RELIABILITY MODELS FOR DATAFLOW COMPUTER SYSTEMS

Krishna M. Kavi
Bill P. Buckles

The University of Texas at Arlington
P.O. Box 19015
Arlington, Texas 76019

Final Report
for NASA-Ames Research Contract NAG-2-273

Submitted to: Ken Stevens
MS 233-14
NASA-Ames Research Center
Moffett Field, CA 94035

Reliability Models for Dataflow Computer Systems

The demands for concurrent operation within a computer system and the representation of parallelism in programming languages have yielded a new form of program representation known as dataflow ([DENN 74], [DENN 75], [TREL 82a]). Execution of dataflow programs is data driven; that is, each instruction is enabled for execution just when each required operand has been supplied by the completion of predecessor instructions.

Dataflow systems have received considerable attention during the past several years. There is a growing agreement, particularly in Japan ([TREL 82a], [TREL 82b]), that the next generation computers should be based on non-von Neumann architecture such as dataflow, as these architectures can be designed to deliver billions of operations per second.

However, the investigators of this research are not aware of any work in estimating either the performance or the reliability of this new structure. Because of the complexity of dataflow systems, the presence of multiple processing units and communication circuits, the reliability, performance of the interconnection structures and scheduling schemes in such architectures could become a significant issue.

In this research project we proposed to study the reliability of dataflow computer systems. We proposed to use the dataflow graph as a model to represent asynchronous concurrent computer architectures including dataflow computers. If methods for

analyzing the reliability of a dataflow graph were available, data driven computers can be modeled as dataflow graphs and their reliability can be analyzed.

In order to achieve this goal several intermediate projects were completed.

1. Formalization of dataflow graph models: Much of the research in dataflow processing has dealt with defining the functionality, designing instruction level architecture or specifying programming languages. This has not made urgent the formalization of the dataflow itself. Formalization is necessary in relating dataflow to other computational models, discovering properties of specific instances of dataflow graphs and in performance and reliability analyses. We presented a formal definition of dataflow graph models and derived conditions for deadlock freeness in dataflow graphs. Appendix 1 contains the details of this work.

2. Development of a Dataflow Simulator: Concurrent with the formalization, we designed and developed a dataflow simulator, DFDLS. This simulator can be used to model computer systems such as MIT dataflow processor and UC-Irvine dataflow computer and study the functionality of the simulated systems. DFDLS is written in Pascal on VAX (VMS). Appendix 2 contains a detailed description of the simulator.

3. Isomorphisms between Petri Nets and Dataflow Graphs: Petri nets have been used to represent asynchronous concurrent computations and Petri net analysis techniques are readily available in literature. Dataflow graph models are superior to Petri nets in

their representational power. We felt that if dataflow graphs can be translated in Petri nets, analysis techniques available for Petri nets can be used with dataflow graphs, thus combining the representational ease of dataflow models with the analysis power of Petri nets. We developed isomorphic transformations between the two models. We demonstrated that it is possible to decompose Petri nets into smaller components and determine aspects of the overall behavior from the behavior of components. Appendix 3 contains the details of this work.

Once these projects were completed we set out to develop reliability models for dataflow graphs. Both Markov chain techniques and path enumeration methods can be used to study the reliability of a dataflow graph model. We also developed techniques for reducing large graphs into smaller manageable size graphs. The details of these results are given in Appendix 4.

We feel that this research is significant in that, dataflow graphs can be used as generalized models of computation analogous to Turing machines and Petri nets; Petri net analysis techniques can be carried over to dataflow models; the reliability of data driven computer systems can be analyzed using dataflow graph models.

References

[DENN 74] J.B. Dennis. "First Version of Data flow Procedural Language", Lecture Notes in Computer Science, Vol. 19, Springer Verlag.

[DENN 75] J. B. Dennis and D.P. Misunas. "A Computer Architecture for a Basic Data Flow Processor", Proc. 2nd Annual Symposium on Computer Architecture, (Houston), pp 126-132.

[TREL 82a] P.C. Treleaven, D.R. Brownbridge and R.P. Hopkins. "Data Driven and Demand Driven Computer Architecture", ACM Computing Surveys, March, pp 93-143.

[TREL 82b] P.C. Treleaven and I.G. Lima. "Japan's Fifth Generation Computer Systems", IEEE Computer, Aug., pp 79-88.

APPENDIX 1

A FORMAL DEFINITION OF DATAFLOW GRAPH MODELS

A FORMAL DEFINITION OF DATAFLOW GRAPH MODELS

Krishna M. Kavi
Bill P. Buckles
The University of Texas at Arlington

and

U. Narayan Bhat
Southern Methodist University

This work is supported in part by NASA-Ames Research Center under Contract NAG 2-273.

A FORMAL DEFINITION OF DATAFLOW GRAPH MODELS

Krishna M. Kavi
Bill P. Buckles
and
U. Narayan Bhat

ABSTRACT

In this paper a new model for parallel computations and parallel computer systems, that is based on dataflow principles is presented. Mathematical formulations of uninterpreted dataflow graph models are defined. Necessary and sufficient conditions for liveness and deadlock-freeness in dataflow graphs are derived.

Keywords: Dataflow graphs, Parallel computations, Deadlocks, Liveness, Petri nets.

I. INTRODUCTION

The demands for increasing computation speeds have generated considerable interest in parallel computations, concurrent operations within a computer system, models for representing parallelism in algorithms and new programming languages for such parallel computers [KARP 66, MILL 73]. 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 end of the algorithm, or problem, rather than the particular machine implementation. Generally, the theoretical work in parallelism can be divided into two categories; 1) the study of the computational aspects of algorithms (both arithmetic and control aspects) devised to make use of the parallelism existing in parallel systems; or 2) the study of the performance limits and reliability aspect of parallel computers.

There are a number of quite different theoretical models proposed for representing the computational aspects of parallel processes, among which Petri net models enjoyed continued interest over the past decade. For a comparative study of models of parallel computation, the reader is referred to [MILL 73].

Performance and reliability evaluations of computer systems, including those with multiple processing elements and high-level

of 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 [KLEI 76, TRIV 78].

Petri net models of parallel and asynchronous systems have been extended to include stochastic aspects [MOLL 81, MOLL 82, MARS 83, GEIS 83]. Molloy established an isomorphism between stochastic Petri nets and homogeneous Markov processes, thus making it possible to apply available techniques in the analysis of Markov processes for the analysis of stochastic Petri net models.

In recent years, two new forms of program representations known as dataflow and reduction languages are attracting attention among researchers in USA, UK, France, Japan and other countries. The literature is abundant with proposals for new computer systems based on dataflow and reduction (demand driven) principles (for example [DENN 74, DENN 75, DENN 80]), programming languages (for example, [ACKE 79, ACKE 82, ARVI 78]), distributed computing based on dataflow [MEAS 82], simulation and modeling using dataflow graphs [KAVI 83, SRIN 83, GAUD 84].

In this paper we propose a different model for parallel computations that is based on dataflow principles. At UCLA, Karplus and Ercegovac are studying the use of dataflow concepts for high speed digital simulation [GAUD 84]. Srini used extended

dataflow graphs for analyzing the architecture of highly concurrent computer systems [SRIN 83]. At UTA, we have developed a dataflow simulator (DFDLS [KAVI 83]) that can be used to model computer systems (both hardware and software). The simulator allows hierarchical (or variable resolution [GAUD 84]) and modular refinement of nodes in dataflow graphs; that is, a node can be a primitive dataflow operator or a dataflow subgraph. Modules written in other languages such as Pascal can be used to represent nodes.

Much of the research in dataflow processing has dealt with defining the functionality, designing instruction level architecture, or specifying programming methodologies. This has not made urgent the formalization of the dataflow model itself. Formalization is necessary, however, in relating dataflow to other computation models, discovering properties of specific instances of dataflow graphs (e.g., absence of deadlock), and in performance evaluations. Formalization also makes possible the utilization of dataflow graphs as abstract models of computation analogous to Turing machines and Petri nets. It is from this motivation that the present work stems. A formal set-relationship definition of a specific kind of dataflow graph (an uninterpreted dataflow graph) is presented. An illustration of its use in describing properties is given in the form of a liveness theorem. We anticipate further studies that illustrate its utility as a computation model and in performance analysis.

In the remainder of this paper, we introduce dataflow graphs and how they can be used to model computer systems including parallel processors and even dataflow machines themselves. A dataflow formalism will be introduced for representing dataflow graphs. The similarity between Petri nets (particulary, free choice nets) and our dataflow graph models should be noted. We will also introduce stochastic aspects into our models such that performance and reliability of dataflow graph models of computer systems can be analyzed.

II. THE DATAFLOW CONCEPT

A dataflow graph is a bipartite directed graph where the two types of nodes are called links and actors. Actors describe operations while links receive data from a single actor and transmit values to one or more actors by way of arcs. 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 values on input arcs and produces results on output arcs. Arcs can be control or data arcs. In the case of control arcs, the tokens are of the type Boolean (true or false); for data arcs, the tokens can 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 control input arcs. For a complete description of dataflow concepts, the reader is referred to [TREL 82].

The dataflow model of computation is neither based on memory structures that require inherent state transitions nor depend on history sensitivity. Thus it eliminates some of the inherent von Neumann pitfalls described by Backus [BACK 78]. However, abstract dataflow models, data driven systems and languages based on such systems must encompass sufficiently powerful and general mechanisms to make expression of complex algorithm constructs concise and natual to be of full utility. Dataflow systems must be

capable of efficiently supporting the functionality commonly found in conventional computer systems if they are to be accepted into general use. Support for such features as controlled sharing of global data, virtualization and communication between processes must be addressed to allow data driven systems to be applied to the large class of problems commonly addressed by programmers and system designers. In his dissertation, Landry [LAND 81] surveyed some of the extensions made to the basic dataflow model to satisfy these requirements.

Firing Semantic Set (FSS): The basic firing rule adopted by most dataflow researchers require 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 dataflow graphs represent primitive operations. However, if the nodes are complex procedures, or dataflow subgraphs, a more generalized firing control for both input and output arcs is required. Landry [LAND 81] discussed a comprehensive firing semantic specification for dataflow nodes. The firing semantic set refers to a subset of input arcs that must contain tokens to enable the node. Similarly, a subset of output arcs are required to be empty. When the node is fired, tokens are removed from the input firing semantic set of arcs and tokens are placed on the output firing semantic set of arcs. For different instances of execution of a node, the firing sets may be different, thus introducing nondeterminacy. Our formal

model of dataflow graphs incorporates this generalized firing specification.

SOME DATAFLOW GRAPH MODELS OF COMPUTER SYSTEMS

Here we show how dataflow graphs can be used to model computer systems. The examples include a simple conventional control unit, a fault-tolerant computer based on von Neumann architecture that emphasizes deterministic synchronization among the concurrent tasks and the MIT dataflow processor. Only high-level models are shown in this paper. However, nodes in the graphs can be expanded to represent instruction-set level architecture.

1. Dataflow Model of SIFT: SIFT is an ultrareliable avionics computer built by NASA-Langley Research Center to study the possibility of completely automating commercial aircrafts [WENS 78]. SIFT has chosen to achieve fault tolerance using sottware. The SIFT system consists of several BDX 930 computers built by Bendix Corporation. The software consists of tasks classified as application tasks and executive tasks. Executive tasks are responsible for scheduling tasks on SIFT processors, detecting errors, reconfiguring the system, synchronizing the various processors (to within 50 microseconds) and input/output from the sensors. Examples of application tasks are yaw damper, the pitch inner loop and the roll inner loop.

For the purpose of maximizing determinism in scheduling, computations in SIFT are conducted in regular time segments called frames and subframes. At present, the length of a subframe is 3.2 milliseconds and there are 50 subframes to a frame. Each task runs at a regular rate, the fastest running once every frame and otners running once every other frame or with even longer period. Tasks that run once every frame are scheduled to specific subframes while the slower tasks are scheduled during the remaining subframes.

During a subframe, the status of the processor is saved, and the values produced by the task are broadcast to other processors. The values received from other processors are voted. If an error is detected, the processor marks the faulty processor in its error table and when the number of errors marked for a processor exceeds a threshold, an error is reported. The system is then reconfigured. Fig. 1 shows a high level dataflow model of a SIFT processor. The nodes in the graphs can be further expanded into dataflow subgraphs. Or, the actual code representing the tasks can be used to represent the nodes.

2. Dataflow Model of a Simple Computer System: Baer [BAER 80, p 71] gave a Petri net model representing the control flow in the execution of an instruction in a single accumulator ALU. Fig. 2 shows a dataflow equivalent of the Petri net given by Baer. The nodes in the graph are intentionally named by the events in order

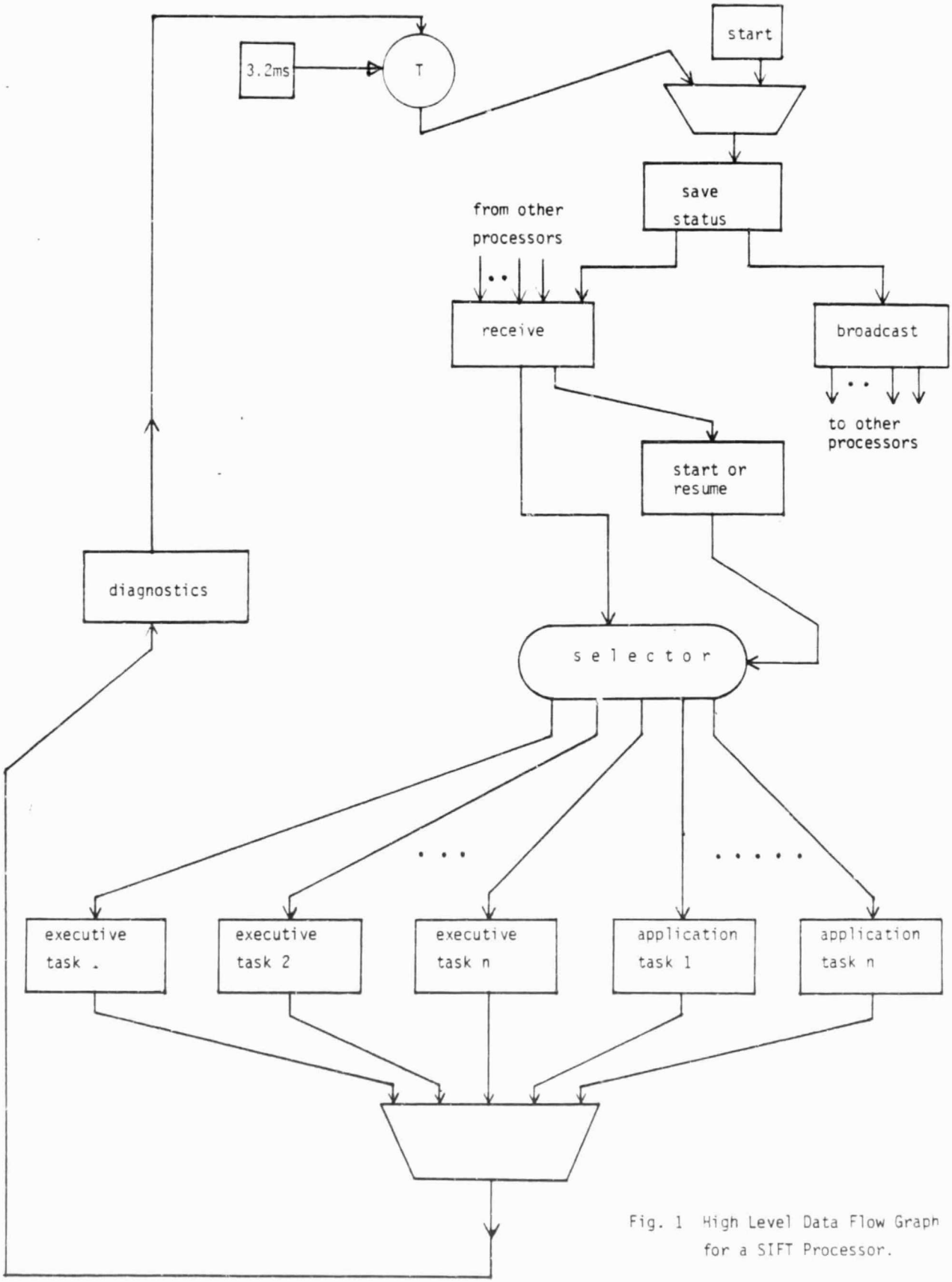


Fig. 1 High Level Data Flow Graph for a SIFT Processor.

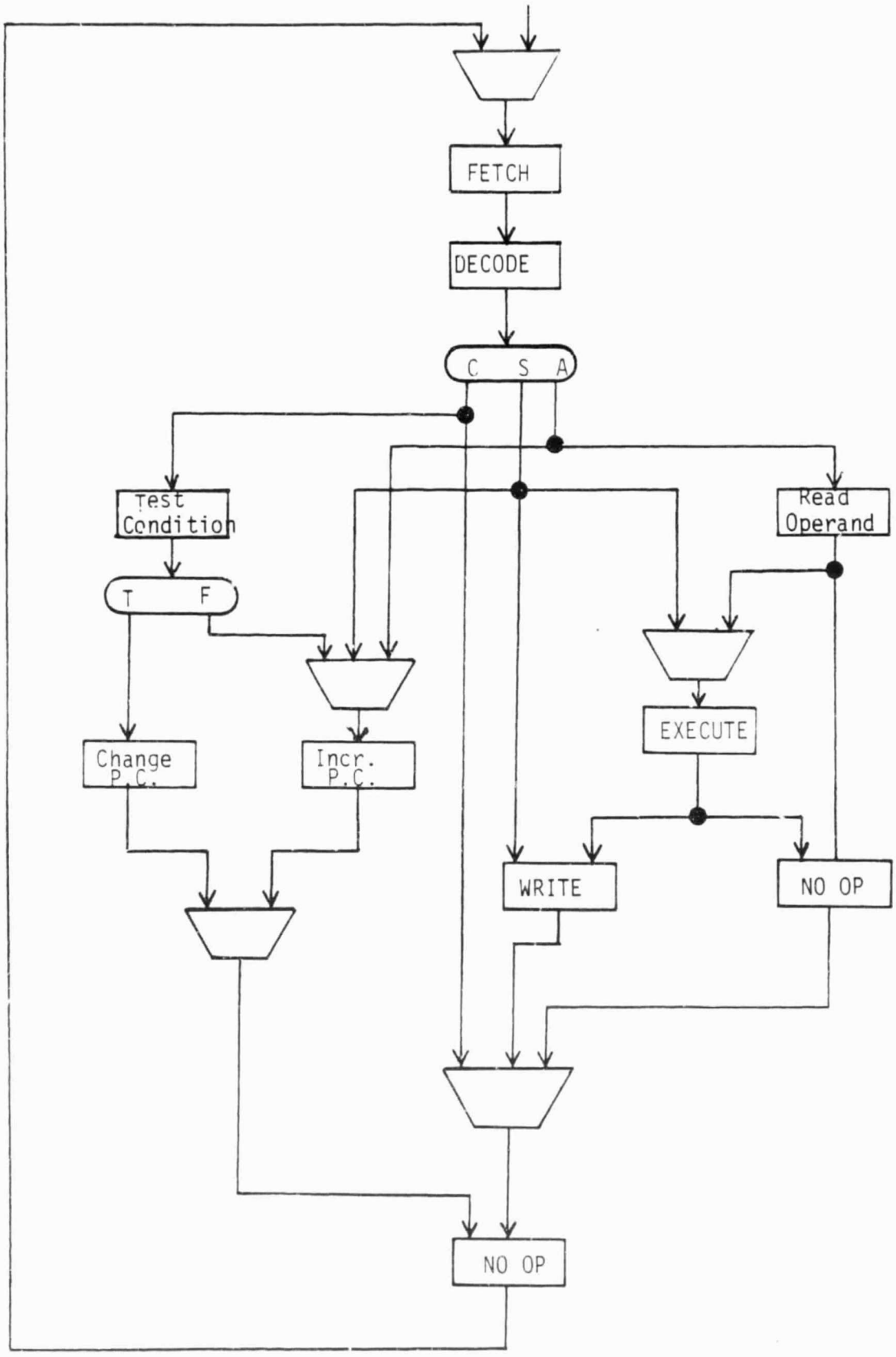


Fig 2 DATAFLOW MODEL OF A COMPUTER SYSTEM

to facilitate interpretation. For the sake of clarity, control arcs and links are not drawn; their presence can be easily seen at the selection (oval shaped) nodes. Funnel shaped actors are enabled when one of its input arcs receives a token. Upon firing, the funnel passes the token on its input arc on to the output arc. Tokens on more than one input arc leads to multiple firings of funnel.

3. Dataflow Model of a Dataflow Computer System: Fig. 3 shows a high-level block diagram of MIT dataflow processor [DENN 75]. A high level dataflow graph model of MIT system is shown in Fig. 4. Memory unit can produce one or more tokens on its output arcs, depending on the number of instructions enabled. The arbitration network schedules one instruction packet at a time; this is done in order to simplify the dataflow graph. Control and Distribution network also receive multiple inputs, but work on one packet at a time. The packet routing networks (arbitration, distribution and control networks) can be represented as dataflow subgraphs detailing how the pakets are routed in the MIT dataflow system.

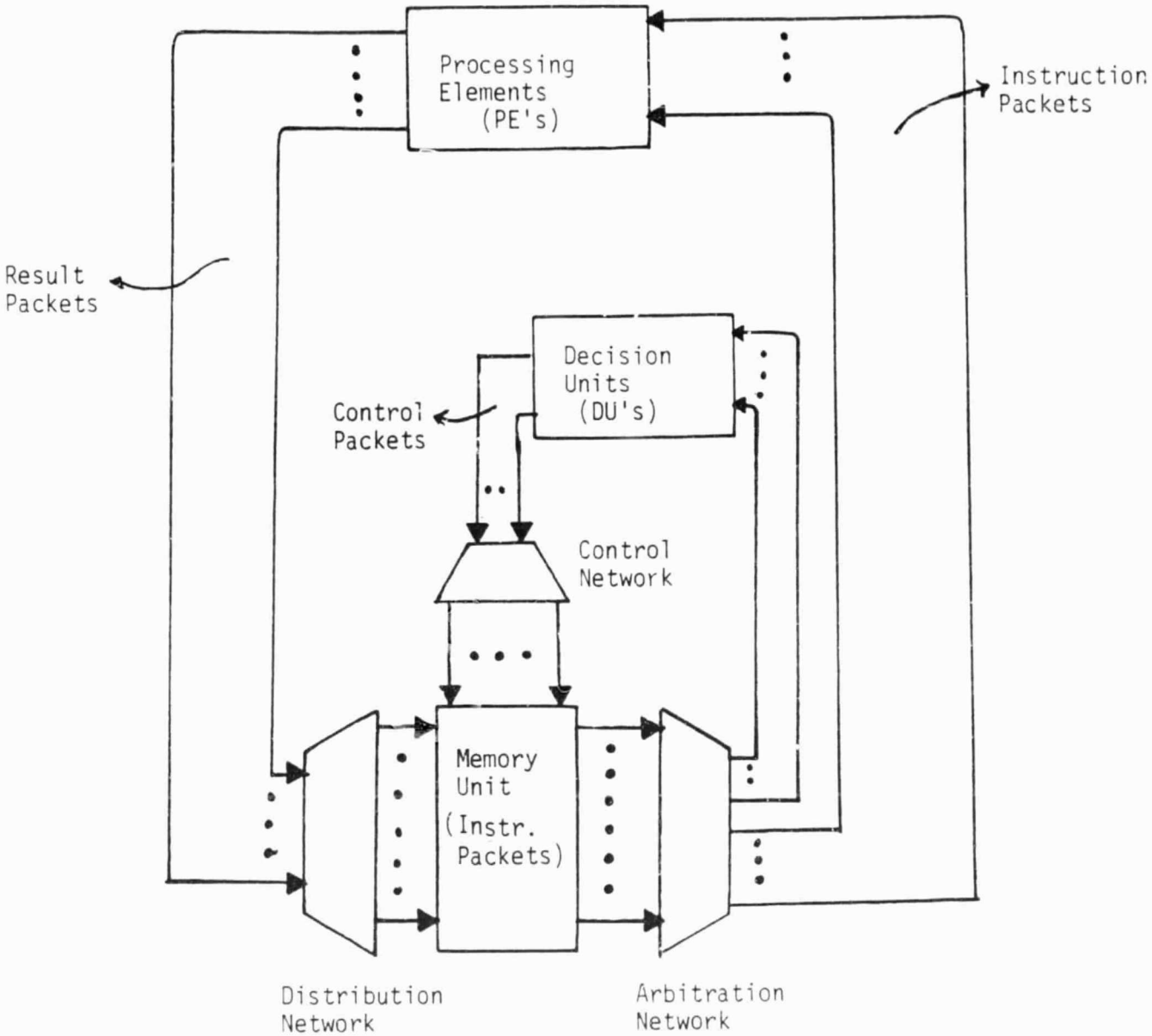


Fig. 3. MIT Data Flow Processor

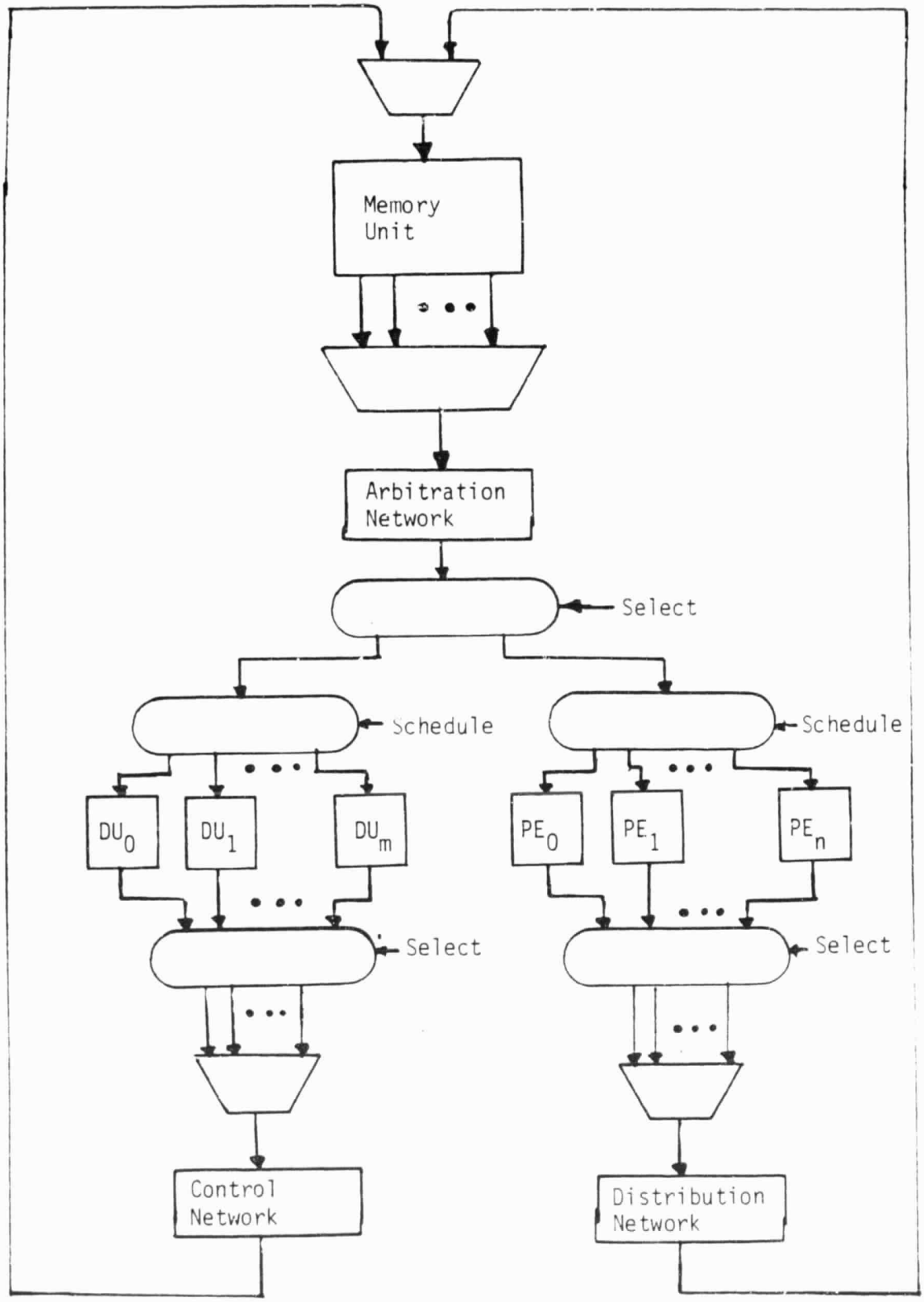


Fig. 4. Dataflow Graph Model for MIT Data Flow Processor

III. DATAFLOW FORMALISM

Definition 1: A Dataflow graph is a bipartite graph where the two types of nodes are called Actors and Links

