Mapcode Theory in Designing Programs

Venkata Rao Kuchibhotla
Government College (A),
Rajahmundry,
A.P., India-533101

B.Gayatri Devi
Godavari Institute of Engineering and Technology,
Rajanagaram,
A.P., India-533296

ABSTRACT
A methodology is proposed for designing a program that takes care of safety (semantics) and termination using Mapcode theory.

General Terms:
Theory of Computation, program designing

Keywords:
partial recursive functions, primal maps, mapcode theory

1. INTRODUCTION

The concept of computability in the theory of computation is introduced to the students through many models like Turing machines, Partial Recursive Functions, $\lambda$-calculus, Post machines, and Register machines. When dealing with practical computing problems, the same students need to learn a programming language and study how to express algorithms in that language. Thus there is a disconnect between theory and practice.

In addition to the problem mentioned above, there are issues like analysis and design of programs which require a study on the principles involved in it. David Gries in his book [4] pointed out that “Programming began as an art, and even to day most people learn only by watching others perform (e.g. a lecturer, a friend) and through habit, with little direction as to the principles involved. . . we are reaching the point where we can begin to teach the principles so that they can be consciously applied”.

In a series of articles [9, 10, 11, 12, 13] and a book [14] a theory called ‘mapcode formalism’ has been presented as an alternative approach so that a student can learn both the theoretical and practical issues in terms of a single framework. In the article [2] a class of maps called primal maps using mapcode theory was introduced and it was proved that the class of primal maps coincide with the class of partial recursive maps.

In continuation of the work, in the article [3] a mathematical framework was proposed that systematically helps in analyzing programs, particularly when dealing with ideas as in [2]. The present article addresses some of the issues related to design of programs. Certain structures are identified in designing a program and certain classes of primal maps are identified based on the programming structure for computing them. It is proposed that the identified classes of maps act as building blocks in designing a program for computing a primal map. This is verified with some of the standard examples discussed in [5] which are used to train students in the discipline of programming. However, it is not claimed that the classification is complete. But this kind of study helps in understanding design of algorithms and ensuring safety and termination simultaneously. This article presents a class of primal maps called maps admitting variations and a program structure which works for any map in the class.

Let $\mathbb{N}$ and $\mathbb{P}$ denote the set of all non-negative integers and the set of all positive integers respectively. Maps under consideration are those maps which are defined on subsets of $\mathbb{N}^p$ into $\mathbb{N}^r$. A function is a map from a subset of $\mathbb{N}^p$ into $\mathbb{N}$. A map $f$ with domain $D \subseteq \mathbb{N}^p$ and codomain $\mathbb{N}^r$ is called a partial map. It will be denoted by either $f : \mathbb{N}^p \to \mathbb{N}^r$ or $f : D \subseteq \mathbb{N}^p \to \mathbb{N}^r$. The domain $D$ of $f$ can also be written as $D_f$ or $dom(f)$ when it is necessary to indicate the dependence of $D$ on $f$. If $D_f = \mathbb{N}^p$, $f$ is said to be total. The symbol $\models$ may be read as ‘is defined as’. The symbols $S, X, T$ are used for specific purposes. They are subsets of $\mathbb{N}^p$ possibly for different values of $n$.

2. MAPCODE FORMALISM

The definition of algorithm and other concepts given below are particular cases of a more general setup presented in [14]. This article is confined to the subsets of $\mathbb{N}^p$ for some $q \in \mathbb{P}$.

**DEFINITION 1.** A map $F : D \subseteq \mathbb{N}^p \to \mathbb{N}^r$ is called a Program Map on $\mathbb{N}^p$. Let $F$ be a program map on $\mathbb{N}^p$.

