A Formal Definition of Data Flow Graph Models

KRISHNA M. KAVI, BILL P. BUCKLES, SENIOR MEMBER, IEEE, AND U. NARAYAN BHAT

Abstract—**In this paper, a new model for parallel computations and parallel computer systems that is based on data flow principles is presented. Uninterpreted data flow graphs can be used to model computer systems including data driven and parallel processors. A data flow graph is defined to be a bipartite graph with actors and links as the two vertex classes. Actors can be considered similar to transitions in Petri nets, and links similar to places. The nondeterministic nature of uninterpreted data flow graphs necessitates the derivation of liveness conditions.**

Index Terms—**Bipartite graphs, data flow graphs, deadlocks, liveness, parallel computations, Petri nets.**

I. Introduction

THE demands for increasing computation speeds have generated considerable interest in parallel computations, concurrent operations within computer systems, models for representing parallelism in algorithms, and new programming languages for such parallel computers [11], [19]. In addition to the design of parallel machines and programming aspects of parallelism, there has been considerable work done in formulating appropriate theoretical models and methods of analysis under which inherent properties of parallelism can be precisely defined and studied more from the viewpoint of the algorithm or problem than the particular machine implementation. Generally, the theoretical work can be divided into two categories: 1) the study of computational aspects of algorithms (both arithmetic and control) devised to make use of the parallelism existing in parallel systems; or 2) the study of the performance and reliability aspects of parallel computers.

There are a number of different theoretical models proposed for representing the computational aspects of parallel processes, among which Petri net models have enjoyed continued interest over the past decade. For a comparative study of models of parallel computation, the reader is referred to [19].

Performance and reliability evaluations of computer systems; including those with multiple processing elements and redundancy, are generally based on probabilistic models and their analysis. The techniques used in this approach involve the identification of underlying stochastic processes and the determination of their properties. General review of various aspects of these analysis techniques can be found in [15] and [25].

Manuscript received October 30, 1984; revised June 8, 1986. This work was supported in part by NASA Ames Research Center under Grant NAG 2-273.

K. M. Kavi and B. P. Buckles are with the Department of Computer Science Engineering, The University of Texas, Arlington, TX 76019.

U. N. Bhat is with the Department of Statistics, Southern Methodist University, Dallas, TX 75222.

IEEE Log Number 8610934.

Petri net models of parallel and asynchronous systems have been extended to include stochastic aspects [10], [17], [21], [22]. Molloy establishes an isomorphism between stochastic Petri nets and homogeneous Markov processes, thus making it possible to apply Markov techniques for the analysis of stochastic Petri net models.

In recent years a new form of program representation known as data flow has attracted the attention of researchers in the United States, England, France, and Japan. The literature is abundant with proposals for new computer systems based on data flow principles [7], [8], [24], programming languages [1]–[3], distributed computing based on data flow [18], as well as simulation and modeling using data flow graphs [9], [12], [23].

Much of the research in data flow processing has dealt with defining the functionality, designing instruction level architectures, or specifying programming methodologies. This has not made urgent the formalization of the data flow model itself. Formalization is necessary, however, in relating data flow to other computation models, discovering properties of specific instances of data flow graphs (e.g., absence of deadlocks), and in performance evaluation. Formalization also makes possible the utilization of data flow graphs as abstract models of computation analogous to Turing machines and Petri nets. It is from this motivation that the present work stems.

Data flow graphs have been used successfully in the simulation of computer systems [9], [23]. The chief advantage of data flow graphs over other models of parallel processors is their compactness and general amenability to direct interpretation. That is, the translation from the conceived system to a data flow graph is straightforward and, once accomplished, it is equally straightforward to determine by inspection which aspects of the system are represented. Because of the hierarchical nature and the modularity of data flow graphs, both software tasks and hardware units can be modeled in a uniform way using data flow graphs [12]. The formalism presented here and elsewhere [13], [14] can be used to analyze the performance and reliability of computer systems modeled as data flow graphs.

In the remainder of the paper, a formal set-relationship definition of a specific kind of data flow graph (known as an uninterpreted data flow graph) is presented. These definitions are based on the data flow model originally presented by Dennis [6]. An illustration of its use in describing properties is given in the form of a liveness theorem. Stochastic aspects are introduced into the model so that performance and reliability of data flow graph models of computer systems can be analyzed.

0018-9340/86/1100-0940$01.00

II. The Data Flow Concept

A data flow graph is a bipartite directed graph in which the two types of nodes are called links and actors [6]. In Dennis' model, actors describe operations while links receive data from a single actor and transmit values to one or more actors by way of arcs. (Arcs can be considered as channels of communication.) In its basic form, nodes (actors and links) are enabled for execution when all input arcs contain tokens and no output arcs contain tokens. An enabled node consumes tokens on input arcs and produces tokens on output arcs. Arcs can be control or data arcs. In the case of control arcs (which enter or leave control links), the tokens are of the type Boolean (true or false); for data arcs (which enter or leave data links), the tokens are of the type integer, real, or character. Control tokens are introduced to indicate the presence of sequence control; certain actors are enabled only when the right control values appear on the input control arcs. For a complete description of data flow concepts, the reader is referred to [24].

The data flow model of computation is neither based on memory structures that require inherent state transitions nor does it depend on history sensitivity. Thus, it eliminates some of the inherent von Neumann pitfalls described by Backus [4]. Several extensions to basic data flow have been proposed so that data flow techniques can be used in a variety of applications. In his dissertation, Landry [16] surveyed some of these extensions. One is discussed here.

A. Firing Semantic Sets (FSS)

The basic firing rule adopted by most data flow researchers requires that all input arcs contain tokens and that no tokens be present on the output arcs. This provides an adequate sequencing control mechanism when the nodes in data flow graphs represent primitive operations. However, if the nodes are complex procedures, or data flow subgraphs, more generalized firing control for both input and output arcs is required. Landry [16] discussed a comprehensive input firing semantic specification for data flow nodes. The (input) firing semantic set refers to a subset of input arcs that must contain tokens to enable the node. Similarly, an output semantic set can be defined as the subset of output arcs that must be empty. When the node is fired, tokens are removed from arcs of the input firing semantic set and new tokens are placed on arcs of the output firing semantic set. For different instances of the execution of a node, the firing sets may differ, thus introducing nondeterminacy. The formal model of data flow graphs described in this paper incorporates this generalized firing specification.