$$G = < A \cup L, E > \quad (1)$$

Here, A is the set of actors

L is the set of links

E is the set of edges

$$E \subseteq (A \times L) \cup (L \times A) \quad (2)$$

Associated with each actor is a function f that describes the action performed by that node. Arcs are labeled by the type of the tokens carried by the arcs.

S is a non-empty set of links called starting set (input links)

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

T is a non-empty set of links called terminating set (output links)

$$T = \{ l \in L \mid (l,a) \notin E \quad \forall a \in A \} \quad (4)$$

The set of input links to an actor a and output links from an actor a are denoted as I(a) and O(a).

$$I(a) = \{ l \in L \mid (l,a) \in E \} \quad (5)$$

$$O(a) = \{ l \in L \mid (a,l) \in E \} \quad (6)$$

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

If a is an actor then $I(a)^*$ designates the set of links that are input links to a, or input links to the actors that feed the input links of a, and so on (transitive closure).

$$I(a)^* = \{l \in L \mid l \in I(a), \text{ or } l \in I(I(I(a))),..\} \quad (7)$$

Similarly, $O(a)^*$ defines all set of links that are output links from a, or output links from the actors fed by the output links of a, and so on.

$$O(a)^* = \{ l \in L \mid l \in O(a) \text{ or } l \in O(O(O(a))),...\} \quad (8)$$

If $B \subseteq A$ is a subset of actors than I(B) and O(B) define the set of links that are input links and output links of actors belonging to B, respectively.

$$I(B) = \{ l \in L \mid l \in I(b) \text{ for } b \in B \}$$
$$O(B) = \{ l \in L \mid l \in O(b) \text{ for } b \in B \} \quad (9)$$

Definition 2: For a dataflow graph the following conditions are true.

$$\begin{aligned} |I(a)| &> 0 && \text{for all actors } a \in A \\ |I(l)| &= 0 \text{ or } 1 && \text{for all links } l \in L \\ |O(a)| &> 0 && \text{for all actors } a \in A \\ |O(l)| &= 0 \text{ or } 1 && \text{for all links } l \in L \end{aligned} \quad (10)$$

Although this definition seems to lack generality, we have discovered that most dataflow graphs can be rewritten to fit this condition. This restriction allows us to isomorphically map dataflow graphs into free-choice Petri nets and free-choice nets into dataflow graphs [BUCK 84].

Uninterpreted Dataflow Graphs: For the purpose of studying the performance of dataflow graph models of computer systems, we introduce uninterpreted dataflow graphs, in that the actual meaning of the functions performed by the actors and the data carried by the arcs is not relevent. The presence of tokens on arcs are used only as triggering signals to enable nodes.

Note: We will use the term dataflow graph to mean uninterpreted dataflow graph throughout this paper.

Definition 3: A marking is a function

$$M: \quad L \longrightarrow \{ 0,1 \} \tag{11}$$

A link l 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 starting set of links contain tokens.

A terminal marking M_t is a marking in which a subset of the terminating set of links contains tokens.

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

$$F_1 (a,M) \subseteq I(a)$$
$$F_2 (a,M) \subseteq O(a) \tag{12}$$

The input firing semantic set F_1 refers to the subset of input links that must contain tokens to enable the actor; the output firing semantic set F_2 refers to the subset of links that receive tokens when the actor is fired.

Definition 4: A <u>firing</u> is a partial mapping from markings to markings.

An actor a is <u>firable</u> at a marking M if the following conditions hold

$$M(1) = 1 \quad \text{for all } 1 \in F_1(a,M)$$
$$M(1) = 0 \quad \text{for all } 1 \in F_2(a,M) \qquad (13)$$

When the actor is fired, tokens from the firing set $F_1(a,M)$ 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 in marking M can be derived as follows.

$$\begin{aligned} M'(1) &= 0 && \text{if } 1 \in F_1(a,M) \\ &= 1 && \text{if } 1 \in F_2(a,M) \\ &= M(1) && \text{otherwise} \end{aligned} \qquad (14)$$

Such 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 node firing rules are defined.

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

$$F_1(a,M) = I(a) \quad \text{for all } M \qquad (15)$$

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

$$|F_1(a,M)| = 1 \quad \text{for all } M \qquad (16)$$

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

$$F_1(a,M) \subseteq I(a) \quad \text{for all } M \tag{17}$$

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

$$|F_2(a,M)| = 1 \quad \text{for all } M \tag{18}$$

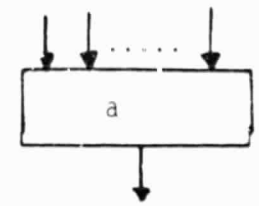
Distributive: When the actor fires, all the output links receive tokens. That is,

$$F_2(a,M) = O(a) \quad \text{for all } M \tag{19}$$

The graphical representations of these possibilities are shown in Fig. 5.

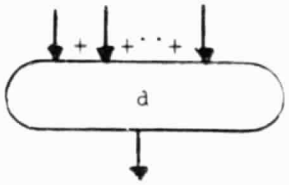
Non-Deterministic Firing Semantics: Since, in our study, uninterpreted dataflow graphs are used for analyzing performance or reliability by applying stochastic methods, F_1, and F_2 are made non-deterministic; for different instances of execution of a node, the firing semantic sets may be different. This eliminates the need for control arcs in dataflow 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 [DENN 74] shown in Fig. 6 (a) will be replaced by Fig. 6 (b). The firing sets F and F are defined as follows.

$$\begin{aligned} F_1 &= I(T) = l_1 \\ F_2 &= O(T) = l_2 \quad \text{with probability } p \\ &= \quad\quad\quad\quad \text{with probability } 1-p \end{aligned} \tag{20}$$



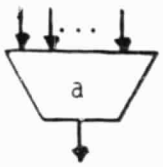
a) Conjunctive

$F_1(a,M) = I(a)$



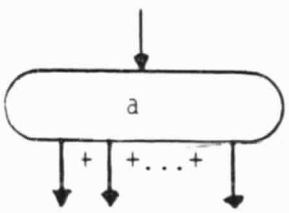
b) Disjunctive

$|F_1(a, M)| = 1$



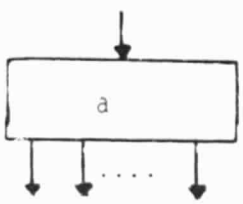
c) Collective

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



d) Selective

$|F_2(a,M)| = 1$

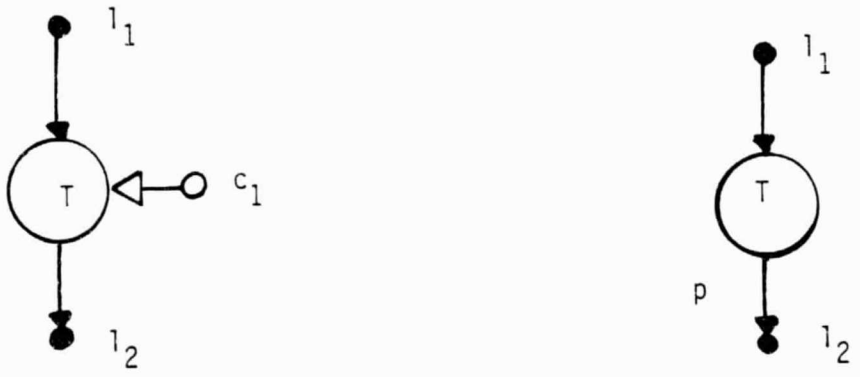


e) Distrubutive

$F_2(a,M) = O(a)$

Fig. 5. Firing Rules

The probability p depends on the frequency of having a TRUE value on the control arc of the T-gate.



(a). T-gate (b). nondeterministic gate

FIG. 6 Nondeterministic Firing Semantics

$P[F_1(a,M)]$ is the probability that in a marking M, $F_1(a,M)$ is the input firing semantic set; this probability function 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. Similarly, $P[F_2(a,M)]$ defines the probability function on the output firing semantic sets; $P[F_2(a,M)]$ is the probability that when the actor a is fired in M, the subset of output links given by $F_2(a,M)$ will receive tokens.

Definition 5: Firing Sequence σ is a sequence of markings resulting from the firing 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 that 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

$$M \xrightarrow{\sigma} M'$$

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.

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

Definition 6: A dataflow graph is said to terminate properly if there exists a firing sequence that leads an initial marking M to a terminal marking M_t.

DEADLOCKS AND LIVENESS IN DATAFLOW GRAPHS

A node that cannot fire in a dataflow graph that models a component of a computer system would seem anamolous and we should be able to identify such situations. Deadlocks can be avoided in dataflow machines by providing feedback to actors using control

tokens [MISU 75]. Since our model omits control tokens, it is necessary to derive the necessary and sufficient conditions for liveness and absence of deadlocks in dataflow graphs. This is similar to the concept of liveness in Petri nets which has been found valuable in modeling deadlock-freeness of operating systems [PETE 77, AGER 79].

Definition 7: An actor a is <u>potentially firable</u> in a marking M, if there exists a marking $M' \in \vec{M}$ such that a is enabled in M', except when M is a terminal marking.
An actor a is said to be <u>blocked</u> in a marking M, if for all markings $M' \in \vec{M}$, a is not enabled.

An actor a is <u>live</u> in M if a is potentially firable in all markings $M' \in \vec{M}$, except when M' is a terminal marking.
A dataflow graph is live in a marking M if a non-empty subset of actors $C \subseteq A$ are live. Liveness in Petri nets require that all transitions be live, implying that C = A. For the present, we will allow a partial liveness in dataflow graphs.
A dataflow graph is blocked in a marking M if the graph is not live in M. That is $C = \phi$.
A dataflow graph is <u>deadlocked</u> if the graph is blocked in a marking $M' \in \vec{M}_0$ where M_0 is an initial marking, except when M' is a terminal marking. We will consider a dataflow graph deadlock-free if it terminates properly.
For some actor $a \in A$ and a forward marking class $\vec{M}$ of some

marking M,

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

$$F_2(a, \vec{M}) = \{ l \in L \mid l \in F_2(a, M') \text{ where } M' \in \vec{M} \text{ and } a \text{ is firable in } M' \} \quad (23)$$

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

Theorem: Suppose $B \subseteq A$ be the subset of actors that are blocked in a nonterminal marking M. Then the dataflow graph is live in M if and only if

$$F_1(A-B, \vec{M}) = F_2(A-B, \vec{M}) \quad (24)$$

Proof: Necessrary Condition: If the dataflow graph is live then equation (24) holds.

This condition is proved by contradiction.

Suppose that the dataflow graph is live, but

$$F_1(A-B, \vec{M}) \neq F_2(A-B, \vec{M})$$

Case 1: There exists a link $l \in F_1(A-B, \vec{M})$ but $l \notin F_2(A-B, \vec{M})$. Since only actors in A-B are live (and hence firable in $\vec{M}$), only $F_2(A-B, \vec{M})$ can contain tokens in all markings $M' \in \vec{M}$. This implies that there exists an actor $b \in A-B$ and a marking $M' \in \vec{M}$ such that the link $l \in F_1(b, M')$ does not contain a token. The

actor b is blocked which is contrary to the assumption that actors in A-B are live.

Case 2: There exists a link $l \in F_2(A\text{-}B, \vec{M})$ but $l \notin F_1(A\text{-}B, \vec{M})$. Since only actors in A-B can fire in all markings $M' \in \vec{M}$, only tokens on links in $F_1(A\text{-}B, \vec{M})$ are consumed.

Since $l \notin F_1(A\text{-}B, \vec{M})$, l will contain an unconsumed token. This would lead to an actor $b \in A\text{-}B$ and a marking $M' \in \vec{M}$ such that $l \in F_2(b, M')$, to be blocked. This is again contrary to the assumption that actors in A-B are live.

Hence, $F_1(A\text{-}B, \vec{M}) = F_2(A\text{-}B, \vec{M})$

<u>Sufficient Condition:</u> If equation (24) holds then the dataflow graph is live.

We will prove this condition by contradiction. Suppose that

$$F_1(A\text{-}B, \vec{M}) = F_2(A\text{-}B, \vec{M})$$

but the dataflow graph is not live. Let $b \in A\text{-}B$ be an actor that is not live in M. That is, there exists a marking $M' \in \vec{M}$ such that b is not potentially firable in M'.

Case 3: The actor b is not firable because, some link $l \in F_1(b,M')$ does not contain a token. Since $F_2(A\text{-}B, \vec{M})$ are the only set of links that can contain tokens in $\vec{M}$, $l \notin F_2(A\text{-}B, \vec{M})$. This is a contradiction since we assumed that equation (24) holds.

Case 4: The actor b is not firable in M' because, for some link $l \in F_2(b,M')$, l contains an unconsumed token. Since only tokens in $F_1(A\text{-}B, \vec{M})$ can be consumed, $l \notin F_1(A\text{-}B, \vec{M})$. This is again contrary to the assumption that equation (24) holds.

Thus, a dataflow graph is live if and only if equation (24) holds.

$$F_1 (A\text{-}B, \vec{M}) = F_2 (A\text{-}B, \vec{M})$$

Corollary 1: If $A\text{-}B \neq \phi$, then $F_2 (A\text{-}B, \vec{M}) \neq \phi$.
If A-B is not null then $F_1(A\text{-}B, \vec{M})$ is not null, since an actor is enabled only if at least one of its input links contain a token. Thus, $F_2 (A\text{-}B, \vec{M})$ is not null.

Corollary 2: If the input firing semantic set F for all arcs is conjunctive and the output firing semantic set F for all arcs is distributive, that is,

$$F_1 (b,M') = I(b) \quad \text{for all } b \in A\text{-}B \text{ and all } M' \in \vec{M}$$
$$F_2 (b,M') = O(b) \quad \text{for all } b \in A\text{-}B \text{ and all } M' \in \vec{M}$$

then, the dataflow graph is live if and only if

$$I(A\text{-}B)^* = O(A\text{-}B)^*$$

The asterisk implies a transitive closure on the input and output links of the actors in A-B.

Remarks: The above theorem does not guarantee that the number of tokens flowing through a dataflow graph remains constant, but that the tokens be conserved over a firing sequence. This allows for some actors consuming more tokens than they produce while other producing more than they consume. Thus in a non-terminating deadlock-free dataflow graphs all firing sequences are repeatable. A Markov process for dataflow graphs can be defined by associating

states with markings. Markings that enable blocked nodes in the dataflow graphs lead to dead states (or failed states).

Dataflow graphs as defined here are <u>safe</u>, since at most one token can be present on a link. Some dataflow researchers have relaxed the firing rules to allow multiple tokens flowing on arcs; arcs are treated as pipelines of tokens. This would lead to <u>k-bounded</u> dataflow graphs where k is the maximum number of tokens that can flow on an arc. Equation (11) must be rewritten as

$$M: \quad L \longrightarrow R^{+} \cup \{0\} \tag{25}$$

equation (13) must be rewritten as

$$\begin{aligned} M(l) &\geqslant 1 \quad \text{for all } l \in F_1(a,M) \\ M(l) &< k \quad \text{for all } l \in F_2(a,M) \end{aligned} \tag{26}$$

and equation (14) as

$$\begin{aligned} M'(l) &= M(l) - 1 \quad \text{if } l \in F_1(a,M) \\ &= M(l) + 1 \quad \text{if } l \in F_2(a,M) \\ &= M(l) \quad \text{otherwise} \end{aligned} \tag{27}$$

The dataflow liveness theorem can be extended to account for k-boundedness. However, we have decided to map dataflow graphs into free-choice Petri nets and use safeness and liveness properties of Petri nets.

IV. CONCLUSIONS

In this paper we have introduced a formal definition of dataflow graphs. Necessary and sufficient conditions for liveness in dataflow graphs are derived. The uninterpreted dataflow graphs can be used to analyze performance and reliability of computer systems including dataflow computers. Stochastic properties can be incorporated into dataflow graphs. We are in the process of deriving isomorphic mappings between Petri nets and uninterpreted dataflow graphs. Such mappings would enable us in carrying mathematical formulations available for Petri nets over to dataflow graph models.

V. REFERENCES

ACKE 79. W. B. Ackermann. "Data Flow Languages", Proc. 1979 NCC, (New York), pp 1087-1095.

ACKE 82. W. B. Ackermann. "Data Flow Languages", IEEE Computer, Feb. 1982, pp 15-25.

AGER 79. T. Agerwala. "Putting Petri Nets to Work", IEEE Computer, Dec. 1979.

ARVI 78. Arvind, K.P. Gostelow and W. Pflouffe. "An Asynchronous Programming Language and Computing Machine", TR 114a, Dept. of Inf. and Comp. Sci., UC-Irvine, Dec. 1978.

BACK 78. J. Backus. "Can Programs be Liberated From von Neumann Style? A Functional Style and Its Algebra of Programs", CACM, Aug. 1978, pp 613-641.

BAER 80. J.L. Baer. Computer Systems Architecture, Computer Science Press, 1980.

BUCK 84. B.P. Buckles, K.M. Kavi and U.N. Bhat. "Isomorphism Between Petri Nets and Dataflow Graphs", In Preparation.

DENN 74. J.B. Dennis. "First Version of Data Flow Procedural Language", Lecture Notes in Computer Science, Vol 19, Springer-Verlag, 1974.

DENN 75. J.B. Dennis and D.P. Misunas. "A Preliminary Architecture for a Basic Data Flow Processor", Proc. 2nd Annl. Symp. on Comp. Arch., (Houston), 1975, pp 126-132.

DENN 80. J.B. Dennis. "Data Flow Supercomputers", IEEE Computer, Nov. 1980, pp 48-56.

GAUD 84. J.L. Gaudiot and M.D. Ercegovac. "Performance Analysis of Data Flow Computers with Variable Resolution Actors", Proc. 4th Intl. Conf. Distr. Comp. Syst., (San Francisco) May 14-18, 1984, pp 2-9.

GEIS 83. R. Geist, K.S. Trivedi, J.B. Dugan and M. Smotherman. "Design of Hybrid Automated Reliability Predictor", Proc. IEEE/AIAA Digital Avionics Syst. Conf., Oct. 1983, pp 16.5.1-16.5.8.

KARP 66. R.M. Karp and R.E. Miller. "Properties of a Model for Parallel Computations: Determinacy, Termination, Queuing", SIAM J. Appl. Math., Vo. 14, No. 6, Nov. 1966, pp 1390-1411.

KAVI 83. K.M. Kavi. "Data Flow Modeling Techniques", Proc. IASTED Intl. Symp. Simulation and Modeling, (Orlando), Nov. 9-11, 1983, pp 1-4.

KLEI 76. L. Kleinrock. "Queuing Systems. Vo. 2. Computer Applications", John Wiley & Sons, 1976.

LAND 81. S.P. Landry. "System Oriented Extension to Data Flow", Ph.D. Dissertation, Dept. of CS, USL (Lafayette, LA), May 1981.

MARS 83. M.A. Marson, G. Balbo and G. Conte. "A Class of Generalized Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems", Proc. ACM SIGMETRICS Conf. Measurment and Modeling, 1983.

MEAS 82. M. Measures, B.D. Shriver and P.A. Carr. "A Distributed Operating System Based on Data Flow Principles", Proc. of COMPCON, Sept. 1982, pp 106-115.

MILL 73. R.E. Miller. "A Comparision of Some Theoretical Models of Parallel Computation", IEEE Tr. Comp., Vol. C-22, No. 8, Aug. 1973, pp 710-717.

MISU 75. D.P. Misunas. "Deadlock Avoidance in Data Flow Architecture", Proc. Symp. Automatic Computation and Control, (Milwaukee, WI), April 1975, pp 337-343.

MOLL 81. M.K. Molloy. "On the Integration of Delayed Throughput Measures in Distributed Processing Models", Ph.D. Dissertation, UCLA, 1981.

MOLL 82. M.K. Molly. "Performance Analysis Using Stochastic Petri Nets", IEEE Tr. Comp. Vol. C-31, No. 9, Sept. 1982, pp 913-917.

PETE 77. J.L. Peterson. "Petri Nets", ACM Comp. Surys., Sept. 1977, pp 223-252.

SRIN 83. V.P. Srini and J.F. Asenjo. "Anaysis of Cray 1S Architecture", Proc. 10th Intl. Symp. on Comp. Arch., (Stockholm) June 13-17, 1983, pp 194-206.

TREL 82. P.C. Treleaven, D.R. Brownbridge and R.P. Hopkins. "Data Driven and Demand Driven Computer Architecture", ACM Comp. Surys., March 1982, pp 93-143.

TRIV 78. K.S. Trivedi. "Analytical Modeling of Computer Systems", IEEE Computer, Oct. 1978, pp 38-56.

WENS 78. J.H. Wensley, et. al. "SIFT: Design and Analysis of a Fault-Tolerant Computer for Aircraft Control", Proc. IEEE Oct. 1978.

APPENDIX 2

SIMULATION OF COMPUTER SYSTEMS USING A DATAFLOW SIMULATOR DFDLS

SIMULATION OF COMPUTER SYSTEMS USING A DATAFLOW SIMULATOR (DFDLS)

Krishna M. Kavi and Robert M. Boyd

Dept. of Computer Science and Engineering
P.O. Box 19015
The University of Texas at Arlington
Arlington, Texas 76019-0015

ABSTRACT

In this paper we describe a dataflow simulator that can be used to simulate computer system as dataflow graphs and test the functionality of the modeled system. High level dataflow graph models of three different computer system are used to illustrate our approach. DFDLS provides the computer systems designer with a CAD environment that enables highly parallel and complex systems to be defined and tested at all levels.

Keywords: Dataflow Graphs, Dataflow Computers, Simulation and Modeling.

