

# Computing with Multiple Discrete Flows\*

To appear in the journal *Differential Equations and Dynamical Systems*.<sup>†</sup>

K. Venkata Rao  
Government College (A)  
Rajahmundry, A.P., India

Kasturi Viswanath<sup>§</sup>  
International Institute of Information Technology, Hyderabad, India

November, 2010

## Abstract

A set  $X$  together with a finite collection of self-maps on  $X$  is called a multiple discrete flow. We develop the theory of fair convergence for multiple flows and show how the UNITY programs of Chandy and Misra can be reformulated in terms of this theory.

## 1 Introduction

In recent years a viewpoint has emerged that views all dynamical evolution as computation and information processing. Thus, instead of saying that the earth moves around the sun obeying the laws of gravitation, one says that the earth senses its environment, processes the information, and computes its way around the sun. In his book *Programming the Universe*, Seth Lloyd [6] says:

*My vision of the world as processing information arose out of my day-to-day work building quantum computers. Since I made this realization, my own vision of the world has changed and evolved. The more*

---

\* A preliminary version of this article was presented at the conference “Managing Complexity in a Distributed World” at the Indian Institute of Science, May 27-31, 2008. Proceedings of the conference were not published.

<sup>†</sup> Key words: mapcode, multiflow, UNITY, nondeterminism, Floyd-Warshall.

<sup>‡</sup> AMS2010 Classification: Primary 68Q10, Secondary 37B99, 68R99.

<sup>§</sup> Financial support by Tata Consultancy Services Ltd is gratefully acknowledged.

*information I process in my own thoughts, the more convinced I am that the theory of the computational universe is the right theory.*

Again, Susan Stepney [9] studies models of computation inspired not only by quantum theory but also biology and the concept of emergent behavior that has resulted from complexity theory. She emphasizes the difference between a computational model, which is a mathematical model, and a computational device, which is a physically or biologically realized entity. She lists several exciting possibilities for research.

This body of work seeks to link real life dynamical evolution with computation by devising new and non-standard models of computing. There are also attempts, not so well-known, to express the now classical theory of computation and algorithmics in terms of discrete dynamical models.

For example, McCarthy [7] observes, with a slight change in notation, “when a block of a program having a single entrance and exit is executed, the effect of the execution on the state vector may be described by a function  $x' = F(x)$  giving the new state vector  $x'$  in terms of the old state vector  $x$ .” In effect he is saying that, if  $X$  denotes the set of all state vectors, then a program may be modeled by a map  $F : X \rightarrow X$ .

Knuth [5] says the same thing explicitly, when he observes that a mathematical theory of algorithms can be built by modeling an algorithm as a map  $F : X \rightarrow X$  together with some additional structure. Given the map  $F$ , computation proceeds by repeatedly changing the state  $x$  to  $F(x)$  till the state does not change any more. This suggestion has been followed up in the work of Viswanath [12, 13] and developed into a full-fledged mathematical introduction to large parts of algorithmics. We shall refer to this work in this article as the ‘mapcode model’, the name given to it in [13].

The mapcode model is deterministic. This means that, given the current state  $x$  the next state  $x'$  can only be  $F(x)$ . However, not all computing needs to be deterministic. The theory of computation [8] studies non-deterministic models of computation and Dijkstra [4] has worked with non-deterministic programs. The question then arises if we can model these ideas also using appropriate generalizations of discrete dynamical systems.

This can be done in two different ways. One way is to replace the state to state map  $F : X \rightarrow X$  with a state to set-of-states map  $\Delta : X \rightarrow \mathcal{P}(X)$ , where  $\mathcal{P}(X)$  denotes the set of all subsets of  $X$ . If the current state in the course of a computation is  $x$  then the next state can be any of the states in the set  $\Delta(x)$ . Thus the current state does not uniquely determine the next state. It has been shown in [11] that this model, with certain additional conditions, gives rise to a theory that is equivalent to that of Dijkstra [4].

There is another way of modeling nondeterminism. That is to replace the single map  $F : X \rightarrow X$  with multiple maps  $F_1, F_2, \dots, F_N$ , all mapping  $X$  to  $X$  and iterating each of them. This models nondeterminism by requiring that if the current state is  $x$  then the next state is got by applying any of the maps  $F_1$  to  $F_N$  to  $x$ .

These two models are not equivalent. One reason is that  $\Delta(x)$  may be the empty set  $\emptyset$  for some  $x$ . Or it could be an infinite set. If  $\Delta(x)$  is non-empty for all  $x$  and the number of elements in  $\Delta(x)$  is bounded above independently of  $x$  then one can construct multiple maps  $F_p, 1 \leq p \leq N$ , so that choosing one of the elements of  $\Delta(x)$  is equivalent to applying one of the maps  $F_p$ . Even in this case, though the two models coincide, the two theories differ in their treatment of convergence. The theory of convergence that naturally arises in the case of the  $\Delta$  model, as developed in [11], is rather different from the convergence theory that suggests itself in the multiple flow case developed here.

These models, together with many variations and interpretations of them, are discussed in the articles of Walicki and Meldal [14] and Armoni and Ben-Ari [2]. While both the articles attempt a comprehensive discussion of nondeterminism in computing, each from its own view-point, it is surprising that neither of them mentions the work of Chandy and Misra [3] even though their book was published 1988.

In their treatise on parallel programming design [3] Chandy and Misra introduce a method of nondeterministic programming using iterated transformations  $\{F_1, \dots, F_N\}$ . They call it **UNITY** - an acronym for **U**nbounded **N**ondeterministic **I**terative **T**ransofmation theor**Y**.

They present a philosophy and methodology of programming that enables them to start with mathematical statements of specification and allows them to progressively refine them to suit given implementation requirements.

As is the practice in computer science **UNITY** is presented in terms of well-formed statements in a specified grammar and the logic is in terms of inference rules. Besides the standard Hoare rules the authors derive some more rules that feature new relations between predicates called ‘unless’, ‘ensures’ and ‘leads-to’. Further the progress of the computation is expressed in terms of infinite execution sequences.

They say in the Preface to their book that courses based on the book are presented to graduate students. We feel that their approach to programming is very illuminating and should be taught more widely. The approach we present here using dynamical systems is at the level of a typical discrete mathematics course and hence should be accessible to many more students.

To show the viability of our model we take up four of the examples studied in [3] and prove them afresh. This makes accessible a whole new area of computation theory to dynamical systems theorists. We hope that this article will

inspire efforts to explore this area further and eventually result in a full fledged course to UNITY parallel programming that is less mathematically demanding.

We now give a brief overview of the structure of the paper. Section 2 models nondeterminism in terms of the maps  $F_1, F_2, \dots, F_N$ , called a multiflow, and the action of an induced monoid on the state space  $X$ . The concept of fair convergence is defined. The definition is distinct from that of [3] but is equivalent to it. Section 3 deals with the idea of computing with multiflows. Sections 4 and 5 address questions of safety and progress respectively. Section 6 presents four examples from the book [3] to make it easier for the reader to compare the two approaches.

In what follows  $\mathbb{N}$  and  $\mathbb{P}$  denote respectively the set of all non-negative integers and the set of all positive integers. The symbol  $\doteq$  may be read as ‘is defined to be’. For any map  $f : X \rightarrow Y$  and  $A \subseteq X$ ,  $f(A) \doteq \{f(x) \mid x \in A\}$ .

