Translating an FP Dialect to $L$ - A Proof of Correctness


*Dennis M. Volpano*

CSE TR 85-001

Oregon Graduate Center
19600 NW Walker Rd.
Beaverton, OR. 97006



### Abstract

A dialect of FP includes FP selectors over tuples and the FP combining forms composition, condition, iteration and tuple construction. The primitives in a dialect are the primitive operations over some abstract data type. In this technical report, the translation of an FP dialect to an abstract imperative language $L$ is formalized. A denotational description of $L$ is given and the translation is proven correct.


### Preliminary definitions

Before proving the correctness of the translation, some definitions and conventions must be given.

*Definition.* (Domains). Let *Id* be a set of cell names (identifiers) and $V$ be a domain of objects. Let *Sexp* be the domain of ***store expressions***. A store expression is a cell name or a sequence of store expressions. Let *Env* denote the domain of environments where an environment is a sequence of cells such that each cell is a triple <CELL, ***name*** , ***contents*** > [1].

*Definition.* Let the signature $\Sigma$ be given by:

$$\mathit{while} : 2 \to 1$$

$$\sigma_0 : 0 \to 1$$

$$\cdot : 2 \to 1$$

$$(\to ;) : 3 \to 1$$

$$[]_n : n \to 1$$

The operator $\sigma_0$ is a family of nullary operators each of which is a primitive operation over some abstract data type. Let $T_\Sigma$ be a $\Sigma$-algebra whose carrier is the set of all $\Sigma$-terms and whose operators are those in $\Sigma$.

*Definition.* Let the signature $\Omega$ be given by:

$$\textbf{whiledo} : 2 \to 1$$

$$\textbf{assign} : 0 \to 1$$

$$\textbf{semi} : 2 \to 1$$

$$\textbf{cond} : 3 \to 1$$

$$\textbf{constr}_n : n \to 1$$

The operator **assign** is a family of nullary operators indexed by elements of $\sigma_0$, *Sexp* and *Id*. For example, if $tl \in \sigma_0$, $x \in Sexp$ and $n \in Id$ then **assign** $(n$ , apply $(tl , x))$ is a nullary operator. Let $T_\Omega$ be an $\Omega$-algebra whose carrier is the set of all $\Omega$-terms and whose operators are those in $\Omega$.

*Conventions.* Unless otherwise noted, $p$ , $f$ and $g$ are $\Sigma$-terms, $l, l_1, l_2, \ldots$ are $\Omega$-terms and $x, x_1, x_2, \ldots$ are store expressions. Upper-case italic symbols $(I, I', \ldots)$ will be used as metavariables ranging over cell names.

*Definition.* The ***apply*** constructor builds an application of a primitive in $\sigma_0$ to a store expression. The usual extractors, ***operator*** and ***operand***, can be used on constructions built with ***apply*** but do not appear in our proof since no further interpretation is given to ***apply*** in the domain of $\Omega$-terms.

*Definition.* Let ***new*** be a function that maps a set of cell names $s$ to a single cell name such that $new\ (s) \notin s$. Let $cells : Sexp \to P\ (Id)$ be a function defined as follows:

$$cells\ I = \{I\}$$

$$cells < x_1, \ldots, x_n > = cells\ x_1 \bigcup \cdots \bigcup cells\ x_n$$

*Definition.* Let $\textbf{store} : Id \to (Env \to Env)$ and $\textbf{fetch} : Sexp \to (Env \to V)$ be functional forms defined as follows:

$$(\textbf{store}\ I) = \text{apndl} \cdot [[\overline{\text{CELL}}, \overline{I}, 1], 2]$$

$$(\textbf{fetch}\ I) = \text{eq} \cdot [\overline{I}, 2 \cdot 1] \to 3 \cdot 1;$$
$$(\textbf{fetch}\ I) \cdot \text{tl}$$

(**fetch** $<x_1, \ldots, x_n>$) = [(**fetch** $x_1$), ..., (**fetch** $x_n$)]

The functional form **fetch** on store expressions behaves like the function "lift" on sequences in [2].

