm{dom}\, g$ holds $g(x) \in f(x)$.

Let f be a function. Note that $\prod^{\cdot} f$ is functional and non empty.

We now state a number of propositions:

[3] The proposition (12) has been removed.

[4] The proposition (17) has been removed.

(25)[5] If $g \in \prod^{\cdot} f$, then $\operatorname{dom} g \subseteq \operatorname{dom} f$ and for every x such that $x \in \operatorname{dom} g$ holds $g(x) \in f(x)$.

(26) $\emptyset \in \prod^{\cdot} f$.

(27) $\prod f \subseteq \prod^{\cdot} f$.

(28) If $x \in \prod^{\cdot} f$, then x is a partial function from $\operatorname{dom} f$ to $\bigcup \operatorname{rng} f$.

(29) If $g \in \prod f$ and $h \in \prod^{\cdot} f$, then $g{+}\cdot h \in \prod f$.

(30) If $\prod f \neq \emptyset$, then $g \in \prod^{\cdot} f$ iff there exists h such that $h \in \prod f$ and $g \leq h$.

(31) $\prod^{\cdot} f \subseteq \operatorname{dom} f \dot{\to} \bigcup \operatorname{rng} f$.

(32) If $f \subseteq g$, then $\prod^{\cdot} f \subseteq \prod^{\cdot} g$.

(33) $\prod^{\cdot} \emptyset = \{\emptyset\}$.

(34) $A \dot{\to} B = \prod^{\cdot} (A \longmapsto B)$.

(35) For all non empty sets A, B and for every function f from A into B holds $\prod^{\cdot} f = \prod^{\cdot} (f \restriction \{x; x$ ranges over elements of A: $f(x) \neq \emptyset\})$.

(36) If $x \in \operatorname{dom} f$ and $y \in f(x)$, then $x \dot{\longmapsto} y \in \prod^{\cdot} f$.

(37) $\prod^{\cdot} f = \{\emptyset\}$ iff for every x such that $x \in \operatorname{dom} f$ holds $f(x) = \emptyset$.

(38) If $A \subseteq \prod^{\cdot} f$ and for all functions h_1, h_2 such that $h_1 \in A$ and $h_2 \in A$ holds $h_1 \approx h_2$, then $\bigcup A \in \prod^{\cdot} f$.

(39) If $g \approx h$ and $g \in \prod^{\cdot} f$ and $h \in \prod^{\cdot} f$, then $g \cup h \in \prod^{\cdot} f$.

(40) If $g \subseteq h$ and $h \in \prod^{\cdot} f$, then $g \in \prod^{\cdot} f$.

(41) If $g \in \prod^{\cdot} f$, then $g \restriction A \in \prod^{\cdot} f$.

(42) If $g \in \prod^{\cdot} f$, then $g \restriction A \in \prod^{\cdot} (f \restriction A)$.

(43) If $h \in \prod^{\cdot} (f{+}\cdot g)$, then there exist functions f', g' such that $f' \in \prod^{\cdot} f$ and $g' \in \prod^{\cdot} g$ and $h = f'{+}\cdot g'$.

(44) For all functions f', g' such that $\operatorname{dom} g$ misses $\operatorname{dom} f' \setminus \operatorname{dom} g'$ and $f' \in \prod^{\cdot} f$ and $g' \in \prod^{\cdot} g$ holds $f'{+}\cdot g' \in \prod^{\cdot} (f{+}\cdot g)$.

(45) For all functions f', g' such that $\operatorname{dom} f'$ misses $\operatorname{dom} g \setminus \operatorname{dom} g'$ and $f' \in \prod^{\cdot} f$ and $g' \in \prod^{\cdot} g$ holds $f'{+}\cdot g' \in \prod^{\cdot} (f{+}\cdot g)$.

(46) If $g \in \prod^{\cdot} f$ and $h \in \prod^{\cdot} f$, then $g{+}\cdot h \in \prod^{\cdot} f$.

(47) For all sets x_1, x_2, y_1, y_2 such that $x_1 \in \operatorname{dom} f$ and $y_1 \in f(x_1)$ and $x_2 \in \operatorname{dom} f$ and $y_2 \in f(x_2)$ holds $[x_1 \longmapsto y_1, x_2 \longmapsto y_2] \in \prod^{\cdot} f$.

5. General theory

Let us consider N, let S be an IC-Ins-separated definite non empty non void AMI over N, and let s be a state of S. The functor $\operatorname{CurInstr}(s)$ yielding an instruction of S is defined by:

(Def. 17) $\operatorname{CurInstr}(s) = s(\mathbf{IC}_s)$.