III. Data Flow Formalism

Definition 1: A *data flow graph* is a bipartite labeled graph where the two types of nodes are called *actors* and *links.*

$$G = \langle A \cup L,\ E \rangle \tag{1}$$

where

$A = \{a_1, a_2, \cdots, a_n\}$ is the set of actors

$L = \{l_1, l_2, \cdots, l_m\}$ is the set of links

$E \subseteq (A \times L) \cup (L \times A)$ is the set of edges.

Actors represent functions and links are treated as place holders of data values (tokens) as they flow from actors to actors. Edges are the channels of communication (like arcs in Dennis' model). S is a subset of links called the *starting set* (input links); these links represent external inputs to a data flow graph (or subgraph).

$$S = \{l \in L \mid (a, l) \notin E, \quad \forall a \in A\}. \tag{2}$$

T is a subset of links called the *terminating set* (output links); these links represent outputs from a data flow graph (or subgraph).

$$T = \{l \in L \mid (l, a) \notin E, \quad \forall a \in A\}. \tag{3}$$

The set of input links to an actor a, and the output links from an actor a are denoted by $I(a)$ and $O(a)$.

$$I(a) = \{l \in L \mid (l, a) \in E\} \tag{4}$$

$$O(a) = \{l \in L \mid (a, l) \in E\}. \tag{5}$$

Similarly, $I(l)$ and $O(l)$ for links can be defined.

The transitive closure on these sets, $I(a)^+$, $I(l)^+$, $O(a)^+$, $O(l)^+$ are defined in the usual manner. For example, $I(a)^+$ denotes the set of links that are inputs to actor a or the input links that are inputs to the actors that feed the input links of a, and so on.

$$I(a)^+ = \{l \in L \mid l \in I(a) \text{ or } l \in I(I(I(a))), \cdots\}. \tag{6}$$

If $B \subseteq A$ is a subset of actors then $I(B)$ and $O(B)$ define the sets of links that are inputs and outputs of actors belonging to B.

$$I(B) = \{l \in L \mid l \in I(b) \quad \text{for } b \in B\} \tag{7}$$

$$O(B) = \{l \in L \mid l \in O(b) \quad \text{for } b \in B\}. \tag{8}$$

A. Uninterpreted Data Flow Graphs

For the purpose of studying the performance of data flow graph models of computer systems, the actual meaning of the functions performed by actors and semantics of the data tokens are not relevant. The presence of tokens in links act as triggering signals to enable nodes. Such data flow graphs will be known as uninterpreted data flow graphs. Throughout this paper the term data flow graph is used to mean an uninterpreted data flow graph.

The data flow graphs in this formal model satisfy the following conditions.

$$\begin{aligned} &|I(a)| > 0 && \text{for all actors } a \in A \\ &|I(l)| = 0 \text{ or } 1 && \text{for all links } l \in L \\ &|O(a)| > 0 && \text{for all actors } a \in A \\ &|O(l)| = 0 \text{ or } 1 && \text{for all links } l \in L. \end{aligned} \tag{9}$$

Although this appears restrictive since the links can have at

most one input actor and one output actor, the authors have successfully translated all data flow graphs by introducing dummy actors (for example, to duplicate an input token onto several output links). This restriction allows for a simpler definition of markings [see (10)–(13)].

Definition 2: A marking is a mapping

$$M : L \rightarrow \{0, 1\}. \tag{10}$$

A link is said to contain a token in a marking M if $M(l) = 1$. An *initial marking* M_0 is a marking in which a subset of the starting set of links contain tokens. A *terminal marking* M_t is a marking in which a subset of the terminating set of links contain tokens.

B. Firing and Firing Semantic Sets

Associated with each actor are two sets of links called *input firing semantic set* F_1 and *output firing semantic set* F_2

$$F_1(a, M) \subseteq I(a)$$

$$F_2(a, M) \subseteq O(a). \tag{11}$$

The input firing semantic set refers to the subset of input links that must contain tokens to enable the actor; the output firing semantic set refers to the subset of links that receive tokens when the actor is fired.

Definition 3: A *firing* is a partial mapping from markings to markings. An actor a is firable at a marking M if the following conditions hold.

$$M(l) = 1 \quad \text{for all } l \in F_1(a, M)$$

$$M(l) = 0 \quad \text{for all } l \in F_2(a, M). \tag{12}$$

When the actor is fired, tokens from the firing set $F_1(a, M)$ of links are consumed and new tokens are placed on each link belonging to the output firing semantic set $F_2(a, M)$. Thus, a new marking M' resulting from the firing of an actor a at marking M can be derived as follows.

$$M'(l) = \begin{cases} 0 & \text{if } l \in F_1(a, M) \text{ and } l \notin F_2(a, M) \\ 1 & \text{if } l \in F_2(a, M) \\ M(l) & \text{otherwise.} \end{cases} \tag{13}$$

A firing of an actor is indicated by $M \xrightarrow{a} M'$.

Depending on whether F_1 and F_2 select only one, a proper subset or the entire set of input and output links the following actor firing rules apply.

Conjunctive: All the input links must contain tokens for the actor to fire. That is,

$$F_1(a, M) = I(a) \quad \text{for all } M. \tag{14}$$

Disjunctive: Only one of the input links must contain a token for the actor to fire. That is,

$$|F_1(a, M)| = 1 \quad \text{for all } M. \tag{15}$$

Collective: One or more of the input links may contain tokens for the actor to fire. That is,

$$F_1(a, M) \subseteq I(a) \quad \text{for all } M. \tag{16}$$

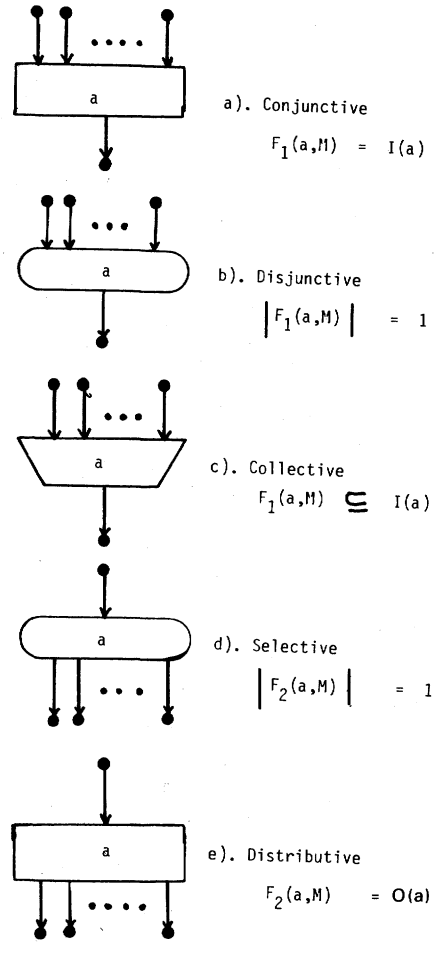


Fig. 1. Firing rules.