SIMULATION OF COMPUTER SYSTEMS USING A DATAFLOW SIMULATOR (DFDLS)

Krishna M. Kavi and Robert M. Boyd
The University of Texas at Arlington

ABSTRACT

In this paper we describe a dataflow simulator that can be used to simulate computer system as dataflow graphs and test the functionality of the modeled system. High level dataflow graph models of three different computer system are used to illustrate our approach. DFDLS provides the computer systems designer with a CAD environment that enables highly parallel and complex systems to be defined and tested at all levels.

Keywords: Dataflow Graphs, Dataflow Computers, Simulation and Modeling.

1. INTRODUCTION

In recent years dataflow models, dataflow languages and computer systems based on dataflow principles are attracting much attention in the United States, Japan and other countries. The major motivation behind the recent surge in dataflow architectures is the desire to enhance performance. It is believed that next generation computer systems will be based on non-von Neumann model of computation, like dataflow, as such systems can be designed to deliver billions of operations per second [TREL 82a]. The second motivation is the desire to exploit VLSI techniques in the design of computers. Because of the regularity of the processing elements and communication circuits, dataflow architectures are suitable for VLSI implementation.

Much of the research in dataflow processing has dealt with defining the functionality, designing instruction level architecture, or specifying programming methodologies. This has not made urgent the formalization of the dataflow model itself. Formalization is necessary, however, in relating dataflow to other computation models, discovering properties of specific instances of dataflow graphs, in simulation and performance evaluation. In a companion paper [KAVI 84], we have presented a formal definition of dataflow graphs.

In this paper we present the application of dataflow graph models in the simulation of computer systems. Because of the hierarchical nature and the modularity of dataflow graphs, both

software tasks and hardware units can be modeled in a uniform way using dataflow graphs [KAVI 83].

1.1. Simulation of computer systems:

Conventional simulation models fall into one of two types: process-oriented and event-driven. Process-oriented simulation consists of modules (programs) that define attributes and activities of the system. A process description comprises declarations, computations and control statements, and sequencing statements. One type of sequencing statement is that used to represent activity execution times such as the "hold(t)" statement of SIMULA. The processes are scheduled, suspended and reactivated in accordance to the process description. In event-driven simulation, all events are ordered by their scheduled times. The next event in the list is executed and the system clock is updated to the time of the event. Most logic simulations are event-driven.

Simulation of computer systems can be at one of four levels: circuit level, gate level, register-transfer level or system level. Each of the levels employs models which are simplifications of those of the preceding levels, both in quantitative terms and in terms of behavior [BREU 72]. Depending on the level of detail, the system is simulated at one of the above four levels. The system being simulated is modeled using a simulation language (e.g. SIMSCRIPT, GPSS). Typically, the model consists of input variables, output variables and the function which describes the transformation of inputs into outputs.

Dataflow languages and models can be effectively used to

simulate a logic system. In its basic form, a dataflow model consists of graphs with nodes that describe the function of a unit and arcs which are inputs to a node or outputs from a node. The main feature of a dataflow model is its hierarchical nature: a node in the model can be a simple node or a dataflow subgraph. Thus such models can be used to selectively change the level of simulation to any level of detail without modifying other parts of the simulation. Also, a dataflow model provides for modular simulation in the sense that any section of the unit can be modeled and tested without modeling the entire system. Dataflow simulation can be classified as a process-oriented simulation. A process description (or node description) comprises computational statements. Control statements are defined by special control nodes and arcs. Sequencing in dataflow is described by the structure of the graph (connections between nodes) and thus an inherent part of the model itself.

At UCLA, Karplus and Ercegovac ([KARP 82], [GAUD 84]) are attempting to use dataflow principles in high speed digital simulations. Srini [SRIN 83] has extended the basic dataflow to model supercomputers like Cray 1S. Another example of a hierarchical simulation tool is LOGOS [HEAT 72]. LOGOS is based on Petri-net concept. The LOGOS representational system uses two directed graphs, one for data flow and the other for control flow. The control flow graph sequences the data transformations and defines the control flow of the model.

1.2. The Dataflow Concept:

A dataflow program is a bipartite directed graph where the two types of nodes are called links and actors. Actors describe operations while links receive data from a single actor and transmit values to one or more actors by way of arcs. In its basic form, nodes (actors and links) are enabled for execution when all input arcs contain tokens and no output arc contains a value. An enabled node consumes values on input arcs and produces results on output arcs. For a complete description of dataflow concept the reader is referred to [TREL 82b].

The dataflow model of computation is neither based on memory structures that require inherent state transitions nor depend on history sensitivity. Thus it eliminates some of the inherent von Neumann pitfalls described by Backus [BACK 78]. However, abstract dataflow models, data driven system and languages based on such systems must encompass sufficiently powerful and general mechanisms to make expression of complex algorithm constructs concise and natual to be of full utility. Dataflow systems must be capable of efficiently supporting the functionality commonly found in conventional computer systems if they are to be accepted into general use. Support for such features as controlled sharing of global data, virtualization and communication between processes must be addressed to allow data driven systems to be applied to the large class of problems commonly addressed by programmers and system designers. In his thesis, Landry [LAND 81] surveyed some of the extensions made to basic dataflow model to satisfy these

requirements. Our formal model of dataflow graphs [KAVI 84] includes several of the exensions.

1.3. Extensions to Dataflow:

Some of the fundamental attributes of the basic dataflow model that makes it appealing for applications to certain class of problems also restricts its utility in others. One of the missing concepts in basic dataflow is memory. There is no easy way of modeling memory units. One can treat the entire memory as a token, but this is cumbersome.

Reference Tokens: At the University of Newcastle upon Tyne, the concept of updatable memory was introduced along with multi-threaded control flow [TREL 79]. This model incorporates all of the fundamental attributes of the dataflow but relaxes the requirement that only tokens which represent values flow along arcs. Instead, reference-tokens and control-tokens are introduced as primitive notions within the model. Reference-tokens are tokens containing names (addresses) used to access memory. Control-tokens are signals used to trigger successor instructions without transporting any data values.

The principle firing rule in this model requires that the combination of data tokens, reference tokens and control tokens specified as being mandatory for firing be present before execution be performed. These input tokens along with the specification of the instruction to be performed is referred to as an instruction-packet. Since the inclusion of reference-tokens implies that data values are stored separate from the instruction

packet, the model dictates that the supporting architecture include mechanisms for collecting reference values, determining when a node is enabled, retrieving data values, and forming instruction packets.

Structured Tokens: In the basic dataflow, the tokens are of scalar type. Almost all implementations of data-driven systems, however, permit tokens to be of more complex structures. In the MIT dataflow system, [DENN 74], data structures are maintained on a heap. The actors are designed so that their execution does not change values on the heap. Instead, whenever a new value is created, a node is added to the heap to represent the new value. New actors like source, append, select, exists are added in order to access the data structures on the heap.

Firing Semantic Set: The basic firing rule adopted by most dataflow researchers requires that all input arcs contain a token and that no tokens be present on the output arcs. When the dataflow graph is simple and the nodes are primitive actors like add or subtract, this provides for an adequate sequencing control mechanism. However, if the nodes are complex procedures or subgraphs, more generalized firing control for both input and output arcs is required. At the University of Southwestern Louisiana [LAND 81] and the University of Texas at Arlington [KAVI 84], a comprehensive and general firing semantic specifications are provided. For these models where the nodes are defined in a way where only a subset of the input arcs must contain tokens and only a subset of output arcs must not contain tokens, the concept

of firing semantic sets are introduced. The input firing set refers to the set of input arcs that will enable a node when tokens are present on each of the arc; the output firing set refers to the set of output arcs that will receive tokens when the node completes execution and hence should not contain tokens before the node is enabled. For different instances of execution of the node, the firing sets may be different, thus introducing non-determinacy and providing for interaction with the supporting environment.

2. DFDLS: An Overview

DFDLS (pronounced as daffodils) is an extensive data flow simulation written in Pascal and is currently available on both DEC 20 and VAX/780 computer systems. Simulation of computer systems, both hardware and software, can be performed using data flow concepts.

The major features of DFDLS are given below.

1. Tokens on data flow arcs can be structured data items. Existing data flow computers and languages permit only elementary data types like integer, real, and characters. DFDLS allows the Pascal data types. At this time, record data types are not processed, but will be included soon.

2. Firing set semantics can be specified in DFDLS. In the basic data flow, a node is enabled for execution only when all input arcs contain tokens and no tokens are present on the output arcs. These firing semantics are extended in the mandatory input values required for the node to execute.

3. Nodes in DFDLS can be data flow subgraphs. In most of the existing data flow systems only primitive functions such as ADD two numbers are permitted. In our system, a node can be a primitive function defined by the system, a data flow subgraph or a Pascal procedure provided by the user. These Pascal procedures are linked dynamically by the runtime environment.

4. The input language is very simple. DFDLS interprets simulation models expressed in our textual language. There is a one-to-one correspondence between the data flow graphs and the textual representation. Thus, data flow graphs can be translated directly into the input language. This also provides for graphical interface that can be designed at a later date.

5. Block structures and Recursion: the present implementation of DFDLS permits a restricted block structure in that, all names must be unique. Recursion is not allowed. However, we are in the process of extending DFDLS to allow more general nesting of blocks and recursion. Because of our data structures and modular design of DFDLS, this addition is straight forward. Separate descriptor tables and node tables will be created for each block and display stack will be used to implement recursion and static scopes.

2.1 BNF For DFDLS Language


```
<program>        ::= <node>  [ <node> ]
<node>           ::= <node_env>  <input_sect>  <output_sect>
<node_env>       ::= NODE:  <node_id> <type_sect> <FSS_sect>  <funct_sect>
<type_sect>      ::= empty  |  TYPE  <any valid PASCAL type section>
<FSS_sect>       ::= FSS:  [ STD  |
                             NON_STD  <FSS_spec>  [ <FSS_spec> ]
<arc_list>       ::= <arc_id>   [ <arc_id> ]
<funct_sect>     ::= FUNCTION:  [ DFG  <block_list>  |
                                  PAS  <NRM_id>      |
                                  LIB  <NRM_id>  ]
<block_list>     ::= <child_node>  [ <child_node> ]
<child_node>     ::= <node_id>
<NRM_id>         ::= NRM:  <program_id>
<program_id>     ::= <name of a user supplied PASCAL procedure
                      or system provided procedure>
<input_sect>     ::= INPUTS  <inputs>
<inputs>         ::= <input_arc>  [ <input_arc> ]
<input_arc>      ::= <arc_id>  :  <type>
<output_sect>    ::= OUTPUTS  <outputs>
<outputs>        ::= <output_arc>  [ <output_arc> ]
<output_arc>     ::= ARC  <arc_id>  :  <type>  <dest_list>
<type>           ::= <valid PASCAL type declared in TYPE section>
<dest_list>      ::= <dest_node>  [ <dest_node> ]
<dest_node>      ::= <node_id>
<node_id>        ::= <id>
<arc_id>         ::= <id>
<id>             ::= <any alphanumeric name of 10 characters or less>
```

2.2 Data Structures In DFDLS

The major data structures in DFDLS are the descriptor table (DT) and the node table (NT). The descriptor table describes the arcs in the data flow program while the node table keeps track of the nodes in a data flow program.

The entries in the descriptor table are described in FIG. 1.

```
DTE_Ptr = ^DTE;
DTE = Record
        Arc_Name : ID;
        Source_NTE : ID;  (* is an output of this NTE *)
   Type_Decl : Boolean; (*true if this DTE is a type declaration*)
   Num_Destinations : Integer;
        Dests_Remaining : Integer;
   This_Arc_Is : Arc_Type;
   Further_Desc : DTE_Ptr;
    Prev   : DTE_Ptr;
    Next   : DTE_Ptr;
    Memptr : Integer;
         Allocated : Boolean;
             Num_Record_Elements : Integer;
             Arr_Dims : Arr_Ptr;
End;  (* DTE *)


ID = Packed Array [1..Max_ID_Len] Of Char;

Arc_Type = (V_Char, V_Int, V_Real, V_Bool,
            V_Alfa, V_Array, V_Unknown, V_Record);

Arr_Ptr = ^Arr_Dim_Desc;
Arr_Dim_Desc = Record
   Stride : Integer;
   Lower  : Integer;
   Upper  : Integer;
   Dim_Ptr : Arr_Ptr;
End; (* Arr_Dim_Desc *)
```

FIG. 1. DESCRIPTOR TABLE ENTRIES

The entries in the node table are described in FIG. 2. A NTE (node table entry) contains the name of the node, the name of the parent node, a list of the children nodes, function description, firing semantic set, pointers to input arc descriptions (in DT), pointers to output arc descriptions (in DT) and destinations for each output arc. When the firing semantic set is satisfied, the node is in "ready" state, and in the "not-ready" state otherwise. If a node is a data flow graph and the internal nodes are currently executing, the state of the node is "children firing". When the node is scheduled for execution, the state is "firing".

For each input arc, the availability of a token on the arc is also maintained. The free state in the output arc description indicates that there exists no value on the output arc. A node can not fire if an output arc contains tokens and one or more destination nodes are using the tokens. This restriction will be relaxed later by treating arcs as queues. The tokens produced will be consumed in FIFO manner.


```
NTE_Ptr = ^NTE;
NTE = Record
  Node_ID : ID;
  Parent_Id : ID;
  Block_Head : Block_List_Ptr;  (* List of Children for a DFG *)
  Funct : Funct_Class;
  FSS_Type : FSS_Class;
  FSS_Head : FSS_Ptr;
  Input_Head : Input_Ptr;
  Output_Head : Output_Ptr;
  Status : Status_Class;
  Next : NTE_Ptr;
  Prev : NTE_Ptr;
  Case fType : Funct_Class  Of
     PAS,DFG : (NRM : ID);
     LIB     : (LNRM : Lib_Class);
End;  (* NTE *)
```

Fig. 2. NODE TABLE ENTRIES

```
FSS_Ptr = ^FSS_Rec;
FSS_Rec = Record;
     Arc_List_Head : Arc_List_Ptr;
     Next : FSS_Ptr
End;  (* FSS_Rec *)

Block_List_Ptr = ^Block_List_Rec;
Block_List_Rec = Record;    (* Child Node of a DFG *)
     Node_ID : ID;
     Next : Block_List_Ptr
End;  (* Block_List_Rec *)

Input_Ptr = ^Input_Rec;
Input_Rec = Record;
     Arc_Name : ID;
     Arc_Availability : Availability_Class;
     Next : Input_Ptr
End;  (* Input_Rec *)

Output_Ptr = ^Output_Rec;
Output_Rec = Record;
     Arc_Name : ID;
     Dests_Head : Dest_Ptr;
     Next : Output_Ptr
End;  (* Output_Rec *)

Dest_Ptr = ^Destination_Rec;
Destination_Rec = Record;
     Node_ID : ID;
     Next : Dest_Ptr;
End;  (* Destination_Rec *)

FSS_Class = (STD, NON_STD, Undefined_FSS);
Funct_Class = (DFG, PAS, LIB, Undefined_Funct_Class);

Arc_List_Ptr = ^Arc_List_Rec;
Arc_List_Rec = Record;
     Arc_Name : ID;
     Next : Arc_List_Ptr;
End;
```

FIG. 2. NODE TABLE ENTRIES (continued)

2.3 Major Modules In DFDLS

The NT-Manager is used to create, search, and find node table entries. One entry exists for each node defined in the data flow program.

The DT-Manager provides functions identical to those of the NT-Manager except the items being manipulated are data descriptors which describe the data tokens.

The Ready List Manager controls the operation of the ready list. It contains those nodes which are ready to be fired and provides functions to insert and delete nodes from the ready list in addition to returning the next node in the Ready List queue. Although a first-in first-out list structure is maintained, prioritized execution is planned in the future.

The Parser consumes the textual input language which describes the data flow graph. As the text is examined, data structures for nodes, data descriptors, and ready list entries are allocated and linked to existing ones.

The Node Manager controls the run-time portion of the simulation. It schedules the next node for execution and informs destination nodes that inputs are available. When the firing semantic set (FSS) has been satisfied, the node is inserted into the rear of the ready list.

The Memory Manager controls the allocation and deallocation of the tokens on one or more of the four heaps created for the primitive data types. More complex data structures are broken into their component forms which are stored in the heaps above.

The Library module provides all of the functions which are to be included with the simulator. Categories include unary and binary input functions as well as multiple input and special functions. The following is a list of library functions and the data types they can operate on.

2.4 Available Functions

Unary Functions

Name	Input	Output
NOT	Boolean	Boolean
NEG	Integer/Real	Integer/Real
SIN	Integer/Real	Real
COS	Integer/Real	Real
TAN	Integer/Real	Real
FIX	Real	Integer
INT	Real	Integer
FLOAT	Integer	Real
CHR	Integer/Real	Character
ORD	Character	Integer

Binary Functions

Name	Inputs	Output
SUB	Integer/Real	Integer/Real
DIV	Integer/Real	Integer/Real
MOD	Integer	Integer
EQ	Int/Real/Char/Bool	Boolean
NE	Int/Real/Char/Bool	Boolean
GT	Int/Real/Char/Bool	Boolean
GE	Int/Real/Char/Bool	Boolean
LT	Int/Real/Char/Bool	Boolean
LE	Int/Real/Char/Bool	Boolean

Multiple Input Functions

Name	Inputs	Output
ADD	Integer/Real	Integer/Real
MULT	Integer/Real	Integer/Real
AND	Boolean	Boolean
OR	Boolean	Boolean
XOR	Boolean	Boolean

Special Library Functions

The T-Gate function simulates the T-gate used for control of arcs within a Data Flow Graph. It accepts 2 input arcs; the first arc is the boolean control arc which will be tested and compared to the true value. The second input is the value which is input and will be placed on the single output arc if the boolean control is true. The F-Gate function acts in a similar matter except that it passes the data value if the boolean control value is false.

2.5 Process Scheduling On VAX VMS

Execution of Pascal procedures involves creating a sub-process under the VAX/VMS operating environment. These procedures are actually programs which have been translated and linked into executable images by the user. All data transfers occur between the simulator and the sub-process occurs through one of several channels which are called mailboxes.

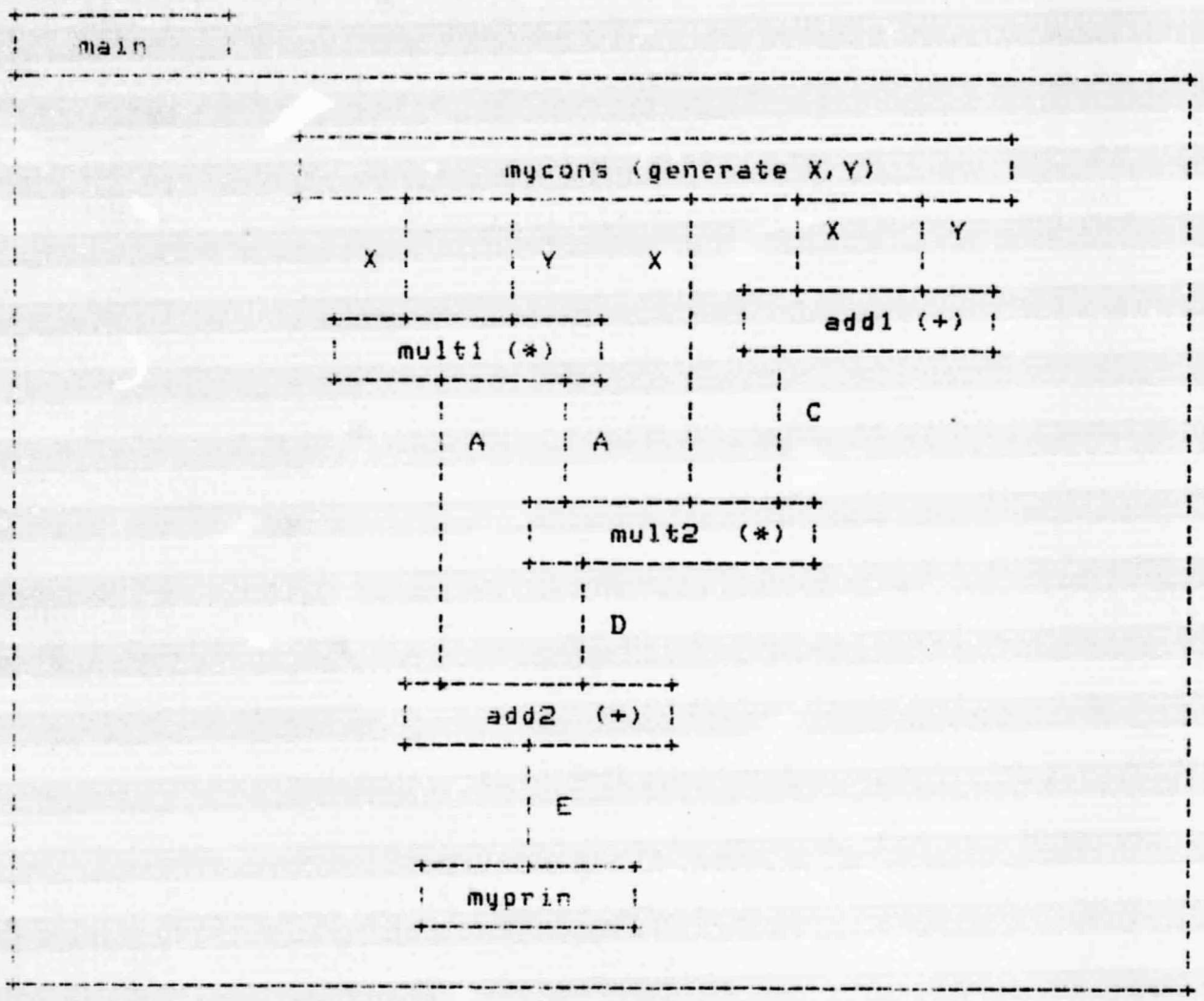
As a node is placed into execution, mailboxes for input, output, and termination messages will be allocated. An sub-process is then created with its input and output files mapped to the those mailboxes and the termination mailbox mapped into a termination channel. Input arcs are sent to the newly created process by means of standard Pascal write statements. The entry for that node is then deleted from the

ready list to prevent it from being scheduled. An entry for the node is placed on the executing list, and an interrupt request is made to the operating system so it may be informed when the process terminates.

Upon process termination, an interrupt occurs which indicates to the simulator that the process has been deleted. The Termination Handler responds by finding the entry for that node in the Executing List and inserting it into the pending list so the node manager may schedule it for further processing. When ready, the node manager calls the unlink procedure which allocates sufficient memory to hold the output arcs from the sub-process and then reads the output arcs from the output mailbox, putting them in memory. Next, the successor nodes are updated to indicate that these input arcs are available and the mailboxes used by this process are returned to the operating system.

2.6 Sample Data Flow Graph

The following Data Flow Graph will be expanded to show the relative ease with which one may transform a graphical DFG into the input text required by the simulator.



DFDLS Input Text for Sample Graph


```
NODE: MAIN;
   NODE_ENV;
      FSS: STD;
      FUNCTION: DFG;
         MULT1
         MULT2
         ADD1
         ADD2
         MYCONS
         MYPRIN
END NODE;

NODE: MULT1;
   NODE_ENV;
      FSS: STD;
      FUNCTION: LIB
        MULT;
   INPUTS;
      X : INTEGER;
      Y : INTEGER;
   OUTPUTS;
      ARC;
         A : INTEGER;
            ADD2;
            MULT2;
END NODE;

NODE: MULT2;
   NODE_ENV;
      FSS: STD;
      FUNCTION: LIB
        MULT;
   INPUTS;
      A : INTEGER;
      X : INTEGER;
      C : INTEGER;
   OUTPUTS;
      ARC;
         D : INTEGER;
            ADD2;
END NODE;

NODE: ADD1;
   NODE_ENV;
      FSS: STD;
      FUNCTION: LIB
         ADD;
   INPUTS;
      X : INTEGER;
      Y : INTEGER;
   OUTPUTS;
      ARC;
         C : INTEGER;
            MULT2;
END NODE;

NODE: ADD2;
   NODE_ENV;
      FSS : STD;
      FUNCTION: LIB
         ADD;
   INPUTS;
      A : INTEGER;
      D : INTEGER;
   OUTPUTS;
      ARC;
         E : INTEGER;
            MYPRIN;
END NODE;

NODE : MYPRIN;
   NODE_ENV;
      FSS : STD;
      FUNCTION: PAS;
         MYPRIN;
   INPUTS;
      E : INTEGER;
   OUTPUTS;
END NODE;

NODE: MYCONS;
   NODE_ENV;
      FSS: STD;
      FUNCTION: PAS;
         MYCOPAR2;
   OUTPUTS;
      ARC;
        X : INTEGER;
          MULT1;  MULT2;  ADD1;
      ARC;
        Y : INTEGER;
          MULT1;  ADD1;
END NODE;
```

3. SOME DATAFLOW GRAPH MODELS OF COMPUTER SYSTEMS

Here we will show how dataflow graphs can be used to model computer systems. The examples include a simple conventional control unit, a fault-tolerant computer based on von Neumann architecture that emphasizes deterministic synchronization among the concurrent tasks and the MIT dataflow processor. Only high level models are shown in this paper. However, nodes in the graphs can be expanded to represent instruction-level architecture.

3.1. Dataflow Model of a Simple Computer System: Baer [BAER 80, p 71] gave 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 dataflow equivalent of the Petri net given by Baer. The nodes in the graph are intentionally named by the events in order to facilitate interpretation. For the sake of clarity, control arcs and links are not drawn; their presence can be easily seen at the selection (oval shaped) nodes. Funnel shaped actors are enabled when one of its input arcs receives a token. Upon firing, the funnel passes the token on its input arc to the output arc. Tokens on more than one input arc leads to multiple firings of funnel.

