STAN-CS-70-171 SU-SEL-70-044

A Survey of Models for Parallel Computing

by

T. H. Bredt

August 1970

Technical Report No. 8

This work was supported in part by the Joint Services Electronic Programs U.S. Army, U.S. Navy, and U.S. Air Force under Contract N-00014-67-A-0112-0044 and by the National Aeronautics and Space Administration under Grant 05-020-377.

DIGITAL SYSTEMS LABORATORY

STANFORD ELECTRONICS LABORATORIES

STANFORD UNIVERSITY • STANFORD, CALIFORNIA

STAN-CS-70-171 SEL-70-044

A SURVEY OF MODELS FOR PARALLEL COMPUTING

by

T. H. Bredt

August 1970

Technical Report No. 8

DIGITAL SYSTEMS LABORATORY

Stanford Electronics Laboratories Computer Science Department

Stanford University

Stanford, California

This work was supported in part by the Joint Services Electronics Programs U.S. Army, U.S. Navy, and U.S. Air Force under Contract N-00014-67-A-0112-0044 and by the Natianal Aeronautics and Space Administration under Grant 05-020-337.

STANFORD UNIVERSITY
Digital Systems Laboratory
Stanford Electronics Laboratories Computer Science Department

Technical Report Number 8

A SURVEY OF MODELS FOR PARALLEL COMPUTING

by

T. H. Bredt

ABSTRACT

The work of Adams, Karp and Miller, Luconi, and Rodriguez on formal models for parallel computations and computer systems is reviewed. A general definition of a parallel schema is given so that the similarities and differences of the models can be discussed. Primary emphasis is on the control structures used to achieve parallel operation and on properties of the models such as determinacy and equivalence. Decidable and undecidable properties are summarized.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

INTRODUCTION

In recent years, a number of articles have appeared in the literature which may be grouped under the classification, models of parallel computing. These papers represent efforts to formalize intuitive notions of parallel computer systems, such as multiprocessor systems and systems with multiple functional units, and also parallel computations, which represent algorithms for solving mathematical problems such as the multiplication of two matrices. Of particular interest in these studies are the nature of the control structures which determine when operations in a system or computation are performed and the properties and characteristics of the models which result in correct operation.

The operation of a parallel computer system or the execution of a parallel computation can be characterized in the following way. First the system or computation must be defined and the initial conditions given. Operators produce changes in a data base. More than one operator may be being executed at a given time. When the execution of each operator is completed, it may be possible to execute other operators. A computation or system terminates its operation when the execution of all operators that are capable of being executed is completed. The time required to execute each operator is assumed to be unbounded but finite.

In this paper, only a portion of the current research in parallel computing is discussed in any detail. We consider the work of Adams [1, 2], Karp and Miller [13, 14, 15], Luconi [16, 17, 18], and Rodrigues [34]. Adams' work is an extension of the model of Karp

and Miller described in [13]. Adams' model is intended to describe parallel computations and not computer systems. The work of Karp and Miller on parallel program schemata extends work of Ianov [11, 12, 35] on sequential schemata to the parallel case. The emphasis here is also on the description of parallel computations. Rodriguez' work uses concepts from Muller's theory of speed independent circuits [30, 31] to develop a model for parallel computations. Luconi's model extends the work of Rodriguez and earlier work by Van Horn [39] and emphasizes the description of computer systems.

Early contributions to the theory of parallel computation are the work of Holt [8], Petri [32, 33], and McNaughton [28]. Petri's work, in particular his concept of Petri nets, has strongly influenced more recent work by Holt [9], Patil [31a], and Shapiro [37]. The work of Karp and Miller on program schemata has been extended by Slutz [38]. Rutledge [36] has developed a model which is another extension of the work by Ianov. Estrin, Martin and others at the University of California at Los Angeles [4, 23, 24, 25, 26, 27] have developed a model which is used mainly for the determination of schedules for computations in a multiprocessor environment. Bredt and McCluskey [5] have applied flow tables introduced by Huffman [10] to describe the control of parallel processes and in particular the control requirements for the mutual exclusion or interlock problem. Ashcroft and Manna [3] have defined a model for parallel computations which applies proof procedures of formal logic and is based on earlier work by Floyd [6] and Manna [19, 20, 21, 22]. It is hoped that in a future version

of this paper an integrated description of the papers mentioned in this paragraph can be given.

BASIC CONCEPTS

Consider the data base for a computation as a set of variables. By a computation, we mean the operation of a computer system or the execution of a parallel algorithm. A computation is said to be determinate or completely functional if the sequence of values associated with each variable in the data base is unique. Determinate computations are considered desirable although intuitively it is possible to have a correct result even though the intermediate value sequences are not unique. Two computations with the same data base are said to be equivalent if both result in the same set of value sequences for each variable in the data base.

There have been two fundamentally different approaches taken in the study of parallel computing. The first defines a model in which it can be proved that every computation which is represented in the model is determinate. This approach is used by Karp and Miller [13] and Adams [1,2]. Adams proves that every computable function (every function which can be computed by a Turing machine) can be represented in his model. This is not true for the model of Karp and Miller in which data-dependent decisions or conditional branches based on the values of variables in the data base are not allowed.

The second approach used is based on the definition of a model or schema in which not all computations are determinate. One theoretical result is the determination of a set of conditions which are sufficient to guarantee that a given computation will be determinate. In addition it can also be shown that under certain conditions either it is or it is not possible to give procedures to test if an arbitrary computation is determinate or if two arbitrary computations are equivalent.

One might well ask why there is interest in such theoretical properties of these models. One reason is that the conditions either implicit in the definition of the model itself or imposed to achieve determinate operation may give valuable insight which can be used in the design of future systems. Control techniques used to enable operations may also be of interest and questions about equivalence are important when transforming representations of computations in the interest of economization or optimization.

GENERAL DEFINITION OF A PROGRAM SCHEMA

In this section, a general definition of a model for parallel computation called a program schema is given. This definition is then modified and extended to describe the models of Adams, Karp and Miller, Luconi, and Rodriguez.

Definition 1:

A program schema or schema $\mathcal{S}$ is defined by a triple

$$\mathcal{S} = (M, A, C)$$

where

$M = \{x_1, x_2, \ldots, x_n\}$ a set of variables

$A = \{a, b, \ldots, c\}$ a set of operators (operations)

C (to be defined) a control

Each operator a has an input set I_a, $I_a \subseteq M$, and an output set O_a, $O_a \subseteq M$.

Associated with each schema is an interpretation defined as follows.

Definition 2:

An interpretation is defined by

1. For each variable x_i, a domain D_i of values which the variable may assume.
2. For each operator a, two functions

   F_a: a computation function which maps values associated with the variables in the input set I_a into values for the variables in the output set O_a.

G_a: a decision-making function (not explicit in all models). The output of this function is used by the control portion of the schema to determine which operations may be performed next.

3. The initial variable values.

A partial interpretation is defined by 1 and 2 above, but not 3.

Definition 3:

A variable history h_i is defined to be the sequence of values associated with the variable x_i during a computation.

Definition 4:

A schema history H is the n-tuple $\langle h_1, h_2, \ldots, h_n \rangle$ consisting of the variable histories for variables $x_1, \ldots, x_n$.

Using these definitions, it is possible to give a more precise definition of determinacy. First, we define the term as it is used by Karp and Miller in their papers on parallel program schemata [14, 15].

Definition 5:

A schema $\mathcal{S}$ is said to be determinate if and only if each interpretation results in a unique schema history.

The following definition will also be used. The phrase "partially interpreted schema" refers to a schema together with a partial interpretation.

Definition 6:

A partially interpreted schema δ is said to be determinate (completely functional) if and only if each set of initial variable values results in a unique schema history.

In the work of Karp and Miller on schemata, the results of a computation must be determinate in the sense of Definition 5. This is directly analogous to mathematical logic where theorems which are are valid must be true under every possible interpretation [29]. Definition 6 corresponds more to our intuitive notion of a computation in which not only the structure of the computation is known but also the functions which define the operations in the computation as well.