## 2 Multiflows and Fair Convergence

**Definition 2.1** Let  $\mathbf{F} = \{F_1, \dots, F_N\}$  be a finite ordered collection of (total) maps from  $X$  to  $X$ . The pair  $(X, \mathbf{F})$  is called a multiple discrete flow, or simply, a multiflow.

**Remark 2.2** If  $\mathbf{F}$  consists of a single map  $F$ , a multiflow reduces to a single discrete flow. In this case the pair  $(X, F)$  is a discrete dynamical system [12]. We also call this a unifold.  $\square$

In the definitions below we express the process of repeatedly applying the maps  $F_p$  on  $X$  in terms of the action of a monoid on  $X$  in the usual way.

**Definitions 2.3** 1. Let  $[1..N] = \{p \mid 1 \leq p \leq N\}$ . Each of the numbers in  $[1..N]$  is called a label.

2. For any positive number  $n$ , a string (finite sequence)  $\mathbf{p} = p_0p_1 \dots p_{n-1}$ , where the  $p_i$ 's range over  $[1..N]$ , is called a label string of length  $n$ . The set  $[1..N]^*$  denotes the set of all label strings over  $[1..N]$ .
3. Given arbitrary strings  $\mathbf{p} = p_0p_1 \dots p_{n-1}$  and  $\mathbf{q} = q_0q_1 \dots q_{m-1}$  in  $[1..N]^*$ ,  $\mathbf{p} \cdot \mathbf{q} \doteq p_0 \dots p_{n-1}q_0 \dots q_{m-1}$  and is called the concatenation of  $\mathbf{p}$  and  $\mathbf{q}$ . We write  $\mathbf{p}^2 = \mathbf{p} \cdot \mathbf{p}$  and more generally  $\mathbf{p}^n = \mathbf{p}^{n-1} \cdot \mathbf{p}$  for  $n \geq 2$ .
4.  $[1..N]^*$  is a monoid with the concatenation operation, the empty string  $\varepsilon$  being the unit.
5. A string  $\mathbf{p}$  is said to be fair if each of the labels from 1 to  $N$  occurs at least once in  $\mathbf{p}$ . If  $\mathbf{p}$  is fair then so are  $\mathbf{q} \cdot \mathbf{p}$  and  $\mathbf{p} \cdot \mathbf{q}$  for every  $\mathbf{q} \in [1..N]^*$ .
6. For  $m \geq 1$ , a string  $\mathbf{p}$  is said to be  $m$ -fair if there exist  $m$  fair strings  $\mathbf{p}_1, \dots, \mathbf{p}_m$  such that  $\mathbf{p} = \mathbf{p}_1 \cdot \dots \cdot \mathbf{p}_m$ . If  $\mathbf{p}$  is  $m$ -fair then so are  $\mathbf{q} \cdot \mathbf{p}$  and  $\mathbf{p} \cdot \mathbf{q}$  for  $\mathbf{q} \in [1..N]^*$ . If  $\mathbf{p}$  is fair then  $\mathbf{p}^m$  is  $m$ -fair. If  $\mathbf{p}$  is  $(m + 1)$ -fair then  $\mathbf{p}$  is also  $m$ -fair.  $\square$

**Remarks 2.4** 1. The first letter in UNITY [3] stands for ‘Unbounded’. The UNITY theory is unbounded because the execution is expressed in terms of the action of infinite sequences of labels on the state space. An infinite sequence of labels is said to be fair if each label occurs infinitely many times in the sequence. Here we have replaced the unbounded theory by the bounded theory of strings and introduced the notion of  $m$ -fairness for strings of labels.

2. Being  $m$ -fair is a stronger condition than requiring that the string must contain  $m$  occurrences of each label.  $\square$

Given  $(X, \mathbf{F})$  we define the action of the monoid  $[1..N]^*$  on  $X$ .

- Definitions 2.5** 1. If  $p$  is a label and  $x \in X$ ,  $x \cdot p \doteq F_p(x)$ . Inductively we define  $x \cdot \mathbf{p} = (x \cdot p_0) \cdot p_1p_2 \dots p_{n-1}$  for a label string  $\mathbf{p} = p_0p_1 \dots p_{n-1}$ . By convention  $x \cdot \varepsilon \doteq x$  for every  $x \in X$ . Clearly  $x \cdot (\mathbf{p} \cdot \mathbf{q}) = (x \cdot \mathbf{p}) \cdot \mathbf{q}$ .
2. If  $A \subseteq X$  and  $\mathbf{p} \in [1..N]^*$ ,  $A \cdot \mathbf{p} \doteq \{x \cdot \mathbf{p} \mid x \in A\}$ . Note that by definition  $\emptyset \cdot \mathbf{p} = \emptyset$ .

3.  $A \subseteq X$  is said to be invariant if  $A \cdot p \subseteq A$  for all labels  $p$ . If  $A$  is invariant,  $A \cdot \mathbf{p} \subseteq A$  for all  $\mathbf{p} \in [1..N]^*$ .
4. Given  $x \in X$ , a label string  $p_0p_1 \dots p_{n-1}$  induces a finite sequence of states called a run starting at  $x$ . This sequence is defined by  $x_0 = x$  and  $x_{i+1} = x_i \cdot p_i$  inductively for every  $0 \leq i \leq n-1$ . The run  $(x_0, x_1, \dots, x_n)$  starting at  $x$  is said to end at  $x_n$ . We note that  $x_1 = x \cdot p_0$ ,  $x_2 = x \cdot p_0p_1$ ,  $\dots$ ,  $x_n = x \cdot p_0p_1 \dots p_{n-1}$ .
5.  $fix(\mathbf{F}) \doteq \{x \mid x \cdot p = x, 1 \leq p \leq N\}$  and is called the set of fixed points of  $\mathbf{F}$ .  $fix(\mathbf{F})$  is an invariant set.
6. A state  $x \in X$  is said to be convergent if there exists a positive integer  $k$ , depending on  $x$ , such that  $x \cdot \mathbf{p} \in fix(\mathbf{F})$  for every  $k$ -fair string  $\mathbf{p} \in [1..N]^*$ . The set of all convergent states is denoted by  $con(\mathbf{F})$ .  $con(\mathbf{F})$  is an invariant set.
7. A round of computation at a state  $x$  means that we take a fair string  $\mathbf{p}$  and change the state  $x$  to  $x \cdot \mathbf{p}$ . So a state is convergent if and only if there is an integer  $k$  such that the state is changed to a fixed point in any  $k$  rounds of computation.
8. Let  $x$  be convergent,  $k$  being the associated integer. Define the limit set  $\Lambda(x)$  at  $x$  by  $\Lambda(x) = \{y \in fix(\mathbf{F}) \mid y = x \cdot \mathbf{p} \text{ for some } k\text{-fair } \mathbf{p}\}$ . If  $x$  is not convergent, define  $\Lambda(x) = \emptyset$ .  $\Lambda(x)$  is called the limit set at  $x$ .  $\square$

### 3 Computing with Multiflows

We are interested in programs that are meant to terminate after a finite amount of time and produce a specified output for a given input. More formally, let  $S$  be the space of legal inputs to a program. The purpose of the program is to produce a specified output  $f(s)$  in a set  $T$  for each given  $s \in S$ . In terms of computer science this map constitutes the specification that the program must satisfy.