3.2. Dataflow Model of SIFT: SIFT is an ultrareliable avionics computer built by NASA-Langley Research Center to study the possibility of completely automating commercial aircrafts [WENS 78]. SIFT has chosen to achieve fault tolerance using software. The SIFT system consists of several BDX 930 computers built by Bendix Corporation. The software consists of tasks classified as

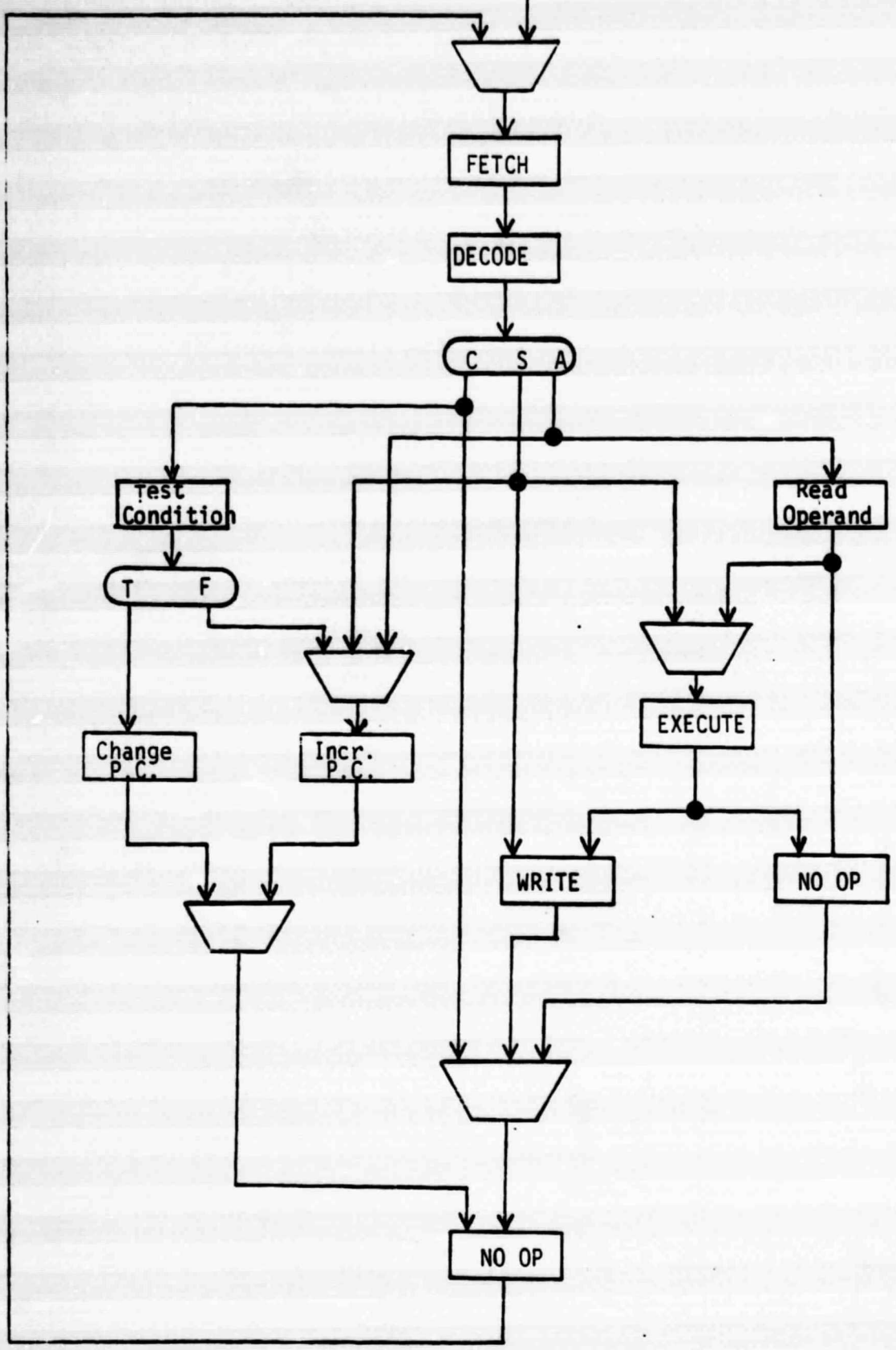


Fig 3. DATAFLOW MODEL OF A COMPUTER SYSTEM

application tasks and executive tasks. Executive tasks are responsible for scheduling tasks on SIFT processors, detecting errors, reconfiguring the system, synchronizing the various active processors (to within 50 microseconds) and input/output from sensors. Examples of application tasks are yaw damper, pitch inner loop and roll inner loop.

For the purpose of maximizing determinism in scheduling, computations in SIFT are conducted in regular time segments called frames and subframes. At present, the length of a subframe is 3.2 milliseconds and there are 50 subframes to a frame. Each task runs at a regular rate, the fastest running once every frame and others running once every other frame or with even longer period. Tasks that run once every frame are scheduled to specific subframes while the slower tasks are scheduled during the remaining subframes.

During a subframe, the status of the processor is saved, and the values produced by the task are broadcast to other processors. The values received from other processors are voted. If an error is detected, the processor marks the faulty processor in its error table and when the number of errors marked for a processor exceeds a threshold, an error is reported. The system is then reconfigured. The processor will execute a new task (or resumes a previous taks). Fig. 4 shows a high level dataflow model of a SIFT processor. The nodes in the graph can be further expanded into dataflow subgraphs, or the actual code representing the task can be used to represent the nodes.

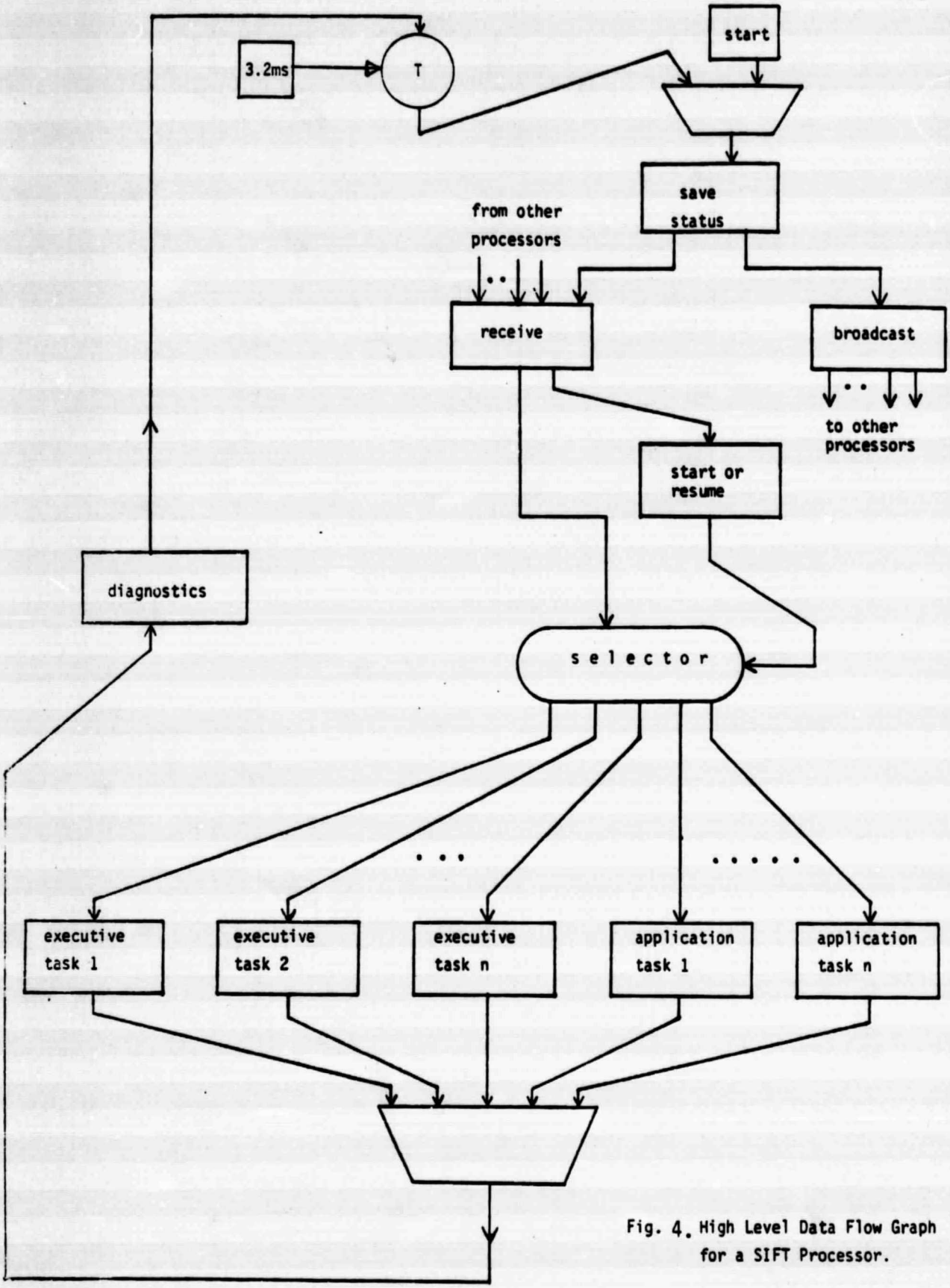


Fig. 4. High Level Data Flow Graph for a SIFT Processor.

3.3. Fault Injection: In the testing of actual circuits or the simulation of systems, it is desirable to provide for fault injection so that common faults can be injected into the system and the fault detection, reconfiguration parts of the system can be studied thoroughly. This capability can be modeled in dataflow as shown in Fig. 5. The nodes labeled Fault-k model simulate the function of the unit when fault-k is present. The selector node has n inputs and one control input. Depending on the value of the token on the control input, one of the n inputs is passed on to the output.

3.4. Dataflow Graph Model of a Dataflow Computer: Fig.6 shows a high level block diagram of the MIT dataflow processor [DENN 75]. A high level dataflow graph model of the MIT system is shown in Fig. 7. Memory unit can produce one or more tokens on its output arcs depending on the number of enabled instructions. The arbitration network schedules one instruction packet at a time and dispatches the instruction to a processing element or a decision unit. This is done in order to simplify the dataflow graph of Fig. 7. The graph can be modified to show concurrent scheduling of several instructions. Control and Distribution networks also receive multiple inputs, but work on one packet at a time. The packet routing networks (arbitration, control and distribution) can be represented as dataflow subgraphs detailing how the packets are routed in the MIT dataflow system.

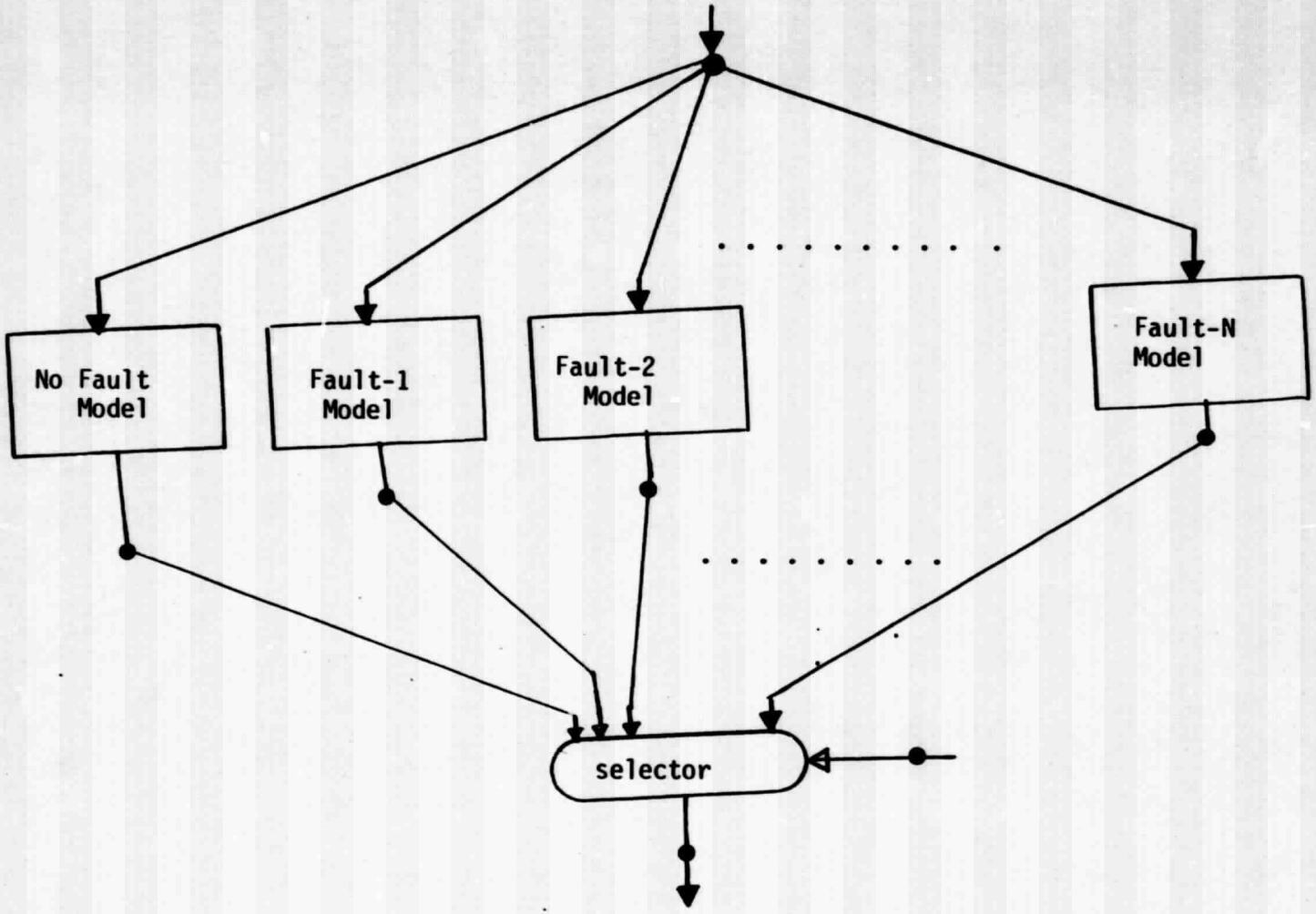


Fig. 5. Dataflow Model for Fault Injection

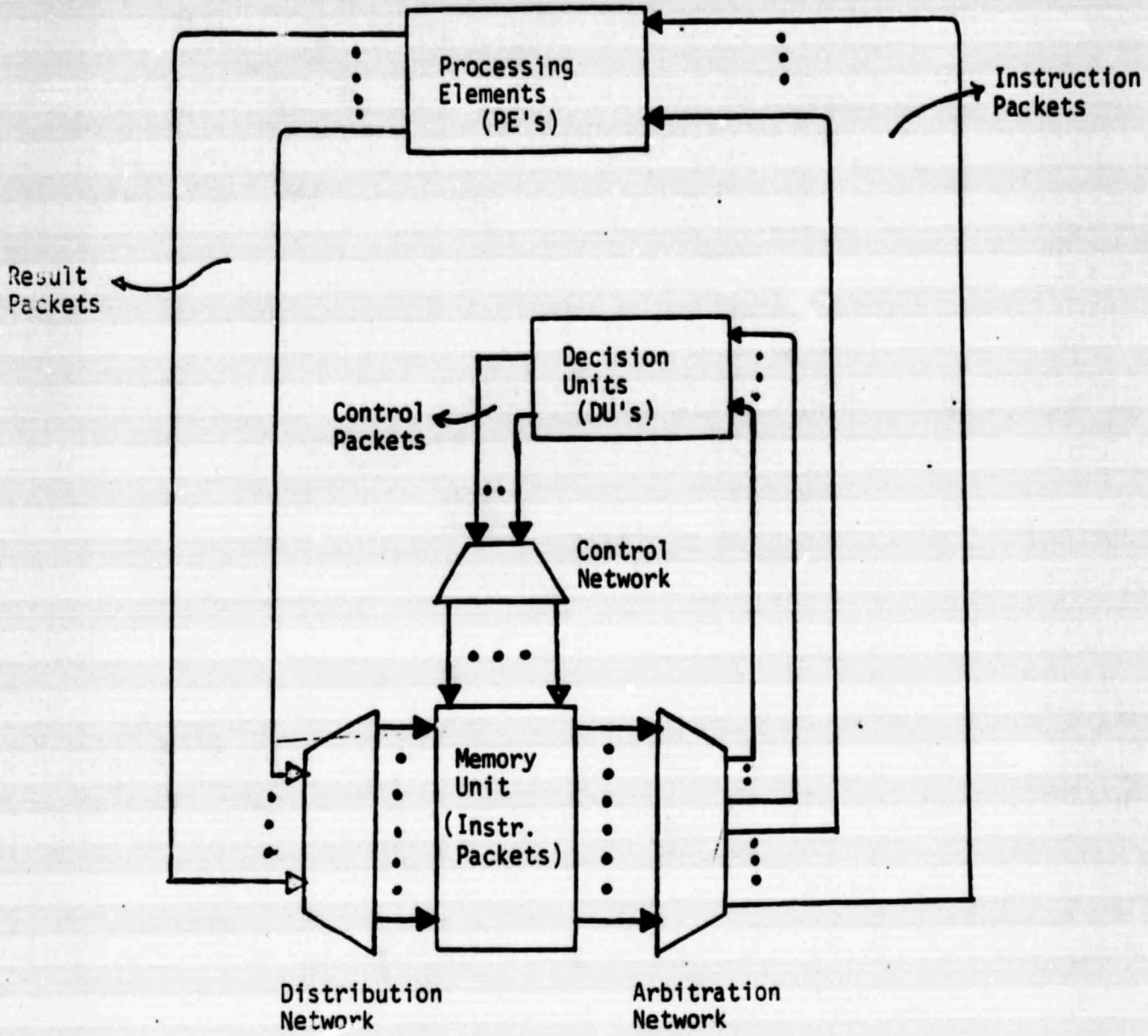


Fig. 6. MIT Data Flow Processor

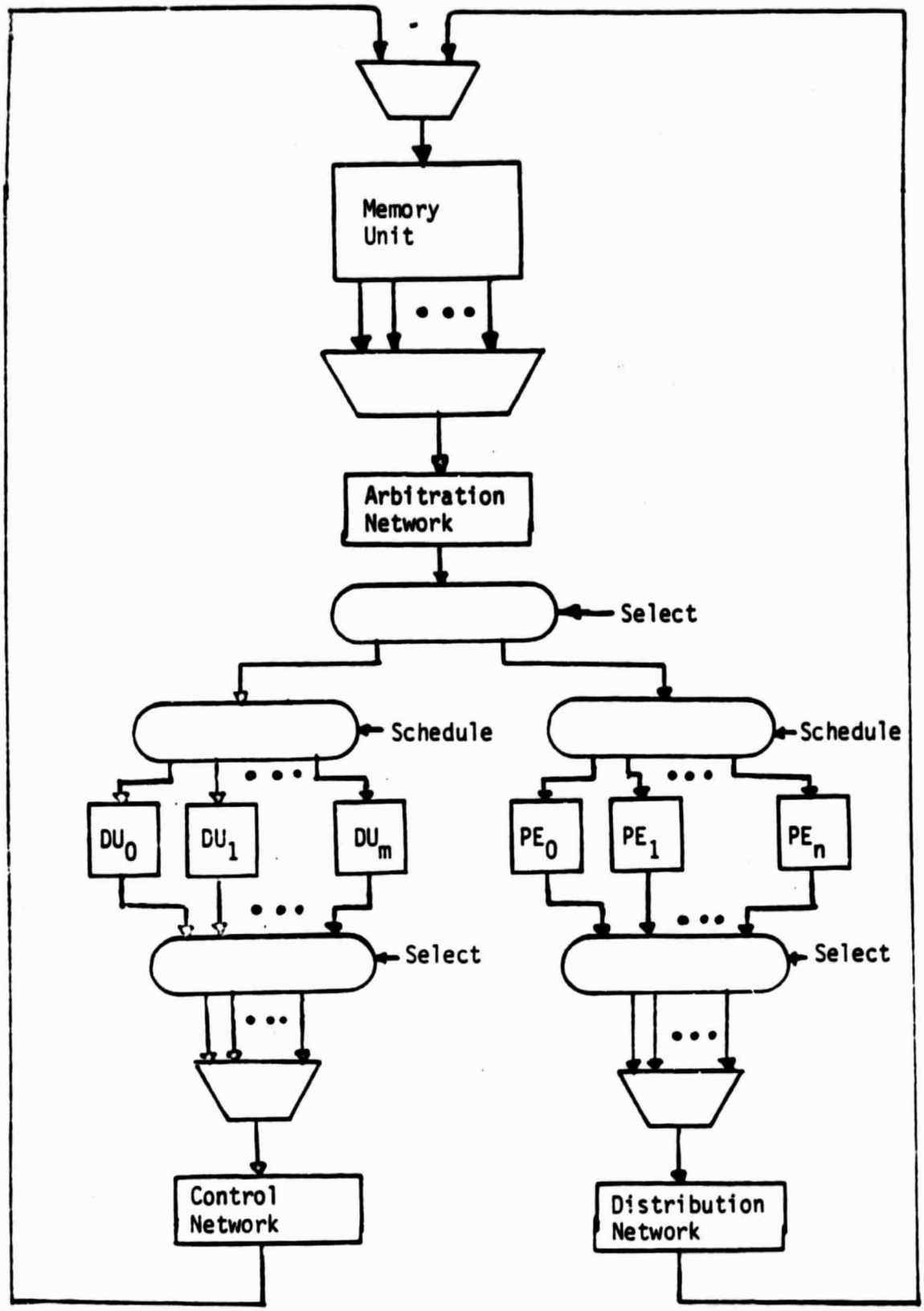


Fig. 7. Dataflow Graph Model for MIT Data Flow Processor

4. SUMMARY AND CONCLUSIONS

In this paper we have described a dataflow simulation that can be used to model computer systems as dataflow graphs and test the functionality of the modeled system. Dataflow graphs can be used to model computer systems including parallel processors and dataflow computers. We plan extend DFDLS to include record structures for tokens, allow recursive definitions for dataflow actors, linking with procedures written in other languages, graphical input, timing and performance measures.

5. REFERENCES

BACK 78 J. Backus. "Can Programs be Liberated From von Neumann Style? A Functional Style and Its Algebra of Programs", CACM, Aug. 1978, pp 613-641.

BAER 80 J.L. Baer. Computer Systems Architecture, Computer Science Press, 1980.

BREU 72 M.A. Breur. "Recent Developments in Design Automation", IEEE Computer, May 1972.

DENN 74 J.B. Dennis. "First Version of Data Flow Procedural Language", Lecture Notes in Computer Science, Vol. 19, Springer Verlag, 1974.

DENN 75 J.B. Dennis and D.P. Misunas. "A Preliminary Architecture for a Basic Data Flow Processor", Proc. 2nd Annl. Symp. on Comp. Arch., (Houston), 1975, pp 126-132.

GAUD 84 J.L. Gaudiot and M.D. Ercegovac. "Performance Analysis of Data Flow Computers with Variable Resolution Actors", Proc. 4th Intl. Conf. Distr. Comp. Syst. (San Francisco), May 14-18, 1984, pp 2-9.

HEAT 72 F.G. Heath and C.W. Rose. "The Case for Integrated Hardware/Software Design wiht CAD Implications", IEEE COMPCON, Sept. 1972.

KARP 82 W.P. Karplus and A. Makoui. "The Role of Data Flow Methods in Continuous System Simulation", Proc. Summer Simulation Conf. (Denver) 1982.

KAVI 83 K.M. Kavi. "Data Flow Modeling Techniques", Proc. IASTED Intl. Symp. on Siml. and Modl. (Orlando), Nov. 9-11, 1983, pp 1-4.

KAVI 84 K.M. Kavi, B.P. Buckles and U.N. Bhat. "Abstract Dataflow Models for Parallel Computer Systems", Submitted to 12th Intl. Symp. on Comp. Arch. (Boston).

LAND 81 S.P. Landry. "System Oriented Extensions to Data Flow", Ph.D. Dissertation, Dept. of C.S., USL, May 1981.

TREL 79 P.C. Treleaven. "Exploiting Program Concurrency in Computing Systems", IEEE Computer, Jan. 1979, pp 42-49.

TREL 82a P.C. Treleaven and B.G. Cone. "Japan's Fifth Generation Computer Systems", IEEE Computer, Aug. 1982, pp 79-88.

TREL 82b P.C. Treleaven, D.R. Brownbridge and R.P. Hopkins. "Data Driven and Demand Driven Computer Architecture", ACM Computing Surveys, March 1982, pp 93-143.

WENS 78 J.H. Wensley, et. al. "SIFT: Design and Analysis of a Fault Tolerant Computer for Aircraft Control", Proc. IEEE, Oct. 1978.

APPENDIX 3

ISOMORPHISMS BETWEEN PETRI NETS AND DATAFLOW GRAPHS

ISOMORPHISMS BETWEEN PETRI NETS AND DATAFLOW GRAPHS[1]

by

Krishna M. Kavi

Billy P. Buckles[2]

University of Texas at Arlington

and

U. Narayan Bhat

Southern Methodist University

Complete Mailing Addresses

Dr. K. M. Kavi and Dr. B. P. Buckles
Computer Science & Engineering Dept.
University of Texas at Arlington
P.O. Box 19015
Arlington, TX 76019
(817)273-3785

Dr. U. N. Bhat
Statistics Department
Southern Methodist University
Dallas, TX 75275
(214)692-2270

---

[1]This work is supported in part by NASA-Ames Research Center under Contract NAG 2-273.