To illustrate these concepts, let us consider a few simple examples expressed, not in terms of schemas, but in terms of ALGOL-like programs with which most readers should be more familiar. A sequential program is shown in Table 1. If the initial value for variables u, x, and y is 0 and the initial value for v is 3, the variable histories for u, v, x, and y during the execution of this program are

Table 1. Example of a Sequential Program

```
    begin  integer u, v, x, y;
       x := u;
       y := v;
iter:  x := x + 1;
       y := y - 1;
       if y ≠ u then go to iter
    end.
```

$$h_u = < 0 >$$

$$h_v = < 3 >$$

$$h_x = < 0, 1, 2, 3, >$$

$$h_y = < 0, 3, 2, 1, 0 >$$

In general if α, β, γ, and δ are arbitrary integers which represent the initial values for variables u, v, x, and y, respectively, then

$$h_u = < \alpha >$$

$$h_v = < \beta >$$

and, if $\alpha < \beta$, then

$$h_x = < \gamma, \alpha, \alpha+1, \ldots, \beta-1, \beta >$$

$$h_y = < \delta, \beta, \beta-1, \ldots, \alpha+1, \alpha >$$

but, if $\alpha \geq \beta$, then

$$h_x = < \gamma, \alpha, \alpha+1, \alpha+2, \ldots >$$

$$h_y = < \delta, \beta, \beta-1, p-2, \ldots >$$

That is, if $\alpha < \beta$, all variable histories are finite and if $\alpha \geq \beta$, the variable histories for x and y are infinite and the execution of the program never terminates. However, in each case, the variable histories are unique and the execution of the program can be said to be determinate.

A second example is shown in Table 2. The reserved words "parbegin" and "parend" designate blocks of statements exactly as do "begin" and "end" ; however, all statements within a block defined by "parbegin" and "parend" may be executed concurrently. This extension to ALGOL has been proposed by Dijkstra [5a]. In this program, the statements x := u and y := v may be executed concurrently and the statements x := x + 1 and y := y - 1 may also be executed concurrently. In both cases, the execution of one statement cannot affect the execution of the other. The execution of the program in Table 2 is also determinate for all possible initial values for the variables and the variable histories are the same as those for the program of Table 1. In the sense that every possible set of initial values results in identical variable histories for the two programs, these programs can be said to be equivalent.

A third example is shown in Table 3. The execution of this program is not determinate if the initial value for u is less than the initial value for v. This follows because the variable history h_x depends on the rate at which the statements in block b2 are executed. Suppose u, x, and y are initially 0 and v has the value 2. Some of the possible variable histories for x are:

$h_x = < 0, 1 >$ $\qquad$ $h_x = < 0, 1, 2, 3 >$

$h_x = < 0, 1, 2, >$ $\qquad$ $h_x = < 0, 1, 2, 3, 4 >$

If the time to execute the statements in b2 is unbounded, the number of possible histories for the variable x is also unbounded.

Table 2. Example of a Program With Concurrent Statement Execution

```
     begin  integer u, v, x, y;
        parbegin
           x := u;
           y := v;
        parend;
iter:   parbegin
           x := x + 1;
           y := y - 1;
        parend;
        if y ≠ u then go to iter
     end .
```

Table 3. Another Example

```
begin  integer u, v, x, y;
   parbegin
      x := u;
      y := v;
   parend;
   parbegin
      b1: begin
     iter1: x := x + 1;
             if y ≠ u then go to iter1
          end;
      b2: begin
     iter2: y := y - 1;
             if y ≠ u then go to iter2
          end
   parend
end .
```

These three examples represent "partially interpreted" programs in the sense that the operations performed by each statement are specified. An "uninterpreted" program for the example of Table 2 is shown in Table 4. The symbol p_1 represents a predicate function which gives the value "true" or "false". In the sense of Definition 5, determinate operation requires that unique variable histories must be obtained for every possible choice of the functions f_1, f_2, f_3, f_4, and the predicate p_1.

CONTROL STRUCTURES

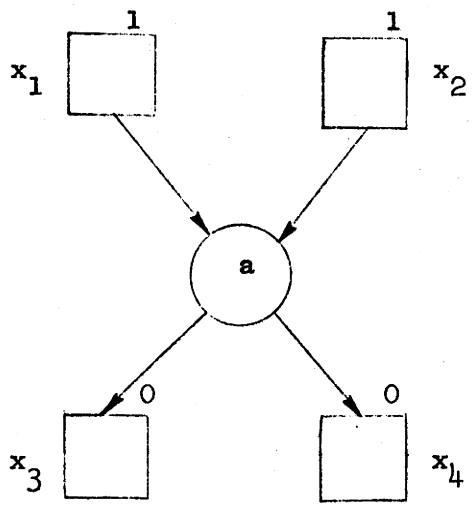
In this section, we consider the form of the control used to permit the initiation and termination of the operations in a schema. For the present, let us consider the variables in a schema to be cells in a memory or register.

Rodriguez

Rodriguez [34] associates status information with each variable. The status information specifies whether a variable is idle (0), ready (1), disabled (-1), or blocked (2). the function G_a of Definition 2 may be considered to map the status values associated with the input set and output set for an operator a into new status values. An example is shown in Fig. 1 where square boxes represent

Table 4. Uninterpreted Version of the Program in Table 2

```
     begin integer u, v, x, y;
        parbegin
           x := f1(u);
           y := f2(v);
        parend;
iter:   parbegin
           x := f3(x);
           y := f4(y);
        parend;
        if p1(u,y) then go to iter
     end .
```



$I_a = \{x_1, x_2\}$ $O_a = \{x_3, x_4\}$

G_a: $x_1 x_2 x_3 x_4$

1 1 0 0 ⟶ 0 0 1 1

Figure 1. Example of a change in status information.

variables and circles represent operators. In this example, status values for the input variables are changed to 0 and the status values for the output variables are changed to 1 when operator a is executed. This is not a coincidence. In fact, with the exception of data-dependent decisions, this is the mechanism used to determine when a particular operation is eligible for initiation. That is, the status of all input variables must be 1 (ready) and the status of all output variables must be 0 (idle). When the operation is performed, the status of all input variables is changed to 0 and the status of output variables to 1. Thus an operation may not be performed a second time until other operations are performed which change the status values of the variables in the input set to 1 and the status of variables in the output set to 0. This control technique has been borrowed from Petri [32, 33]. The 1 status values correspond to the "stones" or "tokens" which determine when the events in a Petri net may occur.

In the Rodriguez model, operators must be chosen from several basic operator types. Computations with data-dependent decisions and iteration can be represented but procedures and recursion cannot be described. Some but not all of the operator types Rodriguez has proposed will now be described.

1. Input Operator

The functions F and G are not defined for this operator. It is used only to provide input data for the model. The status of a variable which is only in the output set of input operators is assumed to be initially equal to 1 to indicate that the variable is ready for use.

2. Output Operator

The function F is not defined for an output operator. The function G changes the status values as defined below.

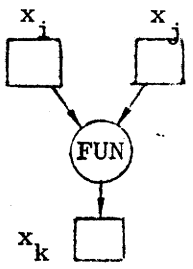
G:

$1 \rightarrow 0$

$-1 \rightarrow 0$

Thus, if ready or disabled status is associated with an input variable, the status is changed to idle.

3. Function Operator



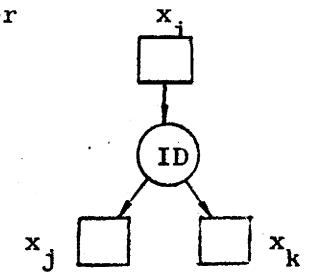
If all input variables have ready status and the output variable has idle status, the function F operates on the values for the input variables x_i and x_j to produce a new value for the output variable x_k. The G function changes status values for x_i, x_j, and x_k as defined below.

