FORMALIZED MATHEMATICS
Volume 7, Number 1, 1998
University of Białystok

# On the Composition of Non-parahalting Macro Instructions

Piotr Rudnicki[1]
University of Alberta
Edmonton

**Summary.** An attempt to use the `Times` macro, [2], was the origin of writing this article. First, the semantics of the macro composition as developed in [23, 3, 4] is extended to the case of macro instructions which are not always halting. Next, several functors extending the memory handling for $\mathbf{SCM}_{\rm FSA}$, [18], are defined; they are convenient when writing more complicated programs. After this preparatory work, we define a macro instruction computing the Fibonacci sequence (see the SCM program computing the same sequence in [10]) and prove its correctness. The semantics of the `Times` macro is given in [2] only for the case when the iterated instruction is parahalting; this is remedied in [17].

MML Identifier: `SFMASTR1`.

The notation and terminology used in this paper are introduced in the following papers: [16], [21], [19], [27], [5], [7], [15], [12], [14], [13], [11], [25], [6], [9], [28], [23], [3], [4], [1], [24], [22], [8], [18], [26], and [20].

## 1. Good Instructions and Good Macro Instruction

Let $i$ be an instruction of $\mathbf{SCM}_{\rm FSA}$. We say that $i$ is good if and only if:

(Def. 1) Macro($i$) is good.

Let $a$ be a read-write integer location and let $b$ be an integer location. One can check the following observations:

* $a$:=$b$ is good,

[1]This work was partially supported by NSERC Grant OGP9207 and NATO CRG 951368.




ISSN 1426–2630

* $\mathrm{AddTo}(a, b)$ is good,
* $\mathrm{SubFrom}(a, b)$ is good, and
* $\mathrm{MultBy}(a, b)$ is good.

Let us note that there exists an instruction of $\mathbf{SCM}_{\mathrm{FSA}}$ which is good and parahalting.

Let $a$, $b$ be read-write integer locations. Observe that $\mathrm{Divide}(a, b)$ is good.

Let $l$ be an instruction-location of $\mathbf{SCM}_{\mathrm{FSA}}$. One can verify that goto $l$ is good.

Let $a$ be an integer location and let $l$ be an instruction-location of $\mathbf{SCM}_{\mathrm{FSA}}$. Note that **if** $a = 0$ **goto** $l$ is good and **if** $a > 0$ **goto** $l$ is good.

Let $a$ be an integer location, let $f$ be a finite sequence location, and let $b$ be a read-write integer location. One can check that $b{:=}f_a$ is good.

Let $f$ be a finite sequence location and let $b$ be a read-write integer location. One can verify that $b{:=}\mathrm{len} f$ is good.

Let $f$ be a finite sequence location and let $a$ be an integer location. One can check that $f{:=}\langle\underbrace{0, \ldots, 0}_{a}\rangle$ is good. Let $b$ be an integer location. Note that $f_a{:=}b$ is good.

Let us note that there exists an instruction of $\mathbf{SCM}_{\mathrm{FSA}}$ which is good.

Let $i$ be a good instruction of $\mathbf{SCM}_{\mathrm{FSA}}$. Note that $\mathrm{Macro}(i)$ is good.

Let $i$, $j$ be good instructions of $\mathbf{SCM}_{\mathrm{FSA}}$. Note that $i;j$ is good.

Let $i$ be a good instruction of $\mathbf{SCM}_{\mathrm{FSA}}$ and let $I$ be a good macro instruction. Note that $i;I$ is good and $I;i$ is good.

Let $a$, $b$ be read-write integer locations. Note that $\mathrm{swap}(a, b)$ is good.

Let $I$ be a good macro instruction and let $a$ be a read-write integer location. One can verify that $\mathrm{Times}(a, I)$ is good.

One can prove the following proposition

(1) For every integer location $a$ and for every macro instruction $I$ such that $a \notin \mathrm{UsedIntLoc}(I)$ holds $I$ does not destroy $a$.