[2]Please address all correspondence regarding this paper to Dr. Bill Buckles at the above address.

Abstract. Dataflow graphs are a generalized model of computation. Uninterpreted dataflow graphs with nondeterminism resolved via probabilities are shown to be isomorphic to a class of Petri nets known as free choice nets. Petri net analysis methods are readily available in the literature and this result makes those methods accessible to dataflow research. Nevertheless, combinatorial explosion can render Petri net analysis inoperative. Using a previously known technique for decomposing free choice nets into smaller components, it is demonstrated that, in principle, it is possible to determine aspects of the overall behavior from the particular behavior of components.

Index Terms. Dataflow graphs, timed Petri nets, free choice nets, performance analysis, isomorphism.

I. INTRODUCTION

Increasing interest in dataflow architectures derives in part from the quest for large improvements in performance through parallelism. This interest has given impetus to the development of new representation methods and languages for parallel algorithms. Our interest is in the dataflow graph and its potential to represent any computational structure including computer architectures. The inherent ability to represent the natural parallelism in architectures other than dataflow machines has been noted by others [9,18].

The chief advantages of dataflow graphs as a computational schema are their compactness and generally amenability to direct interpretation. That is, the translation from the conceived system to a dataflow graph is straightforward and, once accomplished, it is equally straightforward to determine by inspection which aspects of the system are represented [7,8]. Unfortunately, the analysis techniques for data flow graphs are yet not well developed.

A rapid method of closing the gap in analytic methods is to demonstrate cases automatically transformable to Petri nets, the latter having been subject to two decades of study. Certain abstract properties of Petri nets such as liveness and boundedness have immediate relevance in any general computational schema including dataflow graphs. Other properties such as comparative firing frequencies assume relevance with respect to the semantics of the system being modeled. Thus it is clearly of benefit to establish the correspondence between dataflow graphs and Petri nets in order to combine the representational ease of one with the analysis power of the second.

Performance analysis of computer architectures represented as dataflow graphs via Petri nets (more precisely, timed Petri nets) is a goal of this

work. Here, systems are described by using dataflow graphs having abstract data components called tokens. Removing the semantics from data introduces nondeterminism which is compensated by the assignment of probability mass functions to decision points. The result is called an uninterpreted dataflow graph of which important subclasses are isomorphic to subclasses of Petri nets. For those graphs representable in Petri net form, properties such as those mentioned above can be analyzed. In addition, properties dealing with time can be evaluated.

II. DATAFLOW GRAPHS AND PETRI NETS

Formalized treatment of Petri nets is common in the literature [3,10,13,14,16] and will be dealt with briefly.

Definition 1. A Petri net is a quintuple

$$PN = < P, T, D, MP_o, MP_t >$$

where

$P = \{p_1, p_2, \ldots, p_n\}$, a set of places

$T = \{t_1, t_2, \ldots, t_m\}$, a set of transitions

$D \subseteq \{P \times T\} \cup \{T \times P\}$, a set of directed arcs

MP_o is a given initial marking

MP_t is a set of terminal markings.

Here we are chiefly interested in extensions to the basic model that incorporate concepts of time.

Timing information has been incorporated in three ways. Sifakis and others [4,17] associated a nonnegative constant, b, with each place having the semantics that an arriving token was "unavailable" until it had been in the place for a time interval of length b. The two other methods attach timing information directly to transitions. One may associate with a transition a nonnegative constant (timed Petri nets [10,16,19]) or a

probability distribution (stochastic Petri nets [1,5,11,12]). The first case is equivalent to assigning time values to places [17]. In either case, the principal problem to be resolved is when to begin the firing epoch -- upon arrival of the first token or the instant a transition is enabled. One need also consider whether a second or subsequent epoch can begin while one is in progress.

A second problem to resolve is firing conflicts. Those models that depend on fixed firing time generally assign a probability over the marking space from the current to next marking [19]. Stochastic Petri net models generally use the firing rate (based on random firing times) to determine the next marking from the current one [1,11,12]. A difficulty arises if one allows some transitions to have zero firing time. The probability that such transitions will fire once enabled approaches one (1). The solution is to augment the firing rates with transition probabilities as is done in timed Petri nets. Several investigators have noted the direct correspondence between Petri nets with timing information and Markov processes [2,11,12,19]. In this work, timed Petri nets are employed.

Definition 2. A timed Petri net is the pair

$$TPN = < \zeta, f >$$

where

ζ is a PN

$f : T \rightarrow \{R^{+} \cup \{0\}\}$, a firing time function

In addition to analyzing the time properties of nets, a goal of this research is the determination of the overall behavior of a system by the inspection of properties of components. Hack [6] first demonstrated necessary and sufficient conditions for liveness and safeness of a subclass of Petri nets important to this work. Ramchandani [15] achieved related

results for general nets in the more formal context of solutions to Diophantine equations derived from the connectivity of the net. Solutions to the equations results in subnets (more precisely, T-subnets or P-subnets) whose structure [redacted] is that of a marked graph or state machine under some circumstances. Ramamoorthy and Ho [16] developed techniques for cycle time computations for such subnets and Magott [10] transformed the method to a solution of a linear program. We have extended this work by showing how the mean time between events of a net composed of marked graph components can be obtained. Coolahan and Roussopoulos [4] have also developed statistical measures of transition firing frequencies and these are adaptable to our model. Datta and Ghosh [3] developed a labeling method that guarantees liveness for nets with transitions of in-degree (and out-degree) at most two (2).

A formal treatment of dataflow graphs has been lacking in the literature due to the purpose that other investigators have used them. Due to the nature of our study and the need to demonstrate homomorphic structures between dataflow and Petri net models, a formal definition has been developed [8]. The following reviews those results.

Definition 3, A dataflow graph is a labeled bipartite graph where the two types of nodes are called actors and links.

$$DFG = < A \cup L, E, S, T >$$

where

$A = \{a_1, a_2, \ldots, a_n\}$, a set of actors

$L = \{l_1, l_2, \ldots, l_m\}$, a set of links

$E \subseteq (A \times L) \cup (L \times A)$, a set of edges such

that $(a_i,l_k) \in E \wedge (a_j,l_k) \in E \Rightarrow a_i = a_j$

and $(l_i,a_k) \in E \wedge (l_j,a_k) \in E \Rightarrow l_i = l_j$

$S = \{l \in L \mid (a,l) \notin E \text{ for any } a \in A\}$,

a non-empty set of links called the starting set

$T = \{l \in L \mid (l,a) \notin E \text{ for any } a \in A\}$,

a non-empty set of links called the terminating set.

Note that links are limited to a single input and single output. Meeting this restriction may require the introduction of dummy actors (e.g., to duplicate an input token on several output links).

Let $I(a)$, $a \in A$ ($I(l)$, $l \in L$) and $O(a)$, $a \in A$ ($O(l)$, $l \in L$) denote the sets of input and output links (actors) of actor a (link l), respectively. $|I(a)|$ and $|O(a)|$ must be nonzero for each actor while $|I(l)|$ and $|O(l)|$ are at most one. The notation is directly extended to the places and transitions of Petri nets. However, there are no cardinality constraints on the sets denoted.

A marking of a dataflow graph denotes the presence or absence of tokens in links. A marking is a function $M : L \to \{0,1, \ldots, k\}$. When M (or MP for Petri nets) is used it means the vector

$$< M(\ell_1), M(\ell_m), \ldots, M(\ell_m) >$$

A marking is distinguished as an initial marking (terminal marking) $M(\ell) \neq 0 \Rightarrow \ell \in S$ ($M(\ell) \neq 0 \Rightarrow \ell \in T$).

Associated with each actor is an input and output firing set denoting which links enable the actor and which receive tokens when the actor fires. These sets are denoted F_1 and F_2, respectively.

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

$$F_2(a,M) \subseteq O(a)$$

Dataflow graphs exhibit special arcs called control arcs whose purpose is to affect the flow of data at decision points. These do not exist in uninterpreted dataflow graphs used here but a probability mass function over the powerset of O(a) serves the same purpose.

An actor, a, is enabled in marking M if $M(\ell) \neq 0$ for each $\ell \in F_1(a,M)$. The firing of an enabled actor, a, results in a new marking indicated by $M \stackrel{a}{\rightarrow} M'$.

$$M' = M - < I(a) > + < O(a) >$$

where

$< I(a) >$ is a vector in which the ith element is one

if $\ell_i \in F_1(a,M)$

$< O(a) >$ is a vector in which the ith element is one

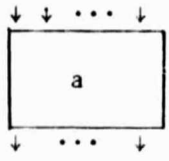
if $\ell_i \in F_2(a,M)$

This can be generalized to a firing sequence, σ, denoted $M \stackrel{\sigma}{\rightarrow} M_p$ where

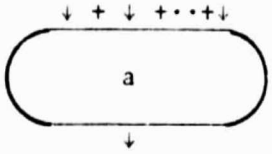
$$M \stackrel{a1}{\rightarrow} M_1 \stackrel{a2}{\rightarrow} M_2 \stackrel{a3}{\rightarrow} \ldots \stackrel{ap}{\rightarrow} M_p$$

The 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 $\vec{M} = \{M' | M \stackrel{\sigma}{\rightarrow} M'\}$.

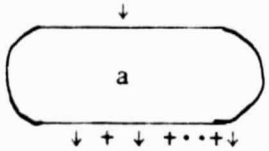
It is simple to extend the semantics given above by assigning a nonnegative real value to each actor representing the time it takes to fire. In the diagrams that follow, the conventions below have been adopted.



an actor for which $F_1(a,M) = I(a)$ and $F_2(a,M) = O(a)$; such an actor is said to have conjunctive input and distributive output



an actor for which $F_1(a,M) \subset I(a)$ and $F_2(a,M) = O(a)$; such an actor is said to have disjunctive input



an actor for which $F_1(a,M) = I(a)$ and $F_2(a,M) \subset O(a)$; such an actor is said to have selective output

Links are represented by solid circles. For the second type of actor (the disjunctive actor) the enabling input link is chosen nondeterministically.

While it is not permitted for an actor to be simultaneously disjunctive and selective, the restriction is not severe. The uninterpreted nature of the data tokens allows such actors to be separated into two actors.

It is not necessary to enumerate each analogous term for Petri nets such as $\vec{MP}$ and $MP \xrightarrow{t} MP'$. Yet it should be noted that the standard firing set semantics are

$$FP_1(t,MP) = I(t)$$

$$FP_2(t,MP) = O(t)$$

The effect of firing an enabled transition, t, is

$$MP' = MP - < I(t) > + < O(t) >$$

III. GRAPH TO NET TRANSFORMATIONS

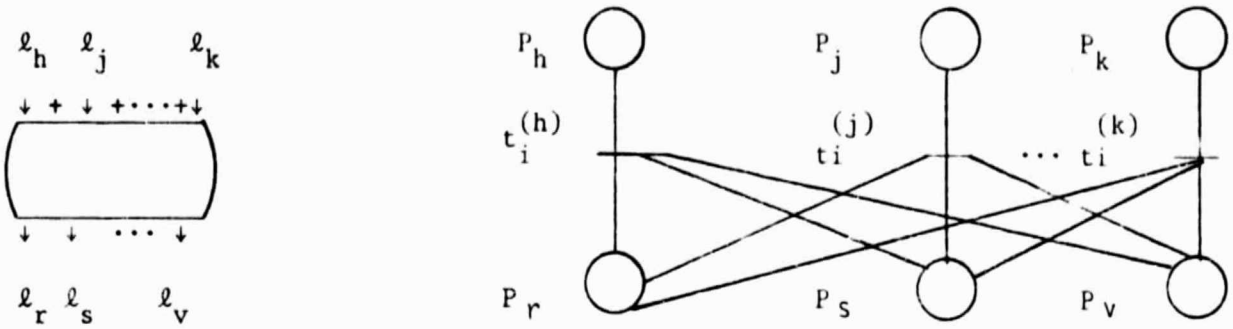
Reduction of graphs representing asynchronous processes to Petri nets occurs frequently in the literature. (See [14] for examples.) Therefore, we will treat the topic informally. Let DFG be an arbitrary uninterpretated dataflow graph.

ALGORITHM A1

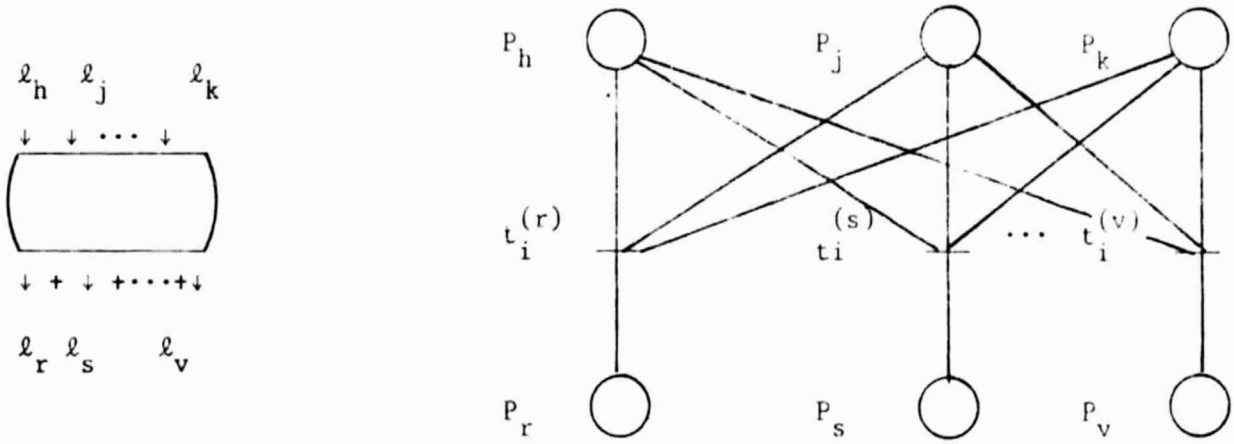
1. Let $MP_o \leftarrow M_o$ and $MP_t \leftarrow M_t$
2. For each $\ell_i \in L$ in DFG, create $p_i \in P$ in the TPN
3. For each conjunctive actor $a_i \in A$ in DFG, create a transition $t_i \in T$ in TPN such that if $\ell_j \in O(a_i)$ then $p_j \in O(t)$, if $\ell_j \in I(a)$ then $p_j \in I(t)$
4. For each disjunctive actor $a_i \in A$, perform the transformation shown in Fig. 1(a). Create a unique transition for each $\ell_j \in I(a_i)$. If $\ell_j \in O(a_i)$ then $p_j \in O(t_i^{(h)})$ for each $t_i^{(h)}$.
5. For each selective actor $a_i \in A$, perform the transformation shown in Fig. 1(b). Create a unique transition for each $\ell_j \in O(a_i)$. If $\ell_j \in I(a_i)$ then $p_j \in I(t_i^{h)})$ for each $t_i^{(h)}$.
6. If α is the time associated with actor a_i, then let $f(t_i) = \alpha$. If more than one transition was derived from a_i, the time associated with each is α.

We will deal with the transformation of the probability mass function (pmf) for selective actors to a pmf for the TPN later.

Let the symbol ~ means "derived from". It may be used with individual components (t ~ a) or entire graphs (TPN ~ DFG). The steps in



(a) Disjunctive Actors



(b) Selective Actors

Fig. 1. Graph to Net Transformations

Algorithm A1 are reversible. Thus it is possible to reconstruct the dataflow graph from the Petri net. In this context, we can use t~a and a~t (p~1 and 1~p) interchangeably. Figures 2 and 3 show a dataflow graph and the Petri net derived from it using the transformation above. Let an arbitrary M be represented by the vector $< m_1, m_2, \ldots, m_n >$ where each m_i is an integer. An arbitrary MP can be represented similarly. Thus when we say M = MP we mean simple vector equality.

Definition 4. A unit disjunctive DFG is one for which $F_1(a,M) \varepsilon I(a)$ for all disjunctive actors. A unit selective DFG is one for which $F_2(a,M) \varepsilon O(a)$ for all selective actors.

THEOREM 1. (isomorphism) Given a unit disjunctive and unit selective DFG, if TPN ~ DFG, M = MP, and $M \overset{\sigma}{\rightarrow} M'$ then there exists σ' such that $MP \overset{\sigma'}{\rightarrow} MP'$ and M' = MP'. Conversely if $MP \overset{\sigma'}{\rightarrow} MP'$ there exists σ such that $M \overset{\sigma}{\rightarrow} M'$, M = MP, and M' = MP'.

PROOF. We prove the first part by demonstrating how to select $t_1, t_2, \ldots, t_k$ in σ' that correspond to $a_1, a_2, \ldots, a_k$ in σ. Let a_1 be the first actor in the sequence $M \overset{a_1}{\rightarrow} M_1 \overset{a_2}{\rightarrow} \ldots \overset{a_k}{\rightarrow} M'$. $M_1 = M - <I(a_1)> + <O(a_1)>$. If a_1 is a conjunctive actor, select $t_1 \sim a_1$. If a_1 is disjunctive select $t_1^{(r)} \sim a_1$ such that $\ell_r \varepsilon F_1(a_1,M)$ is the link that enabled a_1. If a_1 is selective select $t_1^{(r)} \sim a_1$ such that $\ell_r \varepsilon F_2(a_1,m)$ is the link upon which the token is produced. If $MP \overset{t_1}{\rightarrow} MP_1$ then clearly $<I(a_1)> = <I(t_1)>$ and $<O(a_1)> = <O(t_1)>$. Thus $M = MP_1$. Choosing $t_2, t_3, \ldots, t_k$ in the same manner produces MP' = M'. The converse is proved similarly. Q.E.D.

The above theorem demonstrates that properties of a particular dataflow graph can be discovered by examining the derived Petri net.

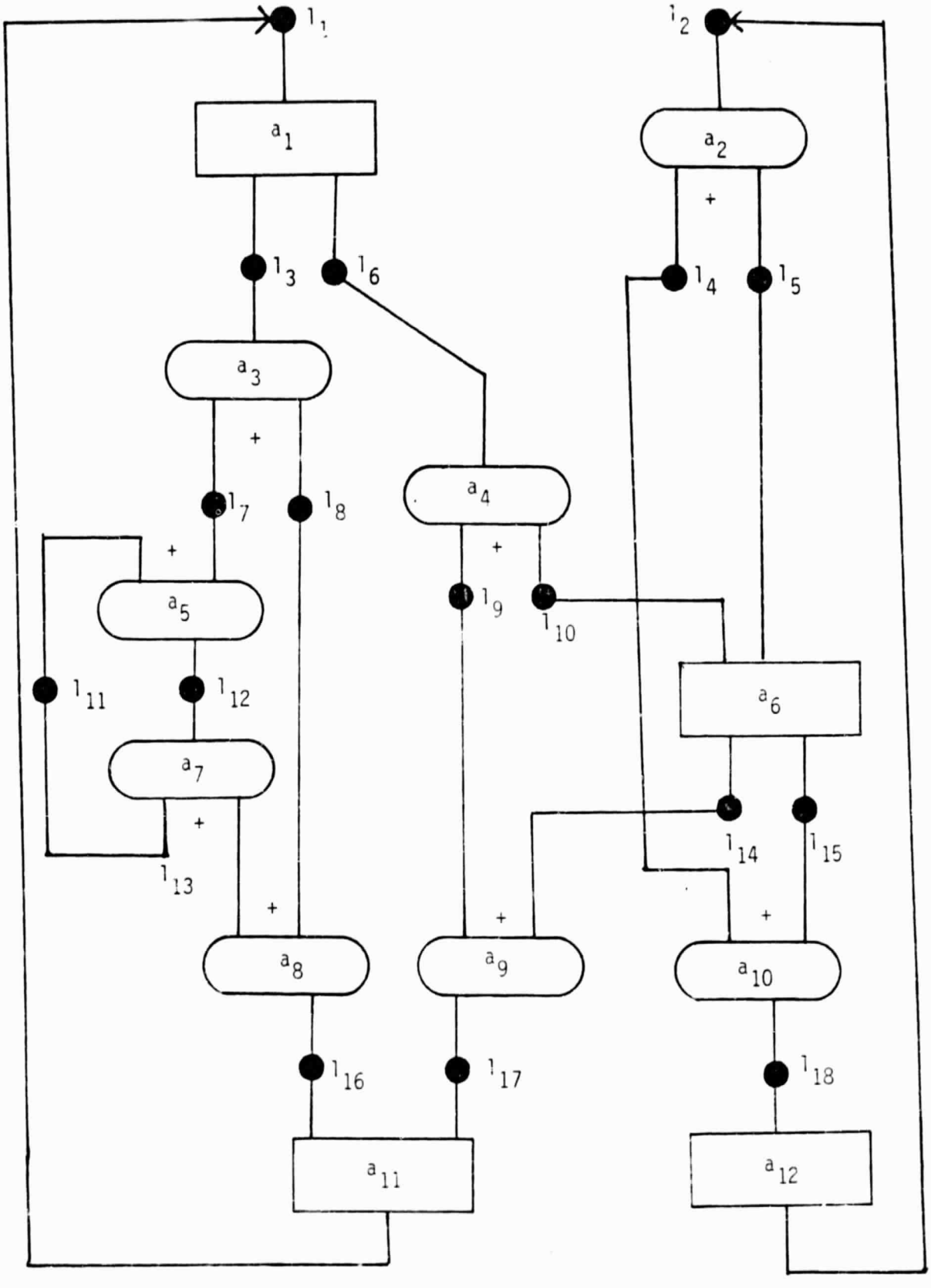


Fig. 2. An Uninterpreted Dataflow Graph

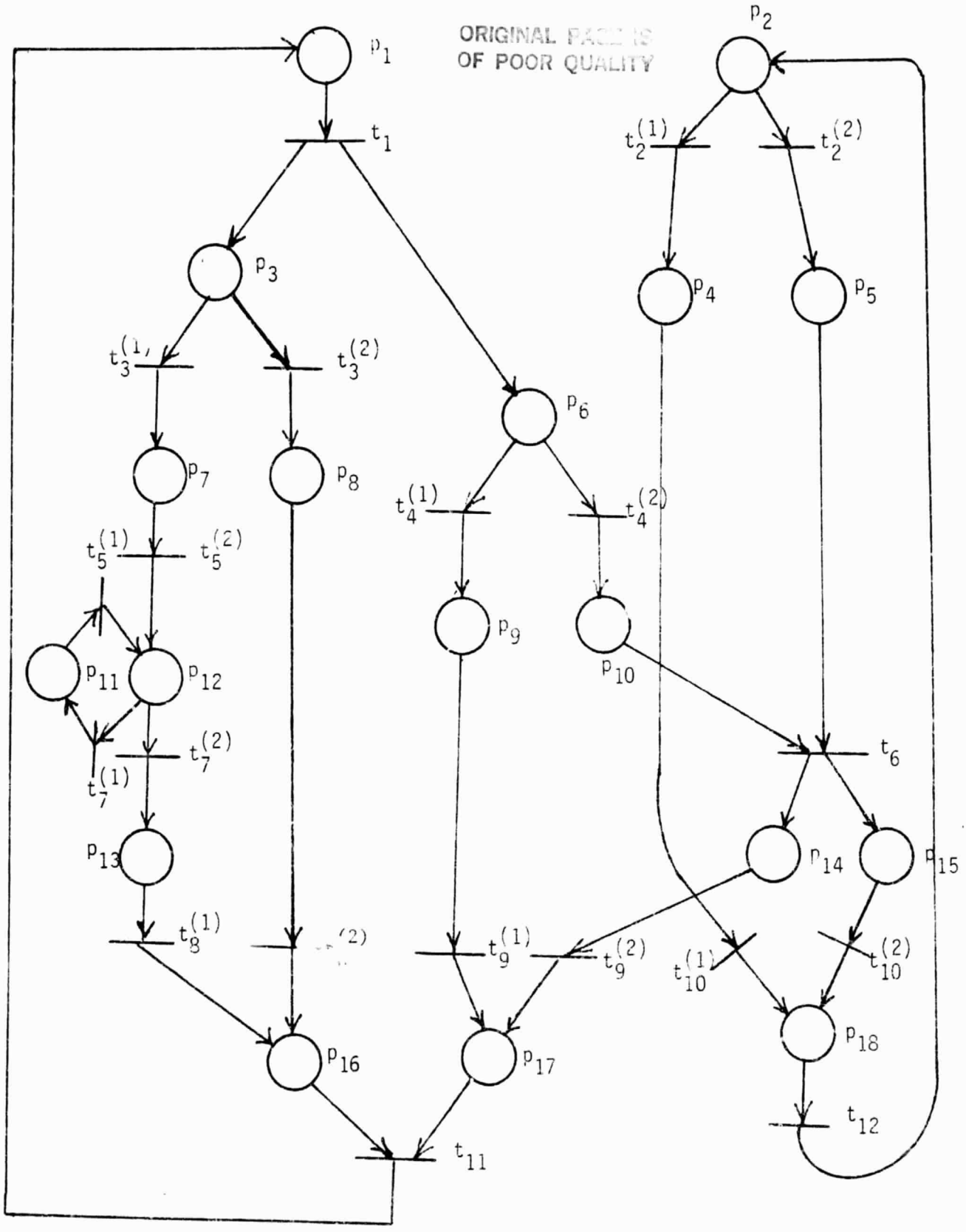


Fig. 3. Derived Petri Net

Before describing these methods, however, we introduce an additional term and its relation to the transformation described.