Let us consider N, let S be an IC-Ins-separated definite non empty non void AMI over N, and let s be a state of S. The functor $\operatorname{Following}(s)$ yields a state of S and is defined by:

(Def. 18) $\operatorname{Following}(s) = \operatorname{Exec}(\operatorname{CurInstr}(s), s)$.

[5] The propositions (23) and (24) have been removed.

Let us consider N, let S be an IC-Ins-separated definite non empty non void AMI over N, and let s be a state of S. The functor Computation(s) yielding a function from $\mathbb{N}$ into $\prod$(the object kind of S) is defined as follows:

(Def. 19) $(\text{Computation}(s))(0) = s$ and for every i holds $(\text{Computation}(s))(i+1) = \text{Following}((\text{Computation}(s))(i))$.

Let us consider N, let S be a non void AMI over N, let f be a function from $\mathbb{N}$ into $\prod$(the object kind of S), and let us consider k. Then $f(k)$ is a state of S.

Let us consider N, let S be a halting IC-Ins-separated definite non empty non void AMI over N, and let I_1 be a state of S. We say that I_1 is halting if and only if:

(Def. 20) There exists k such that $\text{CurInstr}((\text{Computation}(I_1))(k)) = \mathbf{halt}_S$.

Let N be a set and let I_1 be an AMI over N. We say that I_1 is realistic if and only if:

(Def. 21) The instructions of $I_1 \neq$ the instruction locations of I_1.

The following proposition is true

(48) Let S be an IC-Ins-separated definite non empty non void AMI over E. Suppose S is realistic. Then it is not true that there exists an instruction-location l of S such that $\mathbf{IC}_S = l$.

In the sequel S is an IC-Ins-separated definite non empty non void AMI over N and s is a state of S.

One can prove the following two propositions:

(51)[6] For every k holds $(\text{Computation}(s))(i+k) = (\text{Computation}((\text{Computation}(s))(i)))(k)$.

(52) Suppose $i \leq j$. Let given N, S be a halting IC-Ins-separated definite non empty non void AMI over N, and s be a state of S. If $\text{CurInstr}((\text{Computation}(s))(i)) = \mathbf{halt}_S$, then $(\text{Computation}(s))(j) = (\text{Computation}(s))(i)$.

Let us consider N, let S be a halting IC-Ins-separated definite non empty non void AMI over N, and let s be a state of S. Let us assume that s is halting. The functor Result(s) yielding a state of S is defined by:

(Def. 22) There exists k such that $\text{Result}(s) = (\text{Computation}(s))(k)$ and $\text{CurInstr}(\text{Result}(s)) = \mathbf{halt}_S$.

Next we state the proposition

(53) Let S be a steady-programmed IC-Ins-separated definite non empty non void AMI over N, s be a state of S, and i be an instruction-location of S. Then $s(i) = (\text{Following}(s))(i)$.

Let us consider N, let S be a definite non empty non void AMI over N, let s be a state of S, and let l be an instruction-location of S. Then $s(l)$ is an instruction of S.

One can prove the following four propositions:

(54) Let S be a steady-programmed IC-Ins-separated definite non empty non void AMI over N, s be a state of S, i be an instruction-location of S, and given k. Then $s(i) = (\text{Computation}(s))(k)(i)$.

(55) Let S be a steady-programmed IC-Ins-separated definite non empty non void AMI over N and s be a state of S. Then $(\text{Computation}(s))(k+1) = \text{Exec}(s(\mathbf{IC}_{(\text{Computation}(s))(k)}), (\text{Computation}(s))(k))$.

(56) Let S be a steady-programmed IC-Ins-separated halting definite non empty non void AMI over N, s be a state of S, and given k. If $s(\mathbf{IC}_{(\text{Computation}(s))(k)}) = \mathbf{halt}_S$, then $\text{Result}(s) = (\text{Computation}(s))(k)$.

(57) Let S be a steady-programmed IC-Ins-separated halting definite non empty non void AMI over N and s be a state of S. If there exists k such that $s(\mathbf{IC}_{(\text{Computation}(s))(k)}) = \mathbf{halt}_S$, then for every i holds $\text{Result}(s) = \text{Result}((\text{Computation}(s))(i))$.

Let us consider N, let S be a non empty non void AMI over N, and let o be an object of S. Note that ObjectKind(o) is non empty.

[6] The propositions (49) and (50) have been removed.

