Mapcode Computable Functions

Arindama Singh[1]
Department of Mathematics
Indian Institute of Technology Madras
Chennai-600036, India

Abstract

In this paper we discuss an alternative definition of a program. We name this formalization as a mapcode and attempt to explore the theoretical issues of computability by mapcodes. We show that mapcodes not only define programs as mathematical entities in a natural manner, but also they are equivalent to Turing machines. The equivalence is shown by proving that the class of mapcode computable functions coincides with the class of Turing computable functions. A sufficient condition for termination of a mapcode is also proved and a related problem is raised.

Keywords: Computability, Programs, Turing Machines, Mapcodes

1. Introduction

Starting from Turing machines, we have many formalizations of the notion of an algorithm. Besides the theoretical constructs, we have various approaches to programming. A possibility of a unified approach to both the issues of computability and programmming has been expressed by Knuth in the preface to his book [2], though not followed latter in the book. He mentions that "$\cdots$ the concept of algorithm can be firmly grounded in terms of mathematical set teory. Let us formally define a computational method to be a quadruple (Q, I, Ω, f), in which Q is a set containing subsets I and Ω, and f is a fucntion from Q into itself. Furthermore f should leave Ω pointwise fixed, that is, $f(q)$ should equal q for all elements q of Ω. The four quantities Q, I, Ω, f are intended respectively the states of computation, the input, the output, and the computational rule $\cdots$". Taking clue from this observation, Viswanath has shown in his yet unpublished monograph that a modified version of Knuth's mathematical formulation can be used for proving correctness of programs and also for teaching and learning programming by the use of set theory. An introduction to the formal notions of mapcodes may be found in [3].

In this paper we present a formalization of a mapcode and show that mapcode computability is indeed, equivalent to the conventional notion of Turing computability, in a more natural way. We give a brief exposition of the modified version, called the mapcode formalism in Section 2. In Section 3, we define the primitives and describe the formalism in a precise manner. It keeps the general flavour of mapcodes as introduced in [3] but fixes an undesirable generality that each function is mapcode computable. The main result of the paper, i.e., the equivalence of Turing computability and mapcode computability is proved in Section 4. Section 5

[1]Email : asingh@iitm.ac.in

concludes the paper by proving a sufficient condition for termination of mapcodes raising a related problem.

2. A General Formalism

A program takes an input and produces some output; it computes the map that connects the inputs to corresponding outputs. In so doing the program does some computations involvoing not only the inputs or outputs but also some extra symbols, which constitute, so to speak, a language over the tape alphabet of a Turing machine. All these symbols are used to assign values to the program variables. The set of all ordered tuples of values of the program variables is called the state space of a program. The inputs invoke a particular state in the state space; the program transforms this state to another iteratively, and then the output is read from the final state, where the program has terminated. Thus a program involves an *input map* that takes the input to a state, a *program map* that iteratively transforms the state to another which finally stops transforming, and an *output map* which takes the final state to an element of the set of outputs. The scenario is encapsulated by a *mapcode* which has all these maps as its components. Formally,

A mapcode is a six-tuple $\Omega = (X, S, T, F, \rho, \pi)$, where X is a nonempty set, the state space, S is a nonenmpty set, the input set, T is a nonempty set, the output set, $F : X \to X$ is a partial function, the program map, $\rho : S \to X$ is a total function, the input map, and $\pi : X \to T$ is a partial function, the output map.

