

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Project MAC

Computation Structures Group

Memo No. 30

The Design and Transformation of
Asynchronous Computational Structures

(Part II - Output Deterministic Structure Graphs)

by

Fred Luconi

Summary

This paper describes a structural design schema for digital systems. Structures are assembled by interconnecting sets of processing units and control cells. Except for the sequencing constraints implemented by explicit data paths and control links, the processing cells operate asynchronously with respect to one another. The schema consists of a few types of uninterpreted processing cells and control cells plus rules for their interconnection via communication links. The schema is such that if the rules of construction are followed, one is guaranteed to generate a structure which is output functional.

Introduction

In previous works the notion of a computational schema was introduced.^{1,2*} The desire for establishing local rules of construction which could guarantee complete functionality was explained and satisfied; the definition of a well-formed computational schema (wfcs) resulted. It has been shown that the streams of data appearing on each of the communication links of a wfcs during execution are uniquely determined by the initial "state" of the system.

Since links designated as output are a subset of a schema's link set, a wfcs is output functional. The problem to which we now address ourselves is how the wfcs may be generalized to structures which guarantee only output functionality. Such structures are needed to model systems in which processing capabilities are shared by several processes. The structural schemata to be described will provide such a modelling technique.

A structural schema is defined by a set of input links, a set of output links, and a set of enabling links. A function defined over the schema inputs is associated with each enable link. A unique set of output links is associated with each function. When data is supplied to the inputs of a schema and some of the enable links are "activated", the data streams which become defined for the indicated output link sets are uniquely determined by the respective functions. In this sense a structural schema is said to be terminally functional.

A structural schema consists of a data structure, a control structure and an interface structure. The interface structure defines a set of input and output links for the schema. The data structure consists of a set of connected functional

*This paper will assume a knowledge of the techniques and notation used in the references.

operators. These functional operators which may be primitive functional operators or other structural schemata are characterized by a set of input links, a set of output links, and a set of enabling links. When enabled, the functional operator will apply a specified function to the data in its input links and store the results in its output links. No constraints are placed on the time taken for such executions.

Any network of functional operators may provide a specification of various transformations on data stored in the interface structure; each transformation or function is defined by some particular ordering of functional operator application. The specification of such sequencing is called the schema's control structure. Such structures define algorithms for implementing the input-output transformations supplied by the schema. These algorithms may describe parallel processing implementations wherein several functional operators in the data structure may be enabled simultaneously. Even more important is the fact that the services of particular functional operators may be shared by several such processes. The structure of the schema guarantees the non-conflicting use of these operators.

To aid in the understanding of structural schemata, the following interpretation may be helpful. Think of the data-structure as specifying fixed data paths between a set of functional operators. To constrain the sequence in which the operators of the data-structure may be applied, a precedence assignment is defined (the control structure). Together, the data- and control- structures define a functional transformation between input and output data streams. Several precedence assignments may be associated with a given data-structure with each specifying a different functional transformation. If we think of each control structure as defining a computation, then a structural schema may describe a system in which not only are various functional units shared by several computations, but the rules of system construction will guarantee that all such processing can be done concurrently without conflict. This technique enables us to implement

computational systems which provide several input-output transformations simultaneously.

Structural Schemata

A structural schema consists of three connected sub-structures: the data processing structure, the control or sequencing structure(s), and the interface. These three subsystems of a structural schema will now be defined.

The Data-Structure (D-structure)

The D-structure is an ordered 7-tuple $D = \langle F, D, A, E, \delta, \rho, \epsilon \rangle$, where

- 1) $F = \{f_1, f_2, \dots, f_a\}$ is a finite set of functional operators.
- 2) $D = \{l_1, l_2, \dots, l_b\}$ is a finite set of data links.
- 3) $A = \{a_1, a_2, \dots, a_c\}$ is a finite set of arbiters.
- 4) $E = \{e_1, e_2, \dots, e_d\}$ is a finite set of enabling links.
- 5) $\delta = F \rightarrow P(D)^*$ is a function over the set F which defines the set of input links in D associated with each f_i .
- 6) $\rho = F \rightarrow P(D)$ is a function over the set F which defines the set of output links in D associated with each f_i .
- 7) $e = F \rightarrow P(E)$ is a function over the set F which defines the set of enabling links associated with each f_i .

D-structures are represented as a sub-class of computational schemata. The following rules direct the construction of valid D-structures.

- i) Each functional operator $f \in F$ is a valid structural schema or a primitive functional operator having an input vector, $\delta(f)$, an output vector, $\rho(f)$, and one enabling link, $e(f)$. A primitive functional operator is illustrated in figure 1.

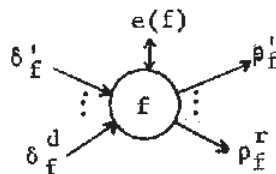


Figure 1

* $P()$ denotes the power set of the argument.