Definition 5. A free choice net is a Petri net for which each arc from a place to a transition is a unique output of the place or the unique input of the transition. More formally,

$$(\forall\ (p,t) \in D)\ (O(p) = \{t\} \text{ or } I(t) = \{p\}).$$

THEOREM 2. If $|I(a)| = 1$ for each selective actor of DFG and TPN ~ DFG then TPN is a timed free choice net.

PROOF. Recall that by definition a DFG link has a single input and a single output actor. Thus, $|O(\ell)| = 1$ for all $\ell \in L$. If $(\ell,a) \in E$, $a \in A$ is conjunctive, $p \sim \ell$, then $|O(p)| = 1$ since for conjunctive actors there is a one-to-one correspondence between $(\ell,a) \in E$ and $(p,t) \in D$. If $(\ell_i, a) \in E$, $a \in A$ is disjunctive, $p_i \sim \ell_i$, then $t^{(i)} \sim a$ is created by step 4 of Algorithm A1 such that $O(p_i) = \{t^{(i)}\}$. If $(\ell_i, a) \in E$, $a \in A$ is selective then $t^{(j)} \sim a$ is created by step 5 of Algorithm A1 such that $I(t^{(j)}) = \{p_i | p_i \sim \ell_i, \ell_i \in I(a)\}$. Clearly $|I(t^{(j)})| = 1$ if $|I(a)| = 1$. Q.E.D.

The unit disjunctive/selective criterion assures a DFG can be converted to a Petri net without combinatorial explosion, $|T| \leq |E|$. If selective actors have a single input, then the DFG is isomorphic to a timed free choice net. This is significant primarily because a considerable body of theory exists for the analysis of free choice nets.

COROLLARY 1. If TPN ~ DFG then

$$(\forall\, t_i, t_j \in T)\ (I(t_i) \cap I(t_j) \neq \Phi \Rightarrow I(t_i) = I(t_j))$$

PROOF. Note that in DFG, $I(a_i)$ iff $a_i \equiv a_j$ because each link has but one output. (It is impossible for distinct actors to share a link.) Therefore, if two transitions share a place they must be derived from the same actor. If a_i is an actor with conjunctive input, there is only one t_i such that $t_i \sim a_i$ and it shares no input. If a_i is an actor with disjunctive input then a unique transition is created for each element of $I(a_i)$. Again $I(t_j) \cap I(t_j) \neq \Phi$ implies $t_i \equiv t_j$. For a selective actor, a_i, for every $\ell_j \in I(a_i)$ then $p_j \in I(t_i^{(h)}$ for every h such that $t_i^{(h)} \sim a_i$. Thus, $I(t_i^{(h)}) = I(t_i^{(k)})$. Q.E.D.

The corollary is well known for free choice nets. Here, however, its proof is based directly on the relationship with dataflow graphs. Its importance is the partitioning of the transitions into blocks $B_1, B_2, \cdots, B_h$ such that if any transition in block B_i is enabled, all are. Further, the firing of a transition in block B_i cannot disable a transition in block B_h if $i \neq h$. This will make it possible to resolve conflicts with probability mass functions over transitions rather than (the ordinary practice) over markings.

IV. PROBABILISTIC TIMING ANALYSIS

An objective of Petri net analysis is to determine overall behavior by decomposing it into components which can be analyzed individually.

Definition 6. A subnet of a Petri net TPN = $\langle P, T, D, MP_o, MP_t, f\rangle$ is another Petri net TPN′ = $\langle P', T', D', MP_o', MP_t', f'\rangle$ such that $P' \subseteq P$, $T' \subseteq T$, $D' = D \cap ((P' \times T') \cup (T' \times P'))$, $MP_o(p) \in MP_o'$ if $p \in P'$, $MP_t(p) \in MP_t'$ if $p \in P'$, and $f'(t) = f(t)$ if $t \in T'$ and undefined otherwise.

Let $I(\cdot)'$ and $O(\cdot)'$ be the input/output sets of TPN′. TPN′ is said to be a T-subnet if for every $t \in T'$, $I(t) = I(t)'$ and $O(t) = O(t)'$. It is said to be a P-subnet if for every $p \in P'$, $I(p) = I(p)'$ and $O(p) = O(p)'$. A net or subnet is said to be strongly connected if for every p_i, $p_j \in P$ there is a directed path from p_i to p_j.

A state machine is a net or subnet for which $|I(t)| \leq 1$ and $|O(t)| \leq 1$ for every $t \in T$. A marked graph is a net or subnet for which $|I(p)| \leq 1$ and $|O(p)| \leq 1$ for every $p \in P$. The incidence matrix $C = [c_{ij}]$ of a Petri net is an n x m matrix where

$$c_{ij} = \begin{cases} -1 \text{ if } (p_i, t_j) \in D \\ 1 \text{ if } (t_j, p_i) \in D \\ 0 \text{ if neither } (p_i, t_j) \in D \text{ nor } (t_j, p_i) \in D \text{ or both are in } D \end{cases}$$

For example, the incidence matrix for the free choice net of Fig. 3 is

$$C = \begin{bmatrix}
-1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & -1
\end{bmatrix}$$

A Petri net is said to be live if for every marking in $\vec{MP}_0$ there are firing sequences that enable each transition. A Petri net is bounded if there is a finite number of tokens in each place given any (perhaps infinite) firing sequence. A Petri net is safe if the number of tokens in any place never exceeds one.

Hack [6] gave necessary and sufficient conditions for the liveness and safeness of a marked free choice net. If $MP_0 = < 1\ 1\ 0\ 0\ \ldots\ 0 >$, Fig. 3 satisfies the conditions. It can be shown [15] that such nets can be decomposed into strongly connected components by finding the simple nonnegative solutions to the system of equations

$$C \cdot Y = 0$$

A solution is simple if it cannot be additively obtained from other solutions. For Fig. 3,

$Y_1 = < 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 0\ 1\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 1\ 1\ 1 >$

$Y_2 = < 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 1 >$

$Y_3 = < 1\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 0\ 1\ 0 >$

$Y_4 = < 1\ 0\ 1\ 1\ 0\ 0\ 1\ 0\ 1\ 1\ 0\ 1\ 1\ 0\ 0\ 1\ 0\ 1\ 1\ 1 >$

$Y_5 = < 1\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 1\ 0\ 0\ 1\ 1\ 0\ 1\ 0\ 0\ 0\ 1\ 0 >$

$Y_6 = < 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0 >$

Each Y-vector designates as set of transitions. For instance, Y_1 designates $\{t_1, t_2^{(2)}, t_3^{(2)}, t_4^{(2)}, t_6, t_8^{(2)}, t_9^{(2)}, t_{10}^{(2)}, t_{11}, t_{12}\}$ and Y_2 designates $\{t_2^{(1)}, t_{10}^{(1)}, t_{12}\}$. Each transition set together with <u>all</u> directly connected places constitutes a strongly connected a T-subnet. For free choice nets, each component is a marked graph.

For components that are timed marked graphs, Ramamoorthy and Ho [16] first showed how to determine the overall cycle time by enumerating the circuits. The time required for each circuit is the sum of the transition times divided by the number of tokens in the circuit. Let K_{ij} be the set of places and transitions in the jth elementary circuit of the ith component.

$$\tau_{ij} = \sum_{t_k \in K_{ij}} f(t_k) / \sum_{p_k \in K_{ij}} MP(p_k)$$

Assuming r circuits and a firing epoch begins as soon as it is enabled, the overall cycle time for the component is $\bar{\tau}_i = \max(\tau_{i1}, \tau_{i2}, \ldots, \tau_{ir})$.

For example, let $MP(p_1) = MP(p_2) = 1$ and $MP(p_i) = 0$ if $i > 2$ for the component, Y_1, of Fig. 3 the elementary circuits are

$$K_{11} = \{p_1\ t_1\ p_3\ t_3^{(2)}\ p_8\ t_8^{(2)}\ p_{16}\ t_{11}\}$$
$$K_{12} = \{p_1\ t_1\ p_6\ t_4^{(2)}\ p_{10}\ t_6\ p_{14}\ t_9^{(2)}\ p_{17}\ t_{11}\}$$
$$K_{13} = \{p_2\ t_2^{(2)}\ p_5\ t_6\ p_{15}\ t_{10}^{(2)}\ p_{18}\ t_{12}\}$$

For concreteness, let $f(t_i^{(j)}) = f(t_i) = \lceil i/3 \rceil$. The overall cycle time $\bar{\tau}_1 = \max(\tau_{11}, \tau_{12}, \tau_{13}) = 12$. For components, Y_1, Y_2, ..., and Y_6, the cycle times are 9,10,14,14, and 5 (if MP(12) = 1), respectively. Magott [10] generalized the technique by showing that the solution could be expressed as a linear program. These methods assume a firing epoch begins as soon as a transition is enabled. When modeling real systems, this is not an unreasonable assumption.

As shown in Corollary 1, free choice net transitions that potentially conflict have the same input place. Because the T-subset components produced as above are strongly connected marked graphs, a place has but one output arc. It follows that a given component is reached from the initial marking by a single transition whose firing is randomly chosen from a conflict set.

With this in mind, we are prepared to deal with the probability mass functions over the output links of selective actors. If the actors are unit selective then $Pr(F_2(a,M)) > 0$ for each $\ell \in F_2(a,M)$ and is zero for nonsingleton elements of the powerset of $O(a)$. If $t \sim a$ and $p \sim \ell \in O(a)$ then let $Pr(t) = Pr(F_2(a,M))$ for ℓ. This contrasts to the normal method of assigning probabilities to markings in the reachability graph but is equivalent due to the partitioning of the transitions described above.

As an illustration of the employment of the foregoing, define $MTTE(t_j)$ to be the mean time to the event of the beginning of the firing epoch for t_j. Assume t_j has s input places with s independent loop-free paths $\sigma_1, \sigma_2, \ldots, \sigma_s$ from MP_o. (Dependent paths can be dealt with but serve no purpose at present.) t_j cannot fire until the last token appears at an input.

$$MTTE(t_j) = \max [G(\sigma_1), G(\sigma_s), \ldots, G(\sigma_s)]$$

$G(\sigma_i)$ is the expected time required to transverse σ_i and incorporates the time intervals the tokens leave the path and traverse circuits.

For clarity, G will be defined with superscripted subscripts distinguished. Let $t_k^{(m)}$ $\underline{adj}$ σ_i mean that $t_k^{(m)} \notin \sigma_i$ while an equivalently subscripted transition is $(t_k^{(h)} \varepsilon \sigma_i, h \neq m)$. By extension, Y_i $\underline{adj}$ σ_i means $t_k^{(m)} \varepsilon Y_i$ and $t_k^{(m)}$ $\underline{adj}$ σ_i. Let

$$\lambda_i = \prod_{t_k^{(m)} \varepsilon Y_i} Pr(t_k^{(m)})$$

if Y_i $\underline{adj}$ σ_j and is undefined otherwise.

$$G(\sigma_j) = \sum_{t_k \varepsilon \sigma_j} f(t_k) + \sum_{t_k^{(m)} \varepsilon \sigma_j} [f(t_k^{(m)}) + A(t_k^{(m)})$$

The latter term, $A(\cdot)$, represents the expected amount of time within components adjacent to σ_j at transition $t_k^{(m)}$. Let $\lambda_1, \lambda_2, \ldots, \lambda_r$ be the probabilities of the r components adjacent at $t_k^{(m)}$. Let $\lambda_{r+1} = Pr(t_k^{(m)})$. (Note that $\sum_{i=1}^{r+1} \lambda_i = 1.$) $\bar{\tau}_i'$ stands for the expected cycle time for component Y_i. That is, $\bar{\tau}_i'$ differs from $\bar{\tau}_i$ in that it takes into account components adjacent to circuits in Y_i.

$$\tau'_{ij} = G(K_{ij}) / \sum_{p_k \varepsilon K_{ij}} MP(p_k)$$

If, to simplify the subscript scheme, $Y_1, Y_2, \ldots, Y_r$ are adjacent to $t_k^{(m)}$,

$$A(t_k^{(m)}) = \lambda_{r+1} \sum_{m_0=0}^{\infty} \sum_{m_1=0}^{m_0} \cdots \sum_{m_{r-1}}^{m_{r-2}} \frac{m_0!}{m_1! m_2! \cdots m_{r-1}! (m_0 - m_1 - \cdots - m_{r-1})!}$$

$$\{[m_1 \bar{\tau}'_1 + m_2 \bar{\tau}'_2 + \cdots + m_{r-1} \bar{\tau}'_{r-1} + (m_0 - m_1 - \cdots - m_{r-1}) \bar{\tau}'_r] \lambda_1\}^{m_1} \lambda_2^{m_2} \cdots \lambda_{r-1}^{m_{r-1}} \lambda_r^{(m_0 - m_1 - \cdots - m_{r-1})}$$

If r = 1 then $A(\cdot)$ reduces to the geometric series multipled by a constant value.

Each path σ_i defines a hierarchy over the set of components. For example, in Fig. 3, the path $\{p_1\ t_1\ p_6\ t_4^{(2)}\ p_{10}\}$ has Y_3 and Y_5 immediately adjacent (at $t_4^{(2)}$) while Y_6 is adjacent to one of the circuits in Y_5. Thus, one must solve Y_6 before Y_5 can be solved.

To illustrate, for Figure 3 let $Pr(t_k^{(1)}) = 1/k$ and $Pr(t_k^{(2)} = 1 - 1/k$. For $MTTE(t_6)$,

$$\sigma_1 = \{p_1\ t_1\ p_6\ t_4^{(2)} p_{10}\}$$

$$\sigma_2 = \{p_2\ t_2^{(2)} p_5\}$$

Table 1 contains the cycles for Y_2, Y_3, Y_5, and Y_6 which are the only components needed. Table 2 contains the relevant intermediate calculations.

$MTTE(t_6) = \max[5.88, 19] = 19$

Given the isomorphism theorem and the equivalence of timing between actors and transitions, it can be concluded that $MTTE(a_j) = MTTE(t_j)$. If the transition is superscripted then $MTTE(a_j) = \min_k \{MTTE(t_j^{(k)})\}$. The mean time to event is but one measure possible. From it, other measures such as mean time between events can be derived and correlated to components in the real system.

TABLE 1. Circuits

K_{ij}	Sequence
K_{21}	$\{p_2\ t_2^{(1)}\ p_4\ t_{10}^{(1)}\ p_{18}\ t_{12}\}$
K_{31}	$\{p1\ t_1\ p_3\ t_3^{(2)}\ p_8\ t_8^{(2)}\ p_{16}\ t_{11}\}$
K_{332}	$\{p_1\ t_1\ p_6\ t_4^{(1)}\ p_9\ t_9^{(1)}\ p_{17}\ t_{11}\}$
K_{51}	$\{p_1\ t_1\ p_3\ t_3^{(1)}\ p_7\ t_5^{(2)}\ p_{12}\ t_7^{(2)}\ p_{13}\ t_8^{(1)}\ p_{16}\ t_{11}\}$
K_{52}	$\{p_1\ t_1\ p_6\ t_4^{(1)}\ p_9\ t_9^{(1)}\ p_{17}\ t_{11}\}$
K_{61}	$\{p_{12}\ t_7^{(1)}\ p_{11}\ t_5^{(1)}\}$

TABLE 2. Intermediate MTTE Calculations

Path	τ'_{ij}	G	$\bar{\tau}'_i$	λ_i
K_{61}	5	5	$\bar{t}'_6=5$	1/7
K_{52}	10	10	--	--
K_{51}	14.97	14.97	$\bar{\tau}'_5=14.97$	1/12
K_{32}	10	10	--	--
K_{31}	9	9	$\bar{\tau}'_3$	1/6
K_{21}	9	9	$\bar{\tau}'_2$	1/2
σ_1	--	5.88	--	--
σ_2	--	19	--	--

V. SUMMARY AND CONCLUSIONS

Dataflow graphs are useful representations for abstract computations, generally rendering models that are easily related to the real system being modeled. Petri nets, while less powerful computationally, have been studied intensively, giving rise to a large body of analytic methods. Here, we have shown that a significant class of dataflow graphs can be effectively transformed to Petri nets. The applications of this class have been primarily in modeling concurrency in computer systems. Thus, the isomorphism between the two computational models allows considerable analytic capability to be employed.

Within the applications context, timed transitions and probabilistic resolution of nondeterminism are introduced. Using these extensions, a measure (Mean Time to Event) was illustrated. The principle behind the derivation was the determination of overall behavior by examination of the contained components. This principle is currently being exploited to determine other properties of the net.

REFERENCES

[1] B. Bayaert, G. Florin, P. Lone, and S. Natkin, "Evaluation of Computer Systems Dependability," Proc. Symp. on Fault Tolerant Computing, 1981, pp. 79-81.

[2] U. N. Bhat, K. M. Kavi, and B. P. Buckles, "Reliability Analysis of Dataflow Graph Models," Submitted for publication.

[3] A. Datta and S.Ghosh, "Synthesis of a Class of Deadlock-free Petri Nets, "J. Ass. Comput. Mach., Vol. 31, No. 3, pp. 486-506,July 1984.

[4] J. E. Coolahan, Jr. and N. Roussopoulos, "Timing Requirements for Time-Driven Systems Using Augmented Petri Nets," IEEE Trans. Software Engineering., Vol. SE-9, No. 5, p. 603-616, Sept. 1983.

[5] R. Geist, K S. Trivedi, J.B. Dugan and M. Smotherman, "Design of the Hybrid Automated Reliability of Predictor," Proc. IEEE/AIAA Digital Avionics Systems Conf., Oct. 1983, pp. 16.5.1-16.5.8.

[6]. M. H. T. Hack, "Analysis of Production Schemata by Petri Nets," Tech. Rep. 94, Project MAC, MIT, 1972, 119p.

[7] K. M. Kavi, "Data Flow Modeling Techniques," Proc. of IASTED Intl. Conf. Sim. Modl., Nov. 1983, pp. 1-4.

[8] K. M. Kavi, B. P. Buckles and U. N. Bhat, "A Formal Definition of Dataflow Graph Models," Submitted to for publication.

[9] W. P. Karplus and A. Makoui, "The Role of Data Flow Methods in Continuous Systems Simulation," Proc. Summer Simulation Conf., Denver, CO, 1982.

[10] J. Magott, "Performance Evaluation of Concurrent Systems Using Petri Nets," Info. Processing Letters, Vol. 18, No. 1, pp. 7-13, Jan. 1984.

[11] M. K. Molloy, "On the Integrating Delay and Throughput Measures in Distributed Processing Models," Ph.D. Dissertation, Univ. Calif. Los Angeles, 1981.

[12] M. K. Molloy, "Performance Analysis Using Stochastic Petri Nets," IEEE Trans. Comput., Vol. C-51, No. 9, pp. 913-917, Sept. 1982.

[13] J. L. Peterson, "Petri Nets," Assoc. Comput. Mach. Surveys, Vol. 9, Sept. 1977, pp. 233-352.

[14] J. L. Peterson, Petri Net Theory and the Modeling of Systems, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[15] C. Ramchandani, "Analysis of Asynchronous Systems by Petri Nets," Tech. Rep. 120, Project MAC, MIT, 1974.

[16] C. V. Ramamoorthy and G. S. Ho, "Performance of Asynchronous Concurrent Systems Using Petri Nets," IEEE Trans. Soft. Engr., Vol. SE-6, pp. 440-449, Sept. 1980.

[17] J. Sifakis, "Performance Evaluation of Systems Using Nets," in Net Theory and Applications, (W. Brower, ed.), Springer-Verlag, NY, 1980, pp. 307-319.

[18] V. P.Srini and J. F.Asenjo, "Analysis of Cray 1S Architecture," Proc. 10th Intern. Symp. on Comput. Arch., June 1983, pp. 194-206.

[19] W. M. Zuberek, "Timed Petri-Nets and Preliminary Performance Evaluation," Proc. 7th Annual Symp. on Comp. Arch., 1980, pp. 88-96.

APPENDIX 4

RELIABILITY ANALYSIS OF DATAFLOW GRAPH MODELS

RELIABILITY ANALYSIS OF DATAFLOW GRAPH MODELS

Krishna M. Kavi

University of Texas at Arlington

U. Narayan Bhat

Southern Methodist University

and

Bill P. Buckles

University of Texas at Arlington

This work is supported in part by NASA-Ames Research Center under Grant NAG-2-273.

All correspondence should be directed to Krishna M. Kavi, Dept. of Computer Science and Engineering, P.O. Box 19015, University of Texas at Arlington, Arlington, Texas 76019-0015.

RELIABILITY ANALYSIS OF DATAFLOW GRAPH MODELS

Krishna M. Kavi
U. Narayan Bhat
Bill P. Buckles

Abstract

A dataflow graph can be used as a generalized Model of computation. Uninterpreted dataflow graphs can be used to analyze the reliability of computer systems including parallel processors and data driven systems. The nondeterminism arising due to the uninterpreted nature is resolved by associating probability distributions with dataflow actors. Both Markov and path ennumeration techniques can be used to study the reliability of dataflow graph models. Large graphs should be reduced to simplify computational complexity.

Key words: Dataflow Graphs, Stochastic Petri Nets, Markov Chains, Reliability, Graph Clustering, Path Ennumeration.

1. INTRODUCTION

The demands for concurrent operation within a computer system and the representation of parallelism in programming languages have yielded a new form of program representation known as dataflow ([DENN 74], [DENN 75], [TREL 82]). Execution of dataflow programs is data driven; that is, each instruction is enabled for execution just when each required operand has been supplied by the completion of the predecessor instruction.

Dataflow systems have received considerable attention during the past several years. However, much of the research in dataflow processing has dealt with defining the functionality, designing instruction level architecture or specifying programming languages. This has not made urgent the formalization of the dataflow itself. Formalization is necessary in relating dataflow to other computation models, discovering properties of specific instances of dataflow graphs and in performance and reliability evaluations. Because of the complexity of dataflow systems, the presence of multiple processing units and communication circuits, the reliability, performance of the interconnection structures, and scheduling schemes in such architectures become significant issues, before dataflow architectures can be used for next generation machines. Formalization of the dataflow model also makes possible the utilization of dataflow graph as an abstract model of computation analogous to Turing machines and Petri nets. In [KAVI 84], we have presented a formal definition of dataflow models, and in [KAVI 85], isomorphic transformation of dataflow graphs into free-choice Petri nets have been presented.

Our interest in dataflow graph is its potential to represent any computation structure, including parallel processing architectures and even dataflow systems. Miller [MILL 73] has surveyed models other than dataflow graphs for representing computational aspects of parallel processes. The chief advantage of dataflow graphs over other models is their compactness and general amenability to direct interpretation. That is, the translation from the conceived system to a dataflow graph is straightforward and, once accomplished, it is equally straightforward to determine by inspection which aspects of the system are represented [KAVI 83]. Dataflow graphs are hierarchical; a node in the graph can be a simple node representing a single hardware unit, or a dataflow subgraph representing an entire processor or an interconnection network. Thus models of computer systems using dataflow graphs can be used to selectively change the level of simulation to any level of detail simply by expanding nodes into dataflow subgraphs without modifying other parts of the model.

In this paper we outline methods for estimating the reliability of a dataflow graph model. Thus, computer systems including parallel processors, dataflow machines and ultrareliable computers with high redundancy can be represented as dataflow graphs, dataflow simulators (for example DFDLS [KAVI 83]) can be used to study their functionality, and the reliability of the simulated computer can be calculated using the techniques described in this paper. We will introduce abstract dataflow graphs in the next section and the techniques for estimating the reliability of dataflow graphs will be described in Section 3.

2. UNINTERPRETED DATAFLOW GRAPHS

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

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

Here, $A = \{a_1, a_2, \ldots, a_\eta\}$ is the set of actors

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

$E = \{A \times L\} \cup \{L \times A\}$ is the set of edges.

S ($S \subset L$) is a non-empty set of links called starting set (input links).

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

T ($T \subset L$) is a non-empty set of links called terminating set (output links).

$$T = \{ l \in L \mid (l,a) \notin E \ \forall a \in A\} \tag{4}$$

The set of input links of an actor a and output links of an actor a are denoted as $I(a)$ and $O(a)$.

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

$$O(a) = \{ l \in L \mid (a,l) \in E\} \tag{6}$$

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

Transitive closure on these sets, $I(a)^+$, $O(a)^+$, $I(l)^+$, $O(l)^+$ for actors and links can be defined; For example, $I(a)^+$ is the set of all links that are inputs to a, or input links to actors that feed the input links of a, and so on.

If $B \subset A$ is a subset of actors then $I(B)$ and $O(B)$ define the set of links that are inputs links and output links of actors in B.

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

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

Definition 2: For a dataflow graph the following conditions are true.

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

Although this definition seems to be restrictive since the links can-not have more than one input actor and one output actor, we have been able to rewrite all dataflow graphs by introducing dummy actors (for example, to duplicate an input token onto several output links). This definition allows us to isomorphically map dataflow graphs into free-choice Petri nets and free-choice nets into dataflow graphs [KAVI 85].