*Definition.* Let $\eta : T_\Omega \rightarrow Sexp$ be a function defined as follows:

$\eta$ **assign** ($I$, apply ($f$, $x$)) = $I$

$\eta$ **semi** ($l_1$, $l_2$) = $\eta\, l_2$

$\eta$ **cond** ($l_1$, $l_2$, $l_3$) = $\eta\, l_2$

$\eta$ **whiledo** ($l_1$, $l_2$) = $\eta\, l_2$

$\eta$ **constr**$_n$ ($l_1, \ldots, l_n$) = $<\eta\, l_1, \ldots, \eta\, l_n>$

Intuitively, $\eta\, l$ is the store expression representing the array of cells in which results would be "deposited" if $l$ were evaluated.

*Definition.* Let $\delta : T_\Omega \times Sexp \rightarrow T_\Omega$ be a function such that $\eta\, \delta(l, x) = x$. The mapping $\delta$ must satisfy an axiom as we shall see. Intuitively, $\delta$ performs a result-cell coercion by forcing the "result" cells of $l$ to be *cells* ($x$).

*Definition.* Let $\psi : T_\Omega \times Sexp \rightarrow T_\Omega$ be a function. Intuitively, $\psi(l, x)$ preserves the meaning of the $\Omega$-term $l$ and preserves the store expression $x$. As we shall see, the mapping $\psi$ must satisfy two axioms.

*Definition.* Let $\Phi : T_\Sigma \times Sexp \times P\,(Id) \rightarrow T_\Omega$ be a function. In $\Phi(f, x, s)$, $f$ is a $\Sigma$-term to be translated, $x$ is a store expression to which $f$ is applied and $s \in P\,(Id)$ is called the *reserved set*. The set $s$ is reserved in the sense that any cell names created by virtue of translating $f$ must be cell names that do not appear in $s$. Let $\Phi$ be defined as follows:

$\Phi(f, x, s)$ = **assign** (*new* ($s$), apply ($f$, $x$))

where $f$ is a primitive in $\sigma_0$.

$\Phi(f \cdot g, x, s)$ =
  **semi** ($\Phi(g, x, s)$, $\Phi(f, \eta\,\Phi(g, x, s), s)$)

$\Phi(p \rightarrow f\, ; g, x, s)$ =
  **cond** ($\Phi(p, x, s)$, $\delta(\Phi(f, x, s), \eta\,\Phi(g, x, s))$,
    $\Phi(g, x, s)$ )

$\Phi(\textit{while}\ p\ f, x, s)$ =
  **whiledo** ($\Phi(p, x, s)$, $\delta(\Phi(f, x, s), x)$)

$\Phi([f_1, \ldots, f_n], x, s)$ =
  **constr**$_n$ ($\psi(\Phi(f_1, x, v_1), x)$, ...,
    $\psi(\Phi(f_n, x, v_n), x)$ )

where $f_1, \ldots, f_n$ are $\Sigma$-terms, $v_1 = s$ and $v_j$ = *cells* $(\eta\,\Phi(f_{j-1}, x, v_{j-1})) \cup v_{j-1}$.

*Definition.* Let $\mu : T_\Omega \rightarrow (Env \rightarrow Env)$ be a "meaning map" or representation function giving meaning to $\Omega$-terms. The meaning of $\Omega$-terms is couched in FP so that the FP algebra can be used in proofs about $\Omega$-terms. Let $\mu$ be defined as follows:[1]

$\mu \llbracket \bot \rrbracket = \bot$

$\mu \llbracket$**assign** ($I$, apply (select$_j$, $<x_1, \ldots, x_n>$))$\rrbracket$ =
  (**store** $I$) $\cdot$ [(**fetch** $x_j$), id]