A structural schema which is used as a functional operator is illustrated in figure 2. A structural schema may provide several functions each of which is activated by a separate enabling link. Associated with each of these functions is a disjoint subset of the operator's output set. This recursive definition of structural schema will allow hierarchical design.

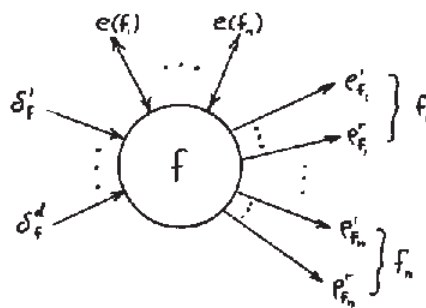


Figure 2

ii) The transformation sets of each operator constrain f_i to be applicable only if a 1 is associated with link $e(f_i)$. When function f_i is applied, the value of $e(f_i)$ is set to "2" and the values of the respective output links are replaced as specified by the function. In a primitive functional operator, each transaction will take the form:

$$\frac{e(f)}{1} \quad \frac{\bar{\delta}(f)}{\bar{S}^1} \rightarrow \frac{e(f)}{2} \quad \frac{\bar{p}(f)}{\bar{t}^1}$$

iii) Like well-formed computational schemata, every D-structure must be transformation lossless. However, unlike wfc's, write-conflicts will be prevented not by restricting the transaction sets of operators but by the inclusion of a set of non-deterministic operators called arbiters. One

arbiter is associated with every data link appearing in the output set of more than one functional operator or monitor operator (see section on interface structure).

As its name would suggest, the arbiter resolves write-conflicts for its associated data link. If f_1 and f_2 have an output link in common ($\rho(f_1) \cap \rho(f_2) \neq \emptyset$), then the enabling links of these two operators will be "connected" to the arbiter, a_i , associated with the "shared" data link, i.e., $e(f_1) \in e(a_i)$ and $e(f_2) \in e(a_i)$. The enabling links associated with arbiters are arranged in pairs. One enabling link pair, will exist for each operator connected to the arbiter. In figure 3, we see that link $e(f_1)$ is a member of $e(a_1)$ and will be referred to as an output link of arbiter a_1 . Corresponding to $e(f_1)$ is the arbiter input link $e'(f_1) \in e(a_1)$. Using the data found in its enabling set, an arbiter performs the non-deterministic function described as follows:

- 1) If all links $e(f)$ associated with arbiter, a_k , have a 0-value, and if link $e'(f_i) \in e(a_k)$ has a 1-value, arbiter a_k will store a 1 in link $e(f_i)$. If more than one of the arbiter's input links ($e'(f_i)$) has a 1-value, the arbiter may service only one request by setting $e(f_i)$ to 1. The operation of the arbiter is non-deterministic because the choice of which input link to service is arbitrary. It is important to note that at most one arbiter output link may have a 1-value at any given time.
- 2) If some link $e(f_i)$ has a 2-value, the arbiter stores a 0-value in that link, and stores a 2-value in link $e'(f_i)$.

If but a single arbiter becomes associated with functional operator f in order to resolve write-conflicts for some member of f 's output vector, the "connection" is simply accomplished by identifying the appropriate arbiter output, $e(f)$, as f 's enabling link. If, however, several of f 's output links are "shared", it then becomes necessary to associate several arbiters with f . Since f has only a single enabling link, a conjunctive input operator (CI-operator) is used to guarantee that f becomes applicable only after being activated by all associated arbiters. If f is connected

to n arbiters, the transaction set of the associated CI-operator appears as:

$\frac{e_1(f)}{\quad}$	$\frac{e_2(f)}{\quad}$	\dots	$\frac{e_n(f)}{\quad}$	$\frac{e(f)}{\quad}$	\rightarrow	$\frac{e_1(f)}{\quad}$	$\frac{e_2(f)}{\quad}$	\dots	$\frac{e_n(f)}{\quad}$	$\frac{e(f)}{\quad}$
1	1	\dots	1	0	\rightarrow	1	1	\dots	1	1
1	1	\dots	1	2	\rightarrow	2	2	\dots	2	0

where $e_i(f)$ is the output of an arbiter and $e(f)$ is the enabling link of operator \underline{f} .