In general, a mapcode is constructed for computing a partial function $f : S \to T$, called the *target function*, that relates inputs to outputs. The input map ρ takes an input $s \in S$ to $\rho(s) \in X$. During computation the program map F goes on transforming $\rho(s)$ to $F(\rho(s)), F(F(\rho(s))), \ldots$ in the state space X, until its termination. Given an $x \in X$, the computation of the mapcode is the sequence of states:

$$x,\ F(x),\ F^2(x),\ F^3(x), \cdots$$

where $F^{n+1}(x) = F(F^n(x))$ and $F^0(x) = x$. The sequence $\{F^n(x)\}_{n=0}^{\infty}$ is called the *orbit of* F for x, and is denoted by $orb_F\, x$, following the terminology of dynamical systems. The computation terminates when F does not change two sucessive terms of the sequence, that is, when for some m, $F^{m+1}(x) = F^m(x)$. To fix the idea, we define the *convergent points* of the program map F as follows:

$$con\, F = \{x \in X : F^n(x) = F^{n+1}(x) \text{ for some } n \in \mathbb{N}\}$$

where $\mathbb{N}$ denotes the set of natural numbers $0, 1, 2, \ldots$.

Let m be the least n such that $F^n(x) = F^{n+1}(x)$ for $x \in con\, F$. Then the finite sequence $(x, F(x), \cdots, F^m(x))$ is called the *trace of* x, and is denoted by $tr_F(x)$. We also say that the trace of x *terminates with* $F^m(x)$. In such a case, the output is $\pi(F^m(x))$ for the given input s, where $\rho(s) = x$. For succinct presentation, we define the *limit map* $F^\infty : con\, F \to X$ by

$$F^\infty(x) = F^m(x) \text{ where } tr_F(x) \text{ terminates with } F^m(x)$$

The input $s \in S$ is taken to $\rho(s)$; it is transformed to $F^\infty(\rho(s))$ when the trace terminates, and then finally, to the output $\pi(F^\infty(\rho(s)))$. Thus, the target function f is computed by the mapcode Ω when $f(s) = \pi(F^\infty(\rho(s)))$ for each $s \in S$.

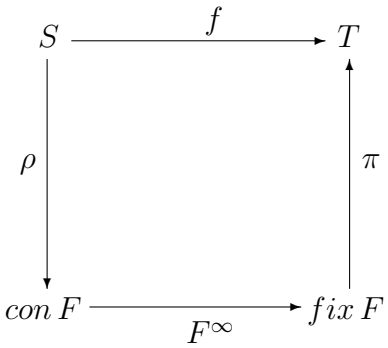
Similar to $con\, F$, the co-domain of the limit map can be restricted. Notice that if y is in the range of F^∞, then $y = F^m(x)$ for an m with $F^m(x) = F^{m+1}(x)$. That is, $F^m(x)$ or y is fixed by F. Defining

$$fix\, F = \{x \in X : F(x) = x\}$$

we see that the range of F^∞ is a subset of $fix\, F$. That is,

$$F^\infty : con\, F \to fix\, F$$

Thus the mapcode $\Omega = (X, S, T, F, \rho, \pi)$ computes the partial function $f : S \to T$ iff the following diagram commutes:



Termination of the mapcode (program) Ω is expressed by the fact that corresponding to the program map F, its limit map F^∞ exists with its domain $con\, F$ being equal to $dom\, F$ and the co-domain $fix\, F$ being nonenmpty. As $con\, F \subseteq dom\, F \neq \emptyset$, termination is proved by showing that $dom\, F \subseteq con\, F$. Partial correctness of the mapcode (program) amounts to commutation of the above diagram. The following example illustrates the formalism. We write $\mathbb{Z}_+$ to denote the set of all positive integers.

Example 2.1 We construct a mapcode for computing $\gcd(x, y)$ for $x, y \in \mathbb{Z}_+$. Here the target function is $\gcd = f : S \to T$ with $S = \mathbb{Z}_+ \times \mathbb{Z}_+$ and $T = \mathbb{Z}_+$. We take the state space $X = S$ and the input function $\rho : S \to X$ as the identity map, i.e., $\rho(x, y) = (x, y)$. We take the output function $\pi : X \to T$ as the first-component projection map, i.e., $\pi(x, y) = x$. We intend to compute the gcd of x and y by starting from the pair (x, y) and ending at (k, p), where $k = \gcd(x, y)$ and p is some positive integer. We define the program map $F : X \to X$ by

$$F(x, y) = \begin{cases} (x - y, y) & \text{if } x > y \\ (x, y - x) & \text{if } x < y \\ (x, y) & \text{if } x = y \end{cases}$$

