JOURNAL OF FORMALIZED MATHEMATICS
Volume 5, Released 1993, Published 2003
Inst. of Computer Science, Univ. of Białystok

Some Remarks on the Simple Concrete Model of Computer

Andrzej Trybulec
Warsaw University
Białystok

Yatsuka Nakamura
Shinshu University
Nagano

Summary. We prove some results on **SCM** needed for the proof of the correctness of Euclid's algorithm. We introduce the following concepts:

- starting finite partial state (Start-At(l)), then assigns to the instruction counter an instruction location (and consists only of this assignment),
- programmed finite partial state, that consists of the instructions (to be more precise, a finite partial state with the domain consisting of instruction locations).

We define for a total state s what it means that s starts at l (the value of the instruction counter in the state s is l) and s halts at l (the halt instruction is assigned to l in the state s). Similar notions are defined for finite partial states.

MML Identifier: `AMI_3`.

WWW: `http://mizar.org/JFM/Vol5/ami_3.html`

The articles [15], [14], [19], [3], [2], [17], [6], [7], [18], [1], [16], [8], [4], [13], [20], [9], [10], [5], [11], and [12] provide the notation and terminology for this paper.

1. A SMALL CONCRETE MACHINE

In this paper i, j, k are natural numbers.

The strict AMI **SCM** over $\{\mathbb{Z}\}$ is defined as follows:

(Def. 1) $\mathbf{SCM} = \langle \mathbb{N}, 0, \text{Instr-Loc}_{\mathbf{SCM}}, \mathbb{Z}_9, \text{Instr}_{\mathbf{SCM}}, \text{OK}_{\mathbf{SCM}}, \text{Exec}_{\mathbf{SCM}} \rangle$.

One can verify that **SCM** is non empty and non void.

Next we state two propositions:

(1) **SCM** is data-oriented.

(2) **SCM** is definite.

One can check that **SCM** is IC-Ins-separated, data-oriented, and definite.

An object of **SCM** is called a data-location if:

(Def. 2) It $\in \text{Data-Loc}_{\mathbf{SCM}}$.

Let s be a state of **SCM** and let d be a data-location. Then $s(d)$ is an integer.

We use the following convention: a, b, c denote data-locations, l_1 denotes an instruction-location of **SCM**, and I denotes an instruction of **SCM**.

Let us consider a, b. The functor $a{:=}b$ yielding an instruction of **SCM** is defined as follows:

(Def. 3) $a{:=}b = \langle 1, \langle a, b\rangle\rangle$.

The functor $\mathrm{AddTo}(a,b)$ yields an instruction of **SCM** and is defined by:

(Def. 4) $\mathrm{AddTo}(a,b) = \langle 2, \langle a, b\rangle\rangle$.

The functor $\mathrm{SubFrom}(a,b)$ yielding an instruction of **SCM** is defined as follows:

(Def. 5) $\mathrm{SubFrom}(a,b) = \langle 3, \langle a, b\rangle\rangle$.

The functor $\mathrm{MultBy}(a,b)$ yielding an instruction of **SCM** is defined by:

(Def. 6) $\mathrm{MultBy}(a,b) = \langle 4, \langle a, b\rangle\rangle$.

The functor $\mathrm{Divide}(a,b)$ yielding an instruction of **SCM** is defined as follows:

(Def. 7) $\mathrm{Divide}(a,b) = \langle 5, \langle a, b\rangle\rangle$.

Let us consider l_1. The functor goto l_1 yields an instruction of **SCM** and is defined by:

(Def. 8) goto $l_1 = \langle 6, \langle l_1\rangle\rangle$.

Let us consider a. The functor **if** $a = 0$ **goto** l_1 yielding an instruction of **SCM** is defined as follows:

(Def. 9) **if** $a = 0$ **goto** $l_1 = \langle 7, \langle l_1, a\rangle\rangle$.

The functor **if** $a > 0$ **goto** l_1 yields an instruction of **SCM** and is defined as follows:

(Def. 10) **if** $a > 0$ **goto** $l_1 = \langle 8, \langle l_1, a\rangle\rangle$.

In the sequel s is a state of **SCM**.
The following propositions are true:

(4)[1] $\mathbf{IC}_{\mathbf{SCM}} = 0$.