$\mu \llbracket$**assign** ($I$, apply ($opr$, $x$))$\rrbracket$ =
  (**store** $I$) $\cdot$ [$opr$ $\cdot$ (**fetch** $x$), id]

$\mu \llbracket$**semi** ($l_1$, $l_2$)$\rrbracket$ = $\mu \llbracket l_2 \rrbracket \cdot \mu \llbracket l_1 \rrbracket$

$\mu \llbracket$**cond** ($l_1$, $l_2$, $l_3$)$\rrbracket$ =
  (**fetch** $\eta\, l_1$) $\cdot\, \mu \llbracket l_1 \rrbracket \rightarrow \mu \llbracket l_2 \rrbracket ; \mu \llbracket l_3 \rrbracket$

$\mu \llbracket$**whiledo** ($l_1$, $l_2$)$\rrbracket$ =
  (**fetch** $\eta\, l_1$) $\cdot\, \mu \llbracket l_1 \rrbracket \rightarrow$
    $\mu \llbracket$**whiledo** ($l_1$, $l_2$)$\rrbracket \cdot \mu \llbracket l_2 \rrbracket$; id

$\mu \llbracket$**constr**$_n$ ($l_1, \ldots, l_n$)$\rrbracket$ =
  $\mu \llbracket l_n \rrbracket \cdot \mu \llbracket l_{n-1} \rrbracket \cdot \cdots \cdot \mu \llbracket l_1 \rrbracket$

The mappings $\delta$ and $\psi$ must satisfy the following axioms:

(**fetch** $x$) $\cdot\, \mu \llbracket \delta(l, x) \rrbracket$ = (**fetch** $\eta\, l$) $\cdot\, \mu \llbracket l \rrbracket$  (A1)

(**fetch** $x$) $\cdot\, \mu \llbracket \psi(l, x) \rrbracket$ = (**fetch** $x$)  (A2)

(**fetch** $\eta\, \psi(l, x)$) $\cdot\, \mu \llbracket \psi(l, x) \rrbracket$ = (**fetch** $\eta\, l$) $\cdot\, \mu \llbracket l \rrbracket$  (A3)

Axiom (A1) is the axiom of "result-cell coercion" and axioms (A2) and (A3) are the axioms of preservation.

[1] The function (**fetch** $\eta\, l$) is interpreted as **fetch** ($\eta\,(l)$).

**Proof of correctness**

The translation is now proven correct by showing that $\Phi$ preserves the meaning of $\Sigma$-terms. The proof proceeds by structural induction on $\Sigma$-terms.

**Theorem**. For any $\Sigma$-term $f$, store expression $x$ and reserve set $s \in P(Id)$,

$$(\mathbf{fetch}\ \eta\,\Phi(f, x, s)) \cdot \mu\,[\![\Phi(f, x, s)]\!]$$
$$= f \cdot (\mathbf{fetch}\ x)$$

*Proof*. Proceed by structural induction on $\Sigma$-terms. As basis cases, consider the FP selectors over tuples and the primitive operations over some abstract data type. If $f$ is a primitive operation and $new(s) = I$ then for any environment $e$:

$$(\mathbf{fetch}\ \eta\,\Phi(f, x, s)) \cdot \mu\,[\![\Phi(f, x, s)]\!] : e$$
$$= (\mathbf{fetch}\ \eta\ \mathbf{assign}\,(I, \text{apply}\,(f, x))) \cdot \mu\,[\![\mathbf{assign}\,(I, \text{apply}\,(f, x))]\!] : e$$
$$= (\mathbf{fetch}\ I) \cdot (\mathbf{store}\ I) \cdot [f \cdot (\mathbf{fetch}\ x), \text{id}] : e$$
$$= (\mathbf{fetch}\ I) \cdot (\mathbf{store}\ I) : < f \cdot (\mathbf{fetch}\ x) : e, e >$$
$$= (\mathbf{fetch}\ I) : << \text{CELL}, I, f \cdot (\mathbf{fetch}\ x) : e >, e >$$
$$= f \cdot (\mathbf{fetch}\ x) : e$$

