JOURNAL OF FORMALIZED MATHEMATICS
Volume 10, Released 1998, Published 2003
Inst. of Computer Science, Univ. of Białystok

On the Composition of Non-parahalting Macro Instructions

Piotr Rudnicki
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 [26], [3], [4] is extended to the case of macro instructions which are not always halting. Next, several functors extending the memory handling for $\mathbf{SCM}_{\rm FSA}$, [19], 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 [9]) 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 [18].

MML Identifier: `SFMASTR1`.

WWW: `http://mizar.org/JFM/Vol10/sfmastr1.html`

The articles [22], [21], [13], [28], [23], [15], [5], [6], [29], [11], [12], [14], [10], [16], [8], [17], [24], [7], [20], [27], [25], [26], [3], [19], [4], [1], and [2] provide the notation and terminology for this paper.

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 verify the following observations:

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

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

Let a, b be read-write integer locations. Observe that Divide(a,b) is good.

Let l be an instruction-location of $\mathbf{SCM}_{\rm FSA}$. Observe that goto l is good.

Let a be an integer location and let l be an instruction-location of $\mathbf{SCM}_{\rm FSA}$. One can verify 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. Note that $b{:=}f_a$ is good.

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

Let f be a finite sequence location and let a be an integer location. Note 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 observe that there exists an instruction of $\mathbf{SCM}_{\mathrm{FSA}}$ which is good.

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

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

Let i be a good instruction of $\mathbf{SCM}_{\mathrm{FSA}}$ and let I be a good macro instruction. One can check 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. Note that $\mathrm{Times}(a,I)$ is good.

We now state the 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 adopt the following rules: s, S are states of $\mathbf{SCM}_{\mathrm{FSA}}$, I, J are macro instructions, I_1 is a good macro instruction, i is a good parahalting instruction of $\mathbf{SCM}_{\mathrm{FSA}}$, j is a parahalting instruction of $\mathbf{SCM}_{\mathrm{FSA}}$, a, b are integer locations, and f is a finite sequence location.

One can prove the following 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 $\mathrm{Initialize}(S)$ and closed on $\mathrm{Initialize}(S)$ and J is closed on $\mathrm{IExec}(I,S)$, then I; J is closed on $\mathrm{Initialize}(S)$.

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

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

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

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

(8) Suppose that

(i) I_1 is parahalting, halting on $\mathrm{Initialize}(s)$, and closed on $\mathrm{Initialize}(s)$, and

(ii) J is parahalting, halting on $\mathrm{IExec}(I_1,s)$, and closed on $\mathrm{IExec}(I_1,s)$.

Then $(\mathrm{IExec}(I_1;J,s))(a) = (\mathrm{IExec}(J,\mathrm{IExec}(I_1,s)))(a)$.

(9) Suppose that

(i) I_1 is parahalting, halting on $\mathrm{Initialize}(s)$, and closed on $\mathrm{Initialize}(s)$, and

(ii) J is parahalting, halting on $\mathrm{IExec}(I_1,s)$, and closed on $\mathrm{IExec}(I_1,s)$.

Then $(\mathrm{IExec}(I_1;J,s))(f) = (\mathrm{IExec}(J,\mathrm{IExec}(I_1,s)))(f)$.

(10) Suppose that

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

(ii) J is parahalting, halting on IExec(I_1,s), and 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, closed on Initialize(s), and 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, halting on Initialize(s), and closed on Initialize(s), then (IExec$(I_1;\, j,s))(a) =$ (Exec$(j,$IExec$(I_1,s)))(a)$.

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

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

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

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

(17) If J is parahalting, halting on Exec$(i,$Initialize$(s))$, and closed on Exec$(i,$Initialize$(s))$, then IExec$(i;\, J,s)\restriction D =$ IExec$(J,$Exec$(i,$Initialize$(s)))\restriction D$, 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.

Next we state three 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 by:

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

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

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

Next we state two propositions:

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

(22) If $n \neq m$, then n^{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^{th}-NotUsed(p) yields an integer location and is defined by:

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

We introduce 1^{st}-NotUsed(p), 2^{nd}-NotUsed(p), 3^{rd}-NotUsed(p) as synonyms of n^{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^{th}-NotUsed(p) is read-write.

4. A Macro for the Fibonacci Sequence

We now state the 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 as follows:

(Def. 5) Fib_macro$(N,r_1) = (N_1{:=}N)$; SubFrom(r_1,r_1); $(n_1{:=}\mathrm{intloc}(0))$; $(a_1{:=}N_1)$; Times$(a_1,$AddTo(r_1,n_1); swap$(r_1,n_1))$; $(N$
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))).