(5) For every **SCM**-state S such that $S = s$ holds $\mathbf{IC}_s = \mathbf{IC}_S$.

Let l_1 be an instruction-location of **SCM**. The functor $\mathrm{Next}(l_1)$ yielding an instruction-location of **SCM** is defined as follows:

(Def. 11) There exists an element m_1 of $\mathrm{Instr\text{-}Loc}_{\mathrm{SCM}}$ such that $m_1 = l_1$ and $\mathrm{Next}(l_1) = \mathrm{Next}(m_1)$.

The following two propositions are true:

(6) For every instruction-location l_1 of **SCM** and for every element m_1 of $\mathrm{Instr\text{-}Loc}_{\mathrm{SCM}}$ such that $m_1 = l_1$ holds $\mathrm{Next}(m_1) = \mathrm{Next}(l_1)$.

(7) For every element i of $\mathrm{Instr}_{\mathrm{SCM}}$ such that $i = I$ and for every **SCM**-state S such that $S = s$ holds $\mathrm{Exec}(I, s) = \mathrm{Exec\text{-}Res}_{\mathrm{SCM}}(i, S)$.

2. Users guide

We now state several propositions:

(8) $(\mathrm{Exec}(a{:=}b, s))(\mathbf{IC}_{\mathbf{SCM}}) = \mathrm{Next}(\mathbf{IC}_s)$ and $(\mathrm{Exec}(a{:=}b, s))(a) = s(b)$ and for every c such that $c \neq a$ holds $(\mathrm{Exec}(a{:=}b, s))(c) = s(c)$.

(9) $(\mathrm{Exec}(\mathrm{AddTo}(a,b), s))(\mathbf{IC}_{\mathbf{SCM}}) = \mathrm{Next}(\mathbf{IC}_s)$ and $(\mathrm{Exec}(\mathrm{AddTo}(a,b), s))(a) = s(a) + s(b)$ and for every c such that $c \neq a$ holds $(\mathrm{Exec}(\mathrm{AddTo}(a,b), s))(c) = s(c)$.

(10) $(\mathrm{Exec}(\mathrm{SubFrom}(a,b), s))(\mathbf{IC}_{\mathbf{SCM}}) = \mathrm{Next}(\mathbf{IC}_s)$ and $(\mathrm{Exec}(\mathrm{SubFrom}(a,b), s))(a) = s(a) - s(b)$ and for every c such that $c \neq a$ holds $(\mathrm{Exec}(\mathrm{SubFrom}(a,b), s))(c) = s(c)$.

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

(11) $(\mathrm{Exec}(\mathrm{MultBy}(a,b),s))(\mathbf{IC}_{\mathbf{SCM}}) = \mathrm{Next}(\mathbf{IC}_s)$ and $(\mathrm{Exec}(\mathrm{MultBy}(a,b),s))(a) = s(a) \cdot s(b)$ and for every c such that $c \neq a$ holds $(\mathrm{Exec}(\mathrm{MultBy}(a,b),s))(c) = s(c)$.

(12)(i) $(\mathrm{Exec}(\mathrm{Divide}(a,b),s))(\mathbf{IC}_{\mathbf{SCM}}) = \mathrm{Next}(\mathbf{IC}_s)$,

(ii) if $a \neq b$, then $(\mathrm{Exec}(\mathrm{Divide}(a,b),s))(a) = s(a) \div s(b)$,

(iii) $(\mathrm{Exec}(\mathrm{Divide}(a,b),s))(b) = s(a) \bmod s(b)$, and

(iv) for every c such that $c \neq a$ and $c \neq b$ holds $(\mathrm{Exec}(\mathrm{Divide}(a,b),s))(c) = s(c)$.

(13) $(\mathrm{Exec}(\mathrm{goto}\ l_1,s))(\mathbf{IC}_{\mathbf{SCM}}) = l_1$ and $(\mathrm{Exec}(\mathrm{goto}\ l_1,s))(c) = s(c)$.