1. $x \in D$ is called a fixed point of $F$ if $F(x) = x$. The set of all fixed points of $F$ is denoted by $fix(F)$. It is called the fixed point set of $F$.
2. If $x \in D$ and $F(x) \in D$, then $F^2(x) = F(F(x))$. More generally, if $x, F(x), \ldots, F^n(x)$ are all in $D$ then $F^{n+1}(x) = F(F^n(x))$. By convention, $F^0(x) = x$ for any $x$, and $F^1(x) = F(x)$.
3. $\text{con}(F) = \{x \in D | F^n(x) \in fix(F) \text{ for some } n \geq 0\}$ is called the convergent set of $F$, and elements of $\text{con}(F)$ are called convergent points of $F$.
4. For any $x \in \text{con}(F)$, let $N(x)$ be the least $n \geq 0$ such that $F^n(x) \in fix(F)$. $N(x)$ is called the runtime of $F$ at $x$.
5. The map $F^\infty : \text{con}(F) \to fix(F)$ defined by $F^\infty(x) = F^{N(x)}(x)$ is called the limit map of $F$. □
Remark 2. If \( F = Id : \mathbb{N}^n \to \mathbb{N}^n \) is the identity map then 
\( \text{con}(F) = \text{fix}(F) = \mathbb{N}^n \) and \( F^\infty = Id. \) Further, the runtime \( N(x) = 0 \) for every \( x \in \mathbb{N}^n. \)

One can think of \( F \) as defining a computation. Knuth [6] says the same thing when he observes that a mathematical theory of algorithms can be built by modeling an algorithm as a map \( F. \) Given a state \( x \in \mathbb{N}^n \) the computation proceeds by evaluating in turn 
\( F(x), F^2(x), \ldots. \) If \( x \in \text{con}(F) \) then there comes a stage when 
\( F^n(x) \in \text{fix}(F). \) The computation is then considered to terminate.

Definition 3. Let \( s : S \subseteq \mathbb{N}^n \to \mathbb{N}^n \) be a partial map and \( \rho : H \subseteq \mathbb{N}^n \to \mathbb{N}^n \) be partial maps. Then the ordered triple \( M = (\rho, F, \pi) \) is called a mapcode machine (over \( \mathbb{N} \)). The maps \( \rho, F, \) and \( \pi \) are called the coding map, program map, and decoding map of the machine respectively.

(2) The set \( \Omega = \{ s \in S \mid \rho(s) \in \text{con}(F), F^\infty(s) \in H \} \) is called the halting set of the mapcode machine \( M. \)

(3) The map \( \phi = \pi \circ F^\infty \circ \rho : \Omega \subseteq \mathbb{N}^n \to \mathbb{N}^n \) is called the semantic map of the mapcode machine \( M. \)

(4) Let \( f : \mathbb{N}^n \to \mathbb{N}^n \) be a partial map with its domain \( D_f, \) and let 
\( M = (\rho, F, \pi) \) be a mapcode machine. \( M \) computes \( f \) or \( M \) is an algorithm for computing \( f, \) if \( D_f = \Omega, \) the halting set of 
\( M \) and \( f(s) = \phi(s) \) for every \( s \in D_f. \)

The terminology of ‘halting set’ and the notations for halting set and semantic map are borrowed from [11]. However, this terminology in the mapcode theory [1] is more general than theirs.

The components of a mapcode machine, namely the maps \( \rho, F, \) and \( \pi, \) constitute its syntax. The semantic map \( \phi \) gives the semantics of the machine.

3. SAFETY AND PROGRESS THEOREMS

Given a mapcode machine \( (\rho, F, \pi) \), to prove that a state \( x \) is convergent and the output is the specified one, there is a need to have some way of showing safety and progress. The concept of invariant principle ensures safety and the concept of a bound function \( \text{[1]} \) is useful for verifying the progress.