Selective: When a fires, only one of the output links receives a token. That is,

$$|F_2(a, M)| = 1 \quad \text{for all } M. \tag{17}$$

Distributive: When a fires, all the output links receive tokens. That is,

$$F_2(a, M) = O(a) \quad \text{for all } M. \tag{18}$$

Graphical representation of these possibilities are shown in Fig. 1.

C. Nondeterministic Firing Semantics

Since uninterpreted data flow graphs are used here, the firing semantic sets F_1 and F_2 are nondeterministic; for different instances of execution of an actor the firing semantic sets can be different. This eliminates the need for control arcs in data flow graph models. The choice arising due to control tokens are incorporated into F_1 and F_2 by associating probability distributions with the firing semantic sets. For example, the t gate [6] shown in Fig. 2(a) is replaced by the actor in Fig. 2(b). The firing sets F_1 and F_2 are defined as

$$F_1(t, M) = I(t) = \{l_1\}$$

$$F_2(t, M) = O(t) = \{l_2\} \quad \text{with probability } p$$

$$= \Phi \quad \text{with probability } 1-p. \tag{19}$$

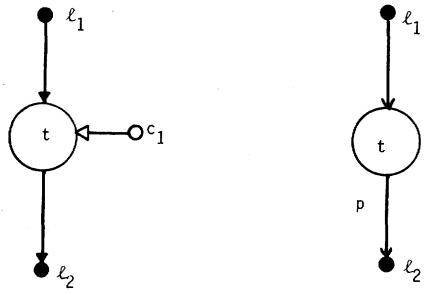


Fig. 2. Representation of a t gate using nondeterministic firing semantics.

The probability p depends on the frequency of having a TRUE value on the control link of the t gate.

$P[F_1(a, M) = f_1]$ is the probability that in marking M, $f_1 \subseteq I(a)$ is the input firing semantic set. The probability determines which subset of tokens on input links (should a choice be made) will be consumed when the actor a is fired in marking M. This probability distribution function is significant for collective actors only. Similarly, $P[F_2(a, M)]$ is the probability distribution on the output firing semantic set; when the actor a fires in M, the links in $f_2 \subseteq O(a)$ will receive tokens with a probability $P[F_2(a, M) = f_2]$. Conditional probabilities can also be defined for the firing semantic sets when $P[F_1(a, M)]$ and $P[F_2(a, M)]$ depend on the firing sets selected by other actors.

Definition 4: A *firing sequence* σ is a sequence of actors, in the order in which the actors are enabled. When actors can be fired concurrently, the order is arbitrary. An actor a is said to belong to σ if a is fired at least once in the firing sequence σ. We say M leads to M' via σ if M' is the new marking that is derived from the marking M when the actors in the firing sequence σ are fired. This is denoted by $M \xrightarrow{\sigma} M'$. The set of markings generated by a firing sequence σ will be known as marking set M^σ.

$$M^\sigma = \{M' \mid M \xrightarrow{\Sigma} M' \quad \text{for any subsequence } \Sigma \text{ that is a prefix of } \sigma\}. \tag{20}$$

If $M \xrightarrow{\sigma} M$ for some nonempty firing sequence σ, then the firing sequence is known as a *firing cycle.*

A *forward marking class* $\vec{M}$ of a marking M is the set of markings which can be derived (or reached) from M via some firing sequence.

$$\vec{M} = \{M' \mid M \xrightarrow{\sigma} M' \quad \text{for some firing sequence } \sigma\}. \tag{21}$$

For some actor $a \in A$ and a marking set M^σ,

$$F_1(a, M)^\sigma = \{l \in L \mid l \in F_1(a, M') \text{ where } M' \in M^\sigma \text{ and } a \text{ is firable in } M'\} \tag{22}$$

$$F_2(a, M^\sigma) = \{l \in L \mid l \in F_2(a, M') \text{ where } M' \in M \text{ and } a \text{ is firable in } M'\}. \tag{23}$$

$F_1(B, M^\sigma)$ and $F_2(B, M^\sigma)$ when B is a subset of actors can be defined similarly.

D. An Example

Baer [5, p. 71] gives a Petri net model representing the control flow in the execution of an instruction in a single accumulator arithmetic and logic unit. Fig. 3 shows a data flow equivalent of the Petri net given by Baer. The actors are intentionally named by the events in order to facilitate interpretation. The data flow graph consists of conjunctive, disjunctive, selective, and distributive actors. Actors labeled as "duplicate" are introduced to satisfy the requirement that links have only one input and one output.

$$G = \langle A \cup L, E \rangle$$

where

$$A = \{a_1, a_2, a_3, \cdots, a_{22}\}$$

$$L = \{l_0, l_1, l_2, \cdots, l_{32}\}$$

$$S = \{l_0\}$$

$$T = \{l_{32}\}.$$

The markings for the data flow graph are listed in Fig. 4. Ones indicate the presence of tokens in links and blanks (actually zeros) represent absence of tokens in links. When multiple actors are enabled in a marking, they are assumed to fire concurrently leading to the next marking. M_0 is the initial marking and M_{14} is the terminal marking. The firing semantic sets for all actors (in the markings in which they are enabled) are shown in Fig. 5. Probabilities associated with the output firing semantic set of selective actors are also shown in Fig. 5. Three firing cycles can be identified.