**Definition 3.1** Let  $S$  and  $T$  be two sets called the set of inputs and the set of outputs respectively. A map  $f : S \rightarrow T$  is called a specification map.  $\square$

The definitions that follow generalize those for the case of a single flow  $(X, F)$  in [13].

**Definitions 3.2** A multiflow machine is a six-tuple  $\mathbf{M} = (S, X, T, \rho, \mathbf{F}, \pi)$  where

1.  $S$  is a set called the input space;
2.  $X$  is a set called the state space;
3.  $T$  is a set called the output space;
4.  $\rho : S \rightarrow X$  is a map called the input map;
5.  $\mathbf{F}$  is an ordered collection  $\{F_1, \dots, F_N\}$  of self-maps on  $X$ ;
6.  $\pi : X \rightarrow T$  is a map called the output map.  $\square$

**Remarks 3.3** 1. Note that a ‘multiflow’ is simply the pair  $(X, \mathbf{F})$  whereas a ‘multiflow machine’ is a six-tuple as above.

2. Even though we have defined above a multiflow machine formally as a six-tuple it is convenient to denote it by  $S \xrightarrow{\rho} X \xrightarrow{\mathbf{F}} X \xrightarrow{\pi} T$ .
3. When the collection  $\mathbf{F}$  consists only of a single map  $F$  a multiflow machine reduces to a mapcode machine defined in [13]. In this context we also call a mapcode machine a uniflow machine.  $\square$

**Definitions 3.4** Suppose  $\mathbf{M} = S \xrightarrow{\rho} X \xrightarrow{\mathbf{F}} X \xrightarrow{\pi} T$  is a multiflow machine.

1. The set  $\{s \mid \rho(s) \in \text{con}(\mathbf{F})\}$  is called the halting set  $\Omega$  of the program.
2. Let  $\mathcal{P}(T)$  denote the class of all subsets of  $T$ . The input-output map computed by the multiflow machine is the map  $\Phi : \Omega \rightarrow \mathcal{P}(T)$  defined by  $\Phi(s) = \pi(\Lambda(\rho(s)))$ .

3. The machine  $\mathbf{M}$  is said to compute the specification map  $f : S \rightarrow T$  if the halting set  $\Omega = S$  and  $\Phi(s) = \{f(s)\}$  for all  $s \in S$ .  $\square$

**Remark 3.5** The expressions ‘halting set’ and ‘input-output map’, and the symbols  $\Omega$  and  $\Phi$  for them, are borrowed from [?]. Our setup is more general than theirs.  $\square$

We now relate the nomenclature introduced above with that of the standard notation in the case of uniflow machines.

**Remark 3.6** Suppose  $\mathbf{F} = \{F\} = \{F_1\}$ . Then the associated monoid  $[1..N]^* = \{1^n \mid n \in \mathbb{N}\}$  where  $1^n$  stands for a string of  $n$  1’s.

1. We have  $x \cdot 1 = F(x)$  and  $x \cdot 1^n = F^n(x)$  for  $x \in X$ .
2.  $fix(F) = \{x \mid F(x) = x\} = fix(\mathbf{F})$ .
3.  $con(F) = \{x \mid F^n(x) \in fix(F) \text{ for some } n \geq 0\} = con(\mathbf{F})$ .
4. If  $x \in con(F)$  and  $F^n(x) \in fix(F)$ , then  $F^n(x) = F^{n+1}(x) = \dots = F^\infty(x)$ , say. The map  $F^\infty : con(F) \rightarrow fix(F)$  is called the limit map of  $F$ . We have  $\Lambda(x) = \{F^\infty(x)\}$ .
5. Consider the uniflow machine  $M = S \xrightarrow{\rho} X \xrightarrow{F} X \xrightarrow{\pi} T$ . Let  $\Omega = \{s \mid \rho(s) \in con(F)\}$ . Then  $\phi = \pi \circ F^\infty \circ \rho$  maps  $\Omega$  to  $T$  and is called the computed map or the input-output map. We have  $\Phi(x) = \{\phi(x)\}$ .
6. Consider a specification map  $f : S \rightarrow T$ . The machine  $M$  computes  $f$  if  $\Omega = S$  and  $\phi = f$  on  $S$ .  $\square$

The next theorem shows that the class of multiflow machines is not more powerful than the class of uniflow machines for computing specification maps.

**Theorem 3.7** *Let  $f : S \rightarrow T$ . If there exists multiflow machine computing  $f$  then there exists a uniflow machine computing  $f$ .*

**Proof** Suppose  $\mathbf{M} = S \xrightarrow{\rho} X \xrightarrow{\mathbf{F}} X \xrightarrow{\pi} T$  computes  $f$ . This implies that  $\rho(s) \in con(\mathbf{F})$  for all  $s$  and that there exists  $k = k(s)$  such that  $\rho(s) \cdot \mathbf{p} \in$

$fix(\mathbf{F})$  for all  $s$  and all  $k$ -fair strings  $\mathbf{p}$ . In addition we have  $\pi(\rho(s) \cdot \mathbf{p}) = f(s)$  for all  $s$ . In particular if  $\mathbf{q}$  is a fair string then for any  $s \in S$ ,  $\pi(\rho(s) \cdot \mathbf{q}^k) = f(s)$  for some  $k$ .

Fix a fair string  $\mathbf{q} = q_0q_1 \dots q_{n-1}$ . Define  $F : X \rightarrow X$  by  $F = F_{q_{n-1}} \circ \dots \circ F_{q_1} \circ F_{q_0}$ . Then for each  $s \in S$ , and for  $k = k(s) \in \mathbb{N}$ ,  $F^k(\rho(s)) \in fix(F)$  and  $\pi(F^k(\rho(s))) = f(s)$ . Hence the uniflow machine  $M = S \xrightarrow{\rho} X \xrightarrow{F} X \xrightarrow{\pi} T$  computes  $f$ .  $\square$

Despite this theorem there are advantages in studying multiflow machines. We spell out some of them.

1. It sometimes happens that the data of a problem naturally comes in the form of a multiflow  $(X, \mathbf{F})$ . If we arbitrarily choose a fair string as in the proof of Theorem 3.7 and reduce the multiflow to a uniflow, some of the information may be lost. For example, in the problem of scheduling a meeting discussed in Section 6.1, each  $F_i$  gives the data of one person and it is necessary to schedule a meeting when all of the  $N$  persons can be present. Suppose one of the persons drops out of reckoning and it is enough for the rest of them to meet. If we only had the uniflow machine that takes into account all the  $N$  people, we would have to retrace our steps and re-do the reduced problem.
2. A multiflow machine is reducible to a uniflow machine in more than one way and different ways of reducing it may give rise to different algorithms and some of them may be more efficient than others. Also, the reduction suggested by Theorem 3.7 may not be the best way in certain circumstances. For example, in the sorting problem discussed in Section 6.3, the multiflow machine is in some sense a seed algorithm and many of the standard sorting algorithms may be obtained from it by reduction in different ways.
3. Suppose the state  $x$  is given by a tuple  $(x_1, x_2, \dots, x_N)$  and  $F_i$  acts only

on  $x_i$ . Then there is the possibility of defining a new map that acts simultaneously on all the components of  $x$ . That is to say instead of applying each of the  $F_i$  serially we can apply all of them in a parallel fashion at the same time, thus speeding up the algorithm.