Definition 4. Let \( f : \mathbb{N}^n \to \mathbb{N}^n \) be a partial map and \( M = (\rho, F, \pi) \) be a mapcode machine with \( \rho : S \subseteq \mathbb{N}^n \to \mathbb{N}^n, F : X \subseteq \mathbb{N}^n \to \mathbb{N}^n, \) and \( \pi : H \subseteq \mathbb{N}^n \to \mathbb{N}^n. \) Suppose that 
\( D_f \subseteq S. \)

(1) A nonempty subset \( V \subseteq X \) is said to be invariant under \( F \) if 
\( F(V) \subseteq V. \)

(2) A subset \( V \subseteq X \) is called an invariant principle for \( f \) at \( s \in D_f, \) if \( V \) is invariant under \( F, \rho(s) \in V, V \cap \text{fix}(F) \subseteq H \) and 
\( \pi(V \cap \text{fix}(F)) = \{ f(s) \}. \)

(3) A map \( \lambda : X \to \mathbb{N} \) is called a bound function for \( F \) if for every 
\( x \in X \) with \( F(x) \in X \) either \( x \in \text{fix}(F) \) or \( \lambda(F(x)) < \lambda(x). \)

(4) For any \( s \in S \) define \( V_s = \{ F^n(s) \mid n \geq 0 \}. \)

Remark 5. (1) If \( \rho(s) \notin X \) then \( F \) is not defined on \( \rho(s). \) Then 
\( V_s = \{ \rho(s) \} \) and \( V_s \) is not invariant under \( F. \) If \( \rho(s) \in X, \) but \( F(\rho(s)) \notin X \) then \( F \) is not defined on \( F(\rho(s)). \) Then 
\( V_s = \{ \rho(s), F(\rho(s)) \} \) and \( V_s \) is not invariant under \( F. \) Continuing the argument inductively, \( V_s \) is invariant under \( F \) if and only if \( F^n(\rho(s)) \in X \) for every \( n \geq 0. \)

(2) Let \( x \in X \) and let \( \lambda(x) = 0. \) Then either \( x \in \text{fix}(F) \) or 
\( F(x) \notin X. \)
More general concept of variations of a map

behind the definition of variations of a map.

This shows that the set of operations involved in generating

The functions zero, succ, and P are called initial functions.

For any set A define a function $\chi_A(x) = 1$ if $x \in A$
and $\chi_A(x) = 0$ if $x \not\in A$. The function $\chi_A$ is called indicator
function of A.

**Definition 9.** The class of primal maps is the smallest class
of maps that contains the initial functions and is closed under just-
taposition, conditional maps and semantic maps of mapcode
machines $(\rho, F, \pi)$.

A set is said to be primal if its indicator function is primal. 

**Remark 10.** The term 'closed under an operation' needs some
explanation. It is explained in the case of semantic maps, and
argument is similar for the other cases. A class C is closed under
semantic maps of mapcode machines if $\phi = \pi \circ F^\infty \circ \rho \in C$ whenever
$\rho, F, \pi \in C$.

It is proved in [2] that the class of primal maps is the class of
partial recursive maps from the classical theory of computation
[3]. This shows that the set of operations involved in generating
partial recursive maps from initial maps can be replaced by the
set of operations involved in generating primal maps.

A class of maps called "maps admitting variations" is introduced
in the next section and a design for computing such maps is proposed.

5. Maps Admitting Variations

For any $x = (x[0], x[1], \ldots, x[k - 1]) \in N^k$ and $y \in N$,
the juxtaposition of $x, y$ is denoted by $(x, y)$ so that $(x, y) =
(x[0], x[1], \ldots, x[k - 1], x[k]) \in N^{k+1}$ with $x[k] = y$.

**Definition 11.** (1) Let $f_1 : D_1 \subseteq N^k \rightarrow N^r$ and $f_2 : D_2 \subseteq
N^{k+1} \rightarrow N^s$. The map $f_1$ is called a variation of $f_2$ if there
exists a map $L : D_2 \subseteq N^{k+1} \times N^s \rightarrow N^s$ as a nontrivial
solution of the equation $f_2(x, y) = L(x, y, f_1(x))$, for any
$(x, y) \in D_1$. The map $L$ is called lifting of $f_1$ to $f_2$ (or
simply a lift map). The equation $f_2(x, y) = L(x, y, f_1(x))$ is
referred as lift equation.