$$\sigma^1 = \{a_1, a_2, a_3, a_4, a_5, a_8, a_{21}, a_{10}, a_{11}, a_{13}, a_{14}, a_{16}, a_{22}\}$$

$$M_1^{\sigma^1} = \{M_1, M_2, M_3, M_4, M_5, M_8, M_9, M_{10}, M_{11}, M_{12}, M_{13}, M_{14}, M_{15}, M_{16}\}$$

$$\sigma^2 = \{a_1, a_2, a_3, a_4, a_6, a_{11}, a_{15}, a_{14}, a_{17}, a_{16}, a_{18}, a_{19}, a_{21}, a_{22}\}$$

$$M_1^{\sigma^2} = \{M_1, M_2, M_3, M_4, M_6, M_{13}, M_{14}, M_{17}, M_{18}, M_{19}, M_{20}, M_{21}, M_{22}\}$$

$$\sigma^3 = \{a_1, a_2, a_3, a_4, a_7, a_9, a_{11}, a_{14}, a_{12}, a_{15}, a_{16}, a_{17}, a_{18}, a_{20}, a_{21}, a_{22}\}$$

$$M_1^{\sigma^3} = \{M_1, M_2, M_3, M_4, M_7, M_{13}, M_{14}, M_{23}, M_{24}, M_{25}, M_{26}, M_{27}, M_{28}, M_{29}, M_{30}\}.$$

The forward marking class of the initial marking M_0 is the entire set of markings

$$\vec{M}_0 = \{M_0, M_1, \cdots, M_{30}\}.$$

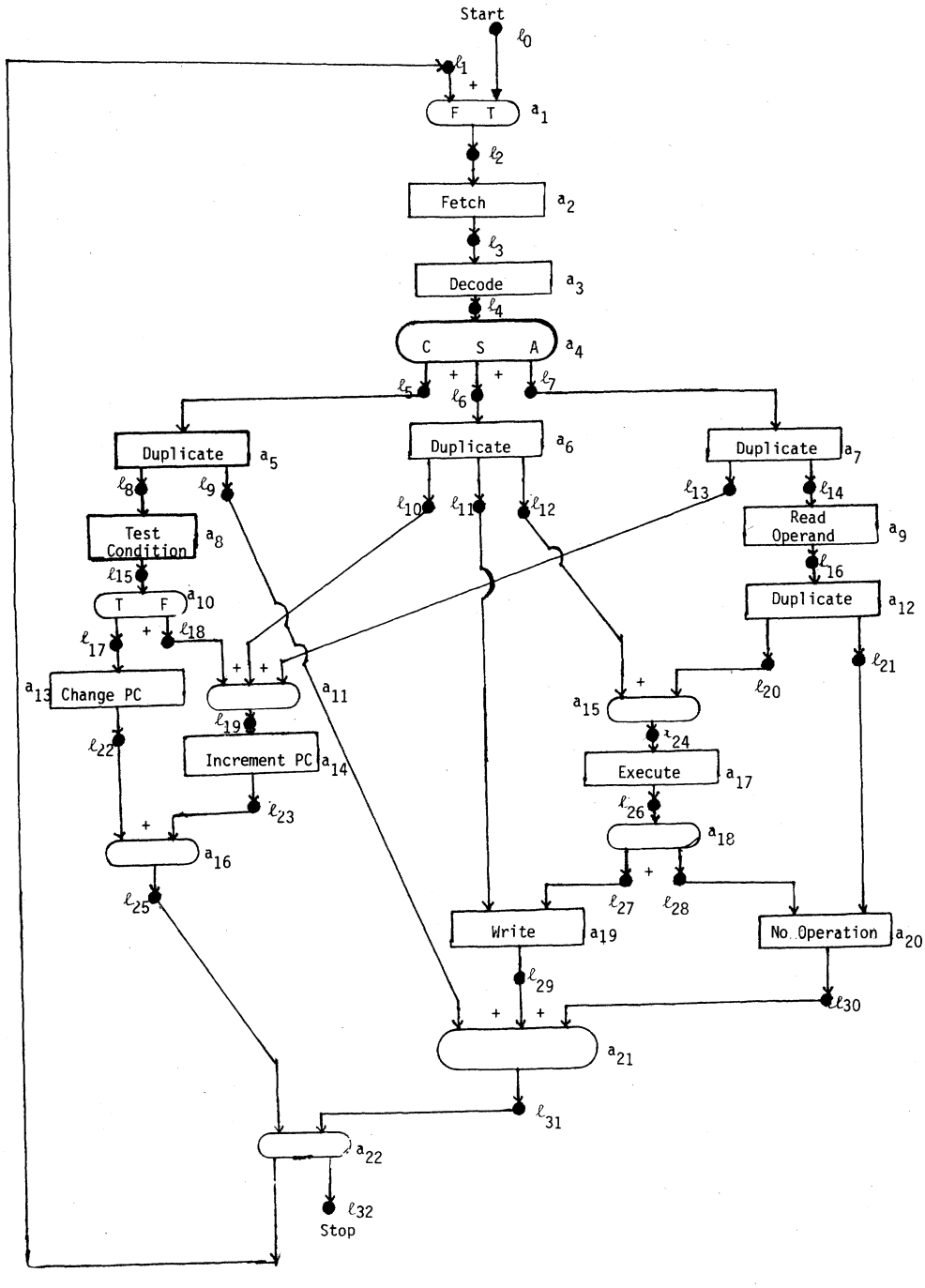


Fig. 3. A data flow diagram of a simple computer.

E. Deadlocks and Liveness in Data Flow Graphs

We should be able to identify situations in which an actor cannot fire. Deadlocks can be avoided in data flow machines by providing feedback using control tokens [20]. To illustrate this, a part of Fig. 3 is redrawn with control arcs and links (Fig. 6). As can be seen, the presence of the same value on control links at actors a_4, a_{15}, a_{18}, and a_{21} permits the selection of the proper path for the flow of tokens. Since the model presented here omits control tokens, it is necessary to derive liveness conditions.

Definition 5: An actor is *potentially firable* in a marking $M(\neq M_t)$, if there exists a marking $M' \in \vec{M}$ such that a is enabled in M'.

An actor is said to be *blocked* in a marking M if for all markings $M' \in \vec{M}$, a is not firable.