If $f$ is an FP selector, say $\text{select}_j$, and $new(s) = I$ then for any environment $e$:

$$(\mathbf{fetch}\ \eta\,\Phi(\text{select}_j, < x_1, \ldots, x_n >, s)) \cdot \mu\,[\![\Phi(\text{select}_j, < x_1, \ldots, x_n >, s)]\!] : e$$
$$= (\mathbf{fetch}\ I) \cdot \mu\,[\![\mathbf{assign}\,(I, \text{apply}\,(\text{select}_j, < x_1, \ldots, x_n >))]\!] : e$$
$$= (\mathbf{fetch}\ I) \cdot (\mathbf{store}\ I) \cdot [(\mathbf{fetch}\ x_j), \text{id}] : e$$
$$= (\mathbf{fetch}\ I) : << \text{CELL}, I, (\mathbf{fetch}\ x_j) : e >, e >$$
$$= (\mathbf{fetch}\ x_j) : e$$

Now suppose that for any $\Sigma$-term $f$, store expression $x$ and reserve set $s$,

$$(\mathbf{fetch}\ \eta\,\Phi(f, x, s)) \cdot \mu\,[\![\Phi(f, x, s)]\!]$$
$$= f \cdot (\mathbf{fetch}\ x)$$

*Composition*.

$$(\mathbf{fetch}\ \eta\,\Phi(f \cdot g, x, s)) \cdot \mu\,[\![\Phi(f \cdot g, x, s)]\!]$$
$$= (\mathbf{fetch}\ \eta\,\Phi(f, \eta\,\Phi(g, x, s), s)) \cdot \mu\,[\![\mathbf{semi}\,(\Phi(g, x, s), \Phi(f, \eta\,\Phi(g, x, s), s))]\!]$$
{defn. of $\eta$ and $\Phi$}

$$= (\mathbf{fetch}\ \eta\,\Phi(f, \eta\,\Phi(g, x, s), s)) \cdot \mu\,[\![\Phi(f, \eta\,\Phi(g, x, s), s)]\!] \cdot \mu\,[\![\Phi(g, x, s)]\!]$$
{defn. of $\mu$}

$$= f \cdot (\mathbf{fetch}\ \eta\,\Phi(g, x, s)) \cdot \mu\,[\![\Phi(g, x, s)]\!]$$
{ind. hyp.}

$$= f \cdot g \cdot (\mathbf{fetch}\ x) \quad \text{\{ind. hyp.\}}$$

*Conditional*.

$$(\mathbf{fetch}\ \eta\,\Phi(p \rightarrow f\,;\,g, x, s))$$
$$= (\mathbf{fetch}\ \eta\ \mathbf{cond}\,(\Phi(p, x, s), \delta(\Phi(f, x, s), \eta\,\Phi(g, x, s)), \Phi(g, x, s)))$$
$$= (\mathbf{fetch}\ \eta\,\delta(\Phi(f, x, s), \eta\,\Phi(g, x, s)))$$
{defn. of $\eta$}

$$= (\mathbf{fetch}\ \eta\,\Phi(g, x, s)) \quad \text{\{defn. of } \delta\text{\}}$$

By definition of $\Phi$,

$$\mu\,[\![\Phi(p \rightarrow f\,;\,g, x, s)]\!]$$
$$= \mu\,[\![\mathbf{cond}\,(\Phi(p, x, s), \delta(\Phi(f, x, s), \eta\,\Phi(g, x, s)), \Phi(g, x, s))]\!]$$
$$= (\mathbf{fetch}\ \eta\,\Phi(p, x, s)) \cdot \mu\,[\![\Phi(p, x, s)]\!] \rightarrow \mu\,[\![\delta(\Phi(f, x, s), \eta\,\Phi(g, x, s))]\!]\,;\ \mu\,[\![\Phi(g, x, s)]\!] \quad \text{\{defn. of } \mu\text{\}}$$