## 2. Composition of Non-parahalting Macro Instructions

For simplicity, we use the following convention: $s$, $S$ denote states of $\mathbf{SCM}_{\mathrm{FSA}}$, $I$, $J$ denote macro instructions, $I_1$ denotes a good macro instruction, $i$ denotes a good parahalting instruction of $\mathbf{SCM}_{\mathrm{FSA}}$, $j$ denotes a parahalting instruction of $\mathbf{SCM}_{\mathrm{FSA}}$, $a$, $b$ denote integer locations, and $f$ denotes a finite sequence location.

We now state a number of propositions:

(2) $(I {+\cdot}\, \mathrm{Start\text{-}At}(\mathrm{insloc}(0))) \restriction D = \emptyset$, where $D = \mathrm{Int\text{-}Locations} \cup \mathrm{FinSeq\text{-}Locations}$.

(3) If $I$ is halting on Initialize($S$) and closed on Initialize($S$) and $J$ is closed on IExec($I, S$), then $I$;$J$ is closed on Initialize($S$).

(4) If $I$ is halting on Initialize($S$) and $J$ is halting on IExec($I, S$) and $I$ is closed on Initialize($S$) and $J$ is closed on IExec($I, S$), then $I$;$J$ is halting on Initialize($S$).

(5) Suppose $I$ is closed on $s$ and $I$+· Start-At(insloc(0)) $\subseteq s$ and $s$ is halting. Let $m$ be a natural number. Suppose $m \leqslant$ LifeSpan($s$). Then (Computation($s$))($m$) and (Computation($s$+·($I$;$J$)))($m$) are equal outside the instruction locations of $\mathbf{SCM}_{\rm FSA}$.

(6) Suppose $I_1$ is halting on Initialize($s$) and $J$ is halting on IExec($I_1, s$) and $I_1$ is closed on Initialize($s$) and $J$ is closed on IExec($I_1, s$). Then LifeSpan($s$+· Initialized($I_1$;$J$)) = LifeSpan($s$+· Initialized($I_1$)) + 1 + LifeSpan(Result($s$+· Initialized($I_1$))+· Initialized($J$)).

(7) Suppose $I_1$ is halting on Initialize($s$) and $J$ is halting on IExec($I_1, s$) and $I_1$ is closed on Initialize($s$) and $J$ is closed on IExec($I_1, s$). Then IExec($I_1$;$J, s$) = IExec($J$, IExec($I_1, s$))+· Start-At($\mathbf{IC}_{{\rm IExec}(J,{\rm IExec}(I_1,s))}$ + card $I_1$).

(8) Suppose that

(i) $I_1$ is parahalting, or halting on Initialize($s$), or closed on Initialize($s$), and

(ii) $J$ is parahalting, or halting on IExec($I_1, s$), or closed on IExec($I_1, s$).

Then (IExec($I_1$;$J, s$))($a$) = (IExec($J$, IExec($I_1, s$)))($a$).

(9) Suppose that

(i) $I_1$ is parahalting, or halting on Initialize($s$), or closed on Initialize($s$), and

(ii) $J$ is parahalting, or halting on IExec($I_1, s$), or closed on IExec($I_1, s$).

Then (IExec($I_1$;$J, s$))($f$) = (IExec($J$, IExec($I_1, s$)))($f$).

(10) Suppose that

(i) $I_1$ is parahalting, or halting on Initialize($s$), or closed on Initialize($s$), and

(ii) $J$ is parahalting, or halting on IExec($I_1, s$), or closed on IExec($I_1, s$).

Then IExec($I_1$;$J, s$)$\restriction D$ = IExec($J$, IExec($I_1, s$))$\restriction D$, where $D$ = Int-Locations $\cup$ FinSeq-Locations.

(11) If $I_1$ is parahalting, or closed on Initialize($s$), or halting on Initialize($s$), then Initialize(IExec($I_1, s$))$\restriction D$ = IExec($I_1, s$)$\restriction D$, where $D$ = Int-Locations $\cup$ FinSeq-Locations.