(14) If $s(a) = 0$, then $(\mathrm{Exec}(\mathbf{if}\ a = 0\ \mathbf{goto}\ l_1,s))(\mathbf{IC}_{\mathbf{SCM}}) = l_1$ and if $s(a) \neq 0$, then $(\mathrm{Exec}(\mathbf{if}\ a = 0\ \mathbf{goto}\ l_1,s))(\mathbf{IC}_{\mathbf{SCM}}) = \mathrm{Next}(\mathbf{IC}_s)$ and $(\mathrm{Exec}(\mathbf{if}\ a = 0\ \mathbf{goto}\ l_1,s))(c) = s(c)$.

(15) If $s(a) > 0$, then $(\mathrm{Exec}(\mathbf{if}\ a > 0\ \mathbf{goto}\ l_1,s))(\mathbf{IC}_{\mathbf{SCM}}) = l_1$ and if $s(a) \leq 0$, then $(\mathrm{Exec}(\mathbf{if}\ a > 0\ \mathbf{goto}\ l_1,s))(\mathbf{IC}_{\mathbf{SCM}}) = \mathrm{Next}(\mathbf{IC}_s)$ and $(\mathrm{Exec}(\mathbf{if}\ a > 0\ \mathbf{goto}\ l_1,s))(c) = s(c)$.

One can verify that **SCM** is halting.

3. Preliminaries

One can prove the following proposition

(18)[2] For all integers m, j holds $m \cdot j \equiv 0 (\mathrm{mod}\, m)$.

The scheme *INDI* deals with natural numbers $\mathcal{A}$, $\mathcal{B}$ and a unary predicate $\mathcal{P}$, and states that:

$\mathcal{P}[\mathcal{B}]$

provided the parameters satisfy the following conditions:

- $\mathcal{P}[0]$,
- $\mathcal{A} > 0$, and
- For all i, j such that $\mathcal{P}[\mathcal{A} \cdot i]$ and $j \neq 0$ and $j \leq \mathcal{A}$ holds $\mathcal{P}[\mathcal{A} \cdot i + j]$.

We now state a number of propositions:

(19) Let X, Y be non empty sets and f, g be partial functions from X to Y. Suppose that for every element x of X and for every element y of Y holds $\langle x, y \rangle \in f$ iff $\langle x, y \rangle \in g$. Then $f = g$.

(20) For all functions f, g and for all sets A, B such that $f \restriction A = g \restriction A$ and $f \restriction B = g \restriction B$ holds $f \restriction (A \cup B) = g \restriction (A \cup B)$.

(21) For every set X and for all functions f, g such that $\mathrm{dom}\, g \subseteq X$ and $g \subseteq f$ holds $g \subseteq f \restriction X$.

(22) For every function f and for every set x such that $x \in \mathrm{dom}\, f$ holds $f \restriction \{x\} = \{\langle x, f(x) \rangle\}$.

(23) For every function f and for every set X such that X misses $\mathrm{dom}\, f$ holds $f \restriction X = \emptyset$.

(24) For all functions f, g and for every set x such that $\mathrm{dom}\, f = \mathrm{dom}\, g$ and $f(x) = g(x)$ holds $f \restriction \{x\} = g \restriction \{x\}$.

(25) For all functions f, g and for all sets x, y such that $\mathrm{dom}\, f = \mathrm{dom}\, g$ and $f(x) = g(x)$ and $f(y) = g(y)$ holds $f \restriction \{x, y\} = g \restriction \{x, y\}$.

(26) Let f, g be functions and x, y, z be sets. If $\mathrm{dom}\, f = \mathrm{dom}\, g$ and $f(x) = g(x)$ and $f(y) = g(y)$ and $f(z) = g(z)$, then $f \restriction \{x, y, z\} = g \restriction \{x, y, z\}$.

(27) For all sets a, b and for every function f such that $a \in \mathrm{dom}\, f$ and $f(a) = b$ holds $a \dot{\longmapsto} b \subseteq f$.

(29)[3] For all sets a, b, c, d and for every function f such that $a \in \mathrm{dom}\, f$ and $c \in \mathrm{dom}\, f$ and $f(a) = b$ and $f(c) = d$ holds $[a \longmapsto b, c \longmapsto d] \subseteq f$.

[2] The propositions (16) and (17) have been removed.

[3] The proposition (28) has been removed.

4. Some Remarks on AMI-Struct

In the sequel N denotes a set.