4. It is frequently easy to think of nondeterministic algorithms. So we may start with one, define a multifold, and refine it step by step to yield a uniflow algorithm.

## 4 Safety Theorems

Programs are studied in terms of safety and progress properties. A safety theorem guarantees that if the program terminates then the output it produces is the correct one. A progress theorem guarantees that as the program keeps acting on an input the state progresses towards termination so that eventually the program terminates.

The formalism of multiflows is very flexible and safety and progress theorems may be devised for each program in a way that suits the context. However, it is illuminating to study the proof methods in a general setting. They help us understand the structure of the action of multiflows. In this section we study safety theorems and take up progress theorems in the next.

We had defined invariant sets earlier in Definitions 2.3. We now define invariant functions.

**Definition 4.1** A map  $\theta : X \rightarrow Y$ , where  $Y$  is any set, is said to be invariant if  $\theta(x \cdot p) = \theta(x)$  for all  $x \in X$  and all  $p \in [1..N]$ . In this case  $\theta(x \cdot \mathbf{p}) = \theta(x)$  for all  $\mathbf{p} \in [1..N]^*$ .  $\square$

**Example 4.2** Given  $(X, \mathbf{F})$ , the limit map  $\Lambda : X \rightarrow \mathcal{P}(fix(\mathbf{F}))$  is an invariant map. Also, if  $\theta$  is invariant, then for every  $x \in con(\mathbf{F})$  and  $y \in \Lambda(x)$  it is true that  $\theta(x) = \theta(y)$ .  $\square$

**Theorem 4.3** Suppose  $S \xrightarrow{\rho} X \xrightarrow{\mathbf{F}} X \xrightarrow{\pi} T$  is a multiframe machine  $M$  and  $f : S \rightarrow T$  is a specification map. Let  $\theta : X \rightarrow T$  be invariant and let  $s \in \Omega$ . If  $\theta(\rho(s)) = f(s)$  and  $\theta(x) = \pi(x)$  for  $x \in \text{fix}(\mathbf{F})$ , then  $\Phi(s) = \{f(s)\}$ .

**Proof** Since  $\rho(s)$  is convergent there exists  $m > 0$  such that

$$\Lambda(\rho(s)) = \{\rho(s) \cdot \mathbf{p} \mid \text{for all } m\text{-fair } \mathbf{p}\}.$$

Using the invariance of  $\theta$  and the two given conditions we see that

$$\Phi(s) = \pi(\Lambda(\rho(s))) = \theta(\Lambda(\rho(s))) = \{\theta(\rho(s))\} = \{f(s)\}. \square$$

We now prove another safety theorem using invariant sets.

**Definition 4.4** Suppose  $S \xrightarrow{\rho} X \xrightarrow{\mathbf{F}} X \xrightarrow{\pi} T$  is a multiframe machine  $M$  and  $f : S \rightarrow T$  is a specification map. An invariant set  $A \subseteq X$  is said to be safe for  $f$  at  $s \in S$  if  $\rho(s) \in A$  and  $\pi(A \cap \text{fix}(\mathbf{F})) = \{f(s)\}$ .  $\square$

**Theorem 4.5** If there exists a safe set  $A$  for  $f$  at  $s \in \Omega$  then  $\Phi(s) = \{f(s)\}$ .

**Proof** Let  $A$  be a safe set for  $f$  at  $s \in \Omega$ . Since  $A$  is invariant, and  $\rho(s) \in A$  we have  $\emptyset \neq \Lambda(\rho(s)) \subseteq A \cap \text{fix}(\mathbf{F})$ . Then  $\emptyset \neq \Phi(s) = \pi(\Lambda(\rho(s))) \subseteq \pi(A \cap \text{fix}(\mathbf{F})) = \{f(s)\}$ . This proves that  $\Phi(s) = \{f(s)\}$ .  $\square$

**Corollary 4.6** If  $A \subseteq \text{con}(\mathbf{F})$  is safe for  $f$  at  $s \in S$  then  $s \in \Omega$  and  $\Phi(s) = \{f(s)\}$ .

## 5 Progress Theorems

To prove that a state  $x$  is convergent we need to have some way of showing that as the labels keep acting on the state, the state progresses towards a fixed point. The concept of a bound function that is useful in the case of uniflows [13] may be extended to be serviceable in the present case also.

**Definition 5.1** A map  $\lambda : X \rightarrow \mathbb{N}$  is called a height function.

**Definition 5.2** Let  $(X, F)$  denote a uniflow and  $(X, \mathbf{F})$  a multiflow. Let  $\lambda$  be a height function on  $X$ .

1.  $\lambda$  is called a bound function for  $F$  if for all  $x \in X$  either  $x \in \text{fix}(F)$  or  $\lambda(F(x)) < \lambda(x)$ .
2.  $\lambda$  is called a bound function for  $\mathbf{F}$  if  $\lambda$  is a bound function for  $F_p$  for all  $p$ .  $\square$

**Remarks 5.3** 1. If  $\lambda$  is a bound function for  $\mathbf{F}$  then  $\lambda(F_p(x)) \leq \lambda(x)$  for all  $x$  and all  $p$ .

2. If  $\lambda$  is a bound function and  $\lambda(x) = 0$  then  $x \in \text{fix}(\mathbf{F})$ .
3. If  $\lambda$  is a bound function so is  $\lambda + c$  where  $c$  is a positive integer. So  $\lambda$  need not be 0 on fixed points.  $\square$

One uses bound functions to prove progress theorems by showing that their values decrease under the action of fair strings.

**Theorem 5.4** Let  $\lambda$  be a bound function for  $(X, \mathbf{F})$ . Then for every  $x$  and every fair string  $\mathbf{p}$ , either  $x \in \text{fix}(\mathbf{F})$  or  $\lambda(x \cdot \mathbf{p}) < \lambda(x)$ . In other words, if  $x$  is not a fixed point for the flow then  $\lambda(x)$  strictly decreases after a round of computation.

**Proof** Take any  $x \in X$  and a fair string  $\mathbf{p} = p_0 p_1 \cdots p_{n-1}$ . Suppose  $x \notin \text{fix}(\mathbf{F})$ . Then there exists a least label  $i$  such that  $x \notin \text{fix}(F_{p_i})$ . (Otherwise  $x$  would be a fixed point for the multiflow.) So  $x \cdot p_0 p_1 \cdots p_{i-1} = x \notin \text{fix}(F_{p_i})$ . Consequently  $\lambda(x \cdot p_0 \cdots p_{n-1}) \leq \lambda(x \cdot p_0 \cdots p_i) = \lambda(x \cdot p_i) < \lambda(x)$ , by the definition of a bound function.  $\square$

Bound functions are not always available to prove progress. In some cases, as in the division example below, we appeal to the notion of a weak bound function.