G: $x_i x_j x_k$

1 1 0 ⟶ 0 0 1
1-1 0 ⟶ 0 0-1
-1 1 0 ⟶ 0 0-1
-1-1 0 ⟶ 0 0-1

The function operator may have many input variables but only one output variable.

4. Identity Operator

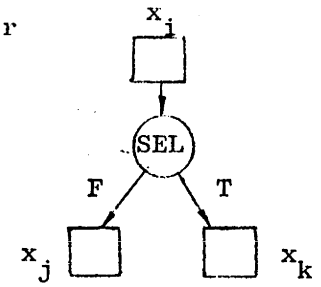


If the input variable has ready status and all output variables have idle status, the value of the input variable is copied by the function F to the output variables. The G function is defined below.

G: $x_i x_j x_k$

1 0 0 ⟶ 0 1 1
-1 0 0 ⟶ 0-1-1

5. Selector Operator



Each selector operator has an associated predicate function p which tests the input variable values. A selector operator may have more than one input variable. The F function copies the input value to x_k if p is true and x_j if p is false.*

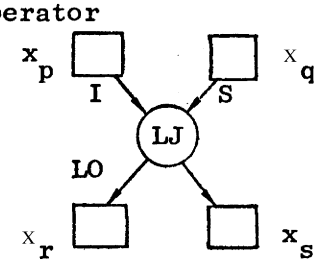
The G function is defined below.

G: $x_i x_j x_k$

$1\ 0\ 0 \longrightarrow \begin{cases} 0\text{-}1\ 1 \text{ if } p(x_i) \text{ is true} \\ 0\ 1\text{-}1 \text{ if } p(x_i) \text{ is false} \end{cases}$

$-1\ 0\ 0 \longrightarrow 0\text{-}1\text{-}1$

6. Loop Junction Operator



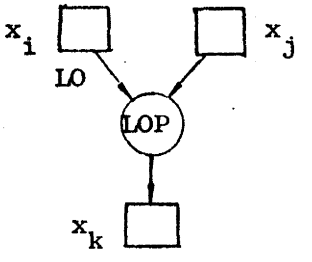
The loop junction operator is used to initiate an iterative computation. The input variable with line labelled I supplies the initial value for the iteration. The variable with line labelled S supplies the value on subsequent or "feedback" iterations. The output variable with label LO must be the

* This is not how Rodriguez defined his selector operator. His operator does not copy the data values. The above form is used to simplify the description of the iteration example which follows.

input to a loop output operator. The F function associated with the loop junction operator takes the value present for the I or S variable and passes it to the output variable x_s, if proper status values are present. The status transitions for a loop junction operator are given below.

G: $x_p x_q x_r x_s$		F:
1 0 0 0	→ 2 0 2 1	$x_s := x_p$
1 1 0 0	→ 2 1 2 1	$x_s := x_p$
1-1 0 0	→ 2 1 2 1	$x_s := x_p$
-1 0 0 0	→ 2 0 2-1	
-1 1 0 0	→ 2 1 2-1	
-1-1 0 0	→ 2-1 2-1	
2 1 0 0	→ 2 o-i 1	$x_s := x_q$
2-1 0 0	→ 0 0 1 0	

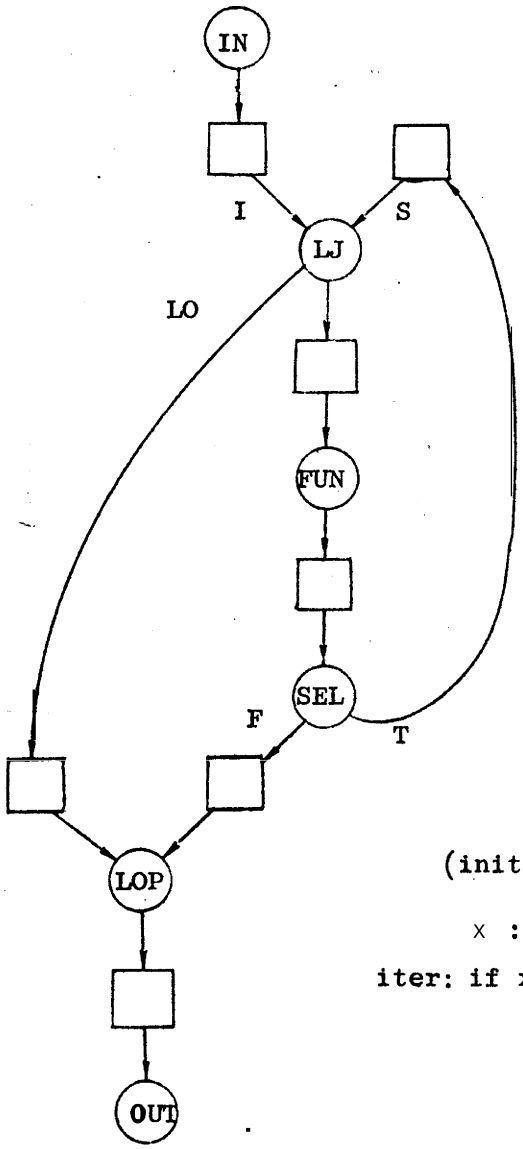
7. Loop Output Operator



The input variable with line labelled LO must be the LO output of a loop junction operator.

G: $x_i x_j x_k$		F:
1 1 0	→ 0 0 1	$x_k := x_j$
1-1 0	→ 0 0-1	
-1 1 0	→ 2 0 0	
-1-1 0	→ 2 0 0	
2 1 0	→ 0 1 0	
2-1 0	→ 0-1 0	

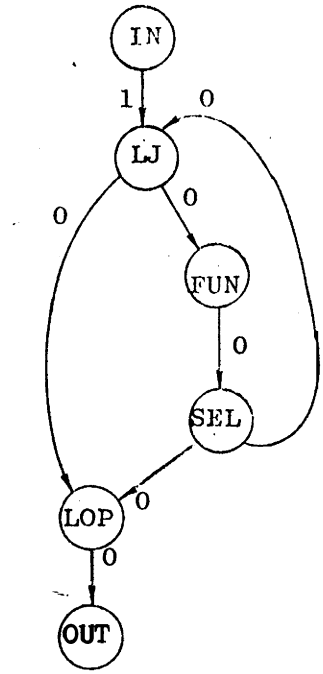
To illustrate the use of these operators, an example of a simple iterative computation, shown in Fig. 2, will be described. The two ALGOL statements describe the computation performed. In Fig. 3, operators are joined directly by arcs and the boxes representing variables are omitted for clarity. Status values are given by the labels on each arc. The initial status values are shown in Fig. 3a. Fig. 3b shows the status values after the first execution of the loop junction operator. The status of the initial value variable has been changed to 2, which blocks further entry to the loop junction until the iteration is complete. The loop junction operator copies the initial data value, in this case 1, into the variable which is the input to the function operator. The function operator is executed next. It subtracts one from the input variable value and places the new value in its output variable. The selector operator may now be executed. The status values after the execution of the selector are shown in Fig. 3c. The selector operator readies the feedback input to the loop junction and disables the input variable for the loop output operator. The loop output operator must be executed next and the status values obtained are shown in Fig. 3d. The loop junction is now executed giving the status values of Fig. 3e. Both the function and loop output operators may now be executed concurrently; the status values obtained are shown in Fig. 3f. The selector operator is now ready to be executed. This time, the test fails and the feedback input to the loop junction is disabled and



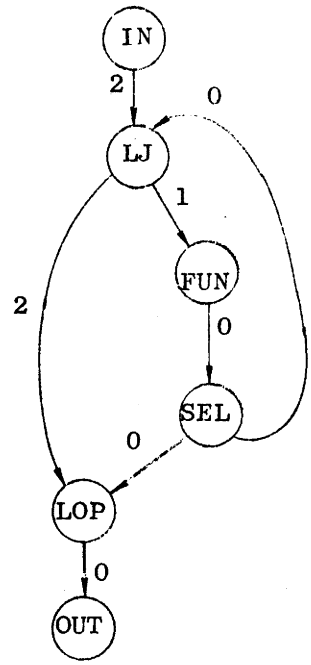

```
(initially x is 1)

   x := x - 1;
iter: if x = 0 then
         begin
            X := x - 1;
            go to iter
         end
```

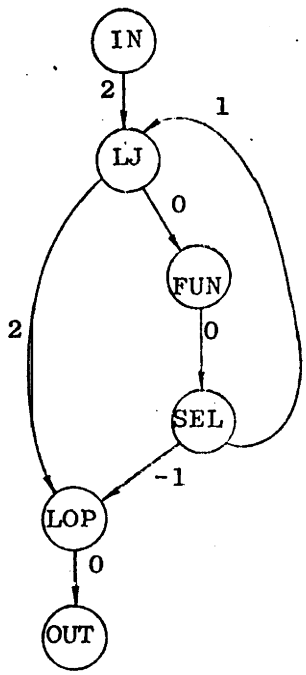
Figure 2. Iteration example in the Rodriguez model.