6. Finite substates

Let N be a set and let S be an AMI over N. The functor $\mathrm{FinPartSt}(S)$ yielding a subset of $\prod^{\cdot}$(the object kind of S) is defined as follows:

(Def. 23) $\mathrm{FinPartSt}(S) = \{p; p$ ranges over elements of $\prod^{\cdot}$(the object kind of S): p is finite$\}$.

Let N be a set and let S be an AMI over N. An element of $\prod^{\cdot}$(the object kind of S) is said to be a finite partial state of S if:

(Def. 24) It is finite.

Let us consider N, let S be an IC-Ins-separated definite non empty non void AMI over N, and let I_1 be a finite partial state of S. We say that I_1 is autonomic if and only if:

(Def. 25) For all states s_1, s_2 of S such that $I_1 \subseteq s_1$ and $I_1 \subseteq s_2$ and for every i holds $(\mathrm{Computation}(s_1))(i)\restriction \mathrm{dom}\, I_1 = (\mathrm{Computation}(s_2))(i)\restriction \mathrm{dom}\, I_1$.

Let us consider N, let S be a halting IC-Ins-separated definite non empty non void AMI over N, and let I_1 be a finite partial state of S. We say that I_1 is halting if and only if:

(Def. 26) For every state s of S such that $I_1 \subseteq s$ holds s is halting.

Let us consider N and let I_1 be an IC-Ins-separated definite non empty non void AMI over N. We say that I_1 is programmable if and only if:

(Def. 27) There exists a finite partial state of I_1 which is non empty and autonomic.

Next we state two propositions:

(58) Let S be an IC-Ins-separated definite non empty non void AMI over N, A, B be sets, and l_1, l_2 be objects of S. Suppose $\mathrm{ObjectKind}(l_1) = A$ and $\mathrm{ObjectKind}(l_2) = B$. Let a be an element of A and b be an element of B. Then $[l_1 \longmapsto a, l_2 \longmapsto b]$ is a finite partial state of S.

(59) Let S be an IC-Ins-separated definite non empty non void AMI over N, A be a set, and l_1 be an object of S. Suppose $\mathrm{ObjectKind}(l_1) = A$. Let a be an element of A. Then $l_1 \dot{\longmapsto} a$ is a finite partial state of S.

Let us consider N, let S be an IC-Ins-separated definite non empty non void AMI over N, let l_1 be an object of S, and let a be an element of $\mathrm{ObjectKind}(l_1)$. Then $l_1 \dot{\longmapsto} a$ is a finite partial state of S.

Let us consider N, let S be an IC-Ins-separated definite non empty non void AMI over N, let l_1, l_2 be objects of S, let a be an element of $\mathrm{ObjectKind}(l_1)$, and let b be an element of $\mathrm{ObjectKind}(l_2)$. Then $[l_1 \longmapsto a, l_2 \longmapsto b]$ is a finite partial state of S.

The following propositions are true:

(60) $\mathbf{AMI}_{\mathrm{t}}$ is realistic.

(61) $\mathbf{AMI}_{\mathrm{t}}$ is programmable.

Let us consider E. Note that $\mathbf{AMI}_{\mathrm{t}}$ is realistic.

Let us consider N. Note that $\mathbf{AMI}_{\mathrm{t}}$ is programmable.

Let us consider E. Observe that there exists an AMI over E which is data-oriented, realistic, and strict.

Let M be a set. One can verify that there exists a non empty non void AMI over M which is data-oriented, realistic, strict, IC-Ins-separated, and definite.

Let us consider N. Note that there exists an IC-Ins-separated definite non empty non void AMI over N which is data-oriented, halting, steady-programmed, realistic, programmable, and strict.

We now state two propositions:

(62) Let S be a non void AMI over N, s be a state of S, and p be a finite partial state of S. Then $s\restriction \mathrm{dom}\, p$ is a finite partial state of S.

(63) For every set N and for every AMI S over N holds $\emptyset$ is a finite partial state of S.

Let us consider N and let S be a programmable IC-Ins-separated definite non empty non void AMI over N. Observe that there exists a finite partial state of S which is non empty and autonomic.

Let N be a set, let S be an AMI over N, and let f, g be finite partial states of S. Then $f{+}\!\cdot g$ is a finite partial state of S.

7. Preprograms

The following propositions are true:

(64) Let S be a halting realistic IC-Ins-separated definite non empty non void AMI over N, l_3 be an instruction-location of S, and l be an element of ObjectKind$(\mathbf{IC}_S)$. Suppose $l = l_3$. Let h be an element of ObjectKind(l_3). If $h = \mathbf{halt}_S$, then for every state s of S such that $[\mathbf{IC}_S \longmapsto l, l_3 \longmapsto h] \subseteq s$ holds $\mathrm{CurInstr}(s) = \mathbf{halt}_S$.