An actor is *live* in M if a is potentially firable in all markings $M' \in \vec{M}$, except when M' is a terminal marking.

Definition 6: A firing sequence σ is said to be *live* in a marking M if all the actors in σ are live in the markings M^σ.

Firing sequences considered here are assumed reachable from an initial marking (that is, $M^\sigma \cap \vec{M} \neq \Phi$). If there are no initial markings in the data flow graph, then a firing sequence is assumed to be initiated in a marking $M \in M^\sigma$. Firing

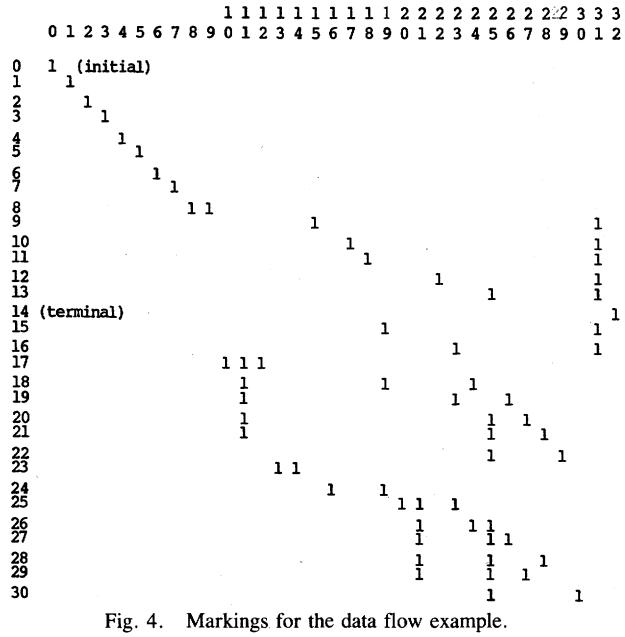


Fig. 4. Markings for the data flow example.

$F_1(a_i, M_j)$	$F_2(a_i, M_j)$
$(a_1, M_0) = l_0$	$(a_1, M_0) = l_2$
$(a_1, M_1) = l_1$	$(a_1, M_1) = l_2$
$(a_2, M_2) = l_2$	$(a_2, M_2) = l_3$
$(a_3, M_3) = l_3$	$(a_3, M_3) = l_4$
$(a_4, M_4) = l_4$	$(a_4, M_4) = l_5$ with p_1
	$= l_6$ with p_2
	$= l_7$ with p_3
$(a_5, M_5) = l_5$	$(a_5, M_5) = l_8, l_9$
$(a_6, M_6) = l_6$	$(a_6, M_6) = l_{10}, l_{11}, l_{12}$
$(a_7, M_7) = l_7$	$(a_7, M_7) = l_{13}, l_{14}$
$(a_8, M_8) = l_8$	$(a_7, M_8) = l_{15}$
$(a_9, M_{23}) = l_{14}$	$(a_9, M_{23}) = l_{16}$
$(a_{10}, M_9) = l_{15}$	$(a_{10}, M_9) = l_{17}$ with p_4
	$= l_{18}$ with p_5
$(a_{11}, M_{11}) = l_{18}$	$(a_{11}, M_{11}) = l_{19}$
$(a_{11}, M_{17}) = l_{10}$	$(a_{11}, M_{17}) = l_{19}$
$(a_{11}, M_{23}) = l_{13}$	$(a_{11}, M_{23}) = l_{19}$
$(a_{12}, M_{24}) = l_{16}$	$(a_{12}, M_{24}) = l_{20}, l_{21}$
$(a_{13}, M_{10}) = l_{17}$	$(a_{13}, M_{10}) = l_{22}$
$(a_{14}, M_{15}) = l_{19}$	$(a_{14}, M_{15}) = l_{23}$
$(a_{14}, M_{18}) = l_{19}$	$(a_{14}, M_{18}) = l_{23}$
$(a_{14}, M_{24}) = l_{19}$	$(a_{14}, M_{24}) = l_{23}$
$(a_{15}, M_{17}) = l_{12}$	$(a_{15}, M_{17}) = l_{24}$
$(a_{15}, M_{28}) = l_{20}$	$(a_{15}, M_{28}) = l_{20}$
$(a_{16}, M_{12}) = l_{22}$	$(a_{16}, M_{12}) = l_{25}$
$(a_{16}, M_{16}) = l_{23}$	$(a_{16}, M_{16}) = l_{25}$
$(a_{16}, M_{19}) = l_{23}$	$(a_{16}, M_{19}) = l_{25}$
$(a_{16}, M_{25}) = l_{23}$	$(a_{16}, M_{25}) = l_{25}$
$(a_{17}, M_{18}) = l_{24}$	$(a_{17}, M_{18}) = l_{26}$
$(a_{17}, M_{26}) = l_{24}$	$(a_{17}, M_{26}) = l_{26}$
$(a_{18}, M_{19}) = l_{26}$	$(a_{18}, M_{19}) = l_{27}$ with p_6
	$= l_{28}$ with p_7
$(a_{18}, M_{27}) = l_{26}$	$(a_{18}, M_{27}) = l_{27}$ with p_8
	$= l_{28}$ with p_9
$(a_{19}, M_{20}) = l_{11}, l_{27}$	$(a_{19}, M_{20}) = l_{29}$
$(a_{20}, M_{28}) = l_{21}, l_{28}$	$(a_{20}, M_{28}) = l_{30}$
$(a_{21}, M_8) = l_9$	$(a_{21}, M_8) = l_{31}$
$(a_{21}, M_{22}) = l_{29}$	$(a_{21}, M_{22}) = l_{31}$
$(a_{21}, M_{30}) = l_{30}$	$(a_{21}, M_{30}) = l_{31}$
$(a_{22}, M_{13}) = l_{25}, l_{31}$	$(a_{22}, M_{23}) = l_1$ with p_{10}
	$= l_{32}$ with p_{11}

Note: $p_1 + p_2 + p_3 = p_4 + p_5 = p_6 + p_7 = p_8 + p_9 = p_{10} + p_{11} = 1$.

Fig. 5. Firing semantic sets for the example.

sequences that are not reachable from an initial marking (implying that the data flow graph is not strongly connected or permanently disabled) do not contribute to the liveness of a data flow graph.