A typical computation of the mapcode $\Omega = (X, S, T, F, \rho, \pi)$ with input $(6, 9)$ looks like:

$F(6,9) = (6,3)$
$F^2(6,9) = F(6,3) = (3,3)$
$F^3(6,9) = F(3,3) = (3,3)$

That is, $F^\infty(6,9) = F^2(6,9) = (3,3)$. And then

$f(6,9) = \pi(F^\infty(\rho(6,9))) = \pi(F^\infty(6,9)) = \pi(3,3) = 3$

The trace $tr_F(6,9) = ((6,9),(6,3),(3,3)) = (x, F(x), F^2(x))$ terminates at $(3,3)$.

A proof that Ω computes f requires showing that

(a) For each $(x,y) \in \mathbb{Z}_+ \times \mathbb{Z}_+$, $tr_F(x,y)$ terminates with some $(z,z) \in \mathbb{Z}_+$,

(b) For such a $z \in \mathbb{Z}_+$, we have $z = gcd(x,y)$.

The requirements (a-b) are the program termination and the partial correctness, respectively. For (a), we observe that application of F on a given pair (x,y) reduces the size of $|x-y|$ if $|x-y| \neq 0$. After a finite number of applications of F, $|x-y|$ is reduced to 0, when the program terminates. Similarly, (b) can be seen easily as $\gcd(x,y) = \gcd(x,x+y)$ for $x \leq y$, $\gcd(x,y) = \gcd(x,x-y)$ for $x \geq y$, and $\gcd(x,y) = x$ for $x = y$.

3. Mapcodes

As presented in Section 2, the notion of mapcode is too general to be called as a model of computation. In its generality, it is capable of computing every function. For example, let $S, T \neq \emptyset$ be sets and $f : S \to T$ be any function. Take the state space $X = S \times T \times \{0,1\}$. Let $\theta \in T$ be a fixed element and $\rho : S \to X$ be defined by $\rho(s) = (s, \theta, 0)$. Define $F : X \to X$ by $F(x,\theta,0) = (x, f(x), 1)$ and $F(x,y,1) = (x,y,1)$. For other elements of X, let F remain undefined. Fix $\pi : X \to T$ by $\pi(x,y,1) = y$. Then clearly, $F^2(x,\theta,0) = F(x,f(x),1) = (x,f(x),1)$. That is, $F^\infty = F$ is well defined with domain and co-domain of F^∞ as

$con\, F = \{(x,\theta,0) : x \in S\} \cup \{(x,y,1) : x \in S, y \in T\}$ and
$fix\, F = \{(x,y,1) : x \in S, y \in T\}$.

Moreover, $f(s) = \pi(F^\infty(\rho(s)))$; the mapcode Ω computes the map f.

This undesirable state of affair is circumvented by fixing the primitives of a mapcode. It is of course enough to define the primitive maps as those which help us to simulate Turing machines (see Section 4). However, in the mapcode settings that would look too artifical. We will rather fix natural maps and prescribe natural rules for extending the primitives. They are described as follows:

The *primitive* program maps are the maps of the type $P1 - P6$ as given below.

$P1$ *Identity Maps*: $i : X \to X$ defined by $i(x) = x$.

$P2$ Constant Maps: For each $y \in X$, $C_y(x) : X \to X$ defined by $C_y(x) = y$.

$P3$ *Projection-like Maps*: $P_m : X \to X$, defined by

$$P_m(x_1, \ldots, x_m, x_{m+1}, \ldots, x_n) = (y_1, \ldots, y_{m-1}, x_m, y_{m+1}, \ldots, y_n)$$

where $X = X_1 \times X_2 \times \cdots \times X_n$ and $y_1 \in X_1, \ldots, y_{m-1} \in X_{m-1}$, $y_{m+1} \in X_{m+1}, \ldots y_n \in X_n$ are fixed elements.