(2) Let $p < n$, and let $f_p : D_p \subseteq N^p \rightarrow N^q$ for $p \leq k \leq n$
be given. If $f_k$ is a variation of $f_{k+1}$ for $p \leq k \leq n - 1$ then the
maps $f_p, f_p \rightarrow f_{k+1}, \ldots, f_{n-1}$ are called variations of $f_k$. 

**Remark 12.** (1) The map $L(x, y, t) = f_2(x, y)$ for every $t$,
is always a solution for the lift equation and it is called a trivial
solution.

(2) Suppose that $f_p, f_{p+1}, \ldots, f_{n-1}$ are variations of $f_k$. Then
$(x_1, x_2, \ldots, x_k) \in \text{dom}(f_k) \Leftrightarrow (x_1, x_2, \ldots, x_k) \in
\text{dom}(f_k)$ for $p \leq k \leq n$.

(3) More general concept of variations of a map $f$ is given as a
collection of maps $f_i$ indexed by elements $i$ of a partial order set
$P$. In this case, $f_i$ is a variation of $f_j$ whenever $j$ is a successor
of $i$ in $P$.

**Example 1.** Finding maximum (or minimum) of $n$ integers
and sum of $n$ integers are simple examples to understand the idea
behind the definition of variations of a map.

(1) **Maximum Problem:** For any $1 \leq k \leq n$, and any
$x = (x[0], x[1], \ldots, x[k - 1]) \in N^k$, define $f_k(x) = \text{Max}(x[0], x[1],
\ldots, x[k - 1])$. Then $f_k : N^k \rightarrow N$. It is easy to see that $\text{Max}(x[0], x[1],
\ldots, x[k]) = \text{Max} \{x[k], \text{Max}(x[0], x[1], \ldots, x[k - 1])\}$
or $f_{k+1}(x, y) = \text{Max}(x, y, f_k(x))$, for $x \in N^k, y \in N$.
More precisely, if $L_{k+1} : N^{k+1} \times N \rightarrow N$ is defined by
$L_{k+1}(x, y, t) = \text{Max}(x[t], y)$, for any $x \in N^k, y, t \in N$, then
$f_{k+1}(x, y) = L_{k+1}(x, y, f_k(x))$. The map $L_{k+1}$ is lifting
of $f_k$ to $f_{k+1}$. The maps $f_1, f_2, \ldots, f_{n-1}$ are variations of
$f = f_n$.

(2) **Summation Problem:** For any $1 \leq k \leq n$, and any $x =$
$(x[0], x[1], \ldots, x[k - 1]) \in N^k$, define $f_k(x) = x[0] + x[1]
\ldots, + x[k - 1]$. Then $f_k : N^k \rightarrow N$. It is easy to see that
$x[0] + x[1] + \ldots, + x[k] = x[k] + (x[0] + x[1] + \ldots, + x[k - 1])$
or $f_{k+1}(x, y) = y + f_k(x)$, for $x \in N^k, y \in N$.
If $L_{k+1} : N^{k+1} \times N \rightarrow N$ is defined by
$L_{k+1}(x, y, t) = y + t$, for any $x \in N^k, y, t \in N$, then
$f_{k+1}(x, y) = L_{k+1}(x, y, f_k(x))$. Then the maps $f_1, f_2, \ldots, f_{n-1}$ are vari-
ations of $f = f_n$.

Two more standard examples in the discipline of programming
are given to fix the idea of variation in mind.