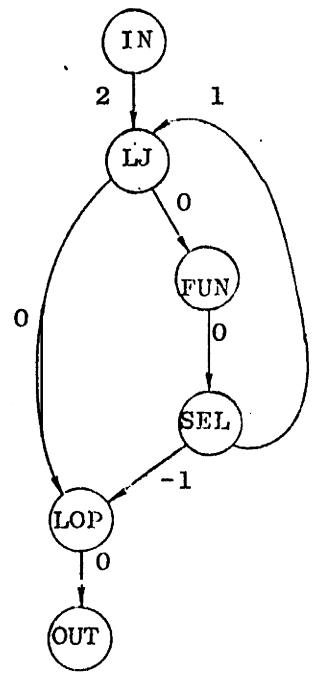
a) initial



b) after LJ

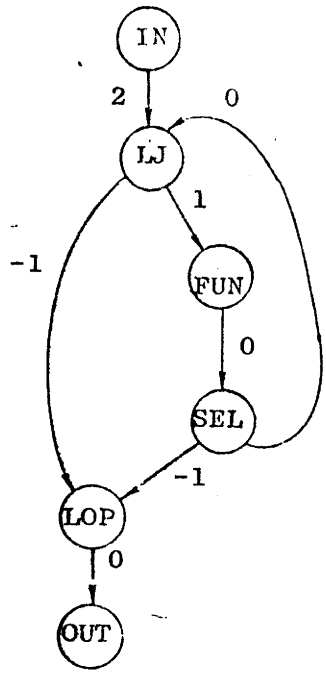


c) after FUN and SEL

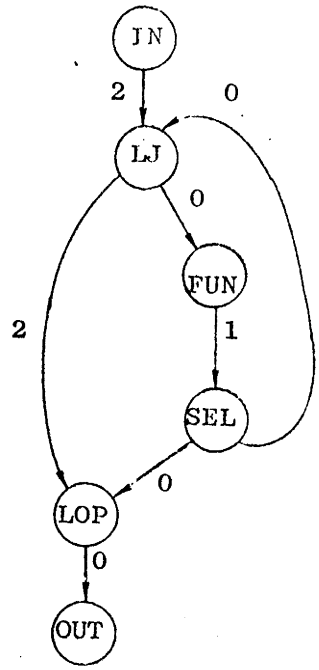


d) after LOP

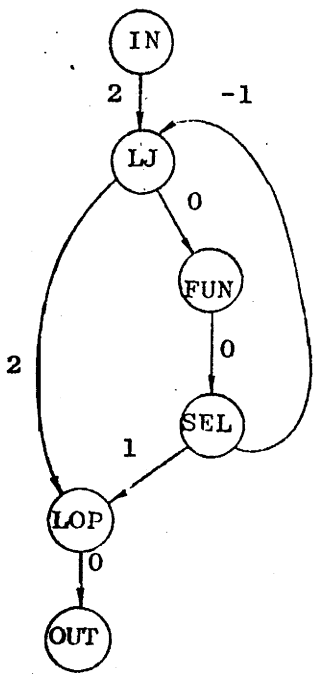
Figure 3. Status value transitions for the example of Fig. 2.



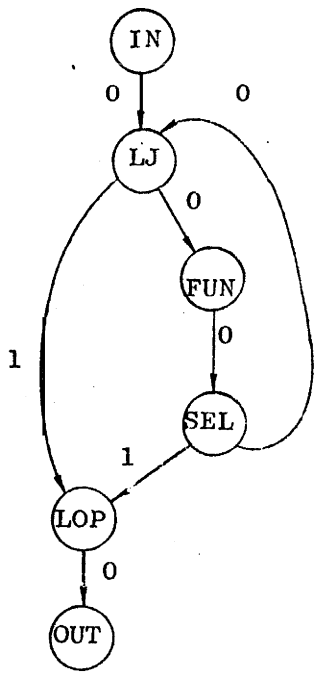
e) after LJ



f) after FUN and LOP



g) after SEL



h) after LOP and LJ

Figure 3. (continued)

the input to the loop output is readied. The status values are shown in Fig. 3g. The loop output is executed next which allows the loop junction to be executed unblocking the initial value input to the iteration. The status values are shown in Fig. 3h. The computation terminates with the execution of the loop output operator followed by an execution of the output operator. At termination (not shown) all variables have idle (0) status values.

Using these techniques for controlling the execution of operations, Rodriguez is able to prove the following theorem.

Theorem 1:

If a computation in the Rodriguez model terminates, it is determinate.

In this theorem, determinate operation implies that the variable histories and the status histories as well are unique. The data functions, F, associated with function operators may be arbitrary.

Rodriguez' results are actually not stated in the above form. His results are based on Muller's definitions of speed independence and use the concept of <u>state</u> of the model as defined by the current variable values rather than the variable histories defined earlier. The fact that once operators are ready to be executed they may not be disabled

corresponds in essence to Muller's concept of semi-modularity [31]. It can be shown that the operation of the Rodriguez model is determinate in the sense that the variable histories are unique*. Rodriguez gives necessary and sufficient conditions for a computation to terminate. These conditions are related to the absence of hang-up states. A hang-up state is entered if the computation terminates such that no operator may be executed and some variable does not have idle status. Rodriguez states an equivalence problem for his model and proves that it is decidable.

Luconi

The work of Luconi [16, 17, 18] differs from that of Rodriguez in the following way. A model corresponding to a partially interpreted schema with variables corresponding to memory or register cells is defined. However, no status information is associated with the variables. Instead, some variables contain data which serves only to determine when operators may be executed. When an operation is performed, the transformation defined by the F and G functions is carried out. Luconi assumes that the output values produced propagate instantaneously (line delays are zero).

The following two conditions are defined which relate to the well-formedness of Luconi schemas.

* Private communication from F. L. Luconi.

Definition 7:

Two operators a and b are said to be conflict-free if and only if whenever a and b may be executed concurrently, any common output variable must receive the same value from each operation.

A slightly stronger condition is that $O_a \cap O_b = \phi$, where ϕ is the empty set.

Definition 8:

Two operators a and b are said to be transformation-lossless if and only if whenever a and b may be executed concurrently, the execution of operator a does not affect the results to be produced by operator b and vice versa.

A slightly stronger condition is that $O_a \cap I_b = \phi$ and $O_b \cap I_a = \phi$. A partially interpreted schema is said to be conflict-free if all pairs of operators are conflict-free; it is said to be transformation-lossless if all pairs of operators are transformation-lossless. Luconi proves the following theorem.

Theorem 2:

Every schema in the Luconi model which is both conflict-free and transformation-lossless is determinate in the sense of Definition 6.

The conflict-free and transformation-lossless conditions are "local" in the sense that they may be tested by examining pairs of operators which may be executed concurrently. Luconi's transformation-lossless condition is essentially the same as the semi-modularity condition of Muller if the variables are interpreted as values on interconnecting lines rather than memory cells. Muller can have no conflicts because the output line for each operator is unique and not shared with other operators.