(65) Let S be a halting realistic IC-Ins-separated definite non empty non void AMI over N, l_3 be an instruction-location of S, and l be an element of ObjectKind$(\mathbf{IC}_S)$. Suppose $l = l_3$. Let h be an element of ObjectKind(l_3). If $h = \mathbf{halt}_S$, then $[\mathbf{IC}_S \longmapsto l, l_3 \longmapsto h]$ is halting.

(66) Let S be a realistic halting IC-Ins-separated definite non empty non void AMI over N, l_3 be an instruction-location of S, and l be an element of ObjectKind$(\mathbf{IC}_S)$. Suppose $l = l_3$. Let h be an element of ObjectKind(l_3). Suppose $h = \mathbf{halt}_S$. Let s be a state of S. If $[\mathbf{IC}_S \longmapsto l, l_3 \longmapsto h] \subseteq s$, then for every i holds $(\mathrm{Computation}(s))(i) = s$.

(67) Let S be a realistic halting IC-Ins-separated definite non empty non void AMI over N, l_3 be an instruction-location of S, and l be an element of ObjectKind$(\mathbf{IC}_S)$. Suppose $l = l_3$. Let h be an element of ObjectKind(l_3). If $h = \mathbf{halt}_S$, then $[\mathbf{IC}_S \longmapsto l, l_3 \longmapsto h]$ is autonomic.

Let us consider N and let S be a realistic halting IC-Ins-separated definite non empty non void AMI over N. One can check that there exists a finite partial state of S which is autonomic and halting.

Let us consider N and let S be a realistic halting IC-Ins-separated definite non empty non void AMI over N. A pre-program of S is an autonomic halting finite partial state of S.

Let us consider N, let S be a realistic halting IC-Ins-separated definite non empty non void AMI over N, and let s be a finite partial state of S. Let us assume that s is a pre-program of S. The functor $\mathrm{Result}(s)$ yielding a finite partial state of S is defined as follows:

(Def. 28) For every state s' of S such that $s \subseteq s'$ holds $\mathrm{Result}(s) = \mathrm{Result}(s') \restriction \mathrm{dom}\, s$.

8. Computability

Let us consider N, let S be a realistic halting IC-Ins-separated definite non empty non void AMI over N, let p be a finite partial state of S, and let F be a function. We say that p computes F if and only if the condition (Def. 29) is satisfied.

(Def. 29) Let x be a set. Suppose $x \in \mathrm{dom} F$. Then there exists a finite partial state s of S such that $x = s$ and $p{+}\!\cdot s$ is a pre-program of S and $F(s) \subseteq \mathrm{Result}(p{+}\!\cdot s)$.

One can prove the following propositions:

(68) Let S be a realistic halting IC-Ins-separated definite non empty non void AMI over N and p be a finite partial state of S. Then p computes $\emptyset$.

(69) Let S be a realistic halting IC-Ins-separated definite non empty non void AMI over N and p be a finite partial state of S. Then p is a pre-program of S if and only if p computes $\emptyset \dot{\longmapsto} \mathrm{Result}(p)$.

(70) Let S be a realistic halting IC-Ins-separated definite non empty non void AMI over N and p be a finite partial state of S. Then p is a pre-program of S if and only if p computes $\emptyset \dot{\longmapsto} \emptyset$.

Let us consider N, let S be a realistic halting IC-Ins-separated definite non empty non void AMI over N, and let I_1 be a partial function from FinPartSt(S) to FinPartSt(S). We say that I_1 is computable if and only if:

(Def. 30) There exists a finite partial state p of S such that p computes I_1.

The following propositions are true:

(71) Let S be a realistic halting IC-Ins-separated definite non empty non void AMI over N and F be a partial function from FinPartSt(S) to FinPartSt(S). If $F = \emptyset$, then F is computable.

(72) Let S be a realistic halting IC-Ins-separated definite non empty non void AMI over N and F be a partial function from FinPartSt(S) to FinPartSt(S). If $F = \emptyset \dot{\longmapsto} \emptyset$, then F is computable.

(73) Let S be a realistic halting IC-Ins-separated definite non empty non void AMI over N, p be a pre-program of S, and F be a partial function from FinPartSt(S) to FinPartSt(S). If $F = \emptyset \dot{\longmapsto} \mathrm{Result}(p)$, then F is computable.