Uninterpreted dataflow graphs: For the purpose of studying the performance and reliability of dataflow graph models of computer systems, we introduce uninterpreted dataflow graphs, in that the actual meaning of the functions performed by the actors and the semantics of the data tokens on arcs is not relevant. The presence of data tokens on arcs are used only as triggering signals to enable actors. We use the term dataflow graph to mean uninterpreted dataflow graph throughout the paper.

Definition 3: A marking is a function $M: L \longrightarrow \{0,1\}$ (10)

A link l 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 (non-empty) subset of starting set of links contain tokens. A terminal marking M_t is a marking in which a (non-empty) subset of terminating set of links contain tokens.

2.1. FIRING AND FIRING SEMANTIC SETS

Associated with each actor are two sets of links called <u>input firing semantic set</u> F_1 and <u>output firing semantic set</u> F_2

$$F_1(a,M) \subseteq I(a)$$
$$F_2(a,M) \subseteq C(a) \qquad (11)$$

The input firing semantic set F_1 refers to the subset of input links that must contain tokens to enable the actor; the output firing semantic set F_2 refers to the subset of links that receive tokens when the actor is fired.

Definition 4: A <u>firing</u> is a partial mapping from markings to markings.

An actor is fireable at a marking M if the following conditions hold

$$M(l) = 1 \quad \text{for all } l \in F_1(a,M)$$
$$M(l) = 0 \quad \text{for all } l \in F_2(a,M) \qquad (12)$$

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

$$M'(l) = 0 \quad \text{if } l \in F_1(a,M)$$
$$= 1 \quad \text{if } l \in F_2(a,M)$$
$$= M(l) \quad \text{otherwise} \qquad (13)$$

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

Depending on whethe. F_1 and F_2 select only one, a proper subset or the entire set of input and output links, the following node firing rules are defined.

Conjunctive: All the input links must contain tokens for the actor to fire. That is, $F_1(a,M) = I(a)$ for all M. (14)

Disjunctive: Only one of the input links must contain a token for the actor to fire. That is, $F_1(a,M) \in I(a)$ for all M (15)

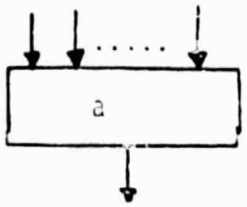
Collective: One or more of the input links may contain tokens for the actor to fire. That is, $F_1(a,M) \subset I(a)$ for all M. (16)

Selective: When the actor fires, only one of the output links receives a token. That is, $F_2(a,M) \in O(a)$ for all M. (17)

Distributive: When the actor fires, all the output links receive tokens. That is, $F_2(a,M) = O(a)$ for all M. (18)

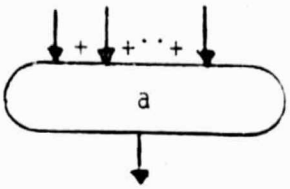
The graphical representations of these possibilities are shown in Fig. 1. In this paper we will not deal with collective actors. They can be replaced by a number of conjunctive actors with inputs corresponding to the subsets of I(a).

NON-DETERMINISTIC FIRING SEMANTICS: Since, in our study, uninterpreted dataflow graph models are used for analyzing reliability by applying stochastic methods, F_1 and F_2 are made non-deterministic; for different instances of execution of an actor, the firing semantic sets may be different. This eliminates the need for control links and arcs in dataflow 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 [DENN 74] shown in Fig. 2(a) will be replaced by Fig. 2(b). The firing semantic sets are



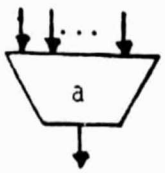
a) Conjunctive

$F_1(a,M) = I(a)$



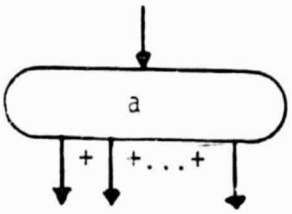
b) Disjunctive

$|F_1(a, M)| = 1$



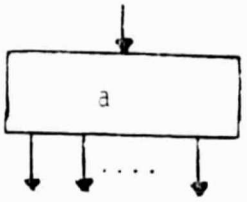
c) Collective

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



d) Selective

$|F_2(a,M)| = 1$



e) Distrubutive

$F_2(a,M) = O(a)$

Fig.1.. Firing Rules

$$F_1 = I(T) = \ell_1$$

$$F_2 = O(T) = \ell_2 \quad \text{with probability } p$$

$$= \phi \quad \text{with probability } 1-p$$

The probability p depends on the frequency of having TRUE value on the control arc of the T-gate.

(a) (b)

Fig. 2. Non-Deterministic Firing

$P[F_1(a,M)]$ is the probability distribution that in marking M, $F_1(a,M)$ is the input firing semantic set; this probability distribution 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. Similarly, $P[E_2(a,M)]$ defines the probability distribution on the output firing semantic set; when a fires in M, the links in $F_2(a,M)$ will receive tokens with a probability $P[F_2(a,M)]$.

3. RELIABILITY ANALYSIS OF DATAFLOW GRAPHS

Network Reliability Analysis: The reliability of communication networks has attracted considerable interest among researchers. When such networks are treated as directed graphs, the reliability can be analyzed by reducing the communication networks to series-parallel networks [MISR 70a], by enumerating paths ([MISR 70b], [FRAT 73]), or by enumurating cut-sets ([LIN 76], [JENS 69]). For a general non series-parallel networks only the last two methods yield exact solutions. However, since they are not tractable for large networks approximate methods become necessary ([JENS 69], [NELS 70], [BATT 71]).

For any m path network the number of non-cancelling terms in the reliability expression resulting from the use of the inclusion-exclusion formula of probability theory [FELL 68, pp 99-108] is $r \leq 2^{m-1}$. However, in practical networks some of the paths are contained in the union of other paths, thus $r << 2^{m-1}$. In [SATY 78], a topological formula and an algorithm are presented for finding the reliability of a network from a given source node to a terminal node, where only non-cancelling terms are included in the formula. This method has been extended to obtain reliability from a single source node to multiple terminal nodes. The reader is referred to [AGRA 84] for a survey of such network reliability methods.

These methods cannot be applied directly to dataflow graphs. Although dataflow graphs are directed graphs and they resemble networks, there exist important differences due to the firing semantics described in the previous section. For example, for

conjunctive actors, all the input links must contain tokens to enable the actor to fire, requiring reliable input paths leading to the actors. Similarly, for distributive actors, all paths starting at the output of the actors are required for the successful completion of the function described by the dataflow graph. Whreas in (communication) networks, only one path is necessary for reliable operation.

A second difference arises due to the nondeterministic nature of dataflow graphs introduced in this paper. Probabilities are associated with paths (section 2), and these probabilities cannot easily be incorporated into the networks obtained from dataflow graphs. For this reason, we introduce a different method of obtaining the reliability of a dataflow graph.

3.1. RELIABILITY OF DATAFLOW GRAPHS

Reliability of a dataflow graph can be defined as the probability of successful completion of a sequence of operations to be performed by the actors of the graph. Thus, if this sequence is identified as the path of a particle traversing the graph, then the reliability is the probability of occurrence of a successful path.

The reliability of a dataflow graph can be determined in two stages. At the first stage, reliabilities of subgraphs (some of them can be single actors) are calculated. At the second stage the reliabilities of the subgraphs are combined appropriately based on the topological structure of the graph. If multiple redundant paths are simultaneously used, the procedure of combining subgraph reliabilities becomes complicated.

In what follows we shall consider the successful conclusion of the operation as the event without any concern to the time needed for the operation. Incorporation of the time element compounds the problem and it will be considered in future research.

Let A_1 , A_2 , ..., A_n be the actors (some of them can be subgraphs) of a dataflow graph and let R_1 , R_2 , ..., R_n be their reliabilities. Unless otherwise stated we will assume that the performance of an actor (or subgraph) is independent of the outcome of any other actor.

Keeping with the analogy of a particle mentioned earlier, the movement of the particle from one node to another will be called a transition. If the transitions among actors in a subgraph $\{A_1 , A_2 , ..., A_n\}$ are hierarchical, say $A_1 \text{-->} A_2 \text{-->} ... \text{-->} A_n$ (or any other arrangement), then the reliability of the subgraph is obtained as the product $\prod_i R_i$. But if the transitions are Markovian with the probability of transition $A_i \text{--> } A_j$ being p_{ij} , then the reliability of the subgraph can be calculated using Markov chain methods.

For instance, consider a Markov chain with two states 1 and 2 (the corresponding subgraph contains two actors). Let the transitions among the states be represented by the elements of the transition probability matrix

	1	2
1	p_{11}	p_{12}
2	p_{21}	p_{22}

where p_{ij} is the probability of transition i --> j. For purposes of reliability modeling we introduce states 1F, 2F and 0 of which

1F and 2F represent failure of actors 1 and 2, and 0 represents the successful completion of operation. Let f_1 and f_2 be the probabilities that actors 1 and 2 fail. Now we can consider an expanded Markov chain whose transition probability matrix is given by

	0	1F	2F	1	2
0	1				
1F		1			
2F			1		
1		f_1		$(1-f_1)p_{11}$	$(1-f_1)p_{12}$
2			f_2	$(1-f_2)p_{21}$	$(1-f_2)p_{22}$

$$= \left[\begin{array}{c|cc} I & 0 \\ \hline \phi & Q \end{array}\right]$$

which can be partitioned as shown. If a particle starts out from state 1 or 2, it is well known [BHAT 84, Chap 4] that the first passage probabilities of the particle into states 1F or 2F (these are the unreliabilities) in n transitions are given by the elements of the second and third columns of the matrix $Q^{n-1} \cdot \bar{\phi}$. The corresponding probabilities for reliable operation without regard to time are given by the elements of the first column of $(I-Q)^{-1} \cdot \bar{\phi}$.

At this point we may note the similarity between dataflow graphs and Petri nets. Stochastic Petri nets ([COOL 83], [GEIS 83], [MOLL 81], [MOLL 82], [SIFA 80], [ZUBE 80]) have been used for performance and reliability analyses of concurrent and asynchronous computer systems. Typically, the markings are mapped into a Markov state space ([MOLL 81], [MOLL 82]). The markings in

a dataflow graph can also be mapped similarly into a Markov state space for the purpose of studying the reliability of the dataflow graph. When dataflow graphs contain strong subgraphs it is convenient to obtain the reliabilities of the subgraphs so that the subgraphs can be replaced by single actors (thus producing a reduced graph). The corresponding markov process can be arranged in a canonical form such that the recurrent equivalent classes can be easily identified. Kavi ([KAVI 79] [BHAT 84, chap 4]) has collected algorithms useful for this purpose. Appendix contains a brief description of these methods.

However, eventhough this method is applicable in the general case, many times it unnecessarily complicates the problem. For a large dataflow graph with m links (the number of links is greater than the number of actors), the Markov chain contains as many as $2^m - 1$ states. So, Markov chain method should be used only to small subgraphs and the reliabilities of subgraphs should be combineed using the technique discussed below.

In the second stage of combining subgraph (or actor) reliabilities to get the reliability of the complete graph we may develop certain rules based on the type of firing rules discussed in section 2.

In addition to the reliabilities R_1 , R_2 , ..., R_n ,also define C_{ij} (i,j = 1, 2, ..., n) as the reliability of the link connecting actor A_i to actor A_j (which can also be considered as the reliability of the communication channel). Let R_{ij} be the reliability of the path A_i --> A_j , including the reliability of A_i , but excluding the reliability of A_j . For the four distinct

types of firings defined in section 2 (collective actors are not considered here), we have the expresssion shown in Fig. 3.

While using these expressions to determine the reliability of a path, the following observations should be noted.

a). Conjunctive and distributive actors are indicative of parallel paths all of which are used. When the paths are independent of each other, the reliability of the graph (or subgraph) consisting of parallel paths is obtained as the product of reliabilities of individual paths.

b). Disjunctive and selective actors result in more than one path, only one of which is used. The reliability of the graph (or subgraph) with such paths is obtained by combining the reliabilities of individual paths using the probabilities of paths as weights.

c). When the paths are not independent, the dependent structure determines how the reliabilities of individual actors (or subgraphs) are combined to get the reliability of the graph.

d). When multiple redundant paths are simultaneously used, the reliability is obtained using the inclusion-exclusion formula of probability theory [FELL 68, pp 99-109].

An algorithm to determine the reliability of a graph can be described as follows.

Algorithm A1

Step1: Identify subgraphs (see Appendix)

Step2: Obtain reliabilities of subgraphs using either Markov chains (Appendix), or using this algorithm recursively.

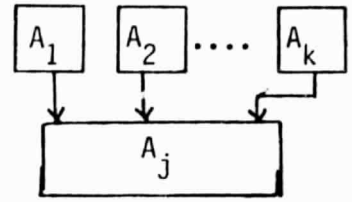
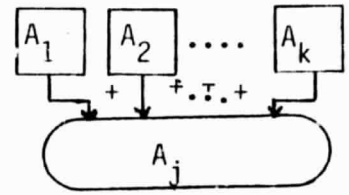
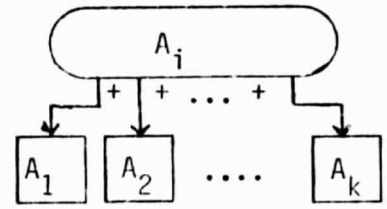
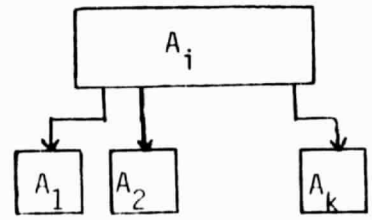
Actor type	Reliability	
1. Conjunctive: $(A_1, A_2, \ldots, A_k) \longrightarrow A_j$	$R_{ij} = R_i C_{ij}$ for $i = 1,2,..,k$	
2. Disjunctive: $A_i \longrightarrow A_j$ with probability p_{ij}	$R_{ij} = p_{ij} R_i C_{ij}$ for $i = 1,2,\ldots,k$	
3. Selective: $A_i \longrightarrow A_j$ with probability p_{ij}	$R_{ij} = p_{ij} R_i C_{ij}$ for $j = 1,2,\ldots,k$	
4. Distributive: $A_i \longrightarrow (A_1, A_2, \ldots, A_k)$	$R_{ij} = R_i C_{ij}$ for $j = 1,2,\ldots,k$	

Fig. 3. Reliability Expressions for Dataflow Actors

Step3: Replace strong subgraphs by single actors producing a reduced graph.

Step4: Identity distinct paths (say using depth-first search)

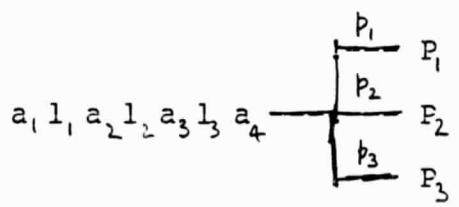
Step5: Combine actor (subgraph) reliabilities using firing types to determine the reliability for each path.

Step6: Combine path reliabilities to give the reliability of the entire graph.

Given below are two examples; one a dataflow graph of a simple computer system (Fig. 4) and the second a dataflow graph representing triple modular redundancy (Fig. 5). In the first example, actors perform independently of each other, whereas in the second example there exists such a dependency among the paths.

Eexample 1: Dataflow Model of a Simple Computer System. Baer [BAER 80, p 71] gave a Petri net model representing the control flow in the execution of an instruction in a single accumulator arithmetic and logic unit. Fig. 4 shows a dataflow equivalent of the Petri net given by Baer. The actors are intentionally named by the events in order to facilitate interpretation. In order to simplify the notations we use a's and l's to represent the actors and links as well as their reliabilities.

The three distinct paths in the graph are identified below.



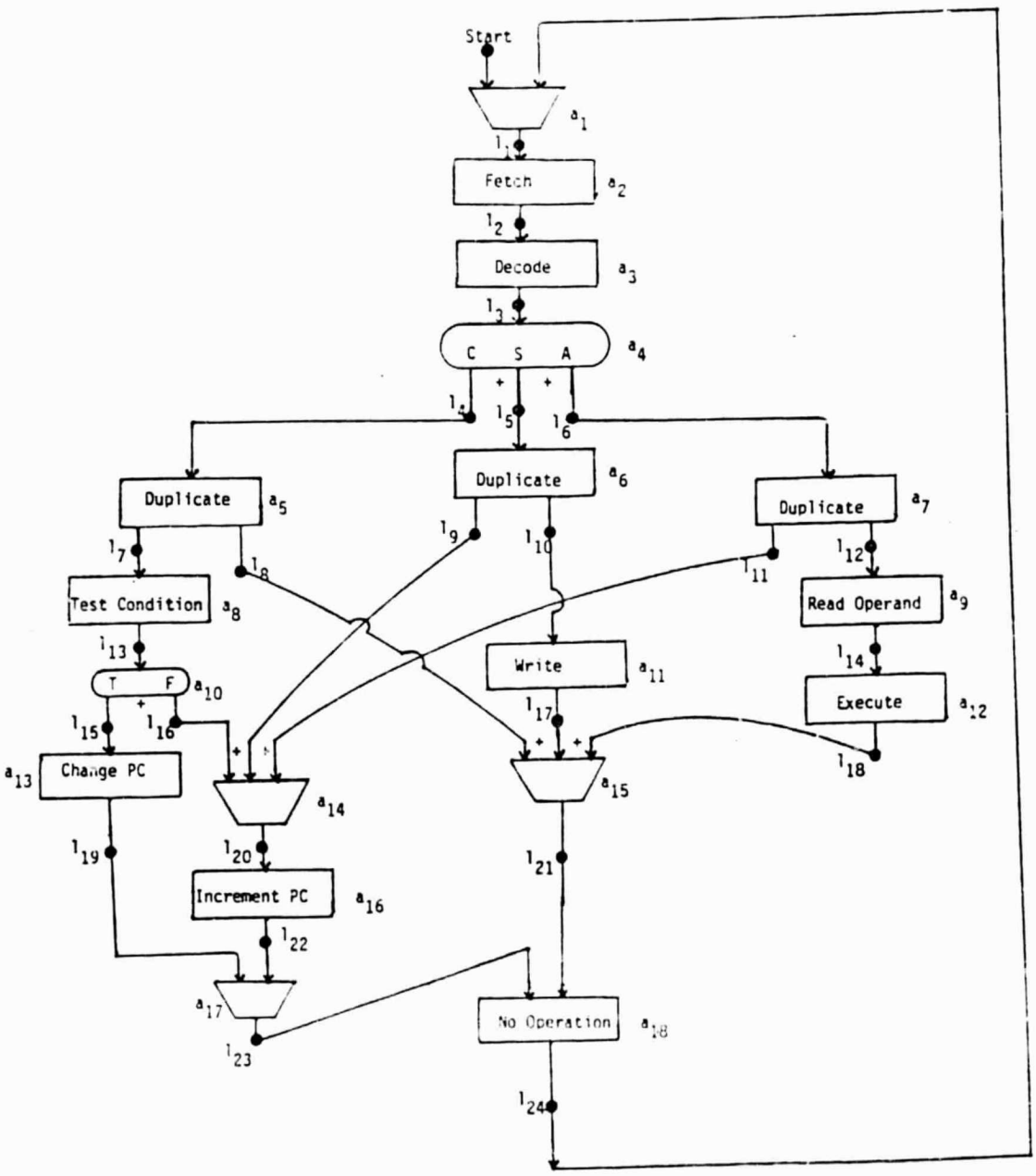
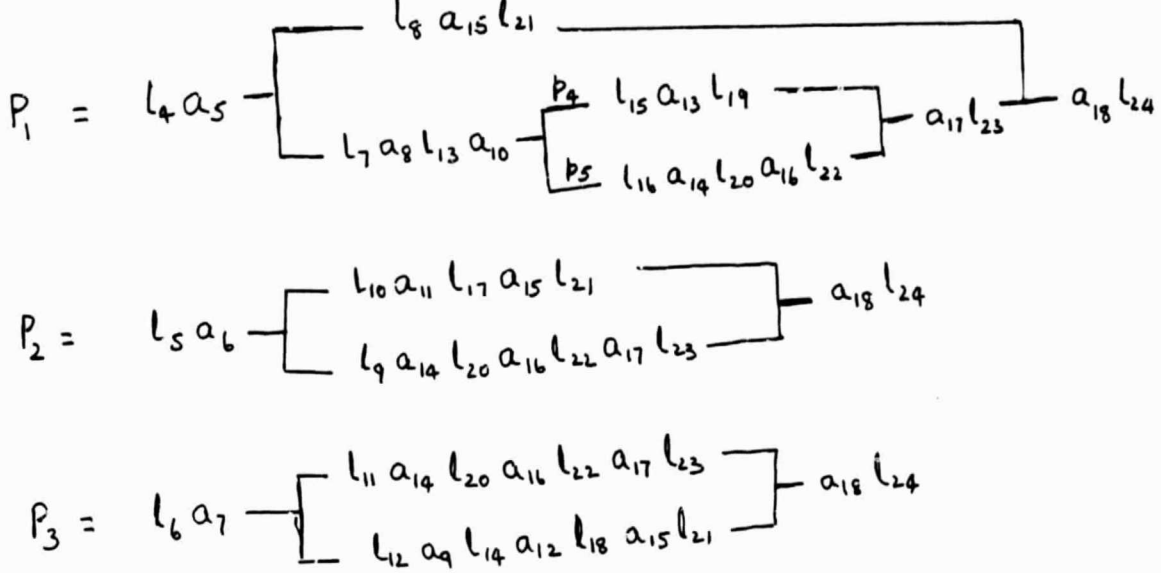


Fig. 4. Dataflow Graph of a Simple Computer System



The probabilities p_1 , p_2 , p_3 indicate the frequency of Conditional, Store and Arithmetic instructions (in a typical program) while p_4 , p_5 are the probabilities that a condition will or will not be satisfied.

The reliabilities of various groups of nodes in subpath P_1 are given below.

$$R_1 = l_8\ a_{15}\ l_{21}$$

$$R_2 = l_7\ a_8\ l_{13}\ a_{10}$$

$$R_3 = l_{15}\ a_{13}\ l_{19}$$

$$R_4 = l_{16}\ a_{14}\ l_{20}\ a_{16}\ l_{22}$$

Let $R^{(1)}$ be reliability of subpath P_1. Then

$$R^{(1)} = l_4\ a_5\ R_1\ R_2\ (p_4\ R_3 + p_5\ R_4)\ a_{17}\ l_{23}\ a_{18}\ l_{24}$$

Let $R^{(2)}$ and $R^{(3)}$ be the reliabilities of subpaths P_2 and P_3 determined in a similar manner. Then the system reliability is

$$R = a_1 l_1 a_2 l_2 a_3 l_3 a_4 (p_1 R^{(1)} + p_2 R^{(2)} + p_3 R^{(3)})$$

Example 2: Dataflow Graph of Triple Modular Redundancy. The most common design to enhance the reliability of a system is to use redundancy so that faults can be masked. In a triple modular redundancy (TMR), the circuit is triplicated and the voter performs the majority function. Fig. 5 shows the dataflow graph of TMR. U_1, U_2, and U_3 are the triplicated units. The function of the voter is shown in detail where N_1 will produce a token on W (working) output when all the three units are functioning and a token on F (failed) output otherwise. N_2 , N_3 , N_4 work similary testing if two units are functioning correctly. Dependency among N_1 , N_2 , N_3 , N_4 should be noted. Because of this dependency, the reliability of paths will have to be considered using a conditional probability argument. Assuming that voter actors (N_1 , N_2 , N_3, N_4) do not fail, and using U_1 , U_2 , U_3 to represent the events that the units are working and U_1^c, U_2^c, U_3^c to represent failures, we have

$$\begin{aligned} R_{TMR} = {} & P(U_1 U_2 U_3) + P\left[U_1 U_2 U_3^c \cap (U_1 U_2 U_3)^c\right] \\ & + P\left[U_1 U_2^c U_3 \cap (U_1 U_2 U_3)^c\right] \\ & + P\left[U_1^c U_2 U_3 \cap (U_1 U_2 U_3)^c\right] \end{aligned}$$

But, $U_1 U_2 U_3^c \subset (U_1 U_2 U_3)^c$

$$P\left[U_1 U_2 U_3^c \cap (U_1 U_2 U_3)^c\right] = P\left[U_1 U_2 U_3^c\right]$$

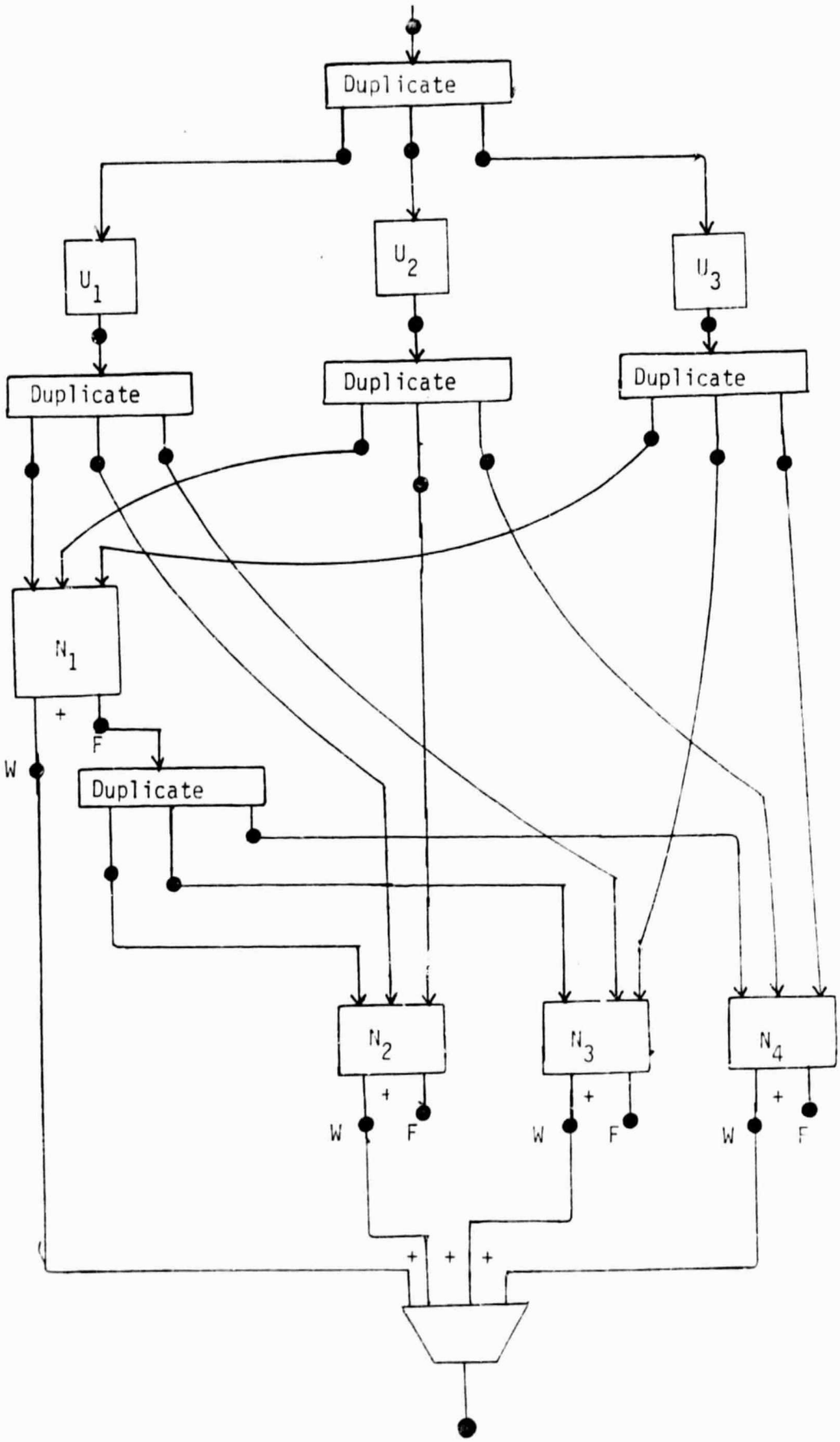


Fig. 5. Dataflow Graph of Triple Modular Redundancy

Similarly after simplifying the other two terms we get

$$R_{TMR} = P(v_1 v_2 v_3) + P(v_1 v_2 v_3^c) + P(v_1 v_2^c v_3) + P(v_1^c v_2 v_3)$$

If each of the units has a reliability of R and they are independent of each other, the reliability of TMR is

$$R_{TMR} = R^3 + 3R^2(1-R) = 3R^2 - 2R^3$$

Let V be reliability of each of the actors in the voter, then

$$R_{TMR} = VR^3 + 3V^2R^2(1-R)$$

When the fault model includes repair or fault coverage, Markov chain techniques would be applicable [GEIS 83].