In order that all functional operators may be activated by a call to a single enabling link, a junction-operator (J-operator) is introduced. When set with a 1-value a J-operator merely distributes 1-values to all the arbiters associated with a given operator, and then waits for the completion detecting 2-value. An n -output J-operator has the following transactions:

$\frac{e'(f)}{\quad}$	$\frac{e'_1(f)}{\quad}$	\dots	$\frac{e'_n(f)}{\quad}$	\rightarrow	$\frac{e'(f)}{\quad}$	$\frac{e'_1(f)}{\quad}$	\dots	$\frac{e'_n(f)}{\quad}$
1	0	\dots	0	\rightarrow	1	1	\dots	1
1	2	\dots	2	\rightarrow	2	0	\dots	0

where $e'_1(f)$ is the input of an arbiter and $e'(f)$ is the pseudo-enabling link of operator \underline{f} .

Figure 3 illustrates a D-structure with all necessary arbiters, CI-operators, and J-operators included. The CI- and J-operators are diagrammed as shaded circles; the arbiters as diamond shaped operators.

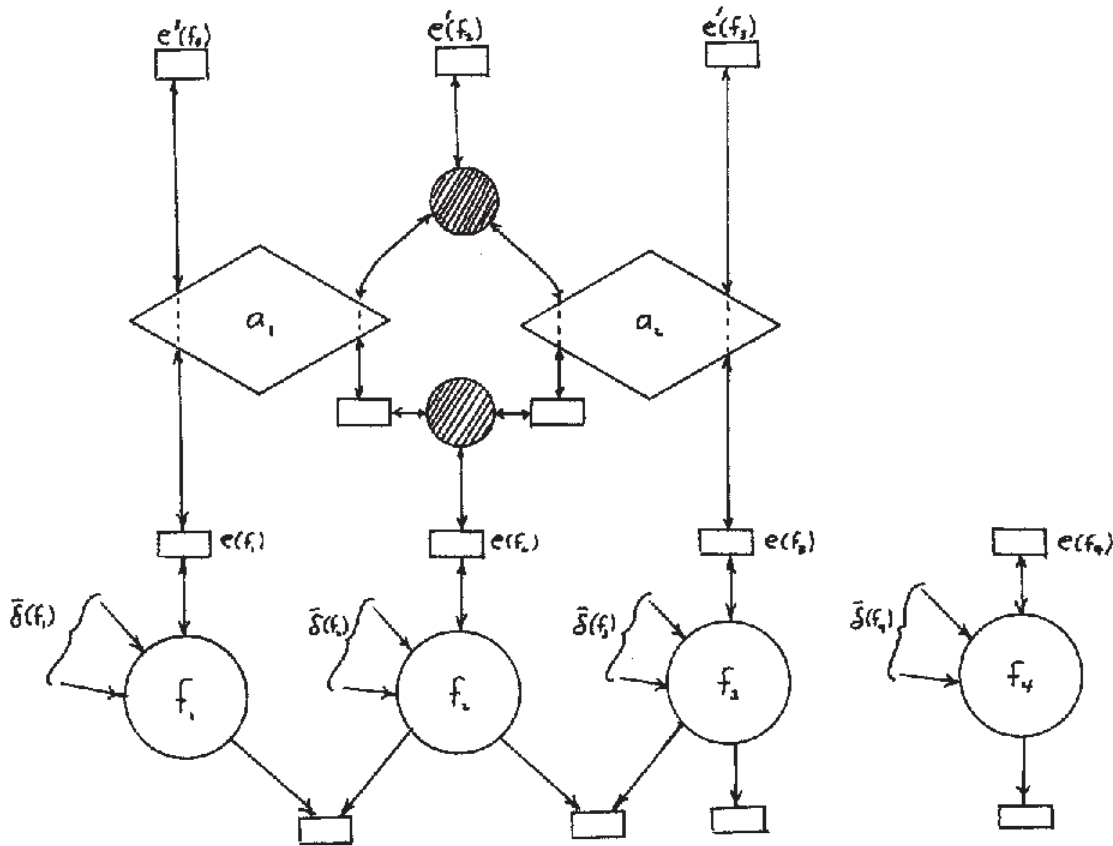


Figure 3

The Interface Structure (I-structure)

The I-structure is an ordered 5-tuple $I = \langle M, T, \delta, \rho, e \rangle^*$ where

- 1) $M = \{m_1, m_2, \dots, m_d\}$ is a finite set of interface monitor operators.
- 2) $T = \{t_1, t_2, \dots, t_e\}$ is a finite set of terminal links.

An I-structure is a well-formed computational schema. Each monitor has one input link, $\delta(m)$, one output link, $\rho(m)$, and one enabling link, $e(m)$. A monitor is illustrated in Figure 4.

* δ, ρ, e are functions over the operator set as defined earlier.