(12) If $I_1$ is parahalting, or halting on Initialize($s$), or closed on Initialize($s$), then (IExec($I_1$;$j, s$))($a$) = (Exec($j$, IExec($I_1, s$)))($a$).

(13) If $I_1$ is parahalting, or halting on Initialize($s$), or closed on Initialize($s$), then (IExec($I_1$;$j, s$))($f$) = (Exec($j$, IExec($I_1, s$)))($f$).

(14) If $I_1$ is parahalting, or halting on Initialize$(s)$, or closed on Initialize$(s)$, then IExec$(I_1;j,s)\upharpoonright D=$ Exec$(j,$IExec$(I_1,s))\upharpoonright D$, where $D=$ Int-Locations $\cup$ FinSeq-Locations.

(15) If $J$ is parahalting, or halting on Exec$(i,$Initialize$(s))$, or closed on Exec$(i,$Initialize$(s))$, then (IExec$(i;J,s))(a)=$ (IExec$(J,$Exec$(i,$Initialize$(s))))(a)$.

(16) If $J$ is parahalting, or halting on Exec$(i,$Initialize$(s))$, or closed on Exec$(i,$Initialize$(s))$, then (IExec$(i;J,s))(f)=$ (IExec$(J,$Exec$(i,$Initialize$(s))))(f)$.

(17) If $J$ is parahalting, or halting on Exec$(i,$Initialize$(s))$, or closed on Exec$(i,$Initialize$(s))$, then IExec$(i;J,s)\upharpoonright D=$ IExec$(J,$Exec$(i,$Initialize$(s)))$ $\upharpoonright D$, where $D=$ Int-Locations $\cup$ FinSeq-Locations.

## 3. Memory Allocation

In the sequel $L$ is a finite subset of Int-Locations and $m$, $n$ are natural numbers.

Let $d$ be an integer location. Then $\{d\}$ is a subset of Int-Locations. Let $e$ be an integer location. Then $\{d,e\}$ is a subset of Int-Locations. Let $f$ be an integer location. Then $\{d,e,f\}$ is a subset of Int-Locations. Let $g$ be an integer location. Then $\{d,e,f,g\}$ is a subset of Int-Locations.

Let $L$ be a finite subset of Int-Locations. The functor RWNotIn-seq $L$ yields a function from $\mathbb{N}$ into $2^{\mathbb{N}}$ and is defined by the conditions (Def. 2).

(Def. 2)(i) (RWNotIn-seq $L)(0)=\{k;k$ ranges over natural numbers: intloc$(k)\notin L\ \wedge\ k\neq 0\}$,

(ii) for every natural number $i$ and for every non empty subset $s_1$ of $\mathbb{N}$ such that (RWNotIn-seq $L)(i)=s_1$ holds (RWNotIn-seq $L)(i+1)=s_1\setminus\{\min s_1\}$, and

(iii) for every natural number $i$ holds (RWNotIn-seq $L)(i)$ is infinite.

Let $L$ be a finite subset of Int-Locations and let $n$ be a natural number. Note that (RWNotIn-seq $L)(n)$ is non empty.

One can prove the following propositions:

(18) $0\notin$ (RWNotIn-seq $L)(n)$ and for every $m$ such that $m\in$ (RWNotIn-seq $L)(n)$ holds intloc$(m)\notin L$.

(19) min(RWNotIn-seq $L)(n)<$ min(RWNotIn-seq $L)(n+1)$.

(20) If $n<m$, then min(RWNotIn-seq $L)(n)<$ min(RWNotIn-seq $L)(m)$.

Let $n$ be a natural number and let $L$ be a finite subset of Int-Locations. The functor $n^{\rm th}$-RWNotIn$(L)$ yields an integer location and is defined as follows:

(Def. 3) $n^{\rm th}$-RWNotIn$(L)=$ intloc(min(RWNotIn-seq $L)(n))$.