Theorem 1: Let $B \subseteq A$ be the subset of actors that belong to a firing cycle σ. The firing cycle σ is live in M if and only if

$$F_1(B, M^\sigma) = F_2(B, M^\sigma). \tag{24}$$

Proof: Necessary condition. If the firing sequence σ is live then (24) holds. This is proved by contradiction.

Suppose that σ is live, but

$$F_1(B, M^\sigma) \neq F_2(B, M^\sigma).$$

Case 1: There exists a link $l \in F_1(B, M^\sigma)$ but $l \notin F_2(B, M^\sigma)$. Since only actors in B and firable in firing cycle σ, only $F_2(B, M^\sigma)$ can receive tokens in all markings $M' \in M^\sigma$. This implies that there exists an actor $b \in B$ and a marking $M' \in M^\sigma$, such that $l \in F_1(b, M')$ does not contain a token. Then actor b will be blocked which is contrary to the assumption that actors in B are live.

Case 2: There exists a link $l \in F_2(B, M^\sigma)$, but $l \notin F_1(B, M^\sigma)$. Since only actors in B can fire in all the markings $M' \in M^\sigma$, only tokens on links $F_1(B, M^\sigma)$ are consumed. Since $l \notin F_1(B, M^\sigma)$, l will contain an unconsumed token. This implies that an actor $b \in B$ will be blocked in a marking $M' \in M^\sigma$ where $l \in F_2(b, M')$. This is again contrary to the assumption that actors in B are live. Hence, $F_1(B, M^\sigma) = F_2(B, M^\sigma)$.

Sufficient Condition. If (24) holds, then the firing sequence σ is live. This is proved by contradiction. That is, $F_1(B, M^\sigma) = F_2(B, M^\sigma)$, but the firing sequence σ is not live. Let $b \in B$ be an actor that is not live in M^σ (that is, there exists a marking $M' \in M^\sigma$ such that b is not potentially firable in M').

Case 3: The actor is not firable because some link $l \in F_1(b, M')$ does not contain a token. Since $F_2(B, M^\sigma)$ is the only set of links that can receive tokens in M^σ, $l \notin F_2(B, M^\sigma)$. This is a contradiction since we assumed that (24) holds.

Case 4: The actor b is not firable in M' because a link $l \in F_2(b, M')$ contains an unconsumed token. Since only tokens on $F_1(B, M^\sigma)$ can be consumed, $l \notin F_1(B, M^\sigma)$. This is again contrary to the assumption that (24) holds. Thus, if

$$F_1(B, M^\sigma) = F_2(B, M^\sigma)$$

then the firing sequence σ is live in M. Q.E.D.

Definition 7: A data flow graph is live (deadlock free) in a marking M if all actors $a \in A$ are live in M.

Theorem 2: A data flow graph is live in a marking M if and only if all firing sequences σ are live in M.

Proof: Necessary condition. If the data flow graph is live in M, then all firing sequences are live in M.

Suppose that the data flow graph is live in M, but there exists a firing sequence σ that is not live in M. By Definition 6, there exists an actor $a \in \sigma$ that is not live in marking $M' \in M^\sigma$. Then by Definition 7, the data flow graph is not live in M.

Sufficient Condition. If all firing sequences σ are live in M, then the data flow graph is live in M. Suppose that all firing

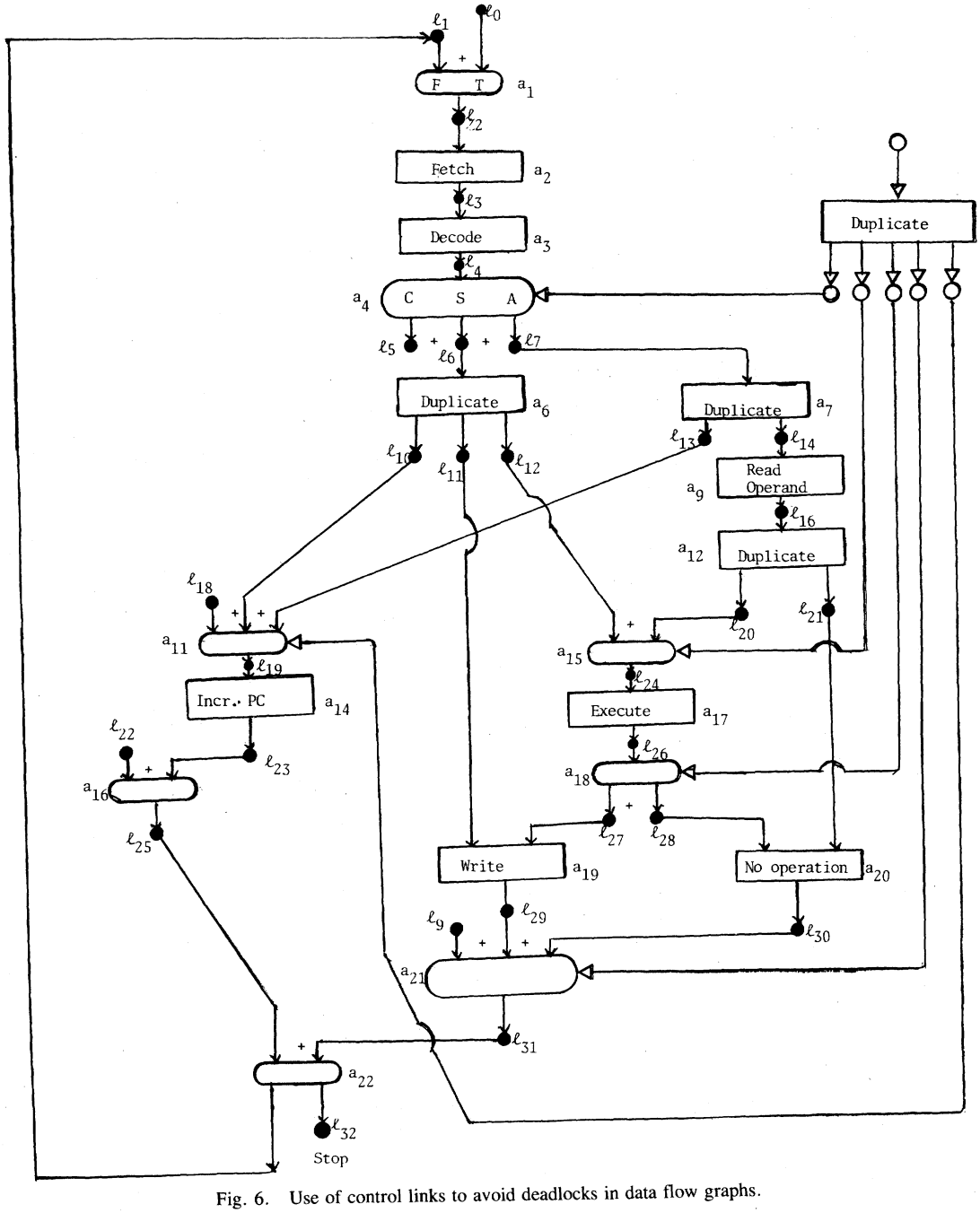


Fig. 6. Use of control links to avoid deadlocks in data flow graphs.