Luconi proves that there is no procedure to determine if an arbitrary, partially interpreted schema is determinate. That is, the decision problem for determinacy is unsolvable.

In Fig. 4, an example of the Luconi model is given which represents the computation for the program in Table 2. The dashed lines indicate portions of the schema the primary function of which is to control when operations may be performed. The initial variable values are shown inside the square boxes. In this example, the control variables (dashed boxes) have their values changed in a manner similar to that used by Rodriguez. That is, before an operator may be executed, certain "input" variables must have the value 1 and certain "output" variables must have the value 0. During the execution of the operator, the "input" values are changed to 0 and the "output" values are changed to 1. We will represent the changes in control

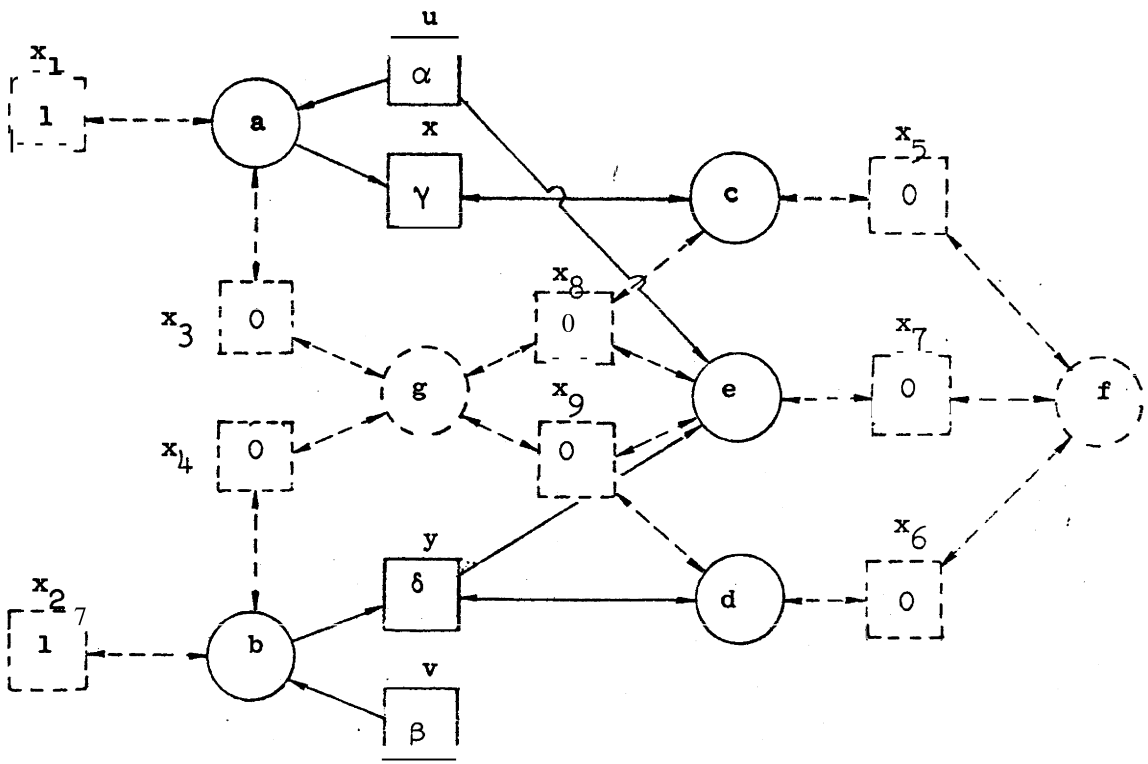


Figure 4. Use of the Luconi model to represent the computation of Table 2.

variable values using the G function and the change in data values using the F function. These functions are defined in Table 5. This example satisfies both the conflict-free and transformation-lossless conditions and is determinate.

In the second part of his thesis, Luconi views the control, which determines when operators may be executed, as being separate from the rest of the schema but still defined as a schema. A schema in this form is called a structural schema and is composed of two parts, an Interpretation-schema (I-schema) and a Control-schema (C-schema). The I-schema performs the computation and the C-schema determines when the operators in the I-schema are enabled.

Associated with each operator in the I-schema is an operator in the C-schema. These operators share a common control variable. Before an operator in the I-schema is eligible to be initiated, the control variable must have the value 0. When the I-schema operator is eligible to be initiated, the value of the control variable is set to 1 by the C-schema operator. When the I-schema operator terminates its execution, it sets the value of the control variable to 2. Fig. 5 shows the interconnection between an I-schema operator and a C-schema operator.

Luconi defines C-schema operators corresponding to Rodriguez' selector, loop junction, loop output, and other operator types. The status values are kept in variables which are part of the C-schema.

Table 5. Function Transitions for the Example of Fig. 4

G_a:	$x_1 x_3$ 1 0 ⟶ 0 1	F_a: x := u
G_b:	$x_2 x_4$ 1 0 ⟶ 0 1	F_b: y := v
G_c:	$x_8 x_5$ 1 0 ⟶ 0 1	F_c: x := x + 1
G_d:	$x_6 x_9$ 1 0 ⟶ 0 1	F_d: y := y - 1
G_e:	if y ≠ u then $x_7 x_8 x_9$ 1 0 0 ⟶ 0 1 1	F_e: null
G_f:	$x_5 x_6 x_7$ 1 1 0 ⟶ 0 0 1	F_f: null
G_g:	$x_3 x_4 x_8 x_9$ 1 1 0 0 ⟶ 0 0 1 1	F_g: null

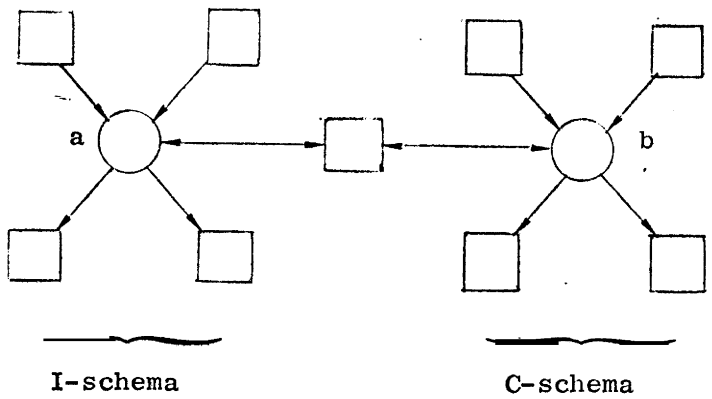
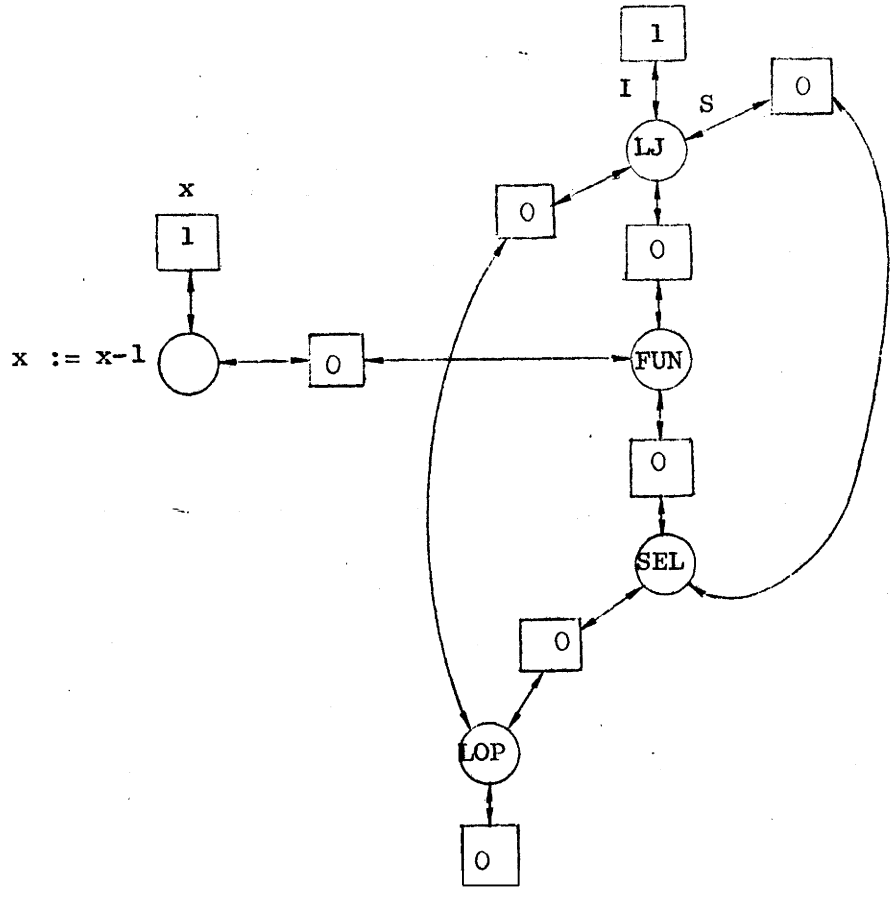


Figure 5. Interconnection of an I-schema operator with a C-schema operator.