Next we state the proposition

(31)[4] For every AMI S over N and for every finite partial state p of S holds $p \in \mathrm{FinPartSt}(S)$.

Let N be a set and let S be an AMI over N. Observe that $\mathrm{FinPartSt}(S)$ is non empty.

We now state two propositions:

(32) For every AMI S over N holds every element of $\mathrm{FinPartSt}(S)$ is a finite partial state of S.

(33) Let S be an AMI over N and F_1, F_2 be partial functions from $\mathrm{FinPartSt}(S)$ to $\mathrm{FinPartSt}(S)$. Suppose that for all finite partial states p, q of S holds $\langle p, q\rangle \in F_1$ iff $\langle p, q\rangle \in F_2$. Then $F_1 = F_2$.

The scheme *EqFPSFunc* deals with a non empty set $\mathcal{A}$ with non empty elements, an AMI $\mathcal{B}$ over $\mathcal{A}$, partial functions $\mathcal{C}$, $\mathcal{D}$ from $\mathrm{FinPartSt}(\mathcal{B})$ to $\mathrm{FinPartSt}(\mathcal{B})$, and a binary predicate $\mathcal{P}$, and states that:

$\mathcal{C} = \mathcal{D}$

provided the parameters meet the following conditions:

- For all finite partial states p, q of $\mathcal{B}$ holds $\langle p, q\rangle \in \mathcal{C}$ iff $\mathcal{P}[p,q]$, and
- For all finite partial states p, q of $\mathcal{B}$ holds $\langle p, q\rangle \in \mathcal{D}$ iff $\mathcal{P}[p,q]$.

Let N be a set with non empty elements, let S be an IC-Ins-separated definite non empty non void AMI over N, and let l be an instruction-location of S. The functor Start-At(l) yielding a finite partial state of S is defined by:

(Def. 12) $\text{Start-At}(l) = \mathbf{IC}_S \mapsto l$.

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

One can prove the following proposition

(34) Let S be an IC-Ins-separated definite non empty non void AMI over N and l be an instruction-location of S. Then $\mathrm{dom}\,\text{Start-At}(l) = \{\mathbf{IC}_S\}$.

Let N be a set, let S be an AMI over N, and let I_1 be a finite partial state of S. We say that I_1 is programmed if and only if:

(Def. 13) $\mathrm{dom}\, I_1 \subseteq$ the instruction locations of S.

Let N be a set and let S be an AMI over N. Note that there exists a finite partial state of S which is programmed.

We now state four propositions:

(35) Let N be a set, S be an AMI over N, and p_1, p_2 be programmed finite partial states of S. Then $p_1 {+\cdot} p_2$ is programmed.

(36) For every non void AMI S over N and for every state s of S holds $\mathrm{dom}\, s =$ the carrier of S.

(37) For every AMI S over N and for every finite partial state p of S holds $\mathrm{dom}\, p \subseteq$ the carrier of S.

(38) Let S be a steady-programmed IC-Ins-separated definite non empty non void AMI over N, p be a programmed finite partial state of S, and s be a state of S. If $p \subseteq s$, then for every k holds $p \subseteq (\mathrm{Computation}(s))(k)$.

Let us consider N, let S be an IC-Ins-separated non empty non void AMI over N, let s be a state of S, and let l be an instruction-location of S. We say that s starts at l if and only if:

[4] The proposition (30) has been removed.

(Def. 14) $\mathbf{IC}_s = l$.

Let us consider N, let S be an IC-Ins-separated halting non empty non void AMI over N, let s be a state of S, and let l be an instruction-location of S. We say that s halts at l if and only if:

(Def. 15) $s(l) = \mathbf{halt}_S$.

We now state the proposition

(39) For every non void AMI S over N and for every finite partial state p of S there exists a state s of S such that $p \subseteq s$.

Let us consider N, let S be a definite IC-Ins-separated non empty non void AMI over N, and let p be a finite partial state of S. Let us assume that $\mathbf{IC}_S \in \operatorname{dom} p$. The functor $\mathbf{IC}_p$ yields an instruction-location of S and is defined as follows:

(Def. 16) $\mathbf{IC}_p = p(\mathbf{IC}_S)$.