sequences σ are live in M, but the data flow graph is not live in M. By Definition 7, there exists an actor $a \in A$, that is not live in a marking $M' \in \vec{M}$. Since all actors must belong to one or more firing sequences, there exists a firing sequence σ, such that $a \in \sigma$ and $M' \in M^{\sigma}$, which is not live in M (by Definition 6). Q.E.D.

Corollary: If a live data flow graph contains no terminal marking B then

$$F_1(A, \vec{M}) = F_2(A, \vec{M}).$$

Proof: If a data flow graph is live then all firing sequences are live. Since there are no terminal markings, all firing sequences must be firing cycles. By Theorem 1,

$$\bigcup_{\sigma}[F_1(B, M^{\sigma})] = \bigcup_{\sigma}[F_2(B, M^{\sigma})]$$

$$F_1(A, \vec{M}) = F_2(A, \vec{M}). \quad \text{Q.E.D.}$$

Example: The data flow graph of Fig. 3 will be used to illustrate the deadlock theorems. For the firing cycle σ^1,

$$F_1(B, M_1^{\sigma^1}) = \{l_1, l_2, l_3, l_4, l_5, l_8, l_9, l_{15}, l_{17}, l_{18}, l_{19}, l_{22}, l_{23}, l_{25}, l_{31}\}$$

$$F_2(B, M_1^{\sigma^1}) = \{l_1, l_2, l_3, l_4, l_5, l_8, l_9, l_{15}, l_{17}, l_{18}, l_{19}, l_{22}, l_{23}, l_{25}, l_{31}\}$$

where B is the set of actors belonging to σ^1. The firing sequence σ^1 is live in M_1 since

$$F_1(B,\ M_1^{\sigma^1}) = F_2(B,\ M_1^{\sigma^1}).$$

For the firing cycle σ^2,

$$F_1(C,\ M_1^{\sigma^2}) = \{l_1,\ l_2,\ l_3,\ l_4,\ l_6,\ l_{10},\ l_{11},\ l_{12},\ l_{19},\ l_{23},\ l_{24},\ l_{25},\ l_{26},\ l_{27},\ l_{29},\ l_{31}\}$$

$$F_2(C,\ M_1^{\sigma^2}) = \{l_1,\ l_2,\ l_3,\ l_4,\ l_6,\ l_{10},\ l_{11},\ l_{12},\ l_{19},\ l_{23},\ l_{24},\ l_{25},\ l_{26},\ l_{27},\ l_{28},\ l_{29},\ l_{31}\}$$

where C is the set of actors belonging to σ^2. As can be seen

$$F_1(C,\ M_1^{\sigma^2}) \neq F_2(C,\ M_1^{\sigma^2})$$

and hence the firing sequence σ^2 is not live in M_1. Similarly, the firing sequence σ^3 is not live in M_1. These firing sequence can be made live by separating the two firing cycles (by creating two distinct execute actors for arithmetic and store instructions).

Remarks: Theorem 1 must be applied to check if firing cycles in a data flow graph are live, given that the cycles are entered. Thus, in the above example, $F_2(a_4,\ M_4) = \{l_5\}$, $\{l_6\}$, $\{l_7\}$, for the firing cycles σ^1, σ^2, σ^3, respectively.

The firing cycle σ^2 becomes live if the conditional probability $P[F_2(a_{18},\ M_{19}) = l_{27}/F_2(a_4,\ M_4) = l_6] = 1$. This is the case when control links and control arcs are used to eliminate deadlocks in data flow graphs [20].

The theorems presented here do not guarantee that the number of tokens flowing through a data flow graph remains constant, but that the tokens are conserved over a firing cycle. This allows for some actors consuming more tokens than they produce and others producing more tokens than they consume. Thus, in a nonterminating deadlock-free data flow graph, all firing sequences are repeatable.

Before using the deadlock theorems, a data flow graph must be transformed to conform to the definitions presented in this paper. The control arcs and links must be removed. Firing cycles may be discovered by examining circuits in the graph. Then, the liveness of the firing cycles can be verified.

IV. Conclusions

In this paper a formal definition of data flow graphs is introduced. The definition is based on the data flow model used originally by Dennis [6]. Necessary and sufficient conditions for liveness in data flow graphs are derived. The uninterpreted data flow graph can be used to model computer systems and the performance and reliability of the modeled system can be analyzed by introducing stochastic properties into data flow graphs. Isomorphic mappings between Petri nets and data flow graphs are presented in a separate paper. Such mappings enable the mathematical formulations used for Petri nets to be employed in analyzing data flow graph models.

References

[1] W. B. Ackermann, "Data flow languages," in *Proc. 1979 NCC,* New York, pp. 1087–1095.

[2] ——, "Data flow languages," *IEEE Computer,* pp. 15–25, Feb. 1982.

[3] Arvind, K. P. Gostelow, and W. Pflouffe, "An asynchronous programming language and computing machine," Dep. Inform. Comput. Sci., Univ. California, Irvine, Tech. Rep. 114a, Dec. 1978.

[4] J. Backus, "Can programs be liberated from von Neumann style? A functional style and its algebra of programs," *CACM,* pp. 613–641, Aug. 1978.

[5] J. L. Baer, *Computer Systems Architecture.* Rockville, MD: Computer Science, 1980.