*P*4 *Comparision Maps*: The maps $K_{FG} : X \to X$ defined by

$$K_{FG}(x) = \begin{cases} a & \text{if } F(x) = G(x) \\ b & \text{if } F(x) \neq G(x) \end{cases}$$

where $a, b \in X$ are fixed elements of X and $F, G : X \to X$ are given maps.

*P*5 *Characteristic Maps of Finite Sets*: The map $\chi_A : X \to X$ defined by

$$\chi_A(x) = \begin{cases} a & \text{if } x \in A \\ b & \text{if } x \notin A \end{cases}$$

where $A \subseteq X$ is a finite set and $a, b \in X$ are fixed elements.

*P*6 *Concatenation-like Maps*: Let $X = \Sigma^*$ for an alphabet Σ and $\sigma \in \Sigma$. Then the maps $L_\sigma, R_\sigma : X \to X$ defined by $L_\sigma(u, \sigma v) = (u\sigma, v)$ and $R_\sigma(u\sigma, v) = (u, \sigma v)$.

Program maps are obtained from primitive program maps recursively by applying the following rules $(R1 - R5)$:

*R*1 *Successor Like Maps*: For $X = \mathbb{N}$, if $\tau : X \to \mathbb{N}$ is a program map, then the map $\sigma : \mathbb{N} \to \mathbb{N}$ defined by $\sigma(x) = \tau(x) + 1$ is a program map.

*R*2 *Product Maps*: If $F_i : X_i \to X_i$, for $i = 1, 2, \ldots, n$ are program maps, $\sigma, \tau : \{1, 2, \ldots, n\} \to \{1, 2, \ldots, n\}$ are permutations, then the map $F : \Pi X_i \to \Pi X_i$ defined by $F(x_1, \ldots, x_n) = (F_{\sigma(1)}(x_{\tau(1)}), \ldots, F_{\sigma(n)}(x_{\tau(n)}))$ is a program map. In particular, the diagonal maps $F(x_1, \ldots, x_n) = (F_1(x_1), \ldots, F_n(x_n))$ are program maps.

*R*3 *Composition Maps*: If $F_i : X_i \to X_i$ and $G : \Pi X_i \to \Pi X_i$ for $i = 1, 2, \ldots, n$ are program maps, then the map $F : \Pi X_i \to \Pi X_i$ defined by $F(x_1, \ldots, x_n) = G(F_1(x_1), \ldots, F_n(x_n))$ is a program map.

*R*4 *Conditional Maps*: Let $X_i \subseteq X$, $i = 1, 2, \ldots, n$ be pairwise disjoint sets and $F_i : X_i \to X$ be program maps. Then the map $F : \cup_{i=1}^n X_i \to X$ defined by $F(x) = F_i(x)$ if $x \in X_i$, is a progam map.

We write such an F as $F = (X_1, F_1)|(X_2, F_2)| \cdots |(X_n, F_n)$

*R*5 *Limit Maps*: If $F : X \to X$ is a program map with a terminating trace for each x in the domain of F, then the limit map $F^\infty : X \to X$ is a program map.

We remark that program maps are, in general, partial functions; thus restriction maps are assumed to be program maps. On the other hand, if we restrict program maps to be total functions, then we would have to add restriction maps as primitive program maps or have a rule for achieving this. Along with program maps, we also

specify the kinds of input and output maps. Though they can be any computable functions, we fix them by keeping an eye on their use. The state space X is viewed as a Cartesian product of other sets with S and as well with T. It would thus be easier formally to take a unified view by fixing X as a product of other finite number of sets with $S \times T$. However, in most circumstances S and T coincide and in those interesting cases, it would be wasteful to work with $S \times T$ rather than with S (or T). Moreover, we see that it is enough to take the other sets in the product as countable sets. We thus take

$$X = X_1 \times X_2 \times \cdots \times X_m \times \overline{S} \times Y_1 \times Y_2 \times \cdots \times Y_n$$

where the nonempty sets X_i and Y_j are countable, and the set $\overline{S}$ is either $S \times T$ or S (or T); the latter is taken when both $S = T$. Notice that this includes the special case $X = S = T$ permitting $m = n = 0$.