We introduce $1^{\mathrm{st}}$-RWNotIn($L$), $2^{\mathrm{nd}}$-RWNotIn($L$), $3^{\mathrm{rd}}$-RWNotIn($L$) as synonyms of $n^{\mathrm{th}}$-RWNotIn($L$).

Let $n$ be a natural number and let $L$ be a finite subset of Int-Locations. One can verify that $n^{\mathrm{th}}$-RWNotIn($L$) is read-write.

We now state two propositions:

(21) $n^{\mathrm{th}}$-RWNotIn$(L) \notin L$.

(22) If $n \neq m$, then $n^{\mathrm{th}}$-RWNotIn$(L) \neq m^{\mathrm{th}}$-RWNotIn$(L)$.

Let $n$ be a natural number and let $p$ be a programmed finite partial state of $\mathbf{SCM}_{\mathrm{FSA}}$. The functor $n^{\mathrm{th}}$-NotUsed($p$) yielding an integer location is defined by:

(Def. 4) $n^{\mathrm{th}}$-NotUsed$(p) = n^{\mathrm{th}}$-RWNotIn(UsedIntLoc($p$)).

We introduce $1^{\mathrm{st}}$-NotUsed($p$), $2^{\mathrm{nd}}$-NotUsed($p$), $3^{\mathrm{rd}}$-NotUsed($p$) as synonyms of $n^{\mathrm{th}}$-NotUsed($p$).

Let $n$ be a natural number and let $p$ be a programmed finite partial state of $\mathbf{SCM}_{\mathrm{FSA}}$. Observe that $n^{\mathrm{th}}$-NotUsed($p$) is read-write.

## 4. A Macro for the Fibonacci Sequence

One can prove the following proposition

(23) $a \in$ UsedIntLoc(swap($a, b$)) and $b \in$ UsedIntLoc(swap($a, b$)).

Let $N$, $r_1$ be integer locations. The functor Fib_macro($N, r_1$) yielding a macro instruction is defined by:

(Def. 5) Fib_macro$(N, r_1) =$

$(N_1{:=}N)$;
SubFrom$(r_1, r_1)$;
$(n_1{:=}\,$intloc$(0))$;
$(a_1{:=}N_1)$;
Times$(a_1,$ AddTo$(r_1, n_1);$ swap$(r_1, n_1))$;
$(N{:=}N_1)$,
where $N_1 = 2^{\mathrm{nd}}$-RWNotIn(UsedIntLoc(swap$(r_1, n_1)$)), $n_1 = 1^{\mathrm{st}}$-RWNotIn ($\{N, r_1\}$), and $a_1 = 1^{\mathrm{st}}$-RWNotIn(UsedIntLoc(swap$(r_1, n_1)$)).

Next we state the proposition

(24) Let $N$, $r_1$ be read-write integer locations. Suppose $N \neq r_1$. Let $n$ be a natural number. If $n = s(N)$, then (IExec(Fib_macro$(N, r_1), s$))$(r_1) =$ Fib$(n)$ and (IExec(Fib_macro$(N, r_1), s$))$(N) = s(N)$.

## References


[1] Noriko Asamoto. Constant assignment macro instructions of $\mathbf{SCM}_{\mathrm{FSA}}$. Part II. *Formalized Mathematics*, 6(**1**):59–63, 1997.

[2] Noriko Asamoto. The `loop` and `Times` macroinstruction for $\mathbf{SCM}_{\mathrm{FSA}}$. *Formalized Mathematics*, 6(**4**):483–497, 1997.

[3] Noriko Asamoto, Yatsuka Nakamura, Piotr Rudnicki, and Andrzej Trybulec. On the composition of macro instructions. Part II. *Formalized Mathematics*, 6(**1**):41–47, 1997.

[4] Noriko Asamoto, Yatsuka Nakamura, Piotr Rudnicki, and Andrzej Trybulec. On the composition of macro instructions. Part III. *Formalized Mathematics*, 6(**1**):53–57, 1997.