[6] J. B. Dennis, "First version of data flow procedural language," *Lecture Notes in Computer Science, Vol. 19.* Berlin: Springer-Verlag, 1974.

[7] J. B. Dennis and D. P. Misunas, "A preliminary architecture for a basic data flow processor," in *Proc. 2nd Symp. Comput. Architect.,* Houston, TX, 1975, pp. 126–132.

[8] J. B. Dennis, "Data flow supercomputers," *IEEE Computer,* pp. 48–56, Nov. 1980.

[9] J.-L. Gaudiot and M. D. Ercegovac, "Performance analysis of data flow computers with variable resolution actors," in *Proc. 4th Int. Conf. Distrib. Comput. Syst.,* San Francisco, CA, May 1984, pp. 2–9.

[10] J. B. Dugan, K. S. Trivedi, R. Geist, and V. F. Nicola, "Extended stochastic Petri nets: Applications and analysis," *Performance 84,* E. Gelenebe, Ed. Amsterdam, The Netherlands: North Holland, 1984.

[11] R. M. Karp and R. E. Miller, "Properties of a model for parallel computations: Determinacy, termination, and queueing," *SIAM J. Appl. Math.,* vol. 14, pp. 1390–1411, Nov. 1966.

[12] K. M. Kavi, "Data flow modeling techniques," in *Proc. IASTED Int. Symp. Simulation and Modeling,* Orlando, FL, Nov. 1983, pp. 1–4.

[13] K. M. Kavi, B. P. Buckles, and U. N. Bhat, "Isomorphisms between Petri nets and data flow graphs," *IEEE Trans. Software Eng.,* to be published.

[14] ——, "Reliability analysis of data flow graph models," Dep. Comput. Sci. Eng., Univ. Texas, Arlington, Tech. Rep. CSE-85-003, 1985.

[15] L. Kleinrock, *Queueing Systems, Vol. 2. Computer Applications.* New York: Wiley, 1976.

[16] S. P. Landry, "System oriented extensions to data flow," Ph.D. dissertation, Dep. Comput. Sci., Univ. Southwestern Louisiana, Lafayette, May 1981.

[17] M. A. Marson, G. Balbo, and G. Conte, "A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems," in *Proc. ACM SIGMETRICS Conf. Measurement and Modeling,* 1983.

[18] M. Measures, B. D. Shriver, and P. A. Carr, "A distributed operating system based on data flow principles," in *Proc. COMPCON,* Sept. 1982, pp. 106–115.

[19] R. E. Miller, "A comparison of some theoretical models of parallel computation," *IEEE Trans. Comput.,* vol. C-22, pp. 710–717, Aug. 1973.

[20] D. P. Misunas, "Deadlock avoidance in data flow architecture," in *Proc. Symp. Automat. Computation and Contr.,* Milwaukee, WI, Apr. 1975, pp. 337–343.

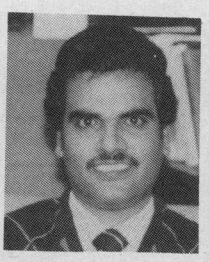
[21] M. K. Molloy, "On the integration of delayed throughput measures in distributed processing models," Ph.D. dissertation, Univ. California, Los Angeles, 1981.

[22] ——, "Performance analysis using stochastic Petri nets," *IEEE Trans. Comput,* vol. C-31, pp. 913–917, Sept. 1982.

[23] V. P. Srini and J. F. Asenjo, "Analysis of Cray 1S architecture," in *Proc. 10th Symp. Comput. Architect.,* Stockholm, Sweden, pp. 194–206, June 1983.

[24] P. C. Treleaven, D. R. Brownbridge, and R. P. Hopkins, "Data driven and demand driven computer architecture," *ACM Comput. Surv.,* Mar. 1982, pp. 93–143.

[25] K. S. Trivedi, "Analytical modeling of computer systems," *IEEE Computer,* pp. 38–56, Oct. 1978.



Krishna M. Kavi received the B.E. degree in electrical engineering from the Indian Institute of Science, the M.S. and Ph.D. degrees in computer science from Southern Methodist University, Dallas, TX, in 1975, 1977, and 1980, respectively.

He is an Associate Professor at the University of Texas, Arlington. Prior to joining the faculty of the University of Texas, he was with the University of Southwestern Louisiana, Lafayette. His interests include data flow architecture, high-level language architecture, distributed operating systems, and

performance evaluation of computer systems.

Dr. Kavi is a member of the IEEE Computer Society, Association for Computing Machinery, Sigma Xi, and Upsilon Pi Epsilon.

Bill P. Buckles (SM'82) received the M.A. degree in operations research, the M.A. degree in computer science, and the Ph.D. degree in operations research from the University of Alabama, Huntsville, AL.

Presently, he is an Associate Professor of Computer Science Engineering at the University of Texas, Arlington. Prior to his appointment, he was a Technical Staff Member at Computer Science Corporation, Science Applications Inc., and General Research Corporation. He has served as Principal Investigator on various National Science Foundation funded projects as well as those supported by industrial research laboratories. He has over one dozen journal publications. Currently, his interests are Petri net modeling as it relates to parallel computing and uncertainty representation in databases using fuzzy set theory.

Dr. Buckles is a senior member of the IEEE Computer Society, the Association for Computing Machinery, and the International Fuzzy Systems Association.

U. Narayan Bhat received the B.A. degree in Mathematics from Madras University, India, the M.A. degree in statistics from Karnatak University, India, and the Ph.D. degree in statistics from the University of Western Australia, Perth, in 1953, 1958, and 1965, respectively.

He has held faculty positions in Karnatak University, 1958–1961, the University of Western Australia, 1965, Michigan State University, East Lansing, 1965–1966, Case Western Reserve University, Cleveland, OH 1966–1969, and Southern Methodist University, Dallas, TX, from 1969 to the present, where he is a Professor of Statistics and Operations Research. He has also held various administrative positions at SMU between 1976–1982, the last being the Vice Provost and the Dean for Graduate Studies. His research interests are in the area of applied probability with particular reference to stochastic modeling of queueing, reliability, and computer systems. He is the author or coauthor of three books and more than 50 research articles.

Dr. Bhat is a member of several professional societies including the American Statistical Association, Operations Research Society of America, and the Institute of Management Sciences.