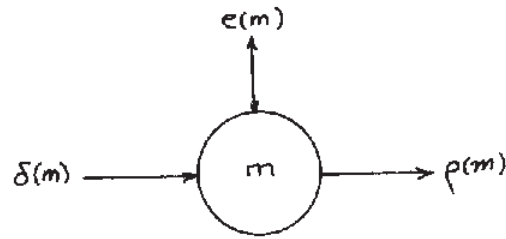


Figure 4

Either $\delta(m)$ or $\rho(m)$ must be a member of the terminal link set T. The transaction of a monitor implements a gated identity function as shown below:

$$\begin{array}{ccccccc}
 \underline{e(m)} & & \underline{\delta(m)} & & \underline{e(m)} & & \underline{\rho(m)} \\
 1 & & x & \rightarrow & 2 & & x
 \end{array}$$

The Control Structure (C-structure)

The C-structure is an ordered 6-tuple $C = \langle S, L, l_e, \delta, \rho, e \rangle$ where

- 1) $S = \{s_1, s_2, \dots, s_f\}$ is a finite set of control operators.
- 2) $L = \{l_1, l_2, \dots, l_g\}$ is a finite set of control links.
- 3) $l_e \in L$ is the control structure enabling link.

A C-structure is a well-formed computational schema. Each control operator $s \in S$ has an input vector, $\delta(s)$, an output vector $\rho(s)$, and an excitatory link, $e(s)$. The input links may be either control links or data-links. A control operator is illustrated in figure 5.

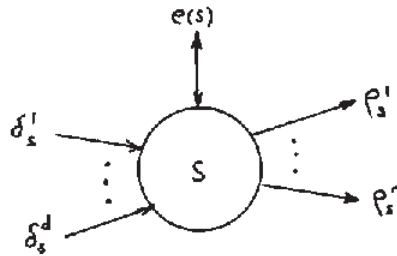


Figure 5

A control operator may apply only if its enabling link has a 0-value. The operator changes this value to a 1-value (which as we shall see causes a functional or monitor operator to become applicable). When a 2-value is returned to the enabling link, the control operator may store new values into the members of its output set and a 0-value into the enabling link. Each transaction appears in two parts as shown:

<u>e(s)</u>	<u>$\bar{s}(s)$</u>	→	<u>e(s)</u>	<u>p(s)</u>
0	\bar{s}^i		1	-
2	\bar{s}^i		0	\bar{t}^i

An example of a useful control schema is described in Appendix I.

The Structural Schema

A structural schema is an ordered 4-tuple $\langle D, I, C, E \rangle$ where D is a data-structure, I is an interface structure, C is a control structure, and E is an enabling link set. By restricting the "connection" of three such structures, we can guarantee the output-functionality of structural schemata.

1) Connection of D and I-structures (D,I-structures)

The monitors of I are used to get information in and out of the D-structure during the course of a computation. The input or output link of each monitor must be a member of the terminal set T; the other link must be a member of the data link set D. If $\rho(m) \in D$, then \underline{m} is an input-monitor, transmitting information from terminal link $\delta(m)$ to the D-structure via data link $\rho(m)$. If $\delta(m) \in D$, \underline{m} is an output-monitor.

If $\rho(m) \in D$ and is shared as an output link by some other operators, an arbiter must be used between \underline{m} and the other operators.

2) Connection of C and the D,I-structure

The control-structure directs the actions of the D,I-structure. This is done by identifying each $e(s)$ of the control structure as an enabling link of the D,I-structure. If an operator, \underline{f} or \underline{m} , in D,I is not connected to an arbitrating configuration, then $e(s)$ is simply identified with the link $e(f)$ or $e(m)$. If \underline{f} or \underline{m} is associated with an arbitrating structure, then the control operators must be connected through that unit. Any number of control operators may be identified with a given $e(f)$ link, but only a single control operator can be identified with a particular $e(m)$ link.

In order to guarantee non-conflicting use of the enabling links, we will make the following restrictions on the class of control-structures which may be connected to a particular D,I-structure. The C and D,I-structures must be related and connected in such a way that a control node connected to some functional operator \underline{f} must become applicable only after the application of other control operators in C have assured that data resides in all the data links input to \underline{f} . Since there can be arbitrary information in the data

links before schema execution, output-functionality can be guaranteed only by assuring that functional operators get proper inputs. The restriction also requires that all initially applicable control operators be connected to input monitors. We will call this set of control operators, the initialization set.

As explained in Appendix I, control operators which implement data-controlled branching are desirable. The one restriction we make on the use of data in D-links by the control operators is: the D-links which are used must be members of the input vector for the functional operator enabled by the control operator. This restriction guarantees that the appropriate branching data will be present when the control operator applies.

Execution of Structural Schemata

1) The structure is initialized by associating values with the terminal links and setting 0-values in the enabling links. The values of the data and control links are arbitrary to the point that operators input to them are not restricted from becoming applicable.

Execution is begun by storing a 1-value in link l_e of the control structure. This should make various members of the initialization set applicable.

2) Whenever an operator is applicable, it may apply itself, revising the values of links in its output and enable sets.