[5] Grzegorz Bancerek. Cardinal numbers. *Formalized Mathematics*, 1(**2**):377–382, 1990.

[6] Grzegorz Bancerek. The fundamental properties of natural numbers. *Formalized Mathematics*, 1(**1**):41–46, 1990.

[7] Grzegorz Bancerek. König's theorem. *Formalized Mathematics*, 1(**3**):589–593, 1990.

[8] Grzegorz Bancerek and Piotr Rudnicki. Development of terminology for **scm**. *Formalized Mathematics*, 4(**1**):61–67, 1993.

[9] Grzegorz Bancerek and Piotr Rudnicki. Two programs for **scm**. Part I - preliminaries. *Formalized Mathematics*, 4(**1**):69–72, 1993.

[10] Grzegorz Bancerek and Piotr Rudnicki. Two programs for **scm**. Part II - programs. *Formalized Mathematics*, 4(**1**):73–75, 1993.

[11] Grzegorz Bancerek and Andrzej Trybulec. Miscellaneous facts about functions. *Formalized Mathematics*, 5(**4**):485–492, 1996.

[12] Czesław Byliński. Finite sequences and tuples of elements of a non-empty sets. *Formalized Mathematics*, 1(**3**):529–536, 1990.

[13] Czesław Byliński. Functions from a set to a set. *Formalized Mathematics*, 1(**1**):153–164, 1990.

[14] Agata Darmochwał. Finite sets. *Formalized Mathematics*, 1(**1**):165–167, 1990.

[15] Agata Darmochwał and Andrzej Trybulec. Similarity of formulae. *Formalized Mathematics*, 2(**5**):635–642, 1991.

[16] Yatsuka Nakamura and Andrzej Trybulec. A mathematical model of CPU. *Formalized Mathematics*, 3(**2**):151–160, 1992.

[17] Piotr Rudnicki. Another `times` macro instruction. *Formalized Mathematics*, 7(**1**):101–105, 1998.

[18] Piotr Rudnicki and Andrzej Trybulec. Memory handling for $\mathbf{SCM}_{\mathrm{FSA}}$. *Formalized Mathematics*, 6(**1**):29–36, 1997.

[19] Yasushi Tanaka. On the decomposition of the states of SCM. *Formalized Mathematics*, 5(**1**):1–8, 1996.

[20] Andrzej Trybulec. Tarski Grothendieck set theory. *Formalized Mathematics*, 1(**1**):9–11, 1990.

[21] Andrzej Trybulec and Yatsuka Nakamura. Some remarks on the simple concrete model of computer. *Formalized Mathematics*, 4(**1**):51–56, 1993.

[22] Andrzej Trybulec and Yatsuka Nakamura. Modifying addresses of instructions of $\mathbf{SCM}_{\mathrm{FSA}}$. *Formalized Mathematics*, 5(**4**):571–576, 1996.

[23] Andrzej Trybulec, Yatsuka Nakamura, and Noriko Asamoto. On the compositions of macro instructions. Part I. *Formalized Mathematics*, 6(**1**):21–27, 1997.

[24] Andrzej Trybulec, Yatsuka Nakamura, and Piotr Rudnicki. The $\mathbf{SCM}_{\mathrm{FSA}}$ computer. *Formalized Mathematics*, 5(**4**):519–528, 1996.

[25] Michał J. Trybulec. Integers. *Formalized Mathematics*, 1(**3**):501–505, 1990.

[26] Zinaida Trybulec. Properties of subsets. *Formalized Mathematics*, 1(**1**):67–71, 1990.

[27] Zinaida Trybulec and Halina Święczkowska. Boolean properties of sets. *Formalized Mathematics*, 1(**1**):17–23, 1990.

[28] Edmund Woronowicz. Relations and their basic properties. *Formalized Mathematics*, 1(**1**):73–83, 1990.


*Received June 3, 1998*