Let us consider N, let S be a definite IC-Ins-separated non empty non void AMI over N, let p be a finite partial state of S, and let l be an instruction-location of S. We say that p starts at l if and only if:

(Def. 17) $\mathbf{IC}_S \in \operatorname{dom} p$ and $\mathbf{IC}_p = l$.

Let us consider N, let S be a definite IC-Ins-separated halting non empty non void AMI over N, let p be a finite partial state of S, and let l be an instruction-location of S. We say that p halts at l if and only if:

(Def. 18) $l \in \operatorname{dom} p$ and $p(l) = \mathbf{halt}_S$.

We now state a number of propositions:

(40) Let S be an IC-Ins-separated definite steady-programmed halting non empty non void AMI over N and s be a state of S. Then s is halting if and only if there exists k such that s halts at $\mathbf{IC}_{(\mathrm{Computation}(s))(k)}$.

(41) Let S be an IC-Ins-separated definite steady-programmed halting non empty non void AMI over N, s be a state of S, p be a finite partial state of S, and l be an instruction-location of S. If $p \subseteq s$ and p halts at l, then s halts at l.

(42) Let S be a halting steady-programmed IC-Ins-separated definite non empty non void AMI over N, s be a state of S, and given k. If s is halting, then $\mathrm{Result}(s) = (\mathrm{Computation}(s))(k)$ iff s halts at $\mathbf{IC}_{(\mathrm{Computation}(s))(k)}$.

(43) Let S be a steady-programmed IC-Ins-separated definite non empty non void AMI over N, s be a state of S, p be a programmed finite partial state of S, and given k. Then $p \subseteq s$ if and only if $p \subseteq (\mathrm{Computation}(s))(k)$.

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

(45) Suppose $i \leq j$. Let S be a halting steady-programmed IC-Ins-separated definite non empty non void AMI over N and s be a state of S. If s halts at $\mathbf{IC}_{(\mathrm{Computation}(s))(i)}$, then s halts at $\mathbf{IC}_{(\mathrm{Computation}(s))(j)}$.

(46) Suppose $i \leq j$. Let S be a halting steady-programmed IC-Ins-separated definite non empty non void AMI over N and s be a state of S. If s halts at $\mathbf{IC}_{(\mathrm{Computation}(s))(i)}$, then $(\mathrm{Computation}(s))(j) = (\mathrm{Computation}(s))(i)$.

(47) 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 halts at $\mathbf{IC}_{(\mathrm{Computation}(s))(k)}$, then for every i holds $\mathrm{Result}(s) = \mathrm{Result}((\mathrm{Computation}(s))(i))$.

(48) Let S be a steady-programmed IC-Ins-separated definite halting non empty non void AMI over N, s be a state of S, l be an instruction-location of S, and given k. Then s halts at l if and only if $(\mathrm{Computation}(s))(k)$ halts at l.

(49) Let S be a definite IC-Ins-separated non empty non void AMI over N, p be a finite partial state of S, and l be an instruction-location of S. Suppose p starts at l. Let s be a state of S. If $p \subseteq s$, then s starts at l.

(50) Let S be an IC-Ins-separated definite non empty non void AMI over N and l be an instruction-location of S. Then $\text{Start-At}(l)(\mathbf{IC}_S) = l$.

Let us consider N, let S be a definite IC-Ins-separated non empty non void AMI over N, let l be an instruction-location of S, and let I be an element of the instructions of S. Then $l \dot{\longmapsto} I$ is a programmed finite partial state of S.

5. Instruction Locations and Data Locations

We now state the proposition

(51) **SCM** is realistic.

Let us observe that **SCM** is steady-programmed and realistic.

Let k be a natural number. The functor $\mathbf{d}_k$ yields a data-location and is defined by:

(Def. 19) $\mathbf{d}_k = 2 \cdot k + 1$.

The functor $\mathbf{i}_k$ yields an instruction-location of **SCM** and is defined by:

(Def. 20) $\mathbf{i}_k = 2 \cdot k + 2$.

In the sequel i, j, k are natural numbers.

We now state four propositions:

(52) If $i \neq j$, then $\mathbf{d}_i \neq \mathbf{d}_j$.

(53) If $i \neq j$, then $\mathbf{i}_i \neq \mathbf{i}_j$.

(54) $\mathrm{Next}(\mathbf{i}_k) = \mathbf{i}_{k+1}$.