4. SUMMARY AND CONCLUSIONS

Much of the research in dataflow processing ignores the need for the formalism of the dataflow model itself. In our research we have developed a formal definition of dataflow graphs. Uninterpreted dataflow graphs can be used to model computer systems including parallel processors and data driven system; the reliability of the modeled computer can be studied by analyzing the dataflow graph. We have introduced stochastic properties with dataflow actors. The reliability of dataflow graphs are analyzed using both Markov and path enumeration techniques. We have outlined how a large dataflow graph can be condensed using graph clustering methods. These clusters can be analyzed separately and the results can be combined to obtain overall system reliability.

The time needed for an actor to complete its operation is not included in our analyses of dataflow graphs. Incorporation of the time element compounds the problem and it will be studied in future research.

5. REFERENCES

[AGRA 84] A. Agrawal and R.E. Barlow. "A Survey of Network Reliability and Domain Theory", Operations Research, Vol. 32, No. 3, May-June 1984, pp 478-526.

[BAER 80] J.L. Baer. Computer Systems Architecture, Computer Science Press, 1980.

[BATT 71] J.R. Batts. "A Computer Program for Approximating System Reliability - Part II", IEEE Tr. Reliability, Vol. R-20, Mar. 1971, pp 88-90.

[BHAT 84] U.N. Bhat. Elements of Applied Stochastic Processes, 2nd Ed., Wiley, 1984.

[CHRI 75] N. Christofides. Graph Theory - An Algorithmic Approach, Academic Press, 1975.

[COOL 83] J.E. Coolahan and N. Roussopoulos. "Timing Requirements for Time-Driven Systems Using Augmented Petri Nets", IEEE Tr. Software Engineering, Vol. SE-9, No. 5, Sept. 1983, pp 603-616.

[DENN 74] J.B. Dennis. "First Version of Data Flow Procedure Language", Lecture Notes in Computer Science, Vol. 19, 1974, Springer-Verlag.

[DENN 75] J.B. Dennis and D.P. Misunas. "A Priliminary Architecture for a Basic Data Flow Processor", Proc. 2nd Annual Symp. Computer Architecture, (Houston), 1975, pp 126-132.

[DIJK 59] E.W. Dijkstra. "A note on two Problems in Connection with Graphs", Numerishe Mathematic, pp 269.

[FELL 68] W. Feller. An Introduction to Probability and Its Applications, Wiley, 1968.

[FRAT 73] L. Fratta and U.G. Montanari. "A Boolean Algebra Method for Computing Terminal Reliability in Communication Network", IEEE Tr. Circuit Theory, Vol. CT-20, May 1973, pp 203-211.

[GEIS 83] R. Geist, K.S. Trivedi, J.B. Dugan and M. Smotherman. "Design of the Hybrid Automated Reliability Predictor", Proc. IEEE/AIAA Digital Avionics Systems Conf., Oct. 1983, pp 16.5.1-16.5.8.

[HARA 72] F. Harary. Graph Theory, Addison Wesley Publishing Co. 1975.

[JENS 69] P.A. Jensen and M. Bellmore. "An Algorithm to Determine the Reliability of a Complex System", IEEE Tr. Reliability, Vol. R-18, Nov. 1968, pp 169-174.

[KAVI 79] K.M. Kavi. "Classification of Markov Chains Using Graph Algorithms" Tech. Rept. CSE 7912, Dept. of CSE, SMU, Dallas, Aug. 1979.

[KAVI 83] K.M. Kavi. "Data Flow Modeling Techniques", Proc. IASTED Intl. Conf. Modeling and Simulation, (Orlando, FL), Nov. 1983, pp 1-4.

[KAVI 84] K.M. Kavi, B.P. Buckles and U.N. Bhat. "A Formal Definition of Dataflow Graph Models", Submitted to for publication.

[KAVI 85] K.M. Kavi, B.P. Buckles and U.N. Bhat. "Isomorphisms Between Petri Nets and Dataflow Graphs", Submitted for publication.

[LIN 76] P.M. Lin, B.J. Leon and T.C. Huang. "A New Algorithm for Symbolic System Reliability Analysis", IEEE Tr. Reliability, Vol. R-25, Apr. 1976, pp 2-15.

[MATU 83] D.W. Matula. "Graph Theoretic Cluster Analysis", in Encyclopedia of Statistical Sciences Eds. N.L. Johnson and S. Koba, Wiley, pp 511-517.

[MILL 73] R.E. Miller. "A Comparison of Some Theoretic Models of Parallel Computation", IEEE Tr. Computers, Vol. C-22, Aug. 1973, pp 710-717.

[MISR 70a] K.B. Misra. "An Algorithm for the Reliability Evaluation of Redundant Networks", IEEE Tr. Reliability, Vol. R-19, Nov. 1970, pp 146-151.

[MISR 70b] K.B. Misra and T.S.M. Rao. "Reliability Analysis of Redundant Networking Using Flow Graphs", IEEE Tr. Reliability, Vol. R-19, Feb. 1970, pp 19-24.

[MOLL 81] M.K. Molloy. "On the Integrating Delay and Throughput Measures in Distributed Processing Models", PhD Dissertation, UCLA, 1981.

[MOLL 82] M.K. Molloy. "Performance Analysis Using Stochastic Petri Nets", IEEE Tr. Computers, Vol. C-51, Sept. 1982, pp 913-917.

[NELS 70] A.C. Nelson Jr., J.R. Batts and R.L.Beadles. "A Computer Program for Approximating System Reliability", IEEE Tr. Reliability, Vol. R-19, May 1970, pp 61-65.

[PERK 83] T.E. Perkins and K.M. Kavi. "A Hueristic Graph Algorithm for Optimizing Program Modularization", Proc. 5th Intl. Conf. Computer Capacity Management, New Orleans, LA, Apr. 1983.

[REIN 77] E.M. Reingold, J. Nievergelt and N. Deo. Combinatorial Algorithms: Theory and Practice, Prentice-Hall, 1977.

[SATY 78] A. Satyanarayana and A. Prabhakar. "New Topologial Formula and Rapid Algorithm for Reliability Analysis of Complex Networks", IEEE Tr. Reliability, Vol. R-27, June 1978, pp 82-100.

[SIFA 80] J. Sifakis. "Performance Evaluation of Systems Using Nets", in Net Theory and Applications, (W. Brower, ed.), Springer-Verlag, 1980, pp 307-319.

[TARJ 72] R.E. Tarjan. "Depth First Search and Linear Graph Algorithms", SIAM J. Computing, 1972, pp 146-160.

[TREL 82] P.C. Treleaven, D.R. Brownbridge and R.P. Hopkins. "Data Driven and Demand Driven Computer Architecture", ACM Computing Surveys, March 1982, pp 93-143.

[ZUBE 80] W.M. Zuberek. "Timed Petri-Nets and Preliminary Performance Evaluation", Proc. 7th Annual Symp. Computer Architecture, 1980, pp 88-96.

APPENDIX: REDUCTION OF DATAFLOW GRPAHS

In this appendix we outline how large dataflow graphs can be reduced to graphs of manageable size. The reduction involves,

- The identification of subgraphs
- The analysis of subgraphs using either Markov chain techniques, or using algorithm A1.
- The replacement of subgraphs by single actors.

The process of identifying subgraphs is similar to clustering of graphs. A cluster is a maximal collection of suitably similar objects drawn from a larger collection of objects [MATU 83]. If the dataflow graph contains strongly connected components, then cliques can be identified using depth-first search ([TARJ 72], [REIN 77]) or reachability sets [CHRI 75]. A survey of such algorithms can be found in [KAVI 79]. Clustering of non-clique subgraphs is usually based on hueristics. In [PERK 83], an algorithm is presented for clustering related nodes based on weight and cost. The weight of a cluster is defined as the ratio of the number of arcs between the nodes in the cluster to the total number of arcs connected to the nodes in the cluster. The cost of a cluster is given by the number of arcs connecting the nodes inside the cluster with the nodes outside the cluster. The algorithm finds maximal subgraphs by growing the clusters until a maximum weight and minimum cost cluster is found. For finding clusters (or subgraphs) in dataflow graphs, the following rules can be used.

1. Maximize the weight of clusters.
2. Minimize the cost of clusters.
3. If a conjunctive actor is included in the cluster, the actors feeding the input links of the conjunctive actor must be included in the cluster.
4. If a distributive actor is included, all the actors fed by the output links from the distributive actor must be included in the cluster.

For the purpose of obtaining a canonical representation, hierarchical relationship among the subgraphs (or actors) can be defined as follows.

1. All actors (or subgraphs) with an outdegree of zero are at level zero in the hierarchy (highest). That is,

   $$H(x_i) = 0 \quad \text{if} \quad \Gamma(x_i) = 0$$

   where $\Gamma(x_i)$ is the set of actors (or subgraphs) that are adjacent to x_i and $H(x_i)$ is the hierarchical level of actor x_i.
2. For an actor with an outdegree greater than zero, the level is given by

   $$H(x_i) = \left[\min_{x_j \in \Gamma(x_i)} \left(H(x_j) \right) \right] - 1$$

The second rule implies that actor x_i (or subgraph) is at a lower level than an actor x_j if an arc (x_i, x_j) exists. The level of an actor (or subgraph) is in fact the negative of the longest path to the actor from an actor at level 0. Dijkstra's ([DIJK 59], [CHRI

75]) shortest path algorithm can be modified to obtain longest paths by assigning a cost of -1 to arcs.

Example: The dataflow graph in Fig. 4 is used to illustrate the reduction approach. Fig. 6 shows three clusters that can be identified based on the hueristics described above. Each cluster can be analyzed using Markov chains. Fig. 7 lists the Markings (hence states) for the three clusters. The state transition matrices appear in Fig. 8. The reliability of each cluster can be obtained by using well established techniques [BHAT 84, Chap. 4] (Fig. 9). These three clusters can now be replaced by single actors as shown in Fig. 10.

Although the clusters identified in this example are simple (and the calculation of reliabilities by ennumerating paths is simpler than the Markov chain technique), we would like to note that in a more complex dataflow graph the graph reduction would lead to computational simplicity. A more complex example would be difficult to present here due to space limitation and the clarity desired.

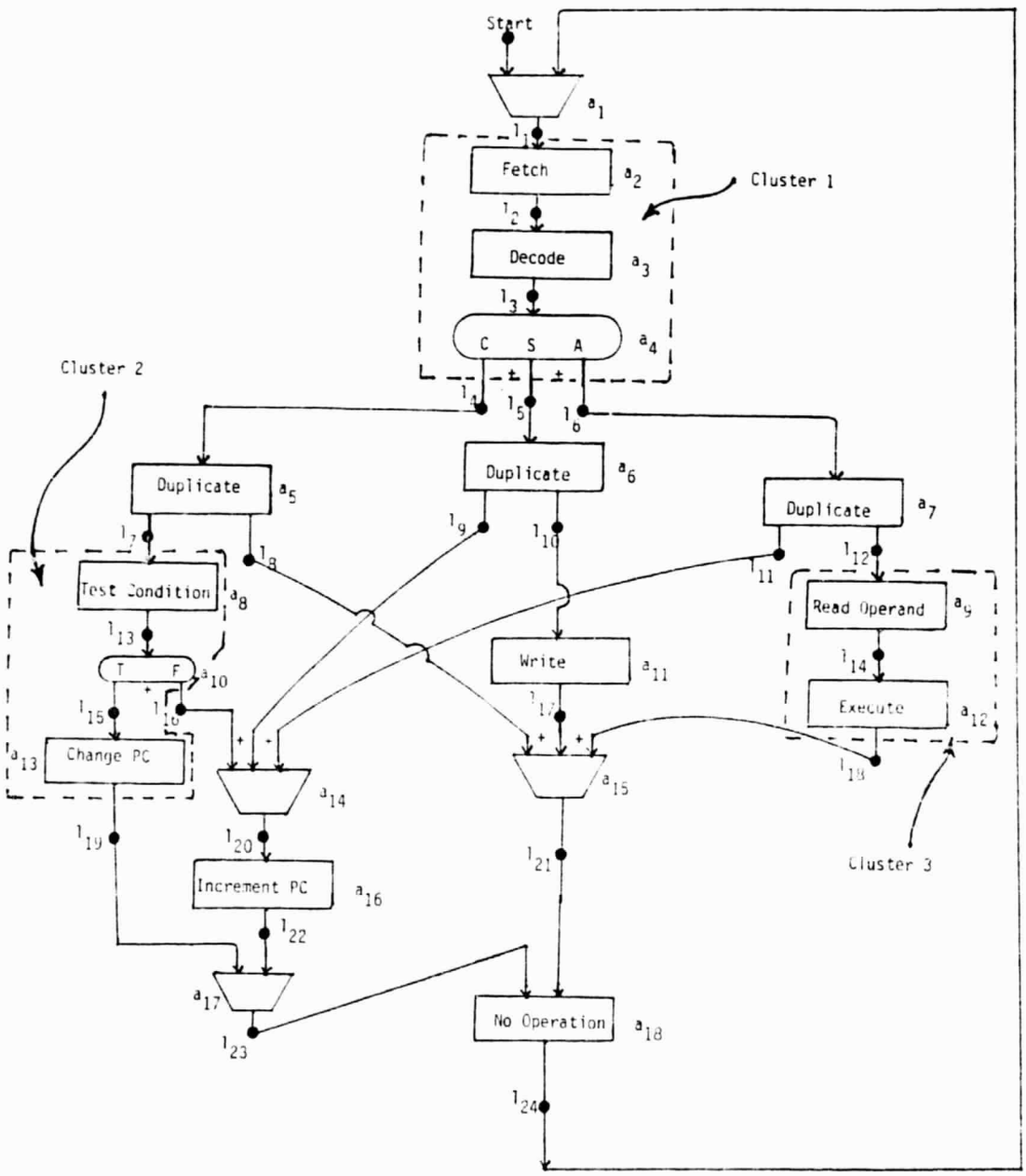


Fig. 6. Three Clusters Identified

Cluster 1:

	l_1	l_2	l_3	l_4	l_5	l_6
M_1:	1	0	0	0	0	0
M_2:	0	1	0	0	0	0
M_3:	0	0	1	0	0	0
M_4:	0	0	0	1	0	0
M_5:	0	0	0	0	1	0
M_6:	0	0	0	0	0	1

Cluster 2:

	l_7	l_{13}	l_{15}	l_{16}	l_{19}
M_1:	1	0	0	0	0
M_2:	0	1	0	0	0
M_3:	0	0	1	0	0
M_4:	0	0	0	1	0
M_5:	0	0	0	0	1

Cluster 3:

	l_{12}	l_{14}	l_{18}
M_1:	1	0	0
M_2:	0	1	0
M_3:	0	0	1

Fig. 7 Markings (States) for Three Clusters

Cluster 1:

	F	M_4	M_5	M_6	M_1	M_2	M_3
F	1	0	0	0	0	0	0
M_4	0	1	0	0	0	0	0
M_5	0	0	1	0	0	0	0
M_6	0	0	0	1	0	0	0
M_1	$(1-l_1a_2)$	0	0	0	0	(l_1a_2)	0
M_2	$(1-l_2a_3)$	0	0	0	0	0	(l_2a_3)
M_3	$(1-l_3a_4)$	$(p_1l_3a_4)$	$(p_2l_3a_4)$	$(p_3l_3a_4)$	0	0	0

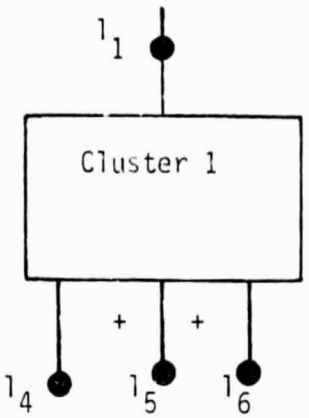
Cluster 2:

	F	M_4	M_5	M_1	M_2	M_3
F	1	0	0	0	0	0
M_4	0	1	0	0	0	0
M_5	0	0	1	0	0	0
M_1	$(1-l_7a_8)$	0	0	0	(l_7a_8)	0
M_2	$(1-l_{13}a_{10})$	$(p_5l_{13}a_{10})$	0	0	0	$(p_4l_{13}a_{10})$
M_3	$(1-l_{15}a_{13})$	0	$(l_{15}a_{13})$	0	0	0

Cluster 3:

	F	M_3	M_1	M_2
F	1	0	0	0
M_3	0	1	0	0
M_1	$(1-l_{12}a_9)$	0	0	$(l_{12}a_9)$
M_2	$(1-l_{14}a_{12})$	$(l_{14}a_{12})$	0	0

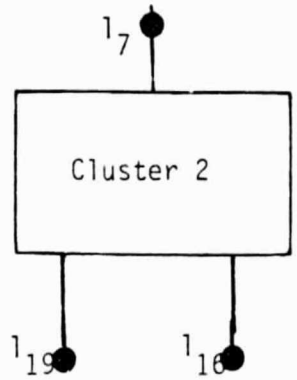
Fig. 8. State Transition Matrices for Three Clusters



$$R(l_1 \longrightarrow l_4) = p_1 \cdot l_1 \cdot a_2 \cdot l_2 \cdot a_3 \cdot l_3 \cdot a_4$$

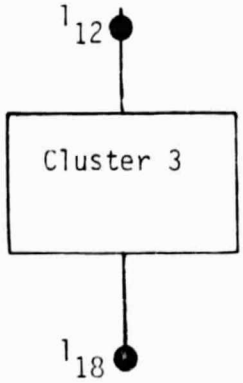
$$R(l_1 \longrightarrow l_5) = p_2 \cdot l_1 \cdot a_2 \cdot l_2 \cdot a_3 \cdot l_3 \cdot a_4$$

$$R(l_1 \longrightarrow l_6) = p_3 \cdot l_1 \cdot a_2 \cdot l_2 \cdot a_3 \cdot l_3 \cdot a_4$$



$$R(l_7 \longrightarrow l_{19}) = p_4 \cdot l_7 \cdot a_8 \cdot l_{13} \cdot a_{10} \cdot l_{15} \cdot a_{13}$$

$$R(l_7 \longrightarrow l_{16}) = p_5 \cdot l_7 \cdot a_8 \cdot l_{13} \cdot a_{10}$$



$$R(l_{12} \longrightarrow l_{18}) = l_{12} \cdot a_9 \cdot l_{14} \cdot a_{12}$$

Fig. 9. Reliabilities of Three Clusters

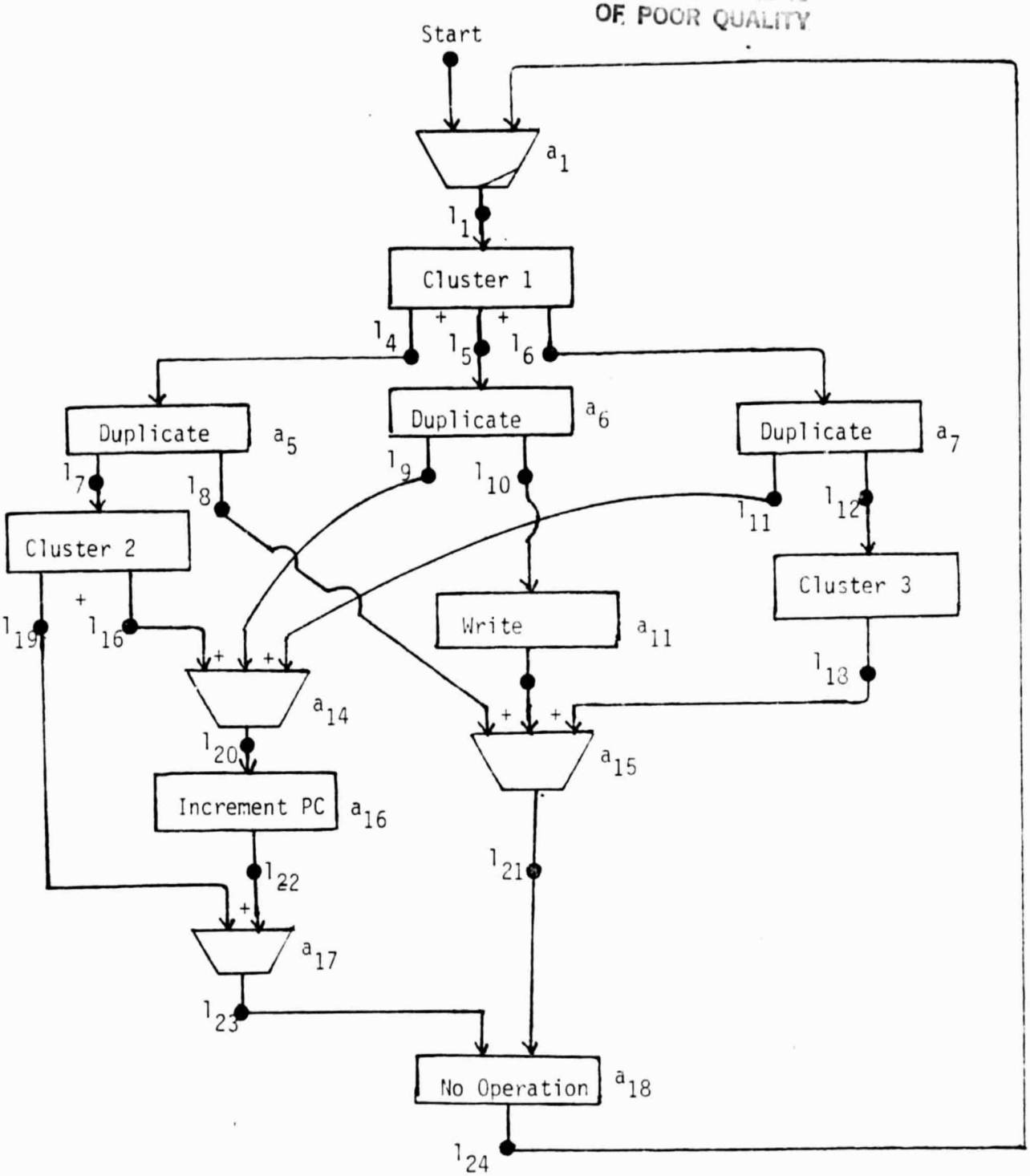


Fig. 10. An Example of Reduced Graph