Input and output maps are identity-like maps or inverse projection maps, also called as cannonical embeddings. Formally,

Input Maps: If $X = X_1 \times \cdots \times X_m \times S \times Y_1 \times \cdots \times Y_n$, then $\rho : S \to X$ defined by $\rho(s) = (x_1, \ldots, x_m, s, y_1, \ldots, y_n)$ is an input map for some fixed elements $x_i \in X_i, y_j \in Y_j$.
Output Maps: If $X = X_1 \times \cdots \times X_m \times T \times Y_1 \times \cdots \times Y_n$, then

$$\pi(x_1, \ldots, x_m, t, y_1, \ldots, y_n) = t$$

is an output map for some fixed elements $x_i \in X_i, y_j \in Y_j$.

The input map ρ is a total function whereas the output map π is a partial function (as in Example 2.1).

4. Mapcode Computability

We show that the notions of computability for both the formalisms of Turing machines and mapcodes coincide. For this purpose, we present a formalization of Turing computability very briefly; for example, see [1].

A Turing machine is a six-tuple $M = (Q, \Sigma, \Gamma, \delta, s, h)$, where Q is the set of states not containing the halt state h, Σ is the input alphabet not containing the blank symbol ƀ, $\Gamma \supseteq \Sigma \cup \{\text{ƀ}, \text{ʟ}, \text{ʀ}\}$ is the tape alphabet with ʟ as the left movement and ʀ as the right movement, and $s \in Q$ is the initial state.

We assume that M is a standard single tape machine extended upto infinity on both left and right, with the transition function $\delta : Q \times \Gamma \to Q \cup \{h\} \times \Gamma$ as a partial function. This means that M, at any instant, can either rewrite the scanned symbol, or move one square to the left, or move one square to the right. An instantaneous description or configuration of M is written as a quadruple $(q, x, \sigma, y) \in Q \times \Gamma^* \times \Gamma \times \Gamma^*$ where M has the current state as q, scanning the symbol σ, to the left of which is the string x, and to the right of which is the string y. We will write the empty string as ϵ. Acceptance of a string is triggered by the fact that by taking a string as an input, it eventually goes to the halt state h. Computation of functions $f : \Sigma^* \to \Sigma_1^*$ are defined for $\Sigma_1 \subseteq \Gamma$ not containg ƀ by requiring that the machine upon taking the input x eventually enters the halt state leaving output as $f(x)$. We refrain from

giving details as to the tape appearances on inputs and outputs. It suffices to mention that while computing the partial function f, the initial configuration of the machine with input x is $(s, \epsilon, \text{b}, x)$ and the final configuration with output $f(x)$ is $(h, \epsilon, \text{b}, f(x))$.

For simulating M that computes a function $f : \Sigma^* \to \Sigma_1^*$ step-by-step, we define a corresponding mapcode $\Omega = (X, S, T, F, \rho, \pi)$ where $S = \Sigma^*, T = \Sigma_1^*, X = Q \cup \{h\} \times \Gamma^* \times \Gamma \times \Gamma^*, \rho : S \to X, \pi : X \to \Sigma_1^*$ with $\rho(u) = (s, \epsilon, \text{b}, u)$ and $\pi(h, \epsilon, \text{b}, v) = v$ when $v \in \Sigma_1^*$. If $v \notin \Sigma_1^*$, we leave π undefined. Finally, define $F : X \to X$ by