A structural schema for the iteration example of Fig. 2 is shown in Fig. 6. The computation proceeds as before except that the function operator (FUN) in the C-schema is responsible for monitoring the execution of the I-schema operator which subtracts one from the current value of the variable x. Notice that data values are no longer passed from operator to operator as in the Rodriguez model.

QUEUES

FIFO (First-In, First-Out) queues have played an important role in the models of Adams [1, 2] and Karp and Miller [13, 14, 15]. They have been used in two different ways. In the first approach, used by Adams and in the Karp and Miller program graph model [13], each variable is considered to be a FIFO queue rather than a simple memory cell. It is required that each queue receive output data from exactly one operator and provide input data for exactly one operator. Adams allows complex data structures as queue entries and associates with each queue status information, which is used for the same purpose as in the Rodriguez model, to control data-dependent branches. In the Adams and Karp and Miller models, operators are ready to be executed when their input queues are non-empty, assuming appropriate status values in the case of Adams. Karp and Miller do not need status information because they do not allow data-dependent decisions in their model.




```
(initially x is 1)

      x := x - 1;

iter : if x = 0 then
          begin
            x := x - 1;
            go to iter
          end
```

Figure 6. A structural schema for the iteration example of Fig. 2 .

Karp and Miller have proved the following theorem.

Theorem 3:

Every computation described in the program graph model is determinate.

Karp and Miller investigate termination properties of their model and also the determination of bounds on the lengths of the queues.

Adams' model is a programming language for describing parallel computations. He allows graph procedures which may be recursive. In addition if, when an operation is initiated, there are sufficient entries in the input queues to permit the operator to be performed more than once, copies of the operator may be created and executed in parallel. Adams proves the following theorem.

Theorem 4:

Every computation described in the Adams model is determinate.

The second way in which queues have been used is in the program schema model proposed by Karp and Miller [14, 15]. Each operator a has an associated queue $\mu(a)$. To aid in understanding how these queues are used, the control structure for the Karp and Miller schema model must be described. The control is a transition system which undergoes changes in control state as the result of the initiation and

termination of operator executions. Suppose that after operator a is executed, operator b is ready to be executed. The function G_a produces a symbol, say a_1, which causes a control-state transition into a state from which operator b may be enabled. This control situation is illustrated by a form of transition diagram shown in Fig. 7. The a_1 outcome from operator a causes the control to enter state q . The enabling of an operator, in this case operator b, is indicated by a transition to another control state. The arc joining the two control states is given a label consisting of the operator name with an overbar, in this case $\bar{b}$. After operator b is enabled, the control enters control state q' from which, in this example, it is possible to enable operator a once more.

The phrase "enable operator b" has the following meaning. Take the values of all input variables (memory cells) for operator b and make these values the next entry in the FIFO queue $\mu(b)$ associated with operator b. The actual execution of operator b is now accomplished in some unspecified manner. When operator b terminates its execution, the queue entry is removed and the output values as determined by the function F_b are assumed to be assigned to the variables in the output set O_b. In addition, the output of the decision-making function G_b causes a control-state transition. A formal definition of the control portion of the Karp and Miller schema model is given below.

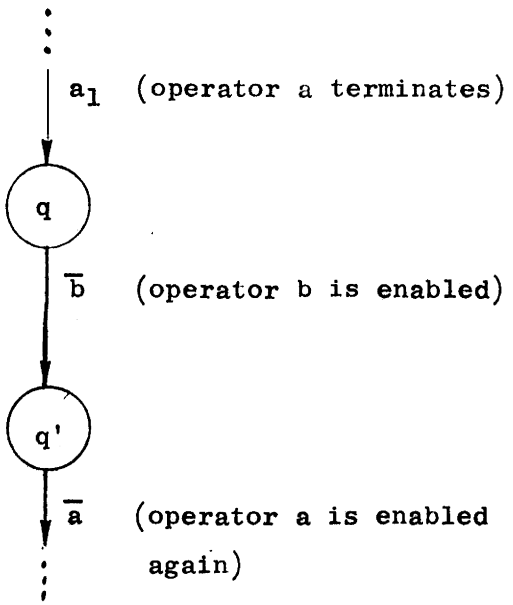


Figure 3. Diagram of control to enable operator b on termination of operator a.

Definition 9:

The control C of a Karp and Miller schema is defined by a quadruple

$$C = (Q , q_0 , \Sigma , \tau)$$

where

Q is a set of control states

q_0 is the initial control state

$\Sigma = \bigcup_{a \in A} \{ \bar{a}, a_1, a_2, \ldots, a_{K(a)} \}$ the control alphabet

$\tau: Q \times \Sigma \longrightarrow Q$ the transition function

The transition $\tau(q,\bar{a})$ specifies the control state entered when operator a is enabled and the queue entry is made for operator a. Transitions $\tau(q,a_i)$, $i = 1, 2, \ldots, K(a)$, specify the control state entered when the execution of operator a is complete. K(a) is the number of data-dependent outcomes for the operator a. Karp and Miller require that $\tau(q,a_i)$, $i = 1, \ldots, K(a)$, be defined for all $q \in Q$ and for all $a \in A$. It is assumed that when an operator is enabled, the operator is executed in a finite but unbounded time.

We now define the state of a schema.

Definition 10:

The state α of a schema is defined by the triple

$$\alpha = (\text{variable values}, q, \mu)$$

where the variable values are the present values of all the variables in M, q is the present control state, and μ represents all queues associated with schema operators.

A Karp and Miller schema for the iteration example of Fig. 2 is shown in Fig. 8.* An illustration of a Karp and Miller schema for the program of Table 2 is shown in Fig. 9.*

The equivalence of schemata is defined as follows.

Definition 11:

Given schemata $\mathcal{S}_1$ and $\mathcal{S}_2$

$$\mathcal{S}_1 = (M, A, C_1)$$

$$\mathcal{S}_2 = (M, A, C_2)$$

$\mathcal{S}_1$ and $\mathcal{S}_2$ are equivalent if and only if for each interpretation, the set of schema histories for $\mathcal{S}_1$ is equal to the set of schema histories for $\mathcal{S}_2$.