**Example 2. Checking Equality Problem:** Choose and fix an
element $x' = (x[0], x[1], \ldots, x[p-1]) \in N^p$. For $k \in \mathbb{P}$ with
$1 \leq k \leq p$, and $x = (x[0], x[1], \ldots, x[k - 1]) \in N^k$, define
$f_k(x) = 1, \text{if } (x[0], x[1], \ldots, x[k - 1]) = (x'[0], x'[1], \ldots, x'[k - 1])$
$0, \text{if } (x[0], x[1], \ldots, x[k - 1]) \neq (x'[0], x'[1], \ldots, x'[k - 1]).$ The maps
$f_1, f_2, \ldots, f_{n-1}$ are variations of $f = f_n$. If $L_{k+1} : N^{k+1} \times N \rightarrow N$ is defined by
$L_{k+1}(x, y, t) = t, \text{if } y = s'[k], \text{and}
L_{k+1}(x, y, t) = 0, \text{if } y \neq s'[k],$ then $f_{k+1}(x, y) = L_{k+1}(x, y, f_k(x))$. The map $L_{k+1}$ is lifting
of $f_k$ to $f_{k+1}$.

**Example 3. Pattern Matching Problem:** The question is : how
many times the pattern $s' = (s[0], s[1], \ldots, s[p-1])$ occur in
the sequence $s = (s[0], s[1], \ldots, s[n - 1])$, assuming that $n$ is
many times larger than $p$.

Consider the notation and terminology used in the example [2].
For any $k \in \mathbb{P}$ with $p \leq k \leq n$, define $f_k : N^k \rightarrow N^p$
by $f_k(x) \cdot$ the number of times the pattern $x'$ occurs in $x$. The maps
$f_1, f_2, \ldots, f_{n-1}$ are variations of $f_k$.

To find $L_{k+1} \rightarrow$ lifting of $f_k$ to $f_{k+1}$, the map $\psi_k$ which picks up
the last $p$ entries of the array $(s[0], s[1], \ldots, s[k - 1])$ is needed.
For any $p \leq k \leq n$, define the map $\psi_k : N^k \rightarrow N^p$ by
$\psi_k(x[0], x[1], \ldots, x[k - 1]) = (x[p], x[k-p+1], \ldots, x[k-1])$.
Then for any $x = (x[0], x[1], \ldots, x[k]) \in N^{k+1}$,
$f_{k+1}(x) = f_k(x[0], x[1], \ldots, x[k - 1] + 1, \text{if } \psi_k(x) \neq x',
\text{then } f_{k+1}(x) = f_k(x[0], x[1], \ldots, x[k - 1]), \text{if } \psi_k(x) \neq x'$. So,$f_{k+1}(x) = f_k(x[0], x[1], \ldots, x[k - 1] + f_k(\psi_k(x)).$
It is now easy to understand that the map $L_{k+1}(x, y, t)$ defined by
$L_{k+1}(x, y, t) = t + f_k \circ \psi_k(x, y)$ is lifting of $f_k$ to $f_{k+1}$.

The purpose of considering variations of $f$ is to split the computation
of $f$ into finite number of simple segments. The following theorem helps reaching this goal.
THEOREM 13. Suppose that \( \{f_0, f_{p+1}, \ldots, f_{n-1}\} \) are variations of \( f = f_n \). Let \( L_{k+1} \) be a lifting of \( f_k \) to \( f_{k+1} \). If \( f_p \) and the map \( L_k \) are primal maps then \( f \) is a primal map.

PROOF. Suppose that \( f_k : \mathbb{N}^k \to \mathbb{N}^r \) for \( p \leq k \leq n \). Given that \( L_{k+1} (x, y, f_k(x)) = L_{k+1} (x, y, f_k(x)) \) for \( p \leq k \leq n - 1 \). Let \( \eta_k : \mathbb{N}^n \to \mathbb{N}^k \) be defined by \( \eta_k (x) = (x[0], x[1], \ldots, x[k-1]) \) for \( x = (x[0], x[1], \ldots, x[n-1]) \in \mathbb{N}^n \). Then the maps \( \eta_k \) are primal maps.