1. $F(p, \epsilon, \text{b}, \sigma u) = (q, \epsilon, \sigma, u)$ if $\delta(p, \text{b}) = (q, \text{R})$
2. $F(p, u, \sigma, v) = (q, u, \tau, v)$ if $\delta(p, \sigma) = (q, \tau)$
3. $F(p, u\tau, \sigma, v) = (q, u, \tau, \sigma v)$ if $\delta(p, \sigma) = (q, \text{L})$
   In particular, $F(p, u\sigma, \text{b}, \epsilon) = (q, u, \sigma, \epsilon)$.
4. $F(p, u, \sigma, \tau v) = (q, u\sigma, \tau, v)$ if $\delta(p, \sigma) = (q, \text{R})$
   In particular, $F(p, u, \sigma, \epsilon) = (q, u\sigma, \text{b}, \epsilon)$.
5. $F(h, u, \sigma, v) = (h, u, \sigma, v)$
6. $F(p, u, \sigma, v)$ is not defined if $\delta(p, \sigma)$ is undefined.

Obviously, the partial function F as defined above is a program map. Further, the program map F simulates one step of computation of M as it is simply a formalization of the notion of one step of a computation of M in terms of its configurations. We note it down as

Lemma 4.1 For each pair of configurations x and y of M, x yields in one step y iff $F(x) = y$.

Moreover, when M stops computing in a halted configuration, the configuration is an element $x \in X$ for which $F(x) = x$. That is, each halted configuration is an element of $fix\, F$. For those configurations that yield no configurations but are not themselves halted configurations, the program map F is undefined. Thus the requirement (b) of program correctness is met by the very definitions of F, ρ and π. It remains to see the termination criterion, the requirement (a).

Lemma 4.2 Let $x \in S$. Then M halts on input x iff $\rho(x) \in con\, F$.

Proof: Suppose M halts on x. The configuration $(s, \epsilon, \text{b}, x)$ yields in m steps the configuration (h, v, σ, w) for some $m \in \mathbb{N}$, $v, w \in \Gamma^*$ and $\sigma \in \Gamma$. By Lemma 4.1,

$$F^m(s, \epsilon, \text{b}, x) = (h, v, \sigma, w) \text{ and } F(h, v, \sigma, w) = (h, v, \sigma, w)$$

Thus,

$$F^{m+1}(s, \epsilon, \text{b}, x) = F^m(s, \epsilon, \text{b}, x) \text{ and } F^\infty(s, \epsilon, \text{b}, x) = (h, v, \sigma, w)$$

Conversely, suppose $(s, \epsilon, \text{b}, x) \in con\, F$. Then

$$F^{m+1}(s, \epsilon, \text{b}, x) = F^m(s, \epsilon, \text{b}, x) \text{ for some } m \in \mathbb{N}$$

Since $F(z) = z$ can only happen for $z = (h, u, \sigma, v)$ for some $u, v \in \Gamma^*$ and $\sigma \in \Gamma$, we have

$$F^m(s, \epsilon, \text{ƀ}, x) = (h, u, \sigma, v)$$

Moreover, $F(y) = z$ iff the configuration y yields in one step the configuration z. Thus, M has a computation which shows that $(s, \epsilon, \text{ƀ}, x)$ yields (h, u, σ, v). That is, M halts on input x.

Lemma 4.3. M computes $f : S \to T$ iff Ω computes f.

Proof: If M computes f, then $(s, \epsilon, \text{ƀ}, x)$ yields in m steps the configuration $(h, \epsilon, \text{ƀ}, f(x))$ for some $m \in \mathbb{N}$. As F simulates M step-by-step, we have $F^m(s, \epsilon, \text{ƀ}, x) = (h, \epsilon, \text{ƀ}, f(x))$. Then

$$F^{m+1}(s, \epsilon, \text{ƀ}, x) = F(h, \epsilon, \text{ƀ}, f(x)) = (h, \epsilon, \text{ƀ}, f(x)) = F^m(s, \epsilon, \text{ƀ}, x)$$

That is, $F^\infty(s, \epsilon, \text{ƀ}, x) = (h, \epsilon, \text{ƀ}, f(x))$ and

$$\pi(F^m(\rho(x))) = \pi(F^\infty(s, \epsilon, \text{ƀ}, x)) = \pi(h, \epsilon, \text{ƀ}, f(x)) = f(x)$$

Therefore, Ω computes f.