Therefore,

$$(\mathbf{fetch}\ \eta\,\Phi(p \rightarrow f\,;\,g, x, s)) \cdot \mu\,[\![\Phi(p \rightarrow f\,;\,g, x, s)]\!]$$

$$= (\textbf{fetch}\ \eta\, \Phi(g, x, s)) \cdot$$
$$(\textbf{fetch}\ \eta\, \Phi(p, x, s)) \cdot \mu \llbracket \Phi(p, x, s) \rrbracket \rightarrow$$
$$\mu \llbracket \delta(\Phi(f, x, s), \eta\, \Phi(g, x, s)) \rrbracket;$$
$$\mu \llbracket \Phi(g, x, s) \rrbracket$$

$$= (\textbf{fetch}\ \eta\, \Phi(p, x, s)) \cdot \mu \llbracket \Phi(p, x, s) \rrbracket \rightarrow$$
$$(\textbf{fetch}\ \eta\, \Phi(g, x, s)) \cdot$$
$$\mu \llbracket \delta(\Phi(f, x, s), \eta\, \Phi(g, x, s)) \rrbracket;$$
$$(\textbf{fetch}\ \eta\, \Phi(g, x, s)) \cdot \mu \llbracket \Phi(g, x, s) \rrbracket$$
{FP algebra}

$$= (\textbf{fetch}\ \eta\, \Phi(p, x, s)) \cdot \mu \llbracket \Phi(p, x, s) \rrbracket \rightarrow$$
$$(\textbf{fetch}\ \eta\, \Phi(f, x, s)) \cdot \mu \llbracket \Phi(f, x, s) \rrbracket;$$
$$(\textbf{fetch}\ \eta\, \Phi(g, x, s)) \cdot \mu \llbracket \Phi(g, x, s) \rrbracket$$
{axiom A1}

$$= p \ \cdot (\textbf{fetch}\ x) \rightarrow f \ \cdot (\textbf{fetch}\ x);\ g \ \cdot (\textbf{fetch}\ x)$$
{ind. hyp.}

$$= (p \rightarrow f\ ;\ g) \cdot (\textbf{fetch}\ x)$$ {FP algebra}

*Iteration*. Proceed by fixpoint induction.

$$(\textbf{fetch}\ \eta\, \Phi(\bot, x, s)) \cdot \mu \llbracket \Phi(\bot, x, s) \rrbracket$$

$$= (\textbf{fetch}\ \eta\, \Phi(\bot, x, s)) \cdot \mu \llbracket \bot \rrbracket$$ {defn. of $\Phi$}

$$= \bot$$ {FP algebra}

$$= \bot \ \cdot (\textbf{fetch}\ x)$$ {FP algebra}

Now the fixpoint inductive hypothesis is given by:

$$(\textbf{fetch}\ \eta\, \Phi(\textit{while}\ p\ f, x, s)) \cdot$$
$$\mu \llbracket \Phi(\textit{while}\ p\ f, x, s) \rrbracket$$
$$= (\textit{while}\ p\ f) \cdot (\textbf{fetch}\ x)$$

Assume the fixpoint inductive hypothesis. Then,

$$(\textbf{fetch}\ \eta\, \Phi(\textit{while}\ p\ f, x, s)) \cdot$$
$$\mu \llbracket \Phi(\textit{while}\ p\ f, x, s) \rrbracket$$

$$= (\textbf{fetch}\ \eta\, \Phi(\textit{while}\ p\ f, x, s)) \cdot$$
$$\mu \llbracket \textbf{whiledo}(\Phi(p, x, s), \delta(\Phi(f, x, s), x)) \rrbracket$$
{defn. of $\Phi$}