With \( s = (s[0], s[1], \ldots, s[n-1]) \in \mathbb{N}^n \), let \( x = (s, k, t) \) be a general element of \( \mathbb{N}^n \times \mathbb{N} \times \mathbb{N}^r \). Consider the partial maps \( \rho : \mathbb{N}^n \to \mathbb{N}^n \times \mathbb{N} \times \mathbb{N}^r \), \( F : \mathbb{N}^n \times \mathbb{N} \times \mathbb{N}^r \to \mathbb{N}^n \times \mathbb{N} \times \mathbb{N}^r \) and \( \pi : \mathbb{N}^n \times \mathbb{N} \times \mathbb{N}^r \to \mathbb{N}^n \) given by

\[
\rho(s) = (s, p, f_p(\eta_k(s))), \\
F(s, k, t) = \begin{cases} 
(s, k + 1, L_{k+1}(\eta_{k+1}(s), t)) & \text{if } k < n, \\
(s, k, t) & \text{otherwise,}
\end{cases}
\]

and \( \pi(s, k, t) = t \).

Then the maps \( \rho, F \) and \( \pi \) are all primal maps. Let the domains of the maps be \( S, X \) and \( H \) respectively. Then

\[
S = \{ s \in \mathbb{N}^n \mid \eta_k(s) \in \text{dom}(f_p) \}, \\
X = \{ (s, k, t) \mid \eta_{k+1}(s, t) \in \text{dom}(L_{k+1}), p \leq k \leq n - 1 \}, \\
H = \mathbb{N}^n \times \mathbb{N} \times \mathbb{N}^r.
\]

Clearly the map \( \lambda(s, k, t) = n - k \) is a bound function. Further,

\[
s \in \Omega \iff \rho(s) \in X, F^k(\rho(s)) \in \text{fix}(F), \text{ for some } k
\]

\[
\begin{align*}
\{ (s, p, f_p) \} & \in X, \\
\{ (s, p + 1, t, \eta_{p+1}) \} & \in X, \\
\cdots & \\
\{ (s, n, t_n) \} & \in X,
\end{align*}
\]

so that \( V_s \cap \text{fix}(F) = \{ (s, m, f_m(s)) \} \), for \( s \in D_{f_m} \). This proves that \( V_s \) is an invariant principle for \( f_m \) and \( (\rho, F, \pi) \) is an algorithm for computing the map \( f_m \). Hence the map \( f_m \) is a primal map.

6. DOCUMENTATION

This section deals with a method of documentation for the mapcode computation. At the outset, the mapcode machine

\[
\rho(s) = (s, p, f_p \circ \eta_k(s)), \\
F(s, k, t) = \begin{cases} 
(s, k + 1, L_{k+1}(\eta_{k+1}(s), t)) & \text{if } k < n, \\
(s, k, t) & \text{otherwise,}
\end{cases}
\]

and \( \pi(s, k, t) = t \).

itself is a documentation for computing \( f \). But tabular form may be a better option. \( a \gets b \) stands for “\( a \) is replaced by \( b \)”.

7. APPLICATION

The proposed computation model will be used in the case of examples discussed earlier. The advantage of this theory is that the safety and termination are ensured by the theorem and there is no need to verify for each example.

(1) Maximum Problem: With the notation and terminology given in the example [1]

(a) the map \( f_1 \) is identity map on \( \mathbb{N} \), so that it is a primal map.

(b) The lift map \( L_k (x, y, t) = \text{Max} \{ y, t \} \) is a primal map.

Then the maximum function \( f \) is a primal map and the mapcode machine computing \( f \) is given by

\[
\rho(s) = (s, 1, s[0]), \\
F(s, k, t) = \begin{cases} 
(s, k + 1, \text{Max} \{ s[k], t \}) & \text{if } k < n, \\
(s, k, t) & \text{otherwise,}
\end{cases}
\]

and \( \pi(s, k, t) = t \).