3) If execution terminates, we require that the final state always be one in which l_e has an associated 2-value and all the other links in L have 0-values.

Concluding Remarks

A structural schema which satisfies all the rules of construction is described in Appendix II. The structure illustrated shows how sequencing of the D-structure operators can be directed by control structures which look like program graphs. It is interesting to see how two such structures (one to calculate a simple arithmetic expression, the other to calculate a square root) can be simultaneously directing the actions of the D-structure.

The theorem on which we are now working is to show that if the terminal links connected to output monitors are defined as system outputs, then all structural schemata are output functional. The theoretical work which leads to the definition of structural schemata and the proof of the functionality theorem will be given in a forthcoming doctoral thesis.

A generalization of structural schemata as now defined is presently being studied. We are interested not only in the "sharing" of functional operators but also in the freedom of allowing a control operator function request to choose from several functional operators.

Interpretations of structural schemata in terms of modern computer technology must also be made. Perhaps new insights into the construction of distributed computational systems may evolve.

Appendix I

A Control Schema

The control schema presented in this section is an adaptation of the thesis research being conducted by Jorge Rodriguez.³ This schema is well-formed and satisfies the construction rules for control structures. The schema, which defines the concept of program graphs, allows easy description of algorithms as directed graphs. The utilization of parallel processing capabilities is easily specified.

A program graph may be constructed out of control links and six types of control operators; these will be explained in detail shortly. For ease of specification the description of control operators will omit the required treatment of enabling links (see section on C-structures). The control operators will be named according to the functional-operators or monitors which they activate.

The transformation sets for each operator type are shown in table 1. Two characteristics are common to all of the operator types. First, every transaction requires that all output links have 0-values; second, for each operator but the loop junction, all input links must have values other than 0.

The function- and junction-type control operators are explained adequately by their transaction sets. The selector operator enables the control sequencing to branch according to data conditions; it is able to read the contents of a data link input to its associated functional operator in the D-structure. Loop junction operators are used to control cyclic processing. Input link δ_f^1 of a loop junction is used to initialize the iteration. Input link δ_f^2 receives control after each iteration. The data-link supplies the information on which loop exiting depends. Loop output operators are provided to define what is to be considered the result of

an iterative process while guaranteeing that the structure is transformation lossless. δ_f^1 of loop output operators must be control output connectors for loop junctions, i.e., ρ_f^4 of loop junctions.

A program graph is a C-structure formed by interconnecting a finite set of control operators according to the following rules:

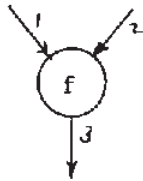
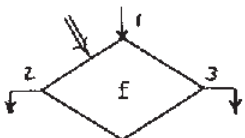
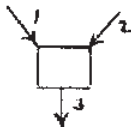
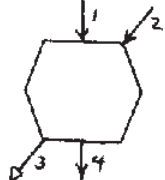
- 1) l_e must be the exclusive input link to a control operator connected to an input monitor.
- 2) Loop junctions and loop output operators must be connected as described above.
- 3) The terminating operation of every program graph must be to store a 2-value in link l_e . This is done by use of the termination operator.

In order to guarantee output functional structural schemata, a few restrictions must be placed on the relationship of a program graph and its connected D,I-structure:

- 1) There must exist an empty- or cleared-value for data links, and the operators of a D,I-structure must be allowed to write only into the links of their output sets which contain this value.
- 2) If a control operator activates an operator of the D,I-structure, the topology of the control structure must be such that previously executed control operators have supplied all necessary data to the input links of the activated functional-operator or monitor.

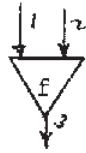
The examples of Appendix II will serve to clarify the concepts of program graph construction and the effect of control structures on structural schema execution.

Table 1

<u>operator types</u>	<u>schematic representation</u>	<u>transaction sets</u>
function operator		$1 \ 1 \ 0 \rightarrow 0 \ 0 \ 1 \ *$ $-1 \ 1 \ 0 \rightarrow 0 \ 0 \ -1$ $1 \ -1 \ 0 \rightarrow 0 \ 0 \ -1$ $-1 \ -1 \ 0 \rightarrow 0 \ 0 \ -1$
selector		$1 \ 0 \ 0 \rightarrow \begin{cases} 0 \ -1 \ 1 \ * \\ 0 \ 1 \ -1 \ * \end{cases}$ $-1 \ 0 \ 0 \rightarrow 0 \ -1 \ -1$
junction		$-1 \ 1 \ 0 \rightarrow 0 \ 0 \ 1$ $1 \ -1 \ 0 \rightarrow 0 \ 0 \ 1$ $-1 \ -1 \ 0 \rightarrow 0 \ 0 \ 1$
loop junction		$1 \ 0 \ 0 \ 0 \rightarrow 2 \ 0 \ 1 \ 1$ $2 \ 1 \ 0 \ 0 \rightarrow 2 \ 0 \ 1 \ 1$ $2 \ -1 \ 0 \ 0 \rightarrow 0 \ 0 \ 2 \ 0$ $-1 \ 0 \ 0 \ 0 \rightarrow 2 \ 0 \ -1 \ -1$ $1 \ 1 \ 0 \ 0 \rightarrow 2 \ 1 \ 1 \ 1$ $1 \ -1 \ 0 \ 0 \rightarrow 2 \ 0 \ 1 \ 1$ $-1 \ -1 \ 0 \ 0 \rightarrow 2 \ 0 \ -1 \ -1$ $-1 \ 1 \ 0 \ 0 \rightarrow 2 \ 1 \ -1 \ -1$