**Definition 5.5** Let  $m$  be a positive integer and  $(X, \mathbf{F})$  a multiflow. A height function  $\lambda$  is called a weak bound function of order  $m$  for  $\mathbf{F}$  if for every  $x \in X$

1.  $\lambda(F_p(x)) \leq \lambda(x)$  for all  $p$ ;
2. for every  $m$ -fair string  $\mathbf{p}$  either  $x \cdot \mathbf{p} \in \text{fix}(\mathbf{F})$  or  $\lambda(x \cdot \mathbf{p}) < \lambda(x)$ .  $\square$

**Remark 5.6** 1. A bound function is a weak bound function of order 1.

2. If  $\lambda$  is a weak bound function of order  $m$  for  $\mathbf{F}$  and  $n > m$  then  $\lambda$  is also a weak bound function of order  $n$ .  $\square$

**Theorem 5.7** *If there is a weak bound function for  $\mathbf{F}$  then  $\text{con}(\mathbf{F}) = X$ .*

**Proof** Suppose  $\lambda$  is a weak bound function of order  $m$  for  $\mathbf{F}$ . Fix  $x \in X$ . If  $\lambda(x) = 0$  then for every  $m$ -fair string  $\mathbf{p}$  either  $x \cdot \mathbf{p} \in \text{fix}(\mathbf{F})$  or  $\lambda(x \cdot \mathbf{p}) \leq -1$ . Since the later is impossible,  $x \cdot \mathbf{p} \in \text{fix}(\mathbf{F})$  for every  $m$ -fair string  $\mathbf{p}$ . This proves that  $x \in \text{con}(\mathbf{F})$ .

Let  $\lambda(x) = k > 0$ . Then for every  $m$ -fair string  $\mathbf{p}$  either  $x \cdot \mathbf{p} \in \text{fix}(\mathbf{F})$  or  $\lambda(x \cdot \mathbf{p}) \leq k - 1$ . Repeating the argument, for every  $2m$ -fair string either  $x \cdot \mathbf{p} \in \text{fix}(\mathbf{F})$  or  $\lambda(x \cdot \mathbf{p}) \leq k - 2$ . So in at most  $k$  steps  $\lambda$  is reduced to 0. By the earlier argument  $x$  is moved to a fixed point. Hence every  $x \in X$  is convergent.  $\square$

There are situations where a single bound function is not available and we have to prove convergence in stages, such as in the sorting example below. The discussion below expresses abstractly the nature of the reasoning.

**Definition 5.8** Given  $(X, \mathbf{F})$  suppose  $A$  and  $B$  are invariant sets such that  $A \supseteq B$ . A height function  $\lambda$  is said to slope to  $B$  on  $A$  if for any  $x \in A$  and any fair string  $\mathbf{p}$  either  $x \in B$  or  $\lambda(x \cdot \mathbf{p}) < \lambda(x)$ .  $\square$

**Remark 5.9** If  $\lambda$  is a bound function for  $\mathbf{F}$  then  $\lambda$  slopes to  $\text{fix}(\mathbf{F})$  on  $X$ .  $\square$

**Lemma 5.10** *Let the notation be as above and suppose  $\lambda$  slopes to  $B$  on  $A$ . If  $x \in A$  and  $\lambda(x) = r$ , then  $x \cdot \mathbf{p} \in B$  for any  $r$ -fair string  $\mathbf{p}$ .*

**Proof** Let  $\mathbf{p} = \mathbf{p}_0\mathbf{p}_1 \dots \mathbf{p}_{r-1}$  where each  $\mathbf{p}_i$  is a fair string. Let  $x_0 = x$  and  $x_i = x \cdot \mathbf{p}_0\mathbf{p}_1 \dots \mathbf{p}_{i-1}$ ,  $i \geq 1$ , so that  $x_r = x_{r-1} \cdot \mathbf{p}_{r-1}$ . Then either  $x = x_0 \in B$  or  $\lambda(x_1) \leq r - 1$ . If  $x \in B$ , since  $B$  is invariant, so does  $x_r$ . If  $x_1 \notin B$  then  $\lambda(x_2) \leq r - 2$ . If this argument continues to the end we see that  $\lambda(x_r) = 0$  and either  $x_r \in B$  or  $\lambda(x_r \cdot \mathbf{q}) \leq -1$  for any fair string  $\mathbf{q}$ . Since the latter is impossible,  $x_r \in B$ .  $\square$

## 6 Examples

The four examples in this section have all been discussed by Chandy and Misra [3]. We choose the very same examples here to illustrate the differences in reasoning methods. The reader is invited to compare the two approaches. Constraints of space prevent us from making a detailed comparison here.

It is worth remarking here that the complexity of a computational problem arises not from the domain where the specification map is defined but from the constraints imposed on how it is to be computed. So, even in such a simple problem as that of finding quotients and remainders, one has to resort to some subtle reasoning to establish safety and progress.

### 6.1 Scheduling a Meeting

There are  $N$  managers in an organization who need to find the earliest date after a given date on which all of them are free to meet. How is the secretary to fix that date?

A simple method is to call each of them one by one. If the first manager gives the earliest date as  $D_1$ , then the second is asked for the first day on or after  $D_1$  when the meeting can take place. In this way all the managers are asked again and again till a date is found. In principle it is possible that no such date can ever be found. So in the formulation of the problem we assume

that given a date  $s$  a common date  $z_s$  exists on or after the date  $s$  and we need to find the first such date. It is intuitively reasonable that if each manager is asked sufficiently many times then a common date can be arrived at.

We need a mathematical formulation for this problem. Let us say that the dates in are given by natural numbers so that the given date is  $s \in \mathbb{N}$ . We can encode the information that is needed from the  $p$ -th manager as a map  $x \rightarrow F_p(x)$ , where  $F_p(x)$  is the first date on or after  $x$  on which day the  $p$ -th manager is free. We capture the intuitive properties of each  $F_p$  by requiring that  $F_p$  is monotone non-decreasing on  $\mathbb{N}$ ,  $F_p(x) \geq x$ , and  $F_p(F_p(x)) = F_p(x)$  for all  $x$  and all  $p$ .

Let  $C = \text{fix}(\mathbf{F})$  and fix  $s \in \mathbb{N}$ . We are given that there exists  $z \in C$  with  $z \geq s$ . We are asked to design a multiflow machine that computes  $\min\{x \in C \mid x \geq s\} = r_s$ , say.

To write down a specification map for this problem we can take  $S = \mathbb{N}$ ,  $T = \mathbb{N}$ , and  $f(s) = r_s$ . In addition, let  $X = \mathbb{N}$ ,  $\rho = \pi = \text{Id}$  the identity map,  $\mathbf{F} = \{F_1, F_2, \dots, F_N\}$ .

**Theorem 6.1** *With definitions as above  $\mathbb{N} \xrightarrow{\text{Id}} \mathbb{N} \xrightarrow{\mathbf{F}} \mathbb{N} \xrightarrow{\text{Id}} \mathbb{N}$  is a multiflow machine computing  $f(s) = r_s$ .*

**Proof** Let  $s \in S$  and  $f(s) = r$ . Define  $A = \{s, s + 1, \dots, r\}$ . The theorem is proved by showing that  $A$  is safe for  $\mathbf{F}$  at  $s$  and that  $\lambda(x) = r - x$  is a bound function for  $\mathbf{F}$ . However, we can understand the essential reasoning without reference to the earlier theorems.

Since  $F_p$  is monotonic on  $A$  and  $F_p(r) = r$  for all  $p$ , if  $s \leq x \leq r$  then  $s \leq x \leq F_p(x) \leq F_p(r) = r$ . So  $A$  is invariant. If  $y$  is reached after one round of computation starting from  $s$  then there can be no fixed point less than  $y$ , as otherwise the computation could not have reached  $y$ . If  $y$  is a fixed point then

$y = r$ . If not we repeat the above argument till  $r$  is reached.  $\square$

## 6.2 Division

Let  $a$  be a non-negative integer and  $b$  a positive integer. Let  $quo(a, b)$  and  $rem(a, b)$  respectively stand for the quotient and remainder when  $a$  is divided by  $b$ . We are asked to compute these two quantities. We consider the maps suggested by Chandy and Misra [3]. It is possible to consider other maps that simplify the work. But our purpose here is to illustrate some finer points of the proof of a progress theorem.