---

* Control transitions which return to the same state are omitted for clarity.

$$A = (M,\ A,\ C)$$

$$M = \{x\}$$

$$A = \{a\}$$

$$C = (\ \{q_0, q_1, q_2\}\ ,\ q_0,\ \{\bar{a}, a_1, a_2\}\ ,\ \tau)$$

$$\tau(q_0, \bar{a}) = q_1$$
$$\tau(q_0, a_1) = \tau(q_0, a_2) = q_0$$
$$\tau(q_1, a_1) = q_0$$
$$\tau(q_1, a_2) = q_2$$
$$\tau(q_2, a_1) = \tau(q_2, a_2) = q_2$$

Interpretation

D_x: integers (x is initially 1)

F_a: x := x - 1

G_a: if x = 0 then a_1 else a_2

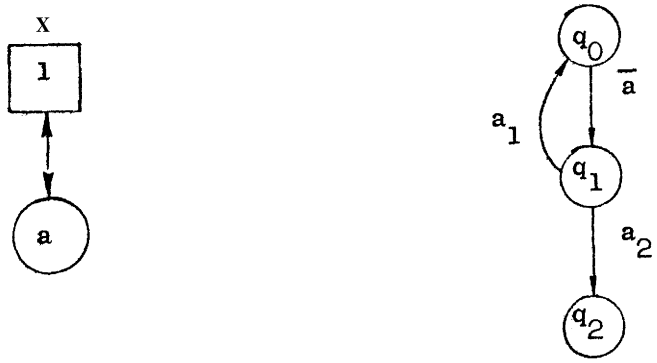
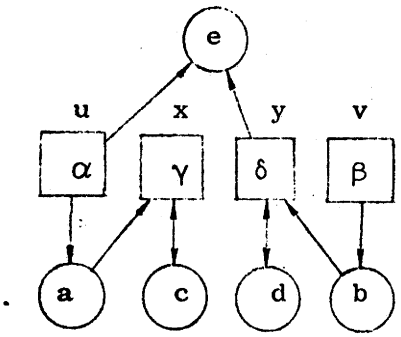


Figure 8. Karp and Miller schema for the iteration example of Fig. 2.



M = u, v, x, y

A = a, b, c, d, e

F_a: x := u

G_a: a_1

F_b: y := v

G_b: b_1

F_c: x := x + 1

G_c: c_1

F_d: y := y - 1

G_d: d_1

F_e: null

G_e: if y ≠ u then e_1 else e_2

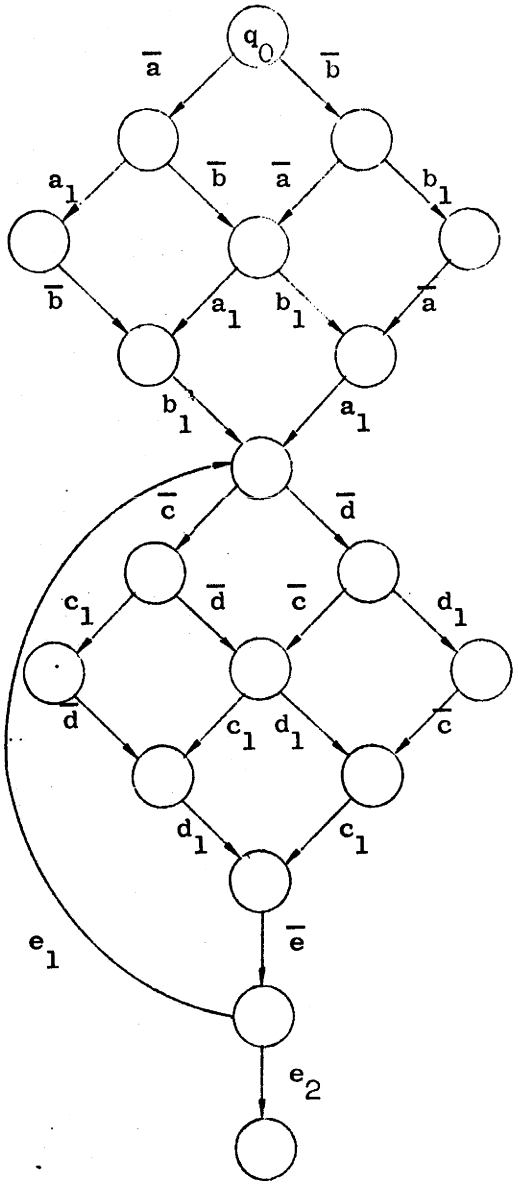


Figure 9. Illustration of a Karp and Miller schema for the program of Table 2.

Notice that this definition requires that $\mathcal{S}_1$ and $\mathcal{S}_2$ have identical variables and operators, only the control may be different. Also, if a schema is determinate the set of schema histories has exactly one member.

Another property of schemata is the boundedness of the operator queues.

Definition 12:

If

$$\sum_{a \in A} [\text{ length of } \mu(a)] \leq K$$

for some integer K, at every stage in the execution of a schema, the schema is said to be bounded. If K = 1, the schema is serial.

In order to specify the class of schemata which is determinate, restrictions on schemata are introduced. In this discussion σ and π represent arbitrary symbols in the control alphabet Σ.

Restriction 1: (persistence)

If $\tau(q,\sigma)$ and $\tau(q,\pi)$ are defined, then $\tau(q,\sigma\pi)$ and $\tau(q,\pi\sigma)$ must be defined.

The persistence restriction requires that once an operator is ready to be initiated, it must remain ready to be initiated.

Restriction 2: (commutativity)

If $\tau(q,\sigma\pi)$ and $\tau(q,\pi\sigma)$ are defined, then $\tau(q,\sigma\pi) = \tau(q,\pi\sigma)$.

The effect of commutativity on the control transition diagram is illustrated in Fig. 10.

Restriction 3: (lossless)

The output set O_a of every operator a must be nonempty ($O_a \neq \phi$).

Let us define a next-state function . which is a function of the present state α and one of the control alphabet symbols defined for the present control state. We write

$\alpha \cdot \bar{a}$ next state entered after operator a is enabled.

$\alpha \cdot a_i$ next state entered after operator a terminates ($1 \leq i \leq K(a)$).

Restriction 4:

If $\alpha \cdot \sigma\pi$ and $\alpha \cdot \pi\sigma$ are defined, then $\alpha \cdot \sigma\pi = \alpha \cdot \pi\sigma$.

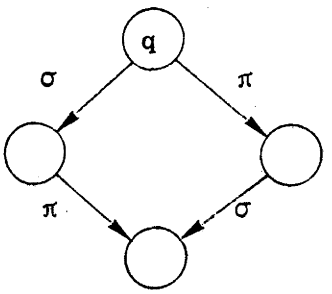


Figure 10. **Effect of commutativity on the control transition diagram.**

Given these restrictions, it is possible to prove the following theorem.

Theorem 4:

Every Karp and Miller schema that satisfies Restrictions 1 - 4 is determinate in the sense of Definition 5.

The schema illustrated in Fig. 8 is a determinate schema. The schema of Fig. 9 is also determinate but it is not lossless (Restriction 3) since $O_e = \phi$. Therefore, Theorem 4 cannot be applied to this example.

The precise statement of the Karp and Miller theorem is slightly different. They prove that a persistent, commutative, and lossless schema is determinate if and only if Restriction 4 holds for every interpretation.

The following two restrictions are useful in establishing further properties of schemata.

Restriction 5: (repetition-free)

If an operator a is executed twice, each variable in its input set must appear in the output set of an operator that is executed between the two executions of operator a.

Restriction 6: (finite-state)

The number of control states in Q is finite.

The following two theorems summarize some of the decidability results for the Karp and Miller model.

Theorem 5:

It is decidable whether

1. A finite-state schema is repetition-free.
2. A finite-state, repetition-free schema is bounded (serial).
3. A given operator a in a finite-state, repetition-free schema is performed a finite number of times in each computation.
4. A persistent, commutative, lossless, repetition-free, finite-state schema is determinate in the sense of Definition 5.

Theorem 6:

It is undecidable whether

1. Two persistent, finite-state schemata are equivalent.
2. Two serial, finite-state schemata are equivalent.

CONCLUSIONS

Richard Hamming has said "the purpose of computing is insight " [7]. We might paraphrase this statement in the following way "the purpose of theory is insight". In this paper, we have attempted to bring together some of the work on the theory of parallel computing with the hope of furthering the insight derived. One general conclusion is that in all these models, by determinate operation, it is either implicit or explicitly required that an operator which is enabled and ready to be executed must not be disabled by the execution of some other operator. We understand that Slutz [38] has been able to weaken this restriction to allow an operator to be disabled if it must eventually be re-enabled. We do not yet understand the details of his result.