The following proposition is true

(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. *Journal of Formalized Mathematics*, 8, 1996. `http://mizar.org/JFM/Vol8/scmfsa7b.html`.

[2] Noriko Asamoto. The `loop` and `times` macroinstruction for $\mathbf{SCM}_{\mathrm{FSA}}$. *Journal of Formalized Mathematics*, 9, 1997. `http://mizar.org/JFM/Vol9/scmfsa8c.html`.

[3] Noriko Asamoto, Yatsuka Nakamura, Piotr Rudnicki, and Andrzej Trybulec. On the composition of macro instructions. Part II. *Journal of Formalized Mathematics*, 8, 1996. `http://mizar.org/JFM/Vol8/scmfsa6b.html`.

[4] Noriko Asamoto, Yatsuka Nakamura, Piotr Rudnicki, and Andrzej Trybulec. On the composition of macro instructions. Part III. *Journal of Formalized Mathematics*, 8, 1996. `http://mizar.org/JFM/Vol8/scmfsa6c.html`.

[5] Grzegorz Bancerek. Cardinal numbers. *Journal of Formalized Mathematics*, 1, 1989. `http://mizar.org/JFM/Vol1/card_1.html`.

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

[7] Grzegorz Bancerek and Piotr Rudnicki. Development of terminology for **scm**. *Journal of Formalized Mathematics*, 5, 1993. `http://mizar.org/JFM/Vol5/scm_1.html`.

[8] Grzegorz Bancerek and Piotr Rudnicki. Two programs for **scm**. Part I - preliminaries. *Journal of Formalized Mathematics*, 5, 1993. `http://mizar.org/JFM/Vol5/pre_ff.html`.

[9] Grzegorz Bancerek and Piotr Rudnicki. Two programs for **scm**. Part II - programs. *Journal of Formalized Mathematics*, 5, 1993. `http://mizar.org/JFM/Vol5/fib_fusc.html`.

[10] Grzegorz Bancerek and Andrzej Trybulec. Miscellaneous facts about functions. *Journal of Formalized Mathematics*, 8, 1996. `http://mizar.org/JFM/Vol8/funct_7.html`.

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

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

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

[14] 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`.

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

[16] Agata Darmochwał and Andrzej Trybulec. Similarity of formulae. *Journal of Formalized Mathematics*, 3, 1991. `http://mizar.org/JFM/Vol3/cqc_sim1.html`.

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

[18] Piotr Rudnicki. Another `times` macro instruction. *Journal of Formalized Mathematics*, 10, 1998. `http://mizar.org/JFM/Vol10/sfmastr2.html`.

[19] Piotr Rudnicki and Andrzej Trybulec. Memory handling for $\mathbf{SCM}_{\mathrm{FSA}}$. *Journal of Formalized Mathematics*, 8, 1996. `http://mizar.org/JFM/Vol8/sf_mastr.html`.

[20] Yasushi Tanaka. On the decomposition of the states of SCM. *Journal of Formalized Mathematics*, 5, 1993. `http://mizar.org/JFM/Vol5/ami_5.html`.

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

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

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

[24] Andrzej Trybulec and Yatsuka Nakamura. Some remarks on the simple concrete model of computer. *Journal of Formalized Mathematics*, 5, 1993. `http://mizar.org/JFM/Vol5/ami_3.html`.

[25] Andrzej Trybulec and Yatsuka Nakamura. Modifying addresses of instructions of $\mathbf{SCM}_{\mathrm{FSA}}$. *Journal of Formalized Mathematics*, 8, 1996. `http://mizar.org/JFM/Vol8/scmfsa_4.html`.

[26] Andrzej Trybulec, Yatsuka Nakamura, and Noriko Asamoto. On the compositions of macro instructions. Part I. *Journal of Formalized Mathematics*, 8, 1996. `http://mizar.org/JFM/Vol8/scmfsa6a.html`.

[27] Andrzej Trybulec, Yatsuka Nakamura, and Piotr Rudnicki. The $\mathbf{SCM}_{\mathrm{FSA}}$ computer. *Journal of Formalized Mathematics*, 8, 1996. `http://mizar.org/JFM/Vol8/scmfsa_2.html`.

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

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

Received June 3, 1998

Published January 2, 2004