1. The input space is  $S = \mathbb{N} \times \mathbb{P}$ .
2. The output space is  $T = \mathbb{N} \times \mathbb{N}$ .
3. The specification map is  $f(a, b) = (quo(a, b), rem(a, b))$ .
4. The state space is  $X = \mathbb{N}^6 = \{x = (a, b, q, r, c, m)\}$ . Suggestively  $q$  is for quotient,  $r$  is for remainder,  $c$  is for counter, and  $m$  is for multiple.
5.  $\rho(a, b) = (a, b, 0, a, 1, b)$ .
6.  $\pi(a, b, q, r, c, m) = (q, r)$ .
7.  $\mathbf{F}$  consists of the three maps defined below.

$$F_1(a, b, q, r, c, m) = \begin{cases} (a, b, q + c, r - m, c, m), & \text{if } r \geq m; \\ (a, b, q, r, c, m), & \text{if } r < m. \end{cases}$$

$$F_2(a, b, q, r, c, m) = \begin{cases} (a, b, q, r, 2c, 2m), & \text{if } r \geq 2m; \\ (a, b, q, r, 1, b), & \text{if } r < 2m. \end{cases}$$

$$F_3(a, b, q, r, c, m) = \begin{cases} (a, b, q, r, 3c, 3m) & \text{if } r \geq 3m; \\ (a, b, q, r, 1, b), & \text{if } r < 3m. \end{cases}$$

$$8. \mathbf{M} = \mathbb{N} \times \mathbb{P} \xrightarrow{\rho} \mathbb{N}^6 \xrightarrow{\mathbf{F}} \mathbb{N}^6 \xrightarrow{\pi} \mathbb{N}^2.$$

**Theorem 6.2** *With definitions as above,  $\mathbf{M}$  is a multiflow machine computing  $f(a, b) = (\text{quo}(a, b), \text{rem}(a, b))$ .*

**Proof** Choose and fix  $(a, b) \in S$  and define

$$A = \{x \mid c \geq 1, m = bc, a = bq + r.\}$$

Then  $\rho(a, b) \in A$  and it is easy to check that all the three maps  $F_1, F_2$  and  $F_3$  leave  $A$  invariant. So it is enough to prove termination and correctness on this set.

Let  $x \in A$ . We show that  $x \in \text{fix}(\mathbf{F})$  if and only if  $r < b = m$ . Let  $r < b = m$ . Then  $r < b = m < 2m < 3m$ , so that  $x = F_1(x) = F_2(x) = F_3(x)$ . Conversely let  $x \in \text{fix}(\mathbf{F})$ . Then  $F_1(x) = x$  implies that  $r < m$  and  $F_2(x) = x$  implies that  $m = b$ . So  $r < b = m$ .

This means that if the computation ends then we have both  $a = bq + r$  as well as  $0 \leq r < b$ . So  $q$  and  $r$  must respectively give the quotient and remainder when  $a$  is divided by  $b$ . This proves correctness.

To prove termination consider the height function  $\lambda(x) = r$ . Note that the value of  $r$  is never increased by any step of the computation. We prove that  $\lambda$  is a weak bound function of order 2. Divide  $A$  into 4 parts as below.

$$\begin{aligned} A_0 &= \{x \mid r < b, m = b\} = \text{fix}(\mathbf{F}) \\ A_1 &= \{x \mid r < b, m > b\} \\ A_2 &= \{x \mid r \geq b, r \geq m\} \\ A_3 &= \{x \mid r \geq b, r < m\} \end{aligned}$$

If  $x \in A_0$  there is nothing to prove. Let  $x \in A_1$ . Then  $F_1(x) = x$  and  $F_2(x) = F_3(x) = (a, b, q, r, 1, b) \in \text{fix}(\mathbf{F})$ . This proves that  $x \cdot \mathbf{p} \in \text{fix}(\mathbf{F})$  for every 1-fair string  $\mathbf{p}$ . So one round of computation changes  $x$  to a fixed point.

Let  $x \in A_2$ . Then  $F_2(x)$  and  $F_3(x)$  both belong to  $A_2$  and  $\lambda(x)$  is unchanged. However  $\lambda(F_1(x)) < \lambda(x)$ . So  $\lambda(x)$  is strictly reduced in one round of computation.

Finally, let  $x \in A_3$ . Then  $F_1(x) = x$  whereas  $F_2(x)$  and  $F_3(x)$  both belong to  $A_2$ . The rest of the argument is a little delicate. Let  $\mathbf{p}$  be any 2-fair string. Then  $\mathbf{p} = \mathbf{q} \cdot \mathbf{r}$  where  $\mathbf{q}$  and  $\mathbf{r}$  are 1-fair. Let  $\mathbf{q} = q_0q_1 \cdots q_{n-1}$ . Let  $q_k$  be the first label that is not equal to 1. Let  $y = x \cdot q_0q_1 \cdots q_k$ . Then  $y \in A_2$ . The string  $q_{k+1} \cdots q_{n-1} \cdot \mathbf{r}$  is 1-fair and its action on  $y$  strictly decreases  $\lambda$ . This means that the action of  $\mathbf{p}$  on  $x$  strictly decreases  $\lambda$ , by the earlier part of the argument.

The three paragraphs above together show that  $\lambda$  is a weak bound function of order 2. Hence termination is guaranteed.  $\square$

### 6.3 Sorting

Given an  $n$ -tuple  $x = (x_1, x_2, \dots, x_n) \in \mathbb{N}^n$  the problem is to sort it in increasing order. This may be done in a naive way. Choose any  $i < n$  and sort the pair  $(x_i, x_{i+1})$ . Choose an  $i$  again and repeat the process. It is intuitively reasonable that if we make a choice of  $i$  sufficiently many times and if each index is chosen enough times then the  $n$ -tuple is eventually sorted.

We need to model this nondeterministic algorithm.

1. Let  $S = T = \mathbb{N}^n$ . The specification map is given by  $f(x) = \text{sort}(x)$ .
2. Let  $X = \mathbb{N}^n$ .
3. Let  $\rho$  and  $\pi$  be the identity maps.
4. Consider  $\mathbf{F} = \{F_p \mid 1 \leq p \leq n - 1\}$ , where  $F_p$  sorts  $x_p$  and  $x_{p+1}$  in  $x$ .
5. Let  $\mathbf{M} = \mathbb{N}^n \xrightarrow{Id} \mathbb{N}^n \xrightarrow{\mathbf{F}} \mathbb{N}^n \xrightarrow{Id} \mathbb{N}^n$ .

The proof of the next theorem is presented without reference to the earlier theorems but the reader will notice that the proof is just a repeated application of Lemma 5.10.