$$= (\textbf{fetch}\ \eta\, \Phi(\textit{while}\ p\ f, x, s)) \cdot$$
$$(\textbf{fetch}\ \eta\, \Phi(p, x, s)) \cdot \mu \llbracket \Phi(p, x, s) \rrbracket \rightarrow$$
$$\mu \llbracket \Phi(\textit{while}\ p\ f, x, s) \rrbracket \cdot$$
$$\mu \llbracket \delta(\Phi(f, x, s), x) \rrbracket;\ \mathrm{id}$$
{defn. of $\mu$}

$$= (\textbf{fetch}\ \eta\, \Phi(p, x, s)) \cdot \mu \llbracket \Phi(p, x, s) \rrbracket \rightarrow$$
$$(\textbf{fetch}\ \eta\, \Phi(\textit{while}\ p\ f, x, s)) \cdot$$
$$\mu \llbracket \Phi(\textit{while}\ p\ f, x, s) \rrbracket \cdot \mu \llbracket \delta(\Phi(f, x, s), x) \rrbracket;$$
$$(\textbf{fetch}\ \eta\, \Phi(\textit{while}\ p\ f, x, s))$$ {FP algebra}

$$= (\textbf{fetch}\ \eta\, \Phi(p, x, s)) \cdot \mu \llbracket \Phi(p, x, s) \rrbracket \rightarrow$$
$$(\textit{while}\ p\ f) \cdot (\textbf{fetch}\ x) \cdot \mu \llbracket \delta(\Phi(f, x, s), x) \rrbracket;$$
$$(\textbf{fetch}\ \eta\, \Phi(\textit{while}\ p\ f, x, s))$$ {fix. ind. hyp.}

$$= (\textbf{fetch}\ \eta\, \Phi(p, x, s)) \cdot \mu \llbracket \Phi(p, x, s) \rrbracket \rightarrow$$
$$(\textit{while}\ p\ f) \cdot (\textbf{fetch}\ \eta\, \Phi(f, x, s)) \cdot$$
$$\mu \llbracket \Phi(f, x, s) \rrbracket;$$
$$(\textbf{fetch}\ \eta\, \Phi(\textit{while}\ p\ f, x, s))$$ {axiom A1}

$$= (\textbf{fetch}\ \eta\, \Phi(p, x, s)) \cdot \mu \llbracket \Phi(p, x, s) \rrbracket \rightarrow$$
$$(\textit{while}\ p\ f) \cdot (\textbf{fetch}\ \eta\, \Phi(f, x, s)) \cdot$$
$$\mu \llbracket \Phi(f, x, s) \rrbracket;$$
$$(\textbf{fetch}\ x)$$ {defn. of $\eta$}

$$= p \ \cdot (\textbf{fetch}\ x) \rightarrow (\textit{while}\ p\ f) \cdot f \ \cdot (\textbf{fetch}\ x);$$
$$(\textbf{fetch}\ x)$$ {ind. hyp.}

$$= (p \rightarrow (\textit{while}\ p\ f) \cdot f\ ;\ \textit{id}) \cdot (\textbf{fetch}\ x)$$
{FP algebra}

$$= (\textit{while}\ p\ f) \cdot (\textbf{fetch}\ x)$$ {FP algebra}

*Construction*. The following proposition is needed in the proof of construction.

**Proposition**. For any $\Omega$-term of the form, $\textbf{constr}_n\,(\psi(l_1, x), \ldots, \psi(l_n, x))$ where $x$ is a store expression and $i \neq j$,

$$(\textbf{fetch}\ \eta\, \psi(l_i, x)) \cdot \mu \llbracket \psi(l_j, x) \rrbracket$$
$$= (\textbf{fetch}\ \eta\, \psi(l_i, x))$$

*Proof*. If $\eta\, \psi(l_i, x) \neq \eta\, \psi(l_j, x)$ then by definition of **fetch** the proposition holds. Suppose $l_i = \Phi(f_i, x, v_i)$ and $l_j = \Phi(f_j, x, v_j)$ where $v_i$ and $v_j$ are reserved sets such that

$v_{k+1} = cells\,(\eta\,\Phi\,(f_k, x, v_k)) \bigcup v_k$. If $i < j$ then without result-cell coercion $\eta\,\psi\,(l_i, x) \in v_j$. Therefore $\eta\,\psi\,(l_j, x) \neq \eta\,\psi\,(l_i, x)$ since $v_j$ is a reserved set. Similarly, if $i > j$ then without coercing result cells $\eta\,\psi\,(l_j, x) \in v_i$. Therefore $\eta\,\psi\,(l_i, x) \neq \eta\,\psi\,(l_j, x)$ since $v_i$ is a reserved set. Hence $\eta\,\psi\,(l_i, x) = \eta\,\psi\,(l_j, x)$ only if $l_i$ and $l_j$ are the products of a result-cell coercion such that $\eta\,\psi\,(l_i, x) = \eta\,\psi\,(l_j, x) = x$. By axiom (A2),

$$(\mathbf{fetch}\ x) \cdot \mu\,[\![\psi\,(l_j, x)]\!] = (\mathbf{fetch}\ x)$$

and since $x = \eta\,\psi\,(l_i, x)$, the proposition holds.

The proof of construction now proceeds as follows:

$$(\mathbf{fetch}\ \eta\,\psi\,(\Phi\,(f_i, x, v_i), x)) \cdot \mu\,[\![\Phi\,([f_1, \ldots, f_n], x, s)]\!]$$

$$= (\mathbf{fetch}\ \eta\,\psi\,(\Phi\,(f_i, x, v_i), x)) \cdot \mu\,[\![\mathbf{constr}_n\,(\psi\,(\Phi\,(f_1, x, v_1), x), \ldots, \psi\,(\Phi\,(f_n, x, v_n), x))]\!] \quad \{\text{defn. of } \Phi\}$$

$$= (\mathbf{fetch}\ \eta\,\psi\,(\Phi\,(f_i, x, v_i), x)) \cdot \mu\,[\![\psi\,(\Phi\,(f_n, x, v_n), x)]\!] \cdot \cdots \cdot \mu\,[\![\psi\,(\Phi\,(f_1, x, v_1), x)]\!] \quad \{\text{defn. of } \mu\}$$

$$= (\mathbf{fetch}\ \eta\,\psi\,(\Phi\,(f_i, x, v_i), x)) \cdot \mu\,[\![\psi\,(\Phi\,(f_i, x, v_i), x)]\!] \cdot \cdots \cdot \mu\,[\![\psi\,(\Phi\,(f_1, x, v_1), x)]\!] \quad \{\text{proposition}\}$$

$$= (\mathbf{fetch}\ \eta\,\Phi\,(f_i, x, v_i)) \cdot \mu\,[\![\Phi\,(f_i, x, v_i)]\!] \cdot \cdots \cdot \mu\,[\![\psi\,(\Phi\,(f_1, x, v_1), x)]\!] \quad \{\text{axiom A3}\}$$

$$= f_i \cdot (\mathbf{fetch}\ x) \cdot \mu\,[\![\psi\,(\Phi\,(f_{i-1}, x, v_{i-1}), x)]\!] \cdot \cdots \cdot \mu\,[\![\psi\,(\Phi\,(f_1, x, v_1), x)]\!] \quad \{\text{ind. hyp.}\}$$

$$= f_i \cdot (\mathbf{fetch}\ x) \quad \{\text{axiom A2, i} - 1 \text{ times}\}$$

This concludes the proof of correctness.

**References**

[1] Backus, J., "Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs", *CACM*, 21, 8, 1978, pp. 613-641.

[2] -----, "Function Level Programs as Mathematical Objects", ACM Conf. on Functional Programming Languages and Computer Architecture, Portsmouth, NH, 1981, pp. 1-10.