In our study of solutions to the mutual exclusion problem [5], we have found examples of systems which were not determinate but which do operate correctly in the sense that the mutual exclusion problem can be solved. This suggests the need for investigation of models which are correct but not necessarily determinate. The work of Ashcroft and Manna [3] is relevant here.

One difficulty with these models, at least with respect to their application in the study of computer systems, is their inadequacy in describing how one operator can prevent another operator, which is being executed at the same time from producing any results. Such an "interrupt" capability exists in most systems and is desirable to prevent time being wasted on the execution of operators when their results are known to be meaningless. For example, a divide by zero should cause the execution

of all operators used in the computation of an arithmetic expression to be terminated.

ACKNOWLEDGEMENTS

The author would like to extend particular thanks to Professor Edward J. McCluskey for his interest in this area, which provided the motivation for this work, and for his many valuable comments. Thanks are also due the authors of the papers surveyed for their time and patience in explaining their results. Any inaccuracies presented are the result of my inability to communicate their intentions. The comments and advice of E. S. Davidson, V. R. Lesser, Z. Manna, and H. S. Stone are also appreciated.

REFERENCES

[1] Adams, D.A. A computation model with data flow sequencing. CS-117, Computer Science Department, Stanford University, Stanford, California. (Dec 1968').

[2] Adams, D.A. A model for parallel computations. Proc. Symp. on Parallel Processor Systems, Technologies, and Applications, Naval Postgraduate School, Monterey, California (in press .

[3] Ashcroft, E. and Manna, Z. Formalization of properties of parallel programs. AIM-110, Artificial Intelligence Project, Stanford University, Stanford, California (Feb 1970).

[4] Baer, J.L. Graph models of computations in computer systems. Report 68-46, Department of Engineering, UCLA, Los Angeles, California. (Oct 1968).

[5] Bredt, T.H. and McCluskey, E.J. A model for parallel computer systems. Technical Report No. 5, Digital Systems Laboratory, Stanford Electronics Laboratories, Stanford University, Stanford, California (Apr 1970)

[5a] Dijkstra, E.W. The structure of the "THE" - multiprogramming system. C. ACM 11, 5 (May 1968), 340-356.

[6] Floyd, R.W. Assigning meanings to programs. Proc. of Symposium on Applied Mathematics, Vol. 19, American Mathematical Society (1967), 19-32

[7] Hamming, R.W. Numerical Methods for Scientists and Engineers. McGraw-Hill, New York (1962).

[8] Holt, A.W. Mem-theory. Technical Documentary Rept. 1, Applied Data Research, Inc., (Aug 1965).

[9] Hold, A.W. Final report. RADC-TR-68-305, Rome Air Development Center, Griffiss Air Force Base, N. Y. (Sept 1968).

[10] Huffman, D.A. The synthesis of sequential switching circuits. in Sequential Machines: Selected Papers, E.F. Moore. (ed.), Addison-Wesley Publishing Co., Inc. (1964), 3-62.

[11] Ianov, I.I. On the equivalence and transformation of program schemata. Doklady, A.N., USSR 113 (1957), 39-42.
Translation: Comm. ACM, 1, 10 (Oct 1958), 3-62.

[12] Ianov, I.I. The logical schemes of algorithms. In Problems of Cybernetics, 1 (1958), 75-127.

[13] Karp, R.M. and Miller, R.E. Properties of a model for parallel computations: determinacy, terminations, queueing. SIAM J. Appl. Math., 14 (Nov 1966), 1390-1411.

[14] Karp, R.M. and Miller, R.E. Parallel program schemata: a mathematical model for parallel computation. IEEE Conference Record of the 8th Annual Symposium on Switching and Automata Theory (Oct 1967), 55-61.

[15] Karp, R.M. and Miller, R.E. Parallel program schemata. J. of Computer and System Sciences 3, 2 (May 1969), 147-195.

[16] Luconi, F.L. Completely functional asynchronous computational structures. IEEE Conference Record of the 8th Annual Symposium on Switching and Automata Theory (Oct 1967), 62-70.

[17] Luconi, F.L. Asychronous computational structures. MAC-TR-49 (Thesis), Massachusetts Institute of Technology, Cambridge, Massachusetts (Feb 1968).

[18] Luconi, F.L. Output functional computational structures. IEEE Conference Record of the 9th Annual Symposium on Switching and Automata Theory (Oct 1968), 76-84.

[19] Manna, Z. Termination of algorithms. Computer Science Department, Carnegie-Mellon University, Pittsburgh, Pennsylvania (Apr 1968).

[20] Manna, Z. Properties of programs and the first-order predicate calculus. J. ACM (Apr 1969).

[21] Manna, Z. The correctness of programs. J. of Computer and System Sciences, 3 (May 1969).

[22] Manna, Z. The correctness of non-deterministic programs. Artificial Intelligence J. 1, 1 (1970).

[23] Martin, D. The automatic assignment and sequencing of computations and systems. UCLA Report No. 66-4 (Jan 1966).

[24] Martin, D. and Estrin, G. Experiments on models of computations and systems. IEEE Transactions on Electronic Computers, EC-16 (Feb 1967), 59-69.

[25] Martin, D. and Estrin, G. Models of computational systems - cyclic to acyclic graph transformations. IEEE Transactions on Electronic Computers, EC-16 (Feb 1967).

[26] Martin, D. and Estrin, G. Models of computations and systems - evaluation of vertex probabilities in graph models of computations. J. ACM, 14, 2 (Apr 1967), 281-299.

[27] Martin, D.F. and Estrin, G. Path length computations on graph models of computations. IEEE Trans. on computers, C-18, 6 (June 1969), 530-536.

[28] McNaughton, R. Badly timed elements and well timed nets. Technical Report No. 65-02, Moore School of Electrical Engineering, University of Pennsylvania (June 1964).

[29] Mendelson, E. Introduction to Mathematical Logic. Van Nostrand Co., Princeton, N.J. (1964).

[30] Miller, R.E. Switching Theory, Vol 11: Sequential Circuits and Machines. John Wiley and Sons, Inc., New York, N.Y. (1965).

[31] Muller, D.E. and Bartky, W.S. A theory of asynchronous circuits. Proc. of an International Symposium on the Theory of Switching, The Annals of the Computation Laboratory of Harvard University, Vol. 29, Part I, Harvard University Press (1959), 204-243.
See Chapter 10 of Miller [30] for a summary.

[31a] Patil, S. Coordination nets. Ph.D. Thesis, Project MAC, Massachusetts Institute of Technology, Cambridge, Mass. (to appear).

[32] Petri, , C.A. Kommunikation mit automaten. Shriften des Reinish-West Falischen Inst. Instrumentelle Math., and der Universitat Bonn, Nr. 2, Bonn (1962)
Translations:
MIT Memorandum MAC-M-212, Project MAC
Applied Data Research

[33] Petri, C.A. Fundamentals of a theory of asynchronous information flow. Proc. IFIP, Munich, Germany (1962), 166-168.

[34] Rodriquez, J.E. A graph model for parallel computations. Ph.D. Thesis, MIT, Department of Electrical Engineering, Cambridge, Massachusetts (Sept 1967).

[35] Rutledge, J.D. On Ianov's program schemata. J. ACM (Jan 1964), 1-9.

[36] Rutledge, J.D. Parallel processes, schemata, and transformations. Draft for NATO conference on computer architecture (Sept 1969).

[37] Shapiro, R.M. and Saint, H. The representation of algorithms. RADC-TR-69-313, Vol. II, Final Technical Report, Rome Air Development Center, New York (Sept 1969).

[38] Slutz, D.R. The flow graph schemata model of parallel computation. MAC-TR-51 (Thesis), MIT, Cambridge, Massachusetts (Sept 1968).

[39] Van Horn, E.C. Computer design for asynchronously reproducible multiprocessing. MAC-TR-34 (Thesis), MIT, Cambridge, Massachusetts (Nov 1966).