**Theorem 6.3** *With definitions as above,  $M$  is a multiflow machine computing  $f(x) = \text{sort}(x)$ .*

**Proof** For any  $x \in X$  let  $\min x$  be the minimum value among all the entries of  $x$ . Let  $k$  be the first index such that  $x_k = \min x$ . Let  $\lambda(x) = k$ . Then  $\lambda(F_{k-1}(x)) = k - 1$  and  $\lambda(F_p(x)) = \lambda(x)$  if  $p \neq k - 1$ . So after one round of computation  $\lambda$  is certain to decrease by 1. Repeating the argument we see that after  $k - 1$  rounds of computation  $\lambda$  takes the value 1. That means that  $\min x$  has reached the first position. No  $F_p$  can change its position further.

We see now that after another  $k - 2$  rounds of computation the second least element of  $x$  will reach its final position at index 2 and so on. So eventually the string is sorted.  $\square$

#### 6.4 Shortest Paths

In this section we assume some familiarity with graphs on the part of the reader.

Let  $S$  be the set of all complete integer weighted directed graphs, without any cycles of negative length, on the vertices  $v_1, v_2, \dots, v_N$ . (In this context the word ‘length’ is synonymous with ‘weight’.) Given any  $G \in S$ , let the weight associated with the edge  $(v_i, v_j)$  be  $W[i, j]$ . Set  $W[i, i] = 0$  for all  $i$ . This construction gives rise to a matrix  $W = W_G$ .

Given  $G \in S$ , and  $1 \leq i, j \leq N$ , let  $D[i, j]$  denote the length of a minimum-length path from  $v_i$  to  $v_j$ . If  $i = j$ ,  $D[i, j] = 0$ . We call  $D[i, j]$  the distance from  $v_i$  to  $v_j$ . This gives rise to a matrix  $D = D_G$ . It is desired to design a multiflow machine to compute the matrix  $D_G$  given  $W_G$  for any  $G$  as above.

What is an intuitive nondeterministic algorithm? Let  $d[i, j]$  be the length of some path from  $v_i$  to  $v_j$  for all  $i$  and  $j$ . For example, we could take  $d[i, j] = W[i, j]$ . We can consider  $d[i, j]$  to be a first estimate of  $D[i, j]$ . Then a better estimate would be  $\min\{d[i, j], d[i, k] + d[k, j]\}$  for some  $k$ . So we can keep choosing triples  $(k, i, j)$  and keep replacing  $d[i, j]$  as above. With luck we will reach a fixed value for every  $i$  and  $j$  and that will be  $D[i, j]$ . We now model this process as a multifold and show that this process does indeed converge and give the result we need.

1.  $S$  is the set of all complete integer weighted directed graphs on  $v_1, v_2, \dots, v_N$  without any cycles of negative length.
2.  $T$  is the space of  $N \times N$  matrices with integer entries.
3. The specification map  $f : S \rightarrow T$  is given by  $f(G) = D_G$ .
4. Let  $X = T$  and let  $d = (d[i, j])$  denote a typical matrix in  $X$ .
5. Given  $G \in S$  the associated matrix  $W_G$  is the coding of  $G$  in  $X$  so that  $\rho(G) = W_G$ .
6. Let  $\pi$  be the identity map given by  $\pi(d) = d$ .
7. For every  $(k, i, j) \in [1..N]^3$ , the associated operation  $F_{(k,i,j)}$  is the replacement of  $d[i, j]$  with  $\min\{d[i, j], d[i, k] + d[k, j]\}$  without changing other entries of the matrix. More formally,  $F_{(k,i,j)}(d) = d'$  where

$$\begin{aligned} d'[i, j] &= \min\{d[i, j], d[i, k] + d[k, j]\} \\ d'[u, v] &= d[u, v] \text{ if } (u, v) \neq (i, j). \end{aligned}$$

Let  $\mathbf{F} = \{F_{(k,i,j)} \mid (k, i, j) \in [1..N]^3\}$ . Let  $p$  be shorthand for  $(k, i, j)$ .

**Theorem 6.4** *With definitions as above  $M = S \xrightarrow{\rho} X \xrightarrow{\mathbf{F}} X \xrightarrow{\pi} T$  is a multifold machine computing  $f(G) = D_G$ .*

**Proof** Choose and fix a graph  $G \in S$ . All definitions to follow are relative to this graph. Let us call  $d \in X$  an admissible matrix if  $d[i, i] = 0$  for all  $i$ ,  $d[i, j]$

is the length of some path from  $v_i$  to  $v_j$  for  $i \neq j$ , and  $d[i, j] \leq W[i, j]$ . If  $d$  is admissible so is  $d \cdot p$  for all  $p$ . If  $A$  is the set of all admissible matrices then  $A$  is invariant.

Define  $\lambda : A \rightarrow \mathbb{Z}$  by  $\lambda(d) = \sum_{i,j} d[i, j]$ . For  $i \neq j$ ,  $d[i, j]$  is length of some path from  $v_i$  to  $v_j$  and similarly  $d[j, i]$  is the length of some path from  $v_j$  to  $v_i$ . So  $d[i, j] + d[j, i]$  is the length of a cycle at  $v_i$  and hence  $d[i, j] + d[j, i] \geq 0$ . It is given that  $d[i, i] = 0$  for all  $i$ . So  $\lambda(d) \geq 0$  for all  $d \in A$ . Hence  $\lambda$  is a height function on  $A$ .

For a given  $p$  suppose  $d \notin \text{fix}(F_p)$ . Then  $d'[i, j] = d[i, k] + d[k, j] < d[i, j]$ . So  $\lambda(d \cdot p) < \lambda(d)$ . Hence  $\lambda$  is a bound function and consequently  $A \subseteq \text{con}(\mathbf{F})$ .

We now show that  $A$  is safe for  $f$  at  $G$  so that  $\Phi(G) = \{f(G)\}$ . Since  $A$  is invariant and  $\rho(G) = W \in A$ , it is enough to prove that  $A \cap \text{fix}(\mathbf{F}) = \{D_G\}$ . Let  $E \in A \cap \text{fix}(\mathbf{F})$ . Since  $E$  is a fixed point we have  $E[i, j] \leq E[i, k] + E[k, j]$  for every  $i, j, k$ . Since  $E$  is admissible  $E[i, j] \leq W[i, j]$  and  $E[i, j]$  is length of some path from  $v_i$  to  $v_j$ . We now prove that  $E = D$ . We prove this by induction on the number of edges of a shortest path from  $v_i$  to  $v_j$  for  $i \neq j$ .

Suppose that the number of edges of a shortest path from  $v_i$  to  $v_j$  is 1. Then  $W[i, j] = D[i, j] \leq E[i, j] \leq W[i, j]$ . So  $D[i, j] = E[i, j]$ . Assume that  $D[i, j] = E[i, j]$  for every  $i, j$  for which the number of edges of a shortest path from  $v_i$  to  $v_j$  is  $\leq m$ , and there is a pair  $i, j$  such that the number of edges of a shortest path from  $v_i$  to  $v_j$  is  $m + 1$ . Let that path be given by  $(v_i = v_{a_0}, v_{a_1}, \dots, v_{a_m}, v_{a_{m+1}} = v_j)$ . It is clear that  $(v_i, v_{a_1}, \dots, v_{a_m})$  is a shortest path from  $v_i$  to  $v_{a_m}$  with number of edges  $m$ , and that  $(v_{a_m}, v_j)$  is a shortest path from  $v_{a_m}$  to  $v_j$  with only one edge. So  $D[i, j] = D[i, a_m] + D[a_m, j]$ . By the induction hypothesis we have  $D[i, a_m] = E[i, a_m]$  and  $D[a_m, j] = E[a_m, j]$ . Then  $D[i, j] \leq E[i, j] \leq E[i, a_m] + E[a_m, j] = D[i, a_m] + D[a_m, j] = D[i, j]$ . Consequently  $D[i, j] = E[i, j]$ . This proves that  $A$  is safe for  $f$  at  $G$ .  $\square$

Let the set  $[1..N]^3$  be ordered lexicographically. For  $1 \leq k \leq N$ , let  $\mathbf{p}_k = (k, 1, 1)(k, 1, 2) \dots (k, N, N)$ . Let the string  $\mathbf{p}$  be defined by  $\mathbf{p} = \mathbf{p}_1\mathbf{p}_2 \dots \mathbf{p}_N$ .