* Operators naming no function in the D,I-structure require no enabling link. Operators which do name a function in the D,I-structure activate the corresponding enabling link only during application of the starred transactions.

loop output



1	-1	0	→	0	0	0
1	1	0	→	0	1	0
-1	-1	0	→	0	-1	0
-1	1	0	→	0	0	0
2	1	0	→	0	0	1 *
2	-1	0	→	0	0	-1
2	0	0	→	0	0	-1

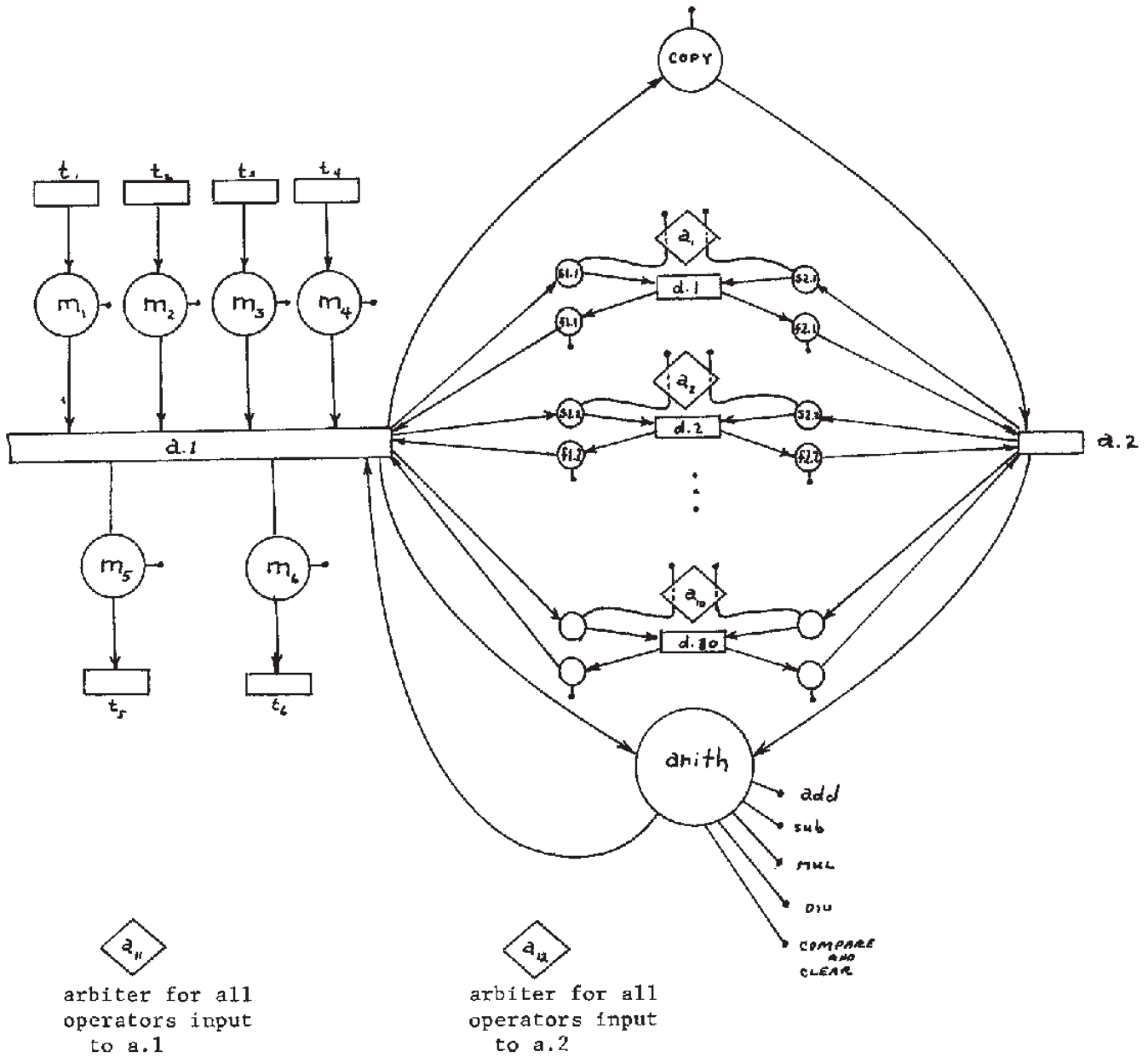
termination



1	0	→	0	2
---	---	---	---	---

Appendix II

Part I - D,I-structure (the naming conventions have been altered to provide a clear exposition)