Documentation of the computation in tabular form is given by

\[
\begin{array}{c|c|c}
\text{Variables} \quad & s[0], s[1], \ldots, s[n-1], \\
\text{Initialisation} \quad & k, t[0], t[1], \ldots, t[r-1], \\
\text{Program} \quad & \text{input for } s[i], \\
\text{Guard} \quad & t = f_p(s[i], \ldots, s[p-1]), \\
\text{Command} \quad & k' = k + 1, \\
\text{Output} \quad & t
\end{array}
\]

(2) Checking Equality Problem: With the terminology and notation given in the example [2]

(a) \( s' = (s'[0], s'[1], \ldots, s'[p-1]) \in \mathbb{N}^p \) is initially chosen and fixed.

(b) The map \( f_1 : \mathbb{N} \to \mathbb{N} \) is given by \( f_1(x) = 1 \), if \( x = s'[0] \), and \( f_1(x) = 0 \), otherwise. It is a primal map.

(c) For any \( (s, y) \in \mathbb{N}^{p-1} \times \mathbb{N} = \mathbb{N}^n \) and \( f \in \mathbb{N} \), the lift map \( L_n \) is given by \( L_n (s, y, t) = t \), if \( y = s'[k] \), and \( L_n (s, y, t) = 0 \), otherwise. Then \( L_n \) is also a primal map.

Then by theorem [13] the function \( f = f_p \) which checks equality is a primal map. The mapcode machine computing \( f \) is given by
\[ \rho(s, s') = \begin{cases} (s, s', f_1(s)) & \text{if } s_0 = s'_0 \\
(s, s', 0, 1) & \text{otherwise} \end{cases} \]

\[ F(s, s', k, t) = \begin{cases} (s, s', k + 1, t), & \text{if } k < p \\
(s, s', k + 1, 0), & \text{if } k < p, s[k] \neq s'[k] \end{cases} \]

\[ \pi(s, s', k, t) = t. \]

In other words,

\[ \rho(s, s') = \begin{cases} (s, s', 0, 1) & \text{if } s_0 = s'_0 \\
(s, s', 0, 1), & \text{otherwise} \end{cases} \]

\[ F(s, s', k, t) = \begin{cases} (s, s', 1, 1), & \text{if } s_0 = s'_0 \\
(s, s', 1, 0), & \text{otherwise} \end{cases} \]

\[ \pi(s, s', k, t) = t. \]

However, a close observation of the machine will reveal that the following machine will also compute the map \( f \).

\[ \rho(s, s') = \begin{cases} (s, s', 0, 1) & \text{if } s_0 = s'_0 \\
(s, s', 0, 1), & \text{otherwise} \end{cases} \]

\[ F(s, s', k, t) = \begin{cases} (s, s', k + 1, t), & \text{if } k < p, s[k] = s'[k] \\
(s, s', n, 0), & \text{if } k < p, s[k] \neq s'[k] \end{cases} \]

\[ \pi(s, s', k, t) = t. \]

This mapcode machine computing \( f_n \) is referred by \( M_2 \).

8. CONCLUSION

The mapcode machine \( M_2 \) involves maps which are again computable with the machine \( M_1 \). Then one has to combine the mapcode machines \( M_1 \) and \( M_2 \) so that relatively elementary operations will be involved for carrying out the computation. This is an essential part of writing a program.

As in the case of ‘maps admitting variations’, some other classes of maps are identified and proposed a mapcode machine for each class. Though the approach is successfully adopted for several standard problems, some of them are even harder, it is not claimed that these classes are exhaustive. These results and the techniques involved in combining two mapcode machines will be published subsequently.

In the present article, simple illustrations are chosen only for better exposition of the content. Two standard examples are chosen to show how the proposed method helps in the construction of an algorithm. However, as quoted by David Gries [4],

One cannot learn to write large programs effectively until one has learned to write small ones effectively.

9. REFERENCES