Conversely, if Ω computes f, then for some $m \in \mathbb{N}$, $F^m(s, \epsilon, \text{ƀ}, x) = F^{m+1}(s, \epsilon, \text{ƀ}, x)$. Moreover, $F^m(s, \epsilon, \text{ƀ}, x)$ must be of the form $(h, \epsilon, \text{ƀ}, f(x))$. Thus, $(s, \epsilon, \text{ƀ}, x)$ yields in m steps the configuration $(h, \epsilon, \text{ƀ}, f(x))$ of the machine M, i.e., M computes f.

Lemma 4.4 Every mapcode computable function is Truring computable.

Proof: We observe that each primitive program map is Turing computable; so are the maps in rules $R1 - R5$. To see, for example, the case of the limit map, let $x \in S$ be an input. If the program map F is Turing computale, let M be a machine that computes it. We design a two-tape machine M' that copies the string on its first tape to the second, simulates M on the first tape, compares the two tapes, and if they match, then it halts, else, it repeats the steps iteratively. It is now obvious that whenever the limit map is well defined, M' halts with $F^\infty(x)$ as its output on the first tape. Therefore, the limit map is Turing computable whenever the program map is. The input maps ρ and output maps π are clearly Turing computable. Since composition of maps is Turing computable, $f = \pi \circ F^\infty \circ \rho$ is Turing computable.

Lemmas 4.1-4.4 prove the following statement:

Theorem 4.1 A partial function is Turing computable iff it is mapcode computable.

5. Conclusions

The results in the last section show the equivalence of Turing computability and mapcode computability. Moreover, a mapcode seems to be a more natural formalization of programs. In a sense, mapcodes unify the abstract notion of computability and the concrete notion of a program. The programs specified by mapcodes can be stated and proved as theorems similar to any branch of mathematics without going through the heavy symbolic formalism of first order logic. For lack of space this point has not been illustarted in this paper. All that one needs there is to show that program maps terminate with finite trace for each possible input and the correct

output is then obtained. Thus teaching of programming become ameneable to any student of mathematics with a minimal knowledge of set theory [3].

Computable sets have also not been discussed in this paper. In fact, similar to the Turing machines, acceptable and decidable languages can be defined for mapcodes. To give a hint, let $\Omega = (X, \Sigma^*, \{0,1\}, F, \rho, \pi)$ be a mapcode, where Σ is an alphabet. Then the language accpeted by Ω can be defined as

$$\mathcal{L}(\Omega) = \{x \in \Sigma^* : \rho(x) \in conF\}$$

Similarly, the language decided by Ω can be defined as

$$\mathcal{L}_D(\Omega) = \{x \in \Sigma^* : \pi(F^\infty(\rho(x))) = 1\}$$

In such a case, we would require the condition that for every $x \in \Sigma^*$, either $\pi(F^\infty(\rho(x))) = 0$ or 1. The set $\{0,1\}$ is taken as the output set whenever we have a decision problem in focus.

Though program maps have been defined recursively it is not yet clear which maps may be considered as program maps. For example, maps from $\mathbb{N}^n$ to $\mathbb{N}$ that are piecewise affine can be shown to be program maps. Note that an affine map is a map $f : \mathbb{N}^n \to \mathbb{N}$ of the form $f(m_1, \ldots, m_n) = k_1 m_1 + \cdots + k_n m_n + p$ for some fixed natural numbers k_i and p. To see that piecewise affine maps are indeed program maps, one uses a Lyapunov function $\lambda : \mathbb{N} \to [0, \infty)$.