Let us consider N, let S be a realistic halting IC-Ins-separated definite non empty non void AMI over N, and let F be a partial function from FinPartSt(S) to FinPartSt(S). Let us assume that F is computable. A finite partial state of S is said to be a program of F if:

(Def. 31) It computes F.

Let N be a set and let S be an AMI over N. An instruction type of S is an element of the instruction codes of S.

Next we state three propositions:

(74) Let S be a realistic halting IC-Ins-separated definite non empty non void AMI over N and F be a partial function from FinPartSt(S) to FinPartSt(S). If $F = \emptyset$, then every finite partial state of S is a program of F.

(75) Let S be a realistic halting IC-Ins-separated definite non empty non void AMI over N and F be a partial function from FinPartSt(S) to FinPartSt(S). If $F = \emptyset \dot{\longmapsto} \emptyset$, then every pre-program of S is a program of F.

(76) Let S be a realistic halting IC-Ins-separated definite non empty non void AMI over N, p be a pre-program of S, and F be a partial function from FinPartSt(S) to FinPartSt(S). If $F = \emptyset \dot{\longmapsto} \mathrm{Result}(p)$, then p is a program of F.

REFERENCES

[1] Grzegorz Bancerek. The fundamental properties of natural numbers. *Journal of Formalized Mathematics*, 1, 1989. `http://mizar.org/JFM/Vol1/nat_1.html`.

[2] Grzegorz Bancerek. König's theorem. *Journal of Formalized Mathematics*, 2, 1990. `http://mizar.org/JFM/Vol2/card_3.html`.

[3] Grzegorz Bancerek and Krzysztof Hryniewiecki. Segments of natural numbers and finite sequences. *Journal of Formalized Mathematics*, 1, 1989. `http://mizar.org/JFM/Vol1/finseq_1.html`.

[4] Czesław Byliński. Basic functions and operations on functions. *Journal of Formalized Mathematics*, 1, 1989. `http://mizar.org/JFM/Vol1/funct_3.html`.

[5] Czesław Byliński. Functions and their basic properties. *Journal of Formalized Mathematics*, 1, 1989. `http://mizar.org/JFM/Vol1/funct_1.html`.

[6] Czesław Byliński. Functions from a set to a set. *Journal of Formalized Mathematics*, 1, 1989. `http://mizar.org/JFM/Vol1/funct_2.html`.

[7] Czesław Byliński. Partial functions. *Journal of Formalized Mathematics*, 1, 1989. `http://mizar.org/JFM/Vol1/partfun1.html`.

[8] Czesław Byliński. Some basic properties of sets. *Journal of Formalized Mathematics*, 1, 1989. `http://mizar.org/JFM/Vol1/zfmisc_1.html`.

[9] Czesław Byliński. A classical first order language. *Journal of Formalized Mathematics*, 2, 1990. `http://mizar.org/JFM/Vol2/cqc_lang.html`.

[10] Czesław Byliński. The modification of a function by a function and the iteration of the composition of a function. *Journal of Formalized Mathematics*, 2, 1990. http://mizar.org/JFM/Vol2/funct_4.html.

[11] Agata Darmochwał. Finite sets. *Journal of Formalized Mathematics*, 1, 1989. http://mizar.org/JFM/Vol1/finset_1.html.

[12] C.C. Elgot and A. Robinson. Random access stored-program machines, an approach to programming languages. *J.A.C.M.*, 11(4):365–399, Oct 1964.

[13] Yatsuka Nakamura. On a mathematical model of CPU and algorithm. Technical report, Shinshu University, Aug 1991.

[14] Andrzej Trybulec. Binary operations applied to functions. *Journal of Formalized Mathematics*, 1, 1989. http://mizar.org/JFM/Vol1/funcop_1.html.

[15] Andrzej Trybulec. Tarski Grothendieck set theory. *Journal of Formalized Mathematics*, Axiomatics, 1989. http://mizar.org/JFM/Axiomatics/tarski.html.

[16] Andrzej Trybulec. Subsets of real numbers. *Journal of Formalized Mathematics*, Addenda, 2003. http://mizar.org/JFM/Addenda/numbers.html.

[17] Zinaida Trybulec. Properties of subsets. *Journal of Formalized Mathematics*, 1, 1989. http://mizar.org/JFM/Vol1/subset_1.html.

[18] Edmund Woronowicz. Relations and their basic properties. *Journal of Formalized Mathematics*, 1, 1989. http://mizar.org/JFM/Vol1/relat_1.html.

Received October 14, 1992

Published January 2, 2004