STRUCTURAL PROPERTIES OF PROCESSES

Dissertation for the Degree of Ph. D.
MICHIGAN STATE UNIVERSITY
JOHN CHRISTIAN HANSEN
1974

This is to certify that the

thesis entitled

STRUCTURAL PROPERTIES OF PROCESSES

presented by

John Christian Hansen

has been accepted towards fulfillment
of the requirements for

Ph.D. degree in Computer Science

M. A. Rahimi

Major professor

Date May 7, 1974

O-7639

ABSTRACT

STRUCTURAL PROPERTIES OF PROCESSES

By

John Christian Hansen

In this thesis, a definition of process, which is both mathematically precise and practically significant, is proposed. Using this definition, a theory of process decomposition, which is similar to that which Hartmanis and Stearns (HART 66) develop for sequential machines, is proposed. A necessary and sufficient condition for the existance of such decompositions is shown.

A parallel decomposition theory more akin to that which might be useful in parallel processing is also discussed. It is shown that there is no algorithm which will detect all cases of this kind of parallelism. However, sufficient conditions for the existance of this kind of decomposition are given.

This thesis also points out how digraph theory serves as a productive tool in the analysis of processes. A special digraph is associated with each process. This digraph is used in the derivation of sufficient conditions for process termination. It also provides the basis for the determination of both computational cost measures and process cost measures. These measures provide the motivation for the development of a general theory of digraph measurement. The

general theory is shown to have some well-known measures as special cases, as well as being the foundation for the development of several new useful measures.

STRUCTURAL PROPERTIES OF PROCESSES

By

John Christian Hansen

A DISSERTATION

Submitted to

Michigan State University

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

Department of Computer Science

TO

MY PARENTS

ACKNOWLEDGMENTS

I am grateful to Dr. Merteza A. Rahimi, my thesis advisor, for his encouragement during the course of this thesis. My thanks also go to Dr. Harry Hedges, Dr. Lewis Greenberg, Dr. Carl V. Page, Dr. Edgar M. Palmer, and Dr. Edward A. Nordhaus for serving on my guidance committee and for their critical review of this work. My special thanks go to my typist, Debbie Claflin, for her speed and accuracy in typing the final draft.

Finally, I am deeply indebted to my wife, Elizabeth, for her critical review of this work which improved the exposition by at least an order of magnitude.

TABLE OF CONTENTS

Chapter 1

INTRODUCTION AND LITERATURE REVIEW

The topics covered in this thesis fall into four major categories; a precise definition of process and the resultant theory, associating digraphs with processes, the concept of measurement in a process, and a generalization of these results to measures on digraphs in general.

Central to the development of computer science theory is the concept of process. P. Brinch Hansen (BRIN 70),E.W. Dijkstra (KIJK 68), J.H. Saltzer (SALT 66) and others have successfully used the concept of process as a basic unit in their descriptions of complex computer systems. Zohar Manna (MANN 70), C.V. Ramamoorthy and M.J. Gonzales (GONZ 70), Leslie Lamport (LAMP 74), M. Lenman (LENM 68), J.B. Dennis (DENN 66), D. Knuth (KNUT 66), and others have successfully used the concept of process as a basic unit from a programming languages standpoint.

A survey of the literature shows that there is a lack of any agreement on a single definition of process. In (DENN 66), Dennis and Van Horn state that:

> ...a process is that abstract entity which moves through the instructions of a procedure as the procedure