In general, we say that $\lambda : X \to [0, \infty)$ is a Lyapunov function for $F : X \to X$ if $F(x) \neq x$ implies the existence of $k \in \mathbb{Z}_+$ such that $\lambda(F^k(x)) \leq \lambda(x) - 1$. Existence of a Lyapunov function for F guarentees termination, i.e., the existence of the limit map F^∞ with $con\, F$ being nonempty as the following theorem shows.

Theorem 5.1 Let $\Omega = (X, S, T, F, \rho, \pi)$ be a mapcode. If a Lyapunov function λ exists for the program map F, then $dom\, F \subseteq con\, F$.

Proof: Let $\lambda : X \to [0, \infty)$ be a Lyapunov function for F. We want to show that each $x \in domF$ is eventually fixed by F. On the contrary, suppose for some $x \in domF$ and for each $n \in \mathbb{N}$, $F^n(x) \neq F^{n+1}(x)$. We define, inductively, a sequence $\{x_m\} \subseteq orb_F\, x$ satisfying the property that $F(x_m) \neq x_m$.

Take $x_0 = x$. Since for each n, $F^n(x) \neq F^{n+1}(x)$, we see that $F(x_0) \neq x_0$. Suppose x_j has already been defined satisfying $F(x_j) \neq x_j$. As λ is a Lyapunov function for F, there is a $k_j \in \mathbb{Z}_+$ such that $\lambda(F^{k_j}(x_j)) \leq \lambda(x_j) - 1$. Take $x_{j+1} = F^{k_j}(x_j)$.

Clerely, $x_m \in orb_F\, x$ for each m. In particular, $x_j = F^{n_j}(x_0)$ for some $n_j \in \mathbb{N}$. Then $x_{j+1} = F^{k_j + n_j}(x_0)$. Now, if $F(x_{j+1}) = x_{j+1}$, then $F^{k_j+n_j+1}(x_0) = F^{k_j+n_j}(x_0)$ contradicting our assumption that $F^n(x) \neq F^{n+1}(x)$. Therefore, $F(x_{j+1}) \neq x_{j+1}$, i.e., the sequence $\{x_m\} \subseteq orb_F\, x$ is well defined and that $F(x_m) \neq x_m$ for each $m \in \mathbb{N}$.

Since $\lambda : X \to [0, \infty)$, there exists an $i \in \mathbb{N}$ such that $\lambda(x_0) \leq i$. Moreover,

$$\lambda(x_{i+1}) \leq \lambda(x_i) - 1 \leq \lambda(x_{i-1}) - 2 \leq \cdots \leq \lambda(x_0) - i - 1 < 0$$

This contradiction shows that for each $x \in dom\, F$, there exists an $n \in \mathbb{N}$ such that $F^n(x) = F^{n+1}(x)$. That is, $dom\, F \subseteq con\, F$.

However, all program maps from $\mathbb{N}^n$ to $\mathbb{N}^n$ (even from $\mathbb{N}$ to $\mathbb{N}$) need not be of this nature. There might be program maps with a terminating trace for each element of the state space though a Lyapunov function may not exist for it. The question is whether this possibility is an actuality. That is, does every program map need to have a Lyapunov function in order to terminate, even when we restrict the state space to $\mathbb{N}$? We conclude the paper with this question.

Acknowledgements: The author thankfully acknowledges the efforts put forth by Professor K. Viswanath of University of Hyderabad, India, in convincing him regarding the usefulness of mapcodes for teaching computer science to students of mathematics.

References

[1] Kozen, D.C., *Automata and Computability*, Springer Undergraduate Texts in Computer Science, Springer-Verlag, New York, 1997.

[2] Knuth D.E., *Fundamental Algorithms: The Art of Computer Programming, Vol.I*, Addison Wesley Pub.Co., 1968.

[3] K.Viswanath, Dynamical Systems and Computer Science, *Mathematics News Letter of the Ramanujan Mathematical Society*, **1(3)**, (2003) 33-48.