Clearly  $\mathbf{p}$  is a fair string. By Theorem 6.4 and the definition of convergence there exists  $k \geq 0$  such that  $W \cdot \mathbf{p}^k = D$ . The surprise is that it is enough to take  $k = 1$ .

### Theorem 6.5 (Floyd-Warshall)

$$W \cdot \mathbf{p} = D.$$

The proof of this theorem is a consequence of the lemma below.

**Lemma 6.6** *Let  $d_k = W \cdot \mathbf{p}_1\mathbf{p}_2 \dots \mathbf{p}_k$  for  $1 \leq k \leq N$ . Then, for  $i \neq j$ ,  $d_k[i, j]$  is the minimum length of the paths from  $v_i$  to  $v_j$  with vertices from  $\{v_i, v_j, v_1, \dots, v_k\}$ .*

**Proof** We shall use induction on  $k$ . For  $k = 1$ ,  $d_1[i, j] = \min\{W[i, j], W[i, 1] + W[1, j]\}$  and hence the claim is true. Assume the result for all values less than  $k$ . Then  $d_k = d_{k-1} \cdot (k11) \dots (kNN)$ . By the induction hypothesis, for any  $i \neq j$ ,  $d_{k-1}[i, j]$  is the minimum length of the paths from  $v_i$  to  $v_j$  with vertices from  $\{v_i, v_j, v_1, \dots, v_{k-1}\}$ . By the definition of  $d_k$  we have  $d_k[i, j] = \min\{d_{k-1}[i, j], d_{k-1}[i, k] + d_{k-1}[k, j]\}$ . It follows that  $d_k[i, j]$  is the minimum length of the paths from  $v_i$  to  $v_j$  with vertices from  $\{v_i, v_j, v_1, \dots, v_k\}$ . The lemma is proved.  $\square$

There is a parallel version of the Floyd-Warshall algorithm. Fix  $k$  and let  $F_k : A \rightarrow A$  be defined by  $F_k(d) = d'$  where

$$d'[i, j] = \min\{d[i, j], d[i, k] + d[k, j]\}$$

for all  $(i, j)$ . We see that  $d'[i, j] = d[i, k]$  and  $d'[i, j] = d[k, i]$ .

So when  $d[i, j]$  is changed no other entry of the matrix is changed. Consequently the action of  $F_k$  is the same as the one got by replacing one at a time each  $d[i, j]$  in some order of the pairs  $(i, j)$ . This implies that  $F_k(d) = d \cdot \mathbf{p}_k$ . In other words, the serial action of the maps  $F_{(k,i,j)}$  for fixed  $k$  and any order

of the pairs  $(i, j)$  is the same as the synchronized action  $F_k$ . We are thus led directly to the next theorem.

### **Theorem 6.7 (Parallel Floyd-Warshall)**

$$(F_N \circ \dots \circ F_2 \circ F_1)(W) = D.$$

## **7 Concluding Remarks**

Developing skill in using a programming language or software is relatively easy. Analyzing programs to assert their properties is harder. Computer scientists appeal to formal logic for this purpose. What we are attempting here is to create a layer of mathematics between the problem and a formal method of writing down and analyzing the computation. These mathematical methods may not meet the standards set by formal methods of verification which are meant to be automated, but nevertheless they would be of help in understanding the issues involved and consequently in designing better programs. Further, a student might find it easier to learn the formal methods after an understanding of the mathematical methods. This is the philosophical underpinning of the book [13] and this is the spirit in which we have taken up a study of UNITY programs.

We propose to continue our study of the multiflow model following the lead of Chandy and Misra till a critical mass of examples is ready to be class tested. We invite dynamical theorists and computer scientists interested in pedagogy to join us.

## **8 Acknowledgements**

We thank Prof. N. Soundararajan and Prof. Kesav Nori for suggesting the present study. Discussions with Prof. Nori helped structure this article. The

referees' comments on earlier versions of this article helped improve the presentation.

## References

- [1] Abelson,H., Sussman, G.J. and Sussman, J.: *Structure and Interpretation of Computer Programs*, Universities Press, Hyderabad, India, 2009.
- [2] Armoni, M. and Ben-Ari,M.: *The concept of nondeterminism: its development and implications for teaching*, Newsletter ACM SIGCSE Bulletin, Volume 41, Issue 2, June 2009.
- [3] Chandy, K.M. and Misra, J.: *Parallel Program Design: A Foundation*, Addison-Wesley, 1988.
- [4] Dijkstra, E.W.: *A Discipline of Programming*, Prentice-Hall, 1976.
- [5] Knuth, D.E.: *The Art of Computer Programming*, Volumes 1 - III, Third Edition, Pearson Education Asia, New Delhi, India, 2002.
- [6] Lloyd,S.: *Programming the Universe: A Quantum Computer Scientist Takes on the Cosmos*, First Vintage Books Edition, USA, 2007.
- [7] McCarthy, J. *Towards a Mathematical Science of Computation*, Information Processing, Proceedings of IFIP Congress 62, ed. C.M.Popplewell, North-Holland, 21-28, 1963.
- [8] Papadimitriou, C.H. and Lewis, H.R.: *Elements of the Theory of Computation*, Second Edition, Pearson Education(Singapore) Pte Ltd., Delhi 110 092, India, 2005.
- [9] Stepney, S.: *Non-standard Computation: An Overview*, <http://www-users.cs.york.ac.uk/susan/complex/nstdcomp.htm>
- [10] Venkata Rao, K. and Viswanath, K.: *Labeled Nondeterminism and UNITY Programs*, talk delivered at the conference “Managing Complexity in a Distributed World” at the Indian Institute of Science, May 27-31, 2008. (Proceedings of the conference were not published.)

- [11] Venkata Rao, K. and Viswanath, K.: *On the Definition of Nondeterministic Mechanisms*, <http://arxiv.org/abs/0906.4216>.
- [12] Viswanath, K.: *Computing with Dynamical Systems*, Journal of Differential Equations and Dynamical Systems, Vol.14(1), pp.1-24, 2006.
- [13] Viswanath, K.: *An Introduction to Mathematical Computer Science*, The Universities Press, Hyderabad, India, 2008.
- [14] M. Walicki, and S. Meldal, “Algebraic Approaches to Nondeterminism: An Overview,” *ACM Computing Surveys*, **29**, 30–81, 1997.
- [15] Wirth, N.: *Computing Science Education: The Road not Taken*, Opening Address, Integrating Technology into Computer Science Education Conference, Aarhus, June 24, 2002.

\* \* \* \* \*