Explanation of Function

- 1) $m_1 \dots m_4$ - input monitors
- 2) m_5, m_6 - output monitors
- 3) S1.i (s2.i) - Functional operator which stores the contents of link a.1 (a.2) into data link d.i.

This action does not clear the contents of link a.1 (a.2).
d.i must be empty (cleared) before the operator can become applicable.
- 4) f1.i (f2.i) - functional operator which fetchs the contents of link d.i into data link a.1 (a.2)

This action clears the contents of d.i.
a.1 (a.2) must be empty before the operator can become applicable.
- 5) copy - Functional operator which copies the contents of a.1 into a.2

This action does not clear the contents of a.1
a.2 must be empty before the operator can become applicable.
- 6) arith - structural schema which adds, subtracts, multiplies or divides the contents of a.1 and a.2 and then stores the result in a.1.

This action clears the contents of a.2
A compare operation is used to supply data to a control structure. After such an action both a.1 and a.2 are cleared.

a.1 and a.2 must be non-empty before the schema can become applicable.

Part II - Sample control structures using the conventions described in Appendix I.

Example 1: calculate the function $(X^2 + Y)(X^2 + Z)$

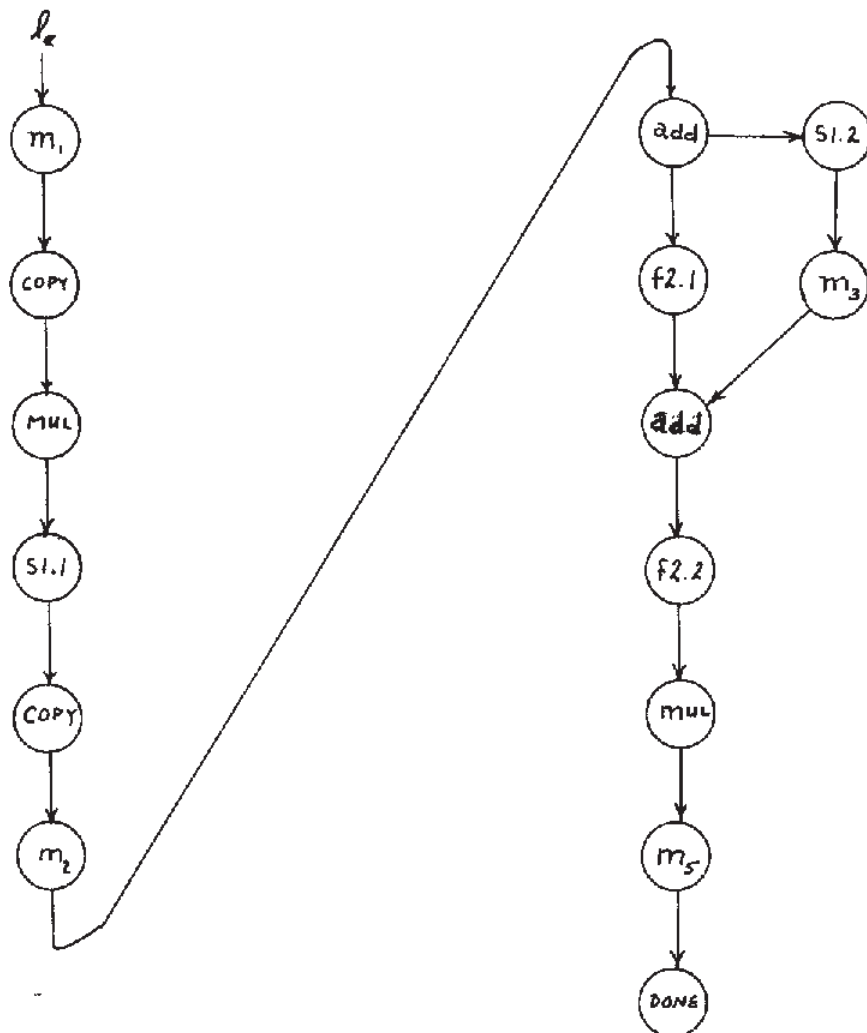
inputs - m_1 : X

outputs - m_5

enabling link - l_e

m_2 : Y

m_3 : Z



Example 2: Calculate the square root of M

inputs - $m_1 : M$

output - m_6

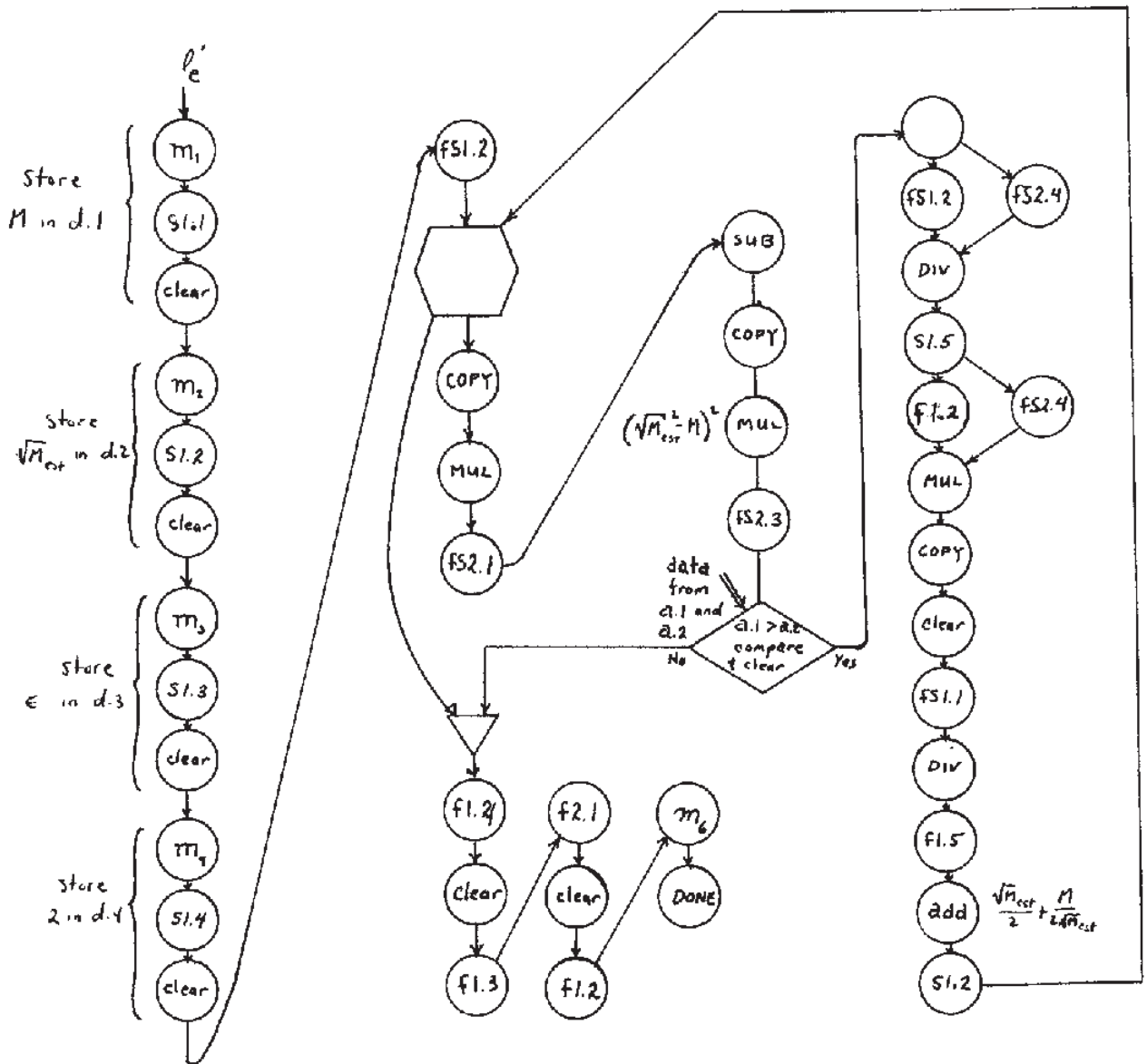
m_2 : initial guess for M

m_3 : allowable squared error, ϵ

$m_4 : 2$

enabling link - ℓ'_e

Note: convention $\textcircled{f_{i,j}}$ corresponds to the read-write function $\textcircled{f_{i,j}} \rightarrow \textcircled{s_{i,j}}$



example 3

It is very important to realize that although the two algorithms specified in examples 1 and 2 specify the use of common computational resources, the two control structures may concurrently be connected to the D,I-structure of part I. The sharing of operators and data links specified by the resulting configuration is guaranteed not to cause output-nondeterminism.

The resulting structural schema may now be diagrammed as an operator which supplies two functions as illustrated below:

