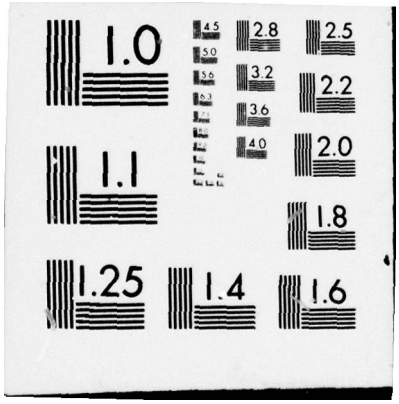
AD-A034 385 HONEYWELL INC MINNEAPOLIS MINN SYSTEMS AND RESEARCH --ETC F/G 9/2
THEORY OF FAULT TOLERANCE. 1976 ANNUAL REPORT. VOLUME 1,(U)
DEC 76 L A JACK, W L HEIMERDINGER, Y W HAN N00014-75-C-0011
UNCLASSIFIED 76SRC/52 NL

I OF I
AD-A
034 385

END
DATE
FILMED
2-15-77
NTIS


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6

THEORY OF FAULT TOLERANCE. 1976 ANNUAL REPORT

VOLUME 1

HONEYWELL, INCORPORATED, MINNEAPOLIS, MINNESOTA

7 DECEMBER 1976

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (WHEN DATA ENTERED)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOV'T ACCESSION NUMBER	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (AND SUBTITLE) THEORY OF FAULT TOLERANCE 1976 ANNUAL REPORT (VOLUME I)		5. TYPE OF REPORT/PERIOD COVERED Annual Report/Aug. to Nov. 6. PERFORMING ORG. REPORT NUMBER 76SRC52
7. AUTHOR(S) L. A. Jack Y. W. Han W. L. Heimerdinger L. L. Kinney		8. CONTRACT OR GRANT NUMBER(S) N00014-75-C-0011
9. PERFORMING ORGANIZATIONS NAME/ADDRESS Honeywell Systems & Research Center 2600 Ridgway Parkway Minneapolis, Minnesota 55413		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME/ADDRESS Office of Naval Research 800 N. Quincy Street Arlington, Virginia 22217		12. REPORT DATE 7 December 1976 13. NUMBER OF PAGES 57
14. MONITORING AGENCY NAME/ADDRESS (IF DIFFERENT FROM CONT. OFF.)		15. SECURITY CLASSIFICATION (OF THIS REPORT) Unclassified 15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (OF THIS REPORT)

Unlimited.

17. DISTRIBUTION STATEMENT (OF THE ABSTRACT ENTERED IN BLOCK 20, IF DIFFERENT FROM REPORT)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (CONTINUE ON REVERSE SIDE IF NECESSARY AND IDENTIFY BY BLOCK NUMBER)

Fault Tolerant Computing
Graphical Modeling
Petri Nets
LOGOS Models
Reduction of Petri Nets
Data Flow Procedure Models
Fault Models

20. ABSTRACT (CONTINUE ON REVERSE SIDE IF NECESSARY AND IDENTIFY BY BLOCK NUMBER)

The 1976 effort focused on exploring Labeled Graph properties which are prerequisites to successfully modeling important fault-tolerant phenomena. The modeling hierarchy task successfully demonstrated that Petri net structures could be reduced while preserving certain properties such as invariance and consistency. The second task which formalized the interface between the data transformation structure and the control structure indicated that a two-graph modeling approach was preferred over a combined

HD-168 REV 11/74

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 55 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (WHEN DATA ENTERED)

single-graph approach. The data-fault task took a preliminary look at classifying various data faults in terms of their functional effects. The results suggest that there are a limited set of data-fault classes which accurately represent the types of data faults experienced in field equipment. Further work is required to complete the study of data faults.

A separate task investigated the feasibility of developing a design for testability methodology which was a Navy-wide need identified at the Navy ATE workshop held at NELC. A framework for such a methodology was proposed, the current state of the art in design for testability was assessed and directions for Research and Development work were recommended.

Honeywell

76SRC52 December 7, 1976

THEORY OF FAULT TOLERANCE

1976 ANNUAL REPORT

VOLUME I

Prepared for

OFFICE OF NAVAL RESEARCH

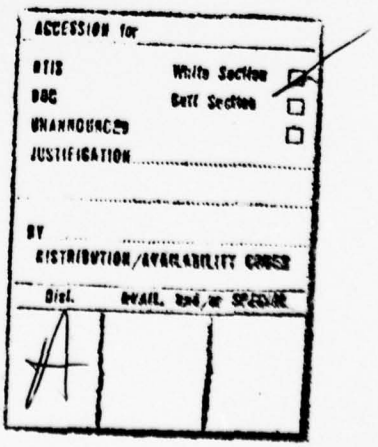
by

L. A. Jack
W. L. Heimerdinger
Y. W. Han
L. L. Kinney

Systems & Research Center
2700 RIDGWAY PARKWAY
MINNEAPOLIS, MINNESOTA 55413

Printed in U.S.A.

CONTENTS



LIST OF FIGURES

INTRODUCTION

This report is a continuation of the ongoing research project aimed at developing a unified theory of fault tolerance for digital systems. From the modeling experience gained in the previous effort, several limitations of the existing modeling approaches were identified. This year's investigation of several prerequisite properties needed for the development of a Labeled Graph model that can represent fault-tolerant phenomena emphasized the following three areas:

- The data-to-control interface
- Hierarchy in a Petri net-representation of control
- The concept of functional data faults

Task 1 studied the relationship between the data transformation aspects and the control aspects of a digital processing system. Task 2 explored the concept of hierarchy as applied to control structure modeling and it also developed an approach for reduction of nets. Finally, Task 3 was a preliminary investigation into the classification and representation of data faults. Each of these efforts are summarized in the remainder of this volume.

In response to the finding at the April Navy ATE workshop held in San Diego, a separate task was initiated to address the design for testability issues. A review of the current design for testability "process" was the focus of this effort. The goal was to assess the feasibility of developing

a design for test methodology and identify the required Research and Development tasks. The results of this effort are presented in Volume II of this report.

TASK I

CONTROL TO DATA INTERFACE

In order to study the critical phenomena in fault tolerant systems, a model must represent both the control and the data aspects. A major portion of any computing system is composed of

- Data
- Its storage
- Data transformation functions

These also contribute to the failure modes of a system. We propose to view a system as composed of a data transformation structure with a superimposed control structure. The data transformation structure is defined to include all the data processing functions, while the control structure includes all the sequencing and state information.

The objective of this task is to explore and refine the interface between the data transformation aspects and the control aspects of a digital system. This task builds on the previous control modeling and data modeling results from the companion AFOSR program. Since a formal data representation was not available from this concurrent effort, we were forced (in this study) to use an informal approach based on the LOGOS and data flow procedure models.

It is assumed that the reader is familiar with the basic notation from both models or can review references 6 and 7. The critical issues which were addressed in this task include:

- Can a clear distinction be made between control and data?
- What are the fundamental relationships between control and data operators?
- Is a single-graph or two-graph model approach better?

Our experience with control structure modeling showed that extensions were needed to adequately deal with the data related issues. The questions which must be addressed are whether to augment the current control graph models or to create a separate data graph.

In order to make such a decision, this task studied the relationship of the data and control structures and experimented with the existing approaches to data graph modeling LOGOS developed by Rose (reference 7) and data flow procedure language developed by Dennis at MIT (reference 4). Although there has been a large amount of work dealing with control modeling and control phenomena, there has been little published work on data process modeling and data phenomena. In general, researchers have chosen to consider the control system rather than concentrating on data issues.

We chose to build on the Wilks' notions described in reference 26. He proposed that a higher order programming language (e.g. ALGOL) could be partitioned into two parts. The "outer syntax" which is concerned

with the organization of the flow of control and the "inner syntax" which is concerned with the operations performed on the data itself. He concluded that it was possible to make a clear separation of inner and outer syntax for a programming language.

From these results we can conjecture that any system can be partitioned into a control structure and the data transformation structure.

RELATIONSHIP BETWEEN THE TWO STRUCTURES

At a given level in the hierarchy of detail we can define a digital computing system to be composed of a set of data inputs, a set of data outputs and a transfer function to map the inputs to the outputs (Figure 1a). Looking at the next lower level of detail, this data processing structure is composed of a set of data operators, intermediate data entities and a control structure which sequences the execution of these more primitive operations. This expansion, as shown in Figure 1b, can be defined recursively on each data transformation operator.

The right portion of Figure 1b represents the control structure which was discussed previously. We now define the control structure to encompass all the information which defines the system state and contributes to the state transition that produces the flow of control.

The left portion of Figure 1b represents the data transformation structure. This structure is defined to encompass the data packets, data structures, and data operators which are utilized in producing the data output from given inputs. Data packets are defined as a quantum of information which

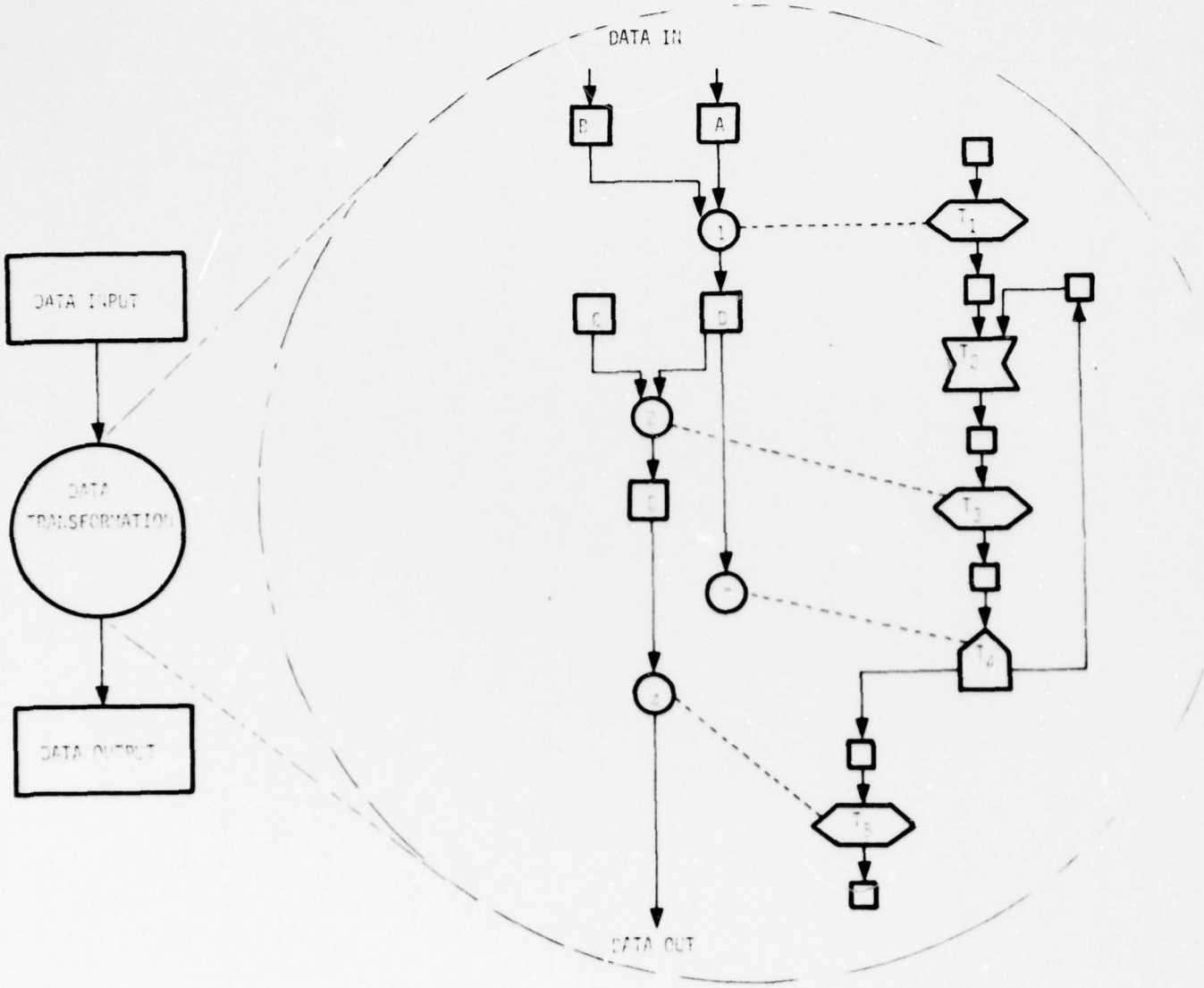


Figure 1. Detailed Expansion of System Model

is stored and can be sampled (i.e., read or tested). A model of a data transformation must be capable of representing the data packets and of organizing the data packets into data structures. It must also be capable of relating the data transformations with the data structures or packets. Figure 2 contains another LOGOS model of both the control and data transformation structure of a given process. The left half of the figure describes data packets A, B, C and D, the data operator MUL, the add operator and the test operator. The relationship between each packet and operator is also shown; for example, data packet D is shown to be both an input to the add operator as well as an output from the add operator. At the same time it is also an input to the test operator which in turn is linked to the control graph.

The operator pairing approach is used by the LOGOS model developed by Rose et al. at Case Western Reserve University. In the LOGOS model, each data operator is paired with a single control operator. When the control operator begins firing, the data operator becomes active and the specified data transformation takes place. When the control operator completes firing and emits a token, the associated data operation is assumed to be complete. Like the DFP model, LOGOS uses a special operator (the predicate operator) to allow data values to steer a control token stream.

Thus, the complete LOGOS model retains separate data and control graphs, linked by operator pairing and predicate operators.

The interface between the two structures is illustrated by the dotted directed arcs. The relationship betwen the operators consists of three types of links:

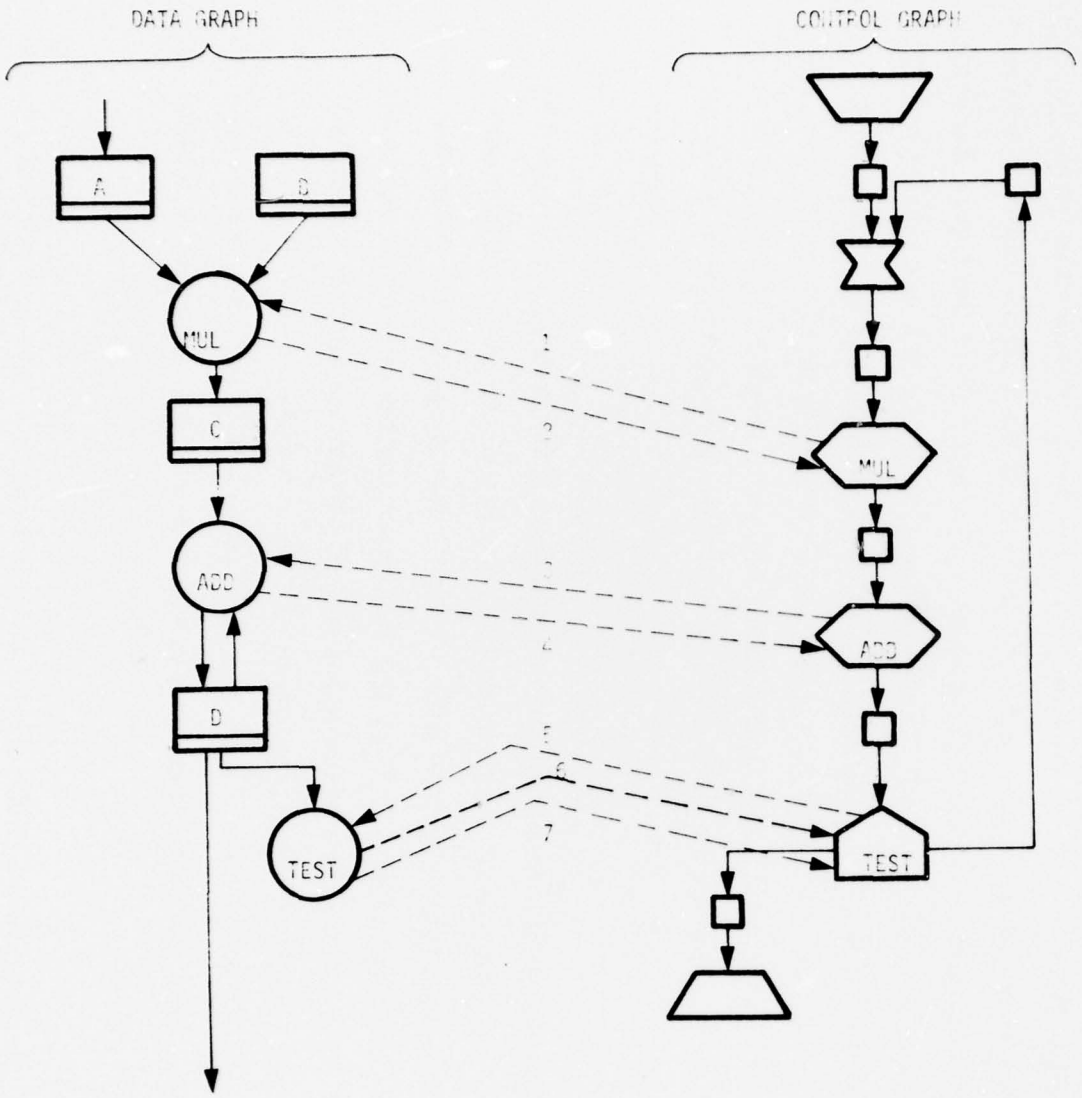


Figure 2. Example Data Transformation Structure and Corresponding Control Structure Pair

- Initiation links (e.g. arc 1 and 3) which enable the data operator when sequenced by the control structure.
- Termination links (e.g. arc 2 and 4) which enable the control operator to complete the firing action given that the data transformation is complete. Note that at the lowest level of detail in synchronous machines the termination link is from the system clock.
- Steering link (e.g. arc 6) which allows data to direct the flow of control at a decision node (e.g. predicate node FES). The steering link is a special addition to the normal initiation/termination links which are associated with each data/control operator pair.

These three links define the narrow interface between the control and data structures. They produce a tightly coupled system which can be separated if timing of critical aspects are considered in the control graph and data flow aspect are considered in the data graph. From preliminary investigations it appears that separate analysis of the data and the control is possible and will significantly reduce the complexity of the analysis routines.

The conversion operator approach is used in the data flow procedure language (DFP) model. The DFP model uses decider operators to convert data value inputs into control token outputs and gate and merge operators to combine control tokens into data token streams. The result, to be discussed next, is a single graph that interweaves what would otherwise be disjoint control and data graphs.

EXAMPLE MODELS

Neither DFP nor LOGOS was intended as a fault-tolerant system model, and each have advantages and shortcomings when applied to fault-tolerant systems. To illustrate the different properties of the two approaches and to effectively study faults in digital systems, we must represent:

- Groupings of data items into data structures
- Relationships between data transformations and data elements so that the influence of erroneous data values is clear
- Data transformations and data elements influenced by each control operator or control path
- Points at which data values can cause diversion of the control stream

The DFP and LOGOS models are probably the most developed graphical models that incorporate both data and control operators, thus an example function is modeled using both models for comparison. Primarily concerned with the structural and philosophical aspects of these two labeled graphs, the conventions used by the original authors were altered where convenient.

The following software algorithm was created to contrast DFP and LOGOS.

```
PROCEDURE     EXAMPLE (M, N, K, W)  ;
INTEGER:  M, N, W  ;
INTEGER CONSTANT:          K  ;
INTEGER ARRAY:        A[0:23], B[0:23]  ;
BEGIN
          WHILE       N ≠ K DO
               BEGIN
                    M ← B(M);
                    N ← A(M);
               END;
          W ← A(M);
END;
```

Figure 3 shows a DFP representation of the above example. Note that ARRAY and CONSTANT work as a memory cell. Although we can envision a token being moved out as the related operators activate, there is another token with the same value to replenish it (just as values are rewritten in core memory cells after a destructive read). It is important to note that a token on a data link has a value (or values for structured data) associated with it, a token on a control link has a Boolean value (true or false) associated with it. This is different from a token in the control graph of a LOGOS representation or in a Petri net, because in both of them only the existence of a token is of concern. Sequence links, however, in which only the existence of a token (or tokens) is of concern can be introduced.

The same example expressed in LOGOS is shown in Figure 4a and 4b. It consists of two parts: the data graph containing the information on data access and data transformation, and the control graph depicting the control flow of the modeled system.

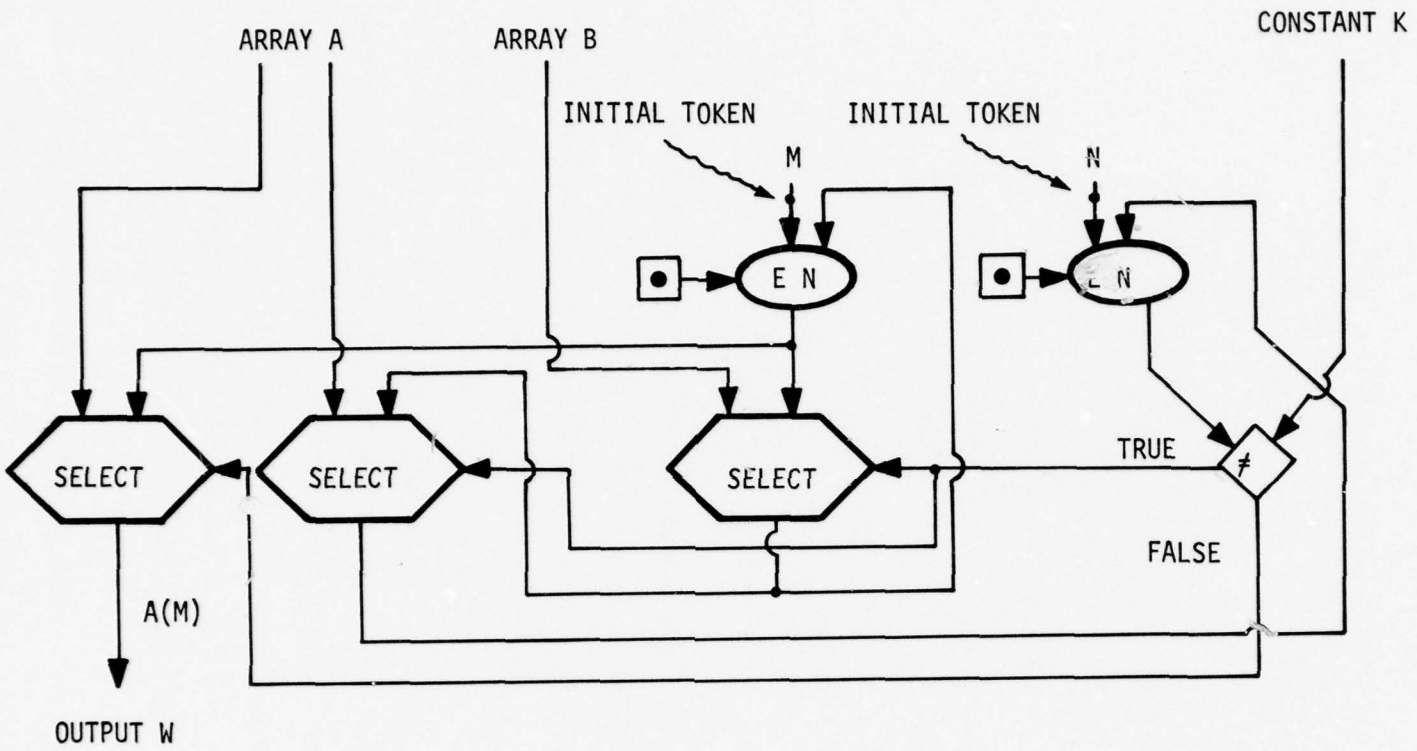
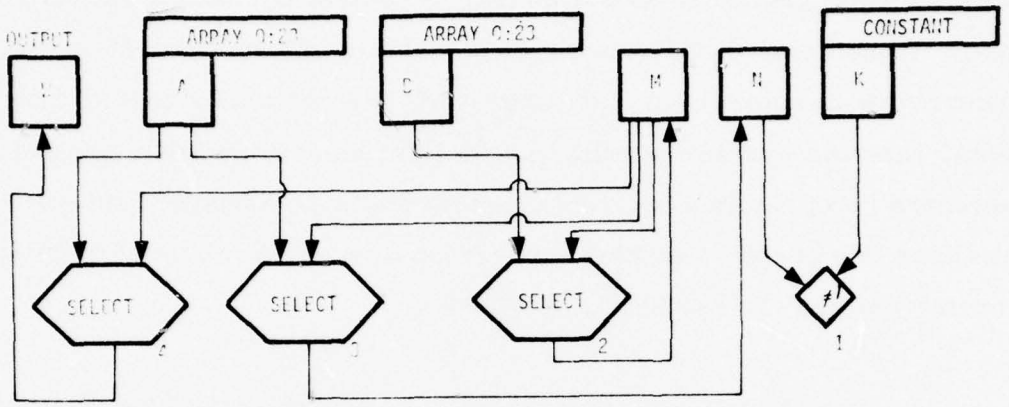
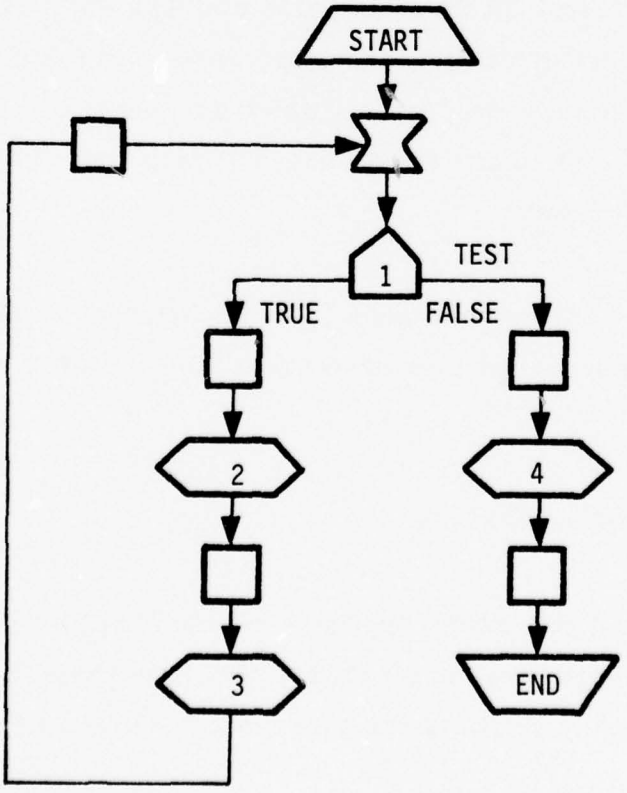


Figure 3. DFP Representation of Example



a. DATA GRAPH



b. CONTROL GRAPH

Figure 4. LOGOS Representation of the Example

Most graphical models of digital systems that represent both control and data have two classes of operator nodes, control operators and data operators. When control operators are connected only to elements exclusively concerned with control and data operators are connected only to data elements, then the system is split into disjoint control and data graphs. There are two principal ways of constructing a data/control interface to link these two graphs, special conversion operators (as used in DFP) and operator pairing (as used in LOGOS).

There are several differences between the data graph and the control graph representations. First, the arcs in the control graph represent a preceding relationship between the operators. In the data graph the arcs represent read or write access relationships between the data packets and the data operators. The data operators themselves do not have any firing constraints like the corresponding control operators. In the data flow procedure language there are data tokens which represent the instance of data. In LOGOS, however, there are no tokens which flow in the data graph as in the control graph.

With the LOGOS model, the properties of the two inherent structures are clear--a simple looping control structure and a simple linear data transformation structure.

COMPARISON OF LOGOS AND DFP

Even though LOGOS is a two-graph approach and DFP is a one-graph approach, and even though there are data tokens which have values in DFP, there are still many similarities between LOGOS and DFP.

The properties of interest compared are:

1. Control Signal Selection of Data Values--The mechanism in the graph by which control signals determine which data item is used from a data structure.

2. Data Steering of Control Sequences--The mechanism by which data values influence the flow of control in the system.

3. Deviation from Real Systems--The degree to which the structure and sequencing of the labeled graph model mirrors the structure and behavior of the actual hardware or software system being represented.

4. Exposure of Parallelism--The extent to which any parallelism or concurrency in the architecture being modeled is reflected in the structure and behavior of the model.

5. Representation of Data Instances--Some models directly represent data storage elements which can be referenced by "read" operations and can receive data with "write" operations. Other models create a new "read-only" data element following each data operation.

6. Determinancy--Determinancy is a labeled-graph property relating to the certainty in the control sequence of a graph. Given an initial marking, a graph is determinant if its firing sequence of the transition is unchangeable and predictable.

7. Safeness--Safeness is a labeled-graph property relating to the number of tokens in a place. A labeled graph is called safe if there can be at most a single token in each of its places at any time.

8. Representation of Synchronous Systems--Synchronous or asynchronous systems differ mostly in the way activation condition is specified; i.e., whether it is a signal coming from a central clock or a signal produced at the completion of the preceding or a combination of both. Labeled graphs are usually used to represent asynchronous systems or events, while it is also possible to represent synchronous systems or events by labeled graphs.

Properties of Data Flow Programs

Some observations of the properties of data flow programs are listed below.

1. Control Signal Selection of Data Values--T-gate, F-gate and merge actors permit the outcomes of deciders to decide the flow of values. If an operator has an input control arc, this operator cannot be enabled until the arc contains a control token.

2. Data Steering of Control Sequences--Decider elements accept data tokens as inputs and use these to output control tokens to initiate control sequences.

3. Deviation from Real Systems--An actor cannot fire if any one of its input arcs lacks a token. In realistic hardware systems every memory cell contains information regardless of whether the information is desired or leftover from previous calculations. On the other hand, this model seems able to reasonably show software programs by envisioning every token placement as a snapshot of the program conditions.

4. Exposure of Parallelism--The computations illustrated by the data flow programs deviate from the original algorithms. Hidden parallelism of the original algorithms are uncovered during the process of constructing data flow programs. As a consequence, this straightforward construction of a data flow program from a system description or an algorithm is not feasible.

5. Representation of Data Instances--Since the firing of an actor or a link removes all the tokens on the input arcs, it can be envisioned as destructive read. Tokens to be used more than once are replicated by multiple links. Data structures are not explicitly modeled.

6. Determinancy--Also, because of the firing rules, data flow programs are deterministic.

7. Safeness--Safeness is enforced by the DFP firing rules.

8. Representation of Synchronous Systems--Synchronous systems or events can be represented by adding a control token generator which has an output arc connecting to every operator. This control token generator generates control tokens as the clock of the simulated systems. It functions as a clock.

Properties of LOGOS

In LOGOS, control and data are separated and are represented by a control graph and a data graph, respectively. Although it may be redundant, sometimes the separation does increase clarity. The properties of LOGOS are listed below:

1. Control Signal Selection of Data Values--Every d-operator (data operator) in a data graph is associated with at least one atomic operator in the corresponding control graph. When the atomic actor is activated, all of its associated data operators start to perform their data access and data transformation. When all the data operators complete their operations, the operation of the atomic operator is then completed. It is important to note that in the original LOGOS every d-operator was uniquely associated with one atomic operator. This one-to-many mapping change was introduced to reduce model complexity.

2. Data Steering of Control Sequences--PRED (predicate) operators in LOGOS are the counterpart of deciders in data flow programs. They are data dependent control branches whose data operator performs a test on its input d-cells (data cells). The outcome of the test decides which control branch will be taken.

3. Deviation from Real System--The deviation problem of data flow programs does not exist in LOGOS.

4. Exposure of Parallelism--Hidden parallelism need not be found to construct a LOGOS model. Note that it is possible to identify hidden parallelism after constructing a LOGOS model, if desired.

5. Representation of Data Instances--Data is modeled in terms of static storage structures. There are no data tokens to represent the creation and movement of data.

6. Determinancy--It is not necessary that a control graph be deterministic.

7. Safeness--Safeness is not necessarily observed in control graphs. In data graphs, operands are stored in data cells and it is non-destructive read. Thus token content does not play a role in data graphs.

8. Representation of Synchronous Systems--Synchronous systems or events can be represented by adding a token generator which has an output arc connecting to every operator of the control graph only. This token generator functions as a clock.

CONCLUSIONS

This task investigated the relationship between the data transformation structure and the control structure which together can completely represent the functions of a digital system. The interface was found to be narrow, consisting of three link functions:

- Initiation
- Termination
- Steering

These functions are well defined and allow separation of data and control into two structures which are not highly dependent from a representation/analysis viewpoint. It appears that meaningful analysis can be performed on each structure independently for data and then control related properties. This is desirable since it significantly reduces the complexity of the graphs and the analysis procedure.

As a result of sample modeling experiments, it can be concluded that a two-graph approach is recommended because:

- Clearer illustration of the inherent structure
- Less complex analysis procedures allowing independent analysis of data and control properties
- Simplified graph notation
- Graphically less complex, more understandable and readable models

These concepts will be formalized when a formal data representation notation has been defined.

TASK II

HIERARCHY IN PETRI NET MODELS OF CONTROL STRUCTURES

INTRODUCTION

A detailed Petri net representation of an actual system usually contains an enormous number of places, tokens and edges. This makes the application of Petri nets time consuming and economically impractical. One approach to overcoming this problem is to hierarchically represent the system at various levels of detail and suppress unnecessary details at higher levels of abstraction. This concept is based on a set of net transformations where a small net can be expanded into a larger net at a lower level of detail, and vice-versa. In addition to a set of transformation rules, a systematic verification method is needed to determine if the parent and transformed net exhibit the same properties.

This task addressed the reduction of a Petri net to a smaller Petri net, such that the smaller net preserves some properties of the original net. In order to reduce them systematically, we need to identify some desirable properties of the nets and to determine mechanically whether a net processes these properties. The desirable properties, consistency, invariance and their relations with liveness, boundedness and transition firing ratios, are summarized. The concept of a threshold number of a net is introduced. If the threshold number of a net can be determined, reduction becomes straightforward.

REDUCTION OF PETRI NETS

Consider a Petri net containing a subnet which interfaces with the rest of the net only through input arcs to transition t_i, the input transition of the subnet, and output arcs from the transition t_o, the output transition of the subnet. In general, we want to determine what modifications can be made to the subnet and still obtain an "equivalent" Petri net. First, we need to define what is meant by equivalence.

Suppose that our primary interest is in the events of the system modeled by the Petri net. The events of the system are represented by the firing of transitions. Hence, for each possible sequence of events in the system, there is a corresponding sequence of transition firings which can occur in the net. Conversely, for each possible sequence of transition firings there will be a sequence of events in the system. (It is assumed that the initial marking for the net corresponds to the initial conditions of the system.)

If we were to insist on having complete information about all events in the system, then it would be necessary to have complete information about all the transitions in the net. However, if only certain events are of interest, then it may be that part of the transitions in the net are superfluous. For example, if in the subnet only the events associated with transitions t_i and t_o are of interest, then it may be possible to reduce the subnet and eliminate some of the transitions within the subnet. To reduce the net, we should be able to handle the following issues:

- What properties must be preserved when making the reduction?
- How can we determine whether a Petri net has these properties?

For purposes of discussing these reductions, we will use the following definition of a subnet.

Definition:

Let $N = \{P, T, A\}$ be a Petri net where P is the set of places, T is the set of transitions and A is a relation, $A \subseteq [PxT] \cup [TxP]$. A transition-to-transition subnet of N is the five-tuple $SN = \{SP, ST, SA, t_i, t_o\}$ where $SP \subseteq P$, $ST \subseteq T$, $SA \subseteq A$ and

- If t_j is in ST, $t_j \neq t_i$, then (t_j, p_k) in A implies that (t_j, p_k) is in SA,
- If t_j is in ST, $t_j \neq t_o$, then (p_k, t_j) in A implies that (p_k, t_j) is in SA,
- There is at least one p_k, such that (p_k, t_i) is in A but not in SA, and
- There is at least one p_k such that (t_o, p_k) is in A but not in SA.

This definition simply says that a transition-to-transition subnet of Petri net consists of a subnet of the net's places and transitions and all of the connections between these places and transitions except for some input edges to t_i and some output edges from t_o. We refer to t_i as the input transition of the subnet and t_o as the output transition. In an analogous fashion we can define transition-to-place, place-to-transition, and place-to-place subnets. Unless otherwise indicated, subnet will mean transition-to-transition subnet.

The reduction problem can now be stated as follows. Let $N = \{P, T, A\}$ be Petri net and $SN = \{SP, ST, SA, t_i, t_o\}$ be a subnet of N. The subnet SN is reduced to a new subnet $SN^1 = \{SP^1, ST^1, SA^1, t_i, t_o\}$ where $SP^1 \subseteq SP$, $ST^1 \subseteq ST$, $SA^1 \subseteq SA$, t_i is in ST^1 and t_o is in ST^1. The rest of N combined with SN^1 produces a reduced net, N^1. Then, N with initial marking M is equivalent with respect to transitions t_i and t_o to N^1 if there is an initial marking M^1 for N^1 such that the set of all possible transition firings of t_i and t_o are the same for N and N^1.

For some simple nets, it is possible to reduce a subnet by inspection. For example, in the net of Figure 5a, let t_i, t_1 and t_o be transitions in the subnet. Clearly, the only possible sequence of firings of t_i and t_o is $t_i, t_o, t_i, t_o, \ldots$ It is also obvious that this is the only possible sequence of firings of t_i and t_o for the net of Figure 5b, hence these two nets are equivalent with respect to t_i and t_o.

As another example of a reduction, consider Figures 5c and 5d. These nets are equivalent and the possible firing sequences are those beginning with t_i in which t_i has fired 0, 1, or 2 more times than t_o. These examples illustrate an important point concerning reduction of subnets. Note that the subnets of Figures 5a and 5c are identical; however, the reduced subnet of Figure 5b is not a valid reduction for the subnet of Figure 5c. The difference between the two cases is determined by the portion of the net external to the subnet--in this case, a single place with tokens--restricting the number of times that the input transition t_i can fire before the output transition t_o must fire. For Figure 5a, the number is 1 while for Figure 5b it is 3. Clearly, the reduced subnet of Figure 5d is valid no matter how many times t_i can fire. Furthermore, the reduction of this subnet can be made independent of the portion of net external to subnet.

There are other examples of simple series and parallel connections of transitions where the reduction can be made by inspection. In some the output transition can fire before the input transition fires. This is analogous to the transient response of systems. In some cases, a net can exhibit a very "nonlinear" transient behavior. It is easy to construct an example where the output transition t_o can continue to fire indefinitely until the input transition t_i is fired. However, once the input transition fires, the output transition can fire one more time and the net is dead thereafter.

In the remainder of this discussion, subnets like the preceding one will not be considered. Only "well-behaved" subnets will be considered. We define a net to be "well-behaved" if it possesses consistency and invariance. These properties are defined below. We will also be confined to the reduction of transition-to-transition subnets. It is, of course, possible to consider the reduction of other types of subnets.

Suppose that to every transition in a net, a variable is assigned. Let the variables be constrained by requiring that, for every place, the sum of the variables assigned to input places be equal to the sum of the variables assigned to output places. If there exists a positive integer (nonzero) solution for the variable, then the net is called consistent. If a net is consistent, then a consistent solution specifies the number of transition firings which occur for some cyclic firing sequence and conversely. (A cyclic firing sequence is a sequence of transition firings which fire each transition and has the same final marking as initial marking.)

The property of invariancy is defined in a dual manner. Variables are assigned to places and are constrained by requiring that, for every transition, the sum of the variables be assigned to output places. If there exists a positive, integer (nonzero) solution for the variables, the net is called invariant. Invariancy is a sufficient, but not necessary, condition for a net to possess a bounded marking. There are examples of nets which are both consistent and invariant and other examples which are neither. Also, there are examples which are one and not the other.

THE THRESHOLD NUMBER OF A PETRI NET

In most Petri nets which represent real systems, the maximum number of firings of a specified input transition without firing a specified output transition is fixed. This number is defined as the threshold number with respect to the input transition and the output transition. It is controlled by the number of tokens in the paths leading from the output transition to the input transition. These paths are called feedback paths in contrast to the paths leading from the input transition to the output transition which are called forward paths. Let us also define a simple path as a path which does not cover any transition or any place more than once. Then, rigorously, a feedback path is a simple path from t_o to t_i; and a forward path is a simple path from t_i to t_o.

In the discussion of threshold number, it is assumed there exist feedback paths to control the token population in nets concerned, since otherwise there exists no threshold number; i.e., it is undefined or infinite. The scope of discussion is on reducible nets only, which are defined as

follows. If a Petri net between an input transition and an output transition can be reduced to a net as shown in Figure 6, formed by a single loop consisting of a forward path and a feedback path, and given an initial marking, the token number in the feedback path is a fixed positive integer, then this net is called completely reducible or reducible for short.

Note that if a net is reducible then the firing ratio between the input and output transition is one-to-one. This assumes that a cyclic firing sequence is being considered; i.e., a firing sequence which returns the net to the same initial conditions. In addition, the maximum number of times that the input transition can fire before the output transition must fire is m, where m is the threshold number. This specifies the "transient behavior" of the net. As discussed previously, only live, consistent, invariant Petri nets are considered. Hence, a completely reducible net

- Is live
- Is consistent
- Is invariant
- Has a 1 to 1 firing ratio between input and output transitions (for cyclic firing sequences)
- Has a positive, constant threshold number

If a net is given, it is easy to check for consistency and invariancy, and thereby check for a 1 to 1 firing ratio between input and output transitions. The reduction of the net is straightforward if a simple procedure is available to determine the value of its positive constant threshold

number. However, the determination of the threshold number appears to be difficult for general reducible nets. The approach investigated to date is to examine the type of coupling between forward and feedback paths and between feedback paths.

The threshold number is related to the minimum number of tokens which can exist on feedback paths. When the input transition fires, a token is removed from each feedback path and a token is added to each forward path. When the output transition fires, the reverse effect takes place. Clearly, the maximum number of times which the input transition can fire without firing the output transition is determined by the minimum number of tokens which can exist on a feedback path. (This is true provided there is no self-loop between the input transition and a place on the feedback path.) However, this number is difficult to determine because the firing of internal transitions in the subnet can increase or decrease the number of tokens on a feedback path. The coupling between forward paths and feedback paths and between feedback paths are discussed below.

Consider the simple subnet of Figure 7. The feedback path and the forward path are coupled at a place. The firing of the internal transition can remove a token from the feedback path which is later returned by the firing of the output transition. The net has an indeterminate threshold number. The input transition can fire an arbitrarily large number of times without firing the output transition and, conversely, the output transition can fire an arbitrarily large number of times without firing the input transition.

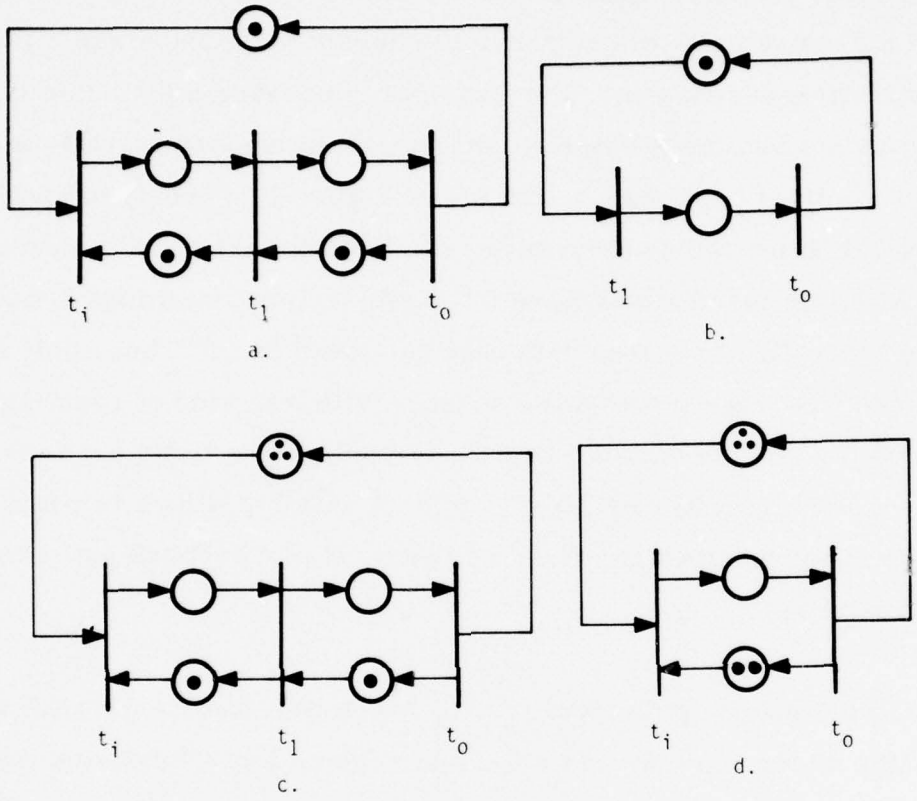


Figure 5. Reductions Which Depend Upon the External Net

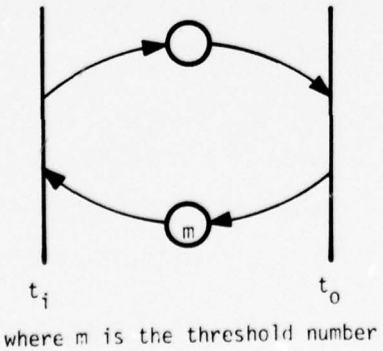


Figure 6. Reducible Net

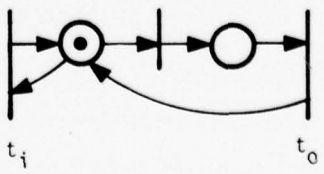


Figure 7. A Net Which Has Undefined Threshold Number

The preceding example shows that the coupling of feedback and forward paths at places can cause the threshold number to be undefined. However, this is not necessarily true. For example, in Figure 8 the same type of coupling exists but, nevertheless, the threshold number is well defined (it is 1). In contrast to Figure 7, the net of Figure 8 is consistent with a unique solution for the assignment of values to the input and output transitions. Although place p in Figure 8 has two output transitions t_i and t_1, they are controlled by a loop with only one token in t_i. Thus, only one of t_i or t_1 can fire when p contains a token. With this kind of coupling, the procedure for determining the threshold number of a Petri net is very involved. For simplicity, we shall only deal with Petri nets in which the type of coupling between any forward path and any feedback path occurs only at transitions.

If there is a feedback path from t_o to t_i which does not couple with any other paths at places, then the only way to place more tokens on that path is to fire the output transition t_o. Hence, if m is the number of tokens on the feedback path, the input transition t_i cannot fire more than m times without firing t_o. If there are two or more feedback paths with coupling at places, then the situation is more complicated. For example, if two feedback paths are coupled from a place p_2 to a place p_1, then all of the tokens can be moved to place p_1. Thus, t_i cannot fire more times than the sum of the tokens on the two paths without firing t_o.

If two feedback paths are coupled from a place to a transition, then the net may not be persistent and the number of times which t_i can fire is not defined; i.e., it is non-deterministic. However, this is not necessarily true for every case of this type of coupling.

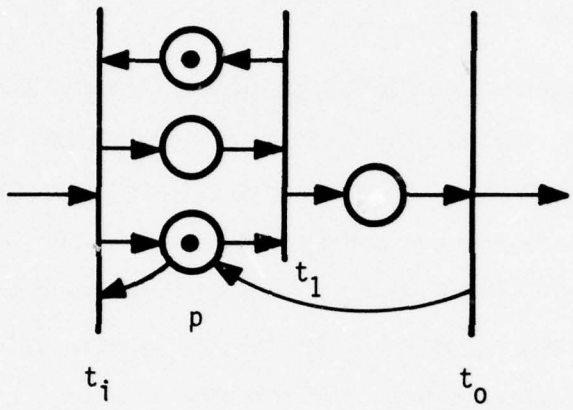


Figure 8. A Net With a Defined Threshold Number

A third type of coupling between feedback paths is transition-to-place. This case is similar to place-to-place coupling. Obviously, the place where the two feedback paths join can receive at most a number of tokens equal to the sum of the number of tokens on the portions of the two paths from t_o to the place. (Note that the tokens on the initial overlapping portion of the paths are counted twice--once for each path.) If we add to this, the number of tokens on the two paths between the place and t_i this gives the number of times t_i can fire without firing t_o. (Note that the tokens on the final overlapping portion of the paths are only counted once.) Again this limit is achievable provided some other feedback path does not have a lower limit.

Finally, the last type of coupling which can occur between feedback paths is the transition-to-transition. In this case, the transition where the paths join cannot fire more times than the minimum of the number of tokens on

the two paths from t_o to that transition. Then, it also follows that t_i cannot fire more than the minimum number of tokens on the feedback paths.

All types of coupling between feedback paths, namely place-to-place, place-to-transition, transition-to-place and transition-to-transition, have been intuitively discussed. Again, for simplicity, we shall deal with only the transition-to-transition coupling between feedback paths. For these restricted Petri nets, the following lemma and theorem illustrate the way to determine threshold numbers. In the discussion, internal transitions of a subnet are all the transitions of the subnet except t_i and t_o.

Lemma 2

Let $\{P, T, A, t_i, t_o\}$ be a reducible subnet with only transition-to-transition coupling between feedback paths and between forward and backward paths. If transition t_i is disabled and can only be enabled by the firing of t_o, then there exists a feedback path which is free of tokens in the subnet.

Proof

Since t_i is disabled, it has at least one input place which is free of tokens. We want to show that there is at least one such place free of tokens no matter what internal transitions fire. If this were not true, there would be two places, say p_1 and p_2, such that a token can be on one or the other but not both. (The argument easily extends if there are more than two places.) However, this implies that there is a path from p_1 to p_2 and also a path from p_2 to p_1. Either p_1 or p_2 or both must be on a feedback path; otherwise, the firing of t_o could never cause the placement of a token on

them and the subsequent enabling of t_i. If both p_1 and p_2 are on feedback paths, then these feedback paths are coupled at places. It is a contradiction. Similarly, if one is on a forward path and the other on a feedback path then the forward path and the feedback path are coupled at places. Again, it is a contradiction. We conclude that there is one input place, say p_1, to t_i which remains free of tokens until after t_o fires.

Let t_1 be the input transition to p_1. Until after t_o fires, t_1 remains disabled. Thus, t_1 always has one input place free of tokens. By the same argument as above, there is at least one place, say p_2, which remains free of tokens until after t_o fires.

Continuing in this way, a feedback path $t_i p_1 t_1 p_2 t_2 \ldots$ can be constructed which is free of tokens. Q. E. D.

Let us define the token content of a feedback path as the sum of tokens in the places of the path. The minimum token content of the feedback paths or minimum token content for short is the minimum value of the token contents. The following theorem says that for a group of reducible Petri nets specified previously, the threshold number is equal to the minimum token content.

Theorem 4

Given a reducible net with only transition-to-transition coupling between feedback paths and between forward and feedback paths, the threshold number, m, is equal to the minimum token content, f, of the feedback paths.

Proof:

(i) $m \leq f$

This can be proved easily from the first two of the following observations:

1. Every firing of t_i removes a token from every feedback path.

2. Every firing of an internal transition does not change the token content of any feedback path. This is true, because only transition-to-transition coupling is considered.

3. Every firing of t_o adds a token to every feedback path. (This condition is *not* needed for this part of the proof; it is mentioned for completeness.) Thus, by firing t_i f times without firing t_o, we shall find that at least one feedback path is free of tokens. Therefore, t_i cannot fire again until after t_o fires. So, $m \leq f$.

(ii) $m \geq f$

The proof is by contradiction. Suppose $m < f$. That means after firing t_i m times without firing t_o, t_i cannot fire any more until the firing of t_o. From observations 1 and 2, and the definition of the minimum token content, it is clear that every feedback path should contain at least one token. But from Lemma 2, there exists a path free of tokens. This is a contradiction. Therefore, $m \geq f$.

It has been shown that $m \leq f$ and $m \geq f$, so $m = f$. Q.E.D. Obtaining the threshold number, m, of a net enables us to reduce the net in the form shown in Figure 6.

CONCLUSIONS

Consistency and invariance have been defined. Their physical meanings are suggested. A scheme for verifying net possession of properties of consistency and invariance has been proposed. By solving the homogeneous consistency equations, the transition firing ratios are also obtained. For a reducible net with only the transition-to-transition coupling, we are able to determine the threshold number, and thus reduce the net in the form shown in Figure 6.

Although only the reduction of a net has been discussed, the techniques for determining the consistency, invariance, transition firing ratios and threshold number of a net also provide a means to some extent for verifying the correctness of expansion. Because after expanding a net, we also can examine whether these properties are preserved.

There are still several problems remaining to be solved. For example, the scheme for determining the threshold number should be generalized to less restricted Petri nets. A mechanical way is required to select the input transitions for a subnet. In addition, the procedures for reducing a subnet with input places and output places has not been considered.

Finally, the investigation of hierarchy in terms of the representation of the data structure has been delayed for subsequent research.

TASK III

DATA FAULTS

INTRODUCTION

Based on the favorable results of using functional fault models to represent control related faults, we propose to extend these concepts to include data-fault phenomena. The objective of this task was to investigate the concept of classification of data faults based on the effect they produce on the behavior of the system. The intent was to identify a preliminary list of data-fault classes which cover the spectrum of fault conditions which occur in real systems.

The results of this effort were limited by two major stumbling blocks. The lack of fault documentation made it difficult to gather information and understand the underlying phenomena of data faults, which would be expected in the field. Secondly, the limited published results in modeling data and data transformations made it difficult to formulate and discuss data-fault issues.

A formal, complete model of data phenomena was needed to provide a foundation and consistent notation to build upon. This was the subject of a parallel task in the AFOSR effort which was unavailable. Thus, this effort began by informally developing the fundamental aspects of a data representation. The intuitive concepts will be formalized as part of the AFOSR report.

Approach

A two-pronged approach was used on this task. It began with an intensive review of some 1,000 failure reports which were collected from the Space Shuttle Engine Control program. These fault reports were chosen for two reasons. First, they are representative of the type of high reliability real-time control systems which required fault tolerant design. Second, it was one of the only programs known to have any formal written fault documentation. At best, this documentation was of limited value without direct contact with a knowledgeable production test engineer.

In parallel, the problem of fault classification was attacked from an abstract tops-down approach. It began with a literature search over data modeling approaches. This proved to be irrelevant. Next, we addressed the basic functions that occur in the data transformation portion of a system. As a foundation, the basic attributes of data were defined. From these, possible fault effects were defined.

ISSUES IN MODELING DATA FAULTS

Data faults can originate in a system from several sources:

- Design and implementation errors in the data transformation structure
- Physical deterioration of the data transformation structure
- Environmental errors propagating into the data transformation structure

Field experience has shown faults to be equally likely from any source until the design errors are debugged. Each of these sources must be considered in developing a fault classification system.

Four important questions which must be addressed in modeling data faults include:

- What are the observable effects of a given data fault?
- What conditions are prerequisites to producing the fault condition?
- Where can the contamination propagate?
- Where is the optimal point for detection/isolation/recovery?

We have observed that most faults eventually result in contamination of data values. But modeling at a level detailed enough to discriminate data values is far too complex, and the detail overwhelms the observable structure. Thus, we need to abstract the concept of data value. There are functionally two alternative approaches:

- Model data value in terms of acceptable range and faulty range of value
- Model data value as a binary correct or incorrect attribute

Either of these approaches is useful for tracing fault propagation. The first approach offers some measure of the problem of detectability based on the size of the range allowable for a given data packet.

Fault conditions can remain latent within the system until the necessary enabling conditions occur to cause a response. These responses can often be masked by the design of the system until they propagate to become visible in another non-faulty part of the system. The model should be capable of describing the conditions necessary for the fault to occur and/or propagate.

RESULTS OF FIELD FAULT REVIEW

From a set of approximately 1,000 reported fault conditions on the Space Shuttle Main Engine Control, we picked approximately 100 interesting cases for detailed review. These were carefully screened to arrive at 30 distinctive fault types. Each of these were reviewed in detail. These faults ranged from design errors where buffer sizes were assigned incorrectly to deterioration faults where instructions were overwritten.

An example of the fault cases reviewed is illustrated in the diagram of Figure 9. In this case, there was an algorithm which was inputting values into a table from an external I/O subsystem. One of the inputs was an index value Ix which was loaded by operation L. The value Ix was then used by operation Lx to index the values into the table which was located in the address space of Lx. The observed faulty phenomena was to find the data being overlaid on part of the instruction stream I1, I2,... This eventually caused the processor to halt when it tried to execute an instruction which had been overwritten to a halt instruction code. Meanwhile many parts of the memory were contaminated as the processor tried to execute the data stored in the instruction locations.

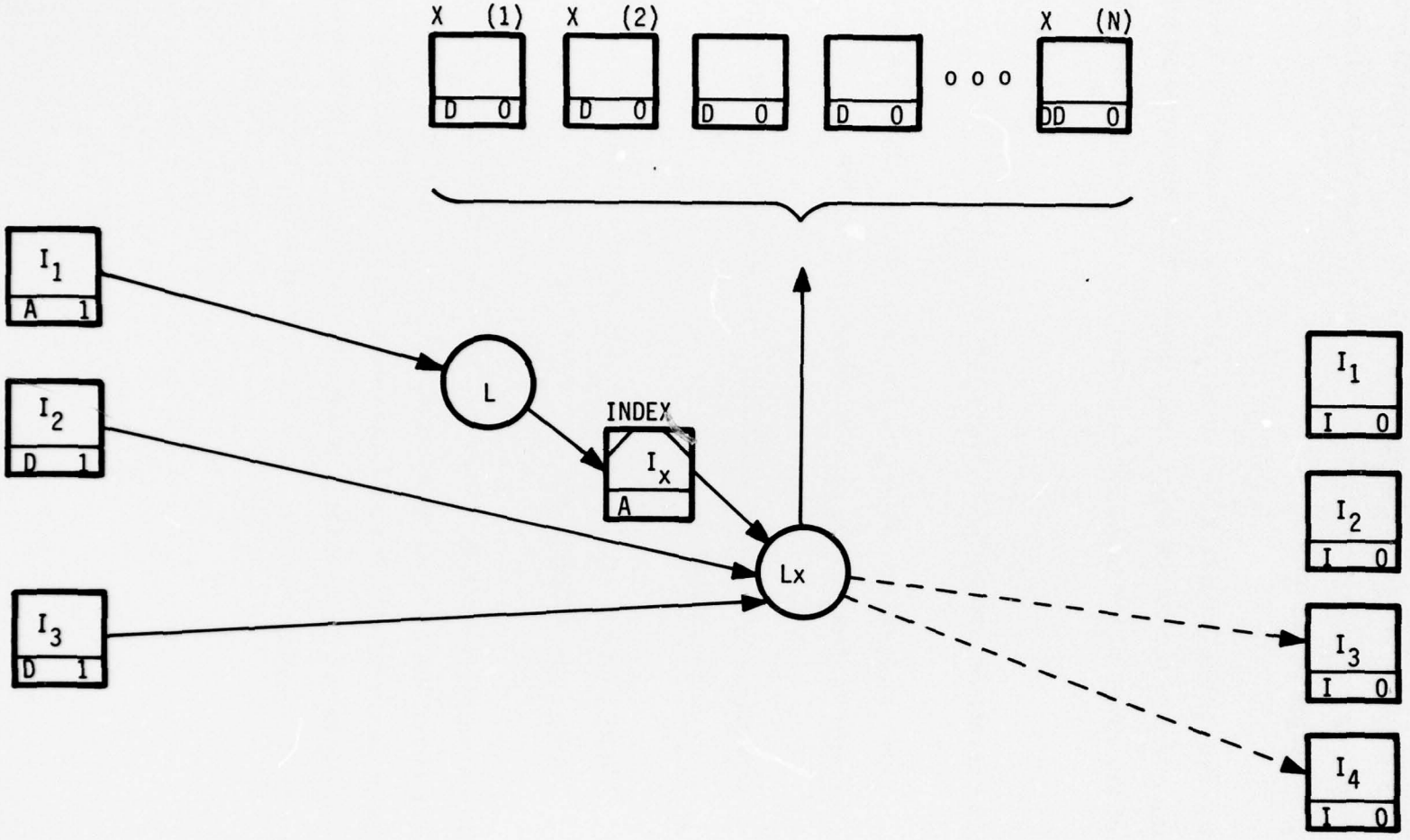


Figure 9. Simplified Data Structure for Sample Fault Case

Surprisingly enough, the source of the fault was an environmental error. The I/O subsystem had lost power and began to deliver faulty data packets I1, I2, I3, ... to the processor.

The faulty-data packet I1 which was interpreted as an index was used as part of the address for future store operations. This produced addresses outside of the range of the table, and allowed data to overwrite instructions.

Several observations can be made about this example.

- Data can exhibit a deterioration phenomena--either as system power is dropping or as it is being overwritten
- Address data is very sensitive data and can cause widespread data contamination
- Fault sources can be difficult to trace (we found half the system faults to be unexplained or poorly understood)

From this example we can conclude that the effect of the fault was the contamination of the instruction stream. It can be seen that this and any other out of range data fault could have been detected by placing a "within limits" check in the Lx function. When data values for Ix exceeded the bounds a fault would be indicated and the operation inhibited.

From the review of failure cases several observations were made:

1. All faults if left unchecked propagate until they affect some data value and/or cause catastrophic loss of control.

2. Design errors are the prevalent failure mode in software systems.

3. Hardware faults can produce data faults independent of control.

4. Software can display deterioration type of failure modes as a result of fault propagations.

5. Since control and data are tightly coupled systems, faults often propagate across the interface.

6. The lack of an accurate method to describe fault phenomena produces unintelligible fault documentation which hinders the understanding and learning from past fault experience.

In conclusion, correlation of fault cases as viewed at the implementation level was small. Although there were many cases of the same type of data fault, there were no patterns which emerged from the 30 sample fault cases. Limitations in fault documentation indicate this is not a fruitful approach and it will not be pursued.

FAULT ABSTRACTION APPROACH

In order to abstract the characteristics of data fault phenomena it was necessary to first address the issue of abstract data modeling. This effort was hindered by the lack of relevant research on data modeling. The work of Mealy in reference 14 was a useful starting point for developing an abstract concept of data.

Data processes will be viewed as composed of data packets, data transformations, and data maps, which define a correspondence between attributes and data packets. For our purposes, the list of data attributes can be simplified to the following five observable attributes of data packets.

- *Identifier* (*name*)
- Value range (allowable values)
- Type (interpretation)
- Generation (instance number)
- Location (physical storage)

The identifier is the name by which the data packet is defined in the system. The value range is defined to be the allowable correct data values which this packet can be assigned by a data transformation. The type attribute defines the way the system will interpret the coded binary bit pattern which makes up the data packet (i.e., real, integer, address, instruction, etc.). The generation attribute captures the dynamic nature of data. For cyclic control structures, this attribute defines which time

frame the data belongs to. Thus we associate an instance number with a given packet of data which corresponds to the time the data was written or updated. Finally the location attribute defines the mapping of data packet into physical storage.

The data transformations are a set of data maps which transform the set of input data into a set of output data. These functions include all the traditional arithmetic and logical functions as well as the accession function, the transfer function, etc. They can be either primitive functions like an ADD or complex functions such as a subroutine algorithm. Data transformations or data operators can be either data attribute preserving or attributed defining operations.

Transformations can be characterized at its interfaces by

- Input data domain
- Input data type
- Output data range
- Output data type
- Specification of the data map function
- Execution time

The input data domain and type define the range of values and data interpretation that the data transformation expects. Likewise for the outputs. The functional specification is generally a lower level description of the function as implemented with more primitive operators. In the case of primitive operators, this corresponds to a truth table or data table.

Faults resulting from deterioration or physical fault-sources along with environmental faults are hypothesized to be associated with changes in data transformation. Faults arising from design errors are hypothesized to be associated with changes in the structure and/or the attributes of data packets. The effects of data faults are proposed to be observable in deviations in the data packet attributes.

From these basic premises we can define fault classes based on the possible faulty effects on these observable attributes. For example, data type mismatch fault is the result of design error in the interpretation and use of a data packet. This type of fault could also be caused by an address transformation error which resulted in fetching the wrong data packet. Another more subtle example would be a faulty data type resulting from reading the incorrect instance of a data location which was being shared with another process.

Other fault classes which were proposed and are under consideration include:

- Value Error--Data Value is out of range
- Access Error--Incorrect Identifier used
- Generation Error--Incorrect Instance of data
- Initialization Error--Start up conditions were not met

Additional effort and a specific data representation are required to refine and formalize these preliminary concepts.

BIBLIOGRAPHY

1. R. M. Karp and R. E. Miller, "Properties of a Model for Parallel Computations: Determinacy, Termination, Queueing," SIAM J. Appl. Math., November 1966, pp. 1390-1411.

2. R. M. Karp and R. E. Miller, "Parallel Program Schemata," J. of Computer & System Sciences, 1969, pp. 147-195.

3. J. D. Noe and G. J. Nutt, "Macro E-Nets for Representation of Parallel Systems," IEEE Trans. on Computers, Vol. C-22, No. 8, August 1973, pp. 718-727.

4. J. B. Dennis and D. P. Misunas, "A Preliminary Architecture for a Basic Data Flow Processor," Proc. 2nd Annual Symposium on Computer Architecture, IEEE, January 1975.

5. G. J. Nutt, "Evaluation Nets for Computer System Performance Evaluation," University of Washington, TR #72-04-03, April 1972.

6. J. B. Dennis, "First Version of a Data Flow Procedure Language," Project MAC, MIT Group Memo 93-1, August 1974.

7. C. W. Rose and M. Albarran, "Modeling and Design Description of Hierarchical Hardware/Software Systems," Proceedings of 12th Design Automation Conference, 1975.

8. M. Hack, "Petri Net Languages," Lab. for Computer Science, MIT, RT-159, March 1976.

9. L. A. Jack, W. L. Heimerdinger and M. D. Johnson, "Theory of Fault Tolerance," Honeywell Annual Report, Systems & Research Center, Honeywell, Inc., September 1975.

10. T. Agerwala, "Towards a Theory for the Analysis and Synthesis of Systems Exhibiting Concurrency," Ph. D. Thesis, Johns Hopkins University, 1975.

11. F. Furtek, "A New Approach to Petri Nets," Project MAC, MIT, Group Memo 123, April 1975.

12. C. Ramchandani, "Analysis of Asynchronous Concurrent Systems by Petri Nets," Project MAC, MIT, TR-120, February 1974.

13. A. W. Holt and F. Commoner, "Events and Conditions: An Approach to the Description and Analysis of Dynamic Systems," Third Semi-Annual Technical Report, Part II, for the Project "Research in Machine-Independent Software Programming," Applied Data Research, Inc., April 1970.

14. G. H. Mealy, "Another Look at Data," IFIPS Proc. FJCC, 1967.

15. M. E. Senko, "Information Systems: Records Relations Sets Entities and Things," Information Systems, Vol. 1, No. 1, Pergamon Press, 1975.

16. M. E. D'Imperio, "Data Structures and Their Representation in Storage," Annual Review of Automatic Programming, Pergamon Press, Oxford, 1968.

17. B. Sundgren, "Conceptual Foundations of the Infological Approach to Data Bases," Data Base Management, edited by J. W. Klimbie, North Holland, 1974.

18. J. Kane and S. Yau, "Concurrent Software Fault Detection," IEEE Transactions on Software Engineering, Vol. SE-1, No. 1, March 1975.

19. J. Earley, "Toward an Understanding of Data Structures," Communications of the ACM, Vol. 14, No. 10, October 1971.

20. I. Miyamoto, "Software Reliability in Online Real Time Environment," Nippon Electric Company, Tokyo, Japan.

21. B. Liskov, "Specification Techniques for Data Abstractions," ACM International Conference on Reliable Software, Los Angeles, California, April 1976.

22. F. E. Allen and J. Cocke, "A Program Data Flow Analysis Procedure," Communications of the ACM, Vol. 19, No. 3, March 1976.

23. C. R. Litecky and G. B. Davis, "A Study of Errors, Error-Proneness and Error Diagnosis in Cobol," CACM, Vol. 19, No. 1, January 1976.

24. E. F. Codd, "A Relational Model of Data for Large Shared Data Banks," CACM, Vol. 13, No. 6, June 1970.

25. L. R. Johnson, "On Operand Structure, Representation, Storage and Search," Research Report RC-603, IBM Corporation, 5 December 1961.

26. M. V. Wilks, "The Outer and Inner Syntax of a Programming Language," The Computer Journal, Vol. 11, No. 3, November 1968.