(55) For every data-location l holds $\mathrm{ObjectKind}(l) = \mathbb{Z}$.

Let l_2 be a data-location and let a be an integer. Then $l_2 \dot{\longmapsto} a$ is a finite partial state of **SCM**.

Let l_2, l_3 be data-locations and let a, b be integers. Then $[l_2 \longmapsto a, l_3 \longmapsto b]$ is a finite partial state of **SCM**.

One can prove the following propositions:

(56) $\mathbf{d}_i \neq \mathbf{i}_j$.

(57) $\mathbf{IC}_{\mathbf{SCM}} \neq \mathbf{d}_i$ and $\mathbf{IC}_{\mathbf{SCM}} \neq \mathbf{i}_i$.

6. Halt Instruction

One can prove the following propositions:

(58) For every instruction I of **SCM** such that there exists s such that $(\mathrm{Exec}(I,s))(\mathbf{IC}_{\mathbf{SCM}}) = \mathrm{Next}(\mathbf{IC}_s)$ holds I is non halting.

(59) For every instruction I of **SCM** such that $I = \langle 0, \emptyset \rangle$ holds I is halting.

(60) $a{:=}b$ is non halting.

(61) $\mathrm{AddTo}(a,b)$ is non halting.

(62) SubFrom(a,b) is non halting.

(63) MultBy(a,b) is non halting.

(64) Divide(a,b) is non halting.

(65) goto l_1 is non halting.

(66) **if** $a = 0$ **goto** l_1 is non halting.

(67) **if** $a > 0$ **goto** l_1 is non halting.

(68) $\langle 0, \emptyset \rangle$ is an instruction of **SCM**.

(69) Let I be a set. Then I is an instruction of **SCM** if and only if one of the following conditions is satisfied:

$I = \langle 0, \emptyset \rangle$ or there exist a, b such that $I = a{:=}b$ or there exist a, b such that $I =$ AddTo(a,b) or there exist a, b such that $I =$ SubFrom(a,b) or there exist a, b such that $I =$ MultBy(a,b) or there exist a, b such that $I =$ Divide(a,b) or there exists l_1 such that $I =$ goto l_1 or there exist a, l_1 such that $I =$ **if** $a = 0$ **goto** l_1 or there exist a, l_1 such that $I =$ **if** $a > 0$ **goto** l_1.

(70) For every instruction I of **SCM** such that I is halting holds $I = \mathbf{halt_{SCM}}$.

(71) $\mathbf{halt_{SCM}} = \langle 0, \emptyset \rangle$.

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. The ordinal numbers. *Journal of Formalized Mathematics*, 1, 1989. http://mizar.org/JFM/Vol1/ordinal1.html.

[3] Grzegorz Bancerek. Sequences of ordinal numbers. *Journal of Formalized Mathematics*, 1, 1989. http://mizar.org/JFM/Vol1/ordinal2.html.

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

[5] 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.

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

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

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

[9] 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.

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

[11] Yatsuka Nakamura and Andrzej Trybulec. A mathematical model of CPU. *Journal of Formalized Mathematics*, 4, 1992. http://mizar.org/JFM/Vol4/ami_1.html.

[12] Yatsuka Nakamura and Andrzej Trybulec. On a mathematical model of programs. *Journal of Formalized Mathematics*, 4, 1992. http://mizar.org/JFM/Vol4/ami_2.html.

[13] Dariusz Surowik. Cyclic groups and some of their properties — part I. *Journal of Formalized Mathematics*, 3, 1991. http://mizar.org/JFM/Vol3/gr_cy_1.html.

[14] Andrzej Trybulec. Enumerated sets. *Journal of Formalized Mathematics*, 1, 1989. http://mizar.org/JFM/Vol1/enumset1.html.

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

[16] Andrzej Trybulec. Tuples, projections and Cartesian products. *Journal of Formalized Mathematics*, 1, 1989. http://mizar.org/JFM/Vol1/mcart_1.html.

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

[18] Michał J. Trybulec. Integers. *Journal of Formalized Mathematics*, 2, 1990. `http://mizar.org/JFM/Vol2/int_1.html`.

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

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

Received October 8, 1993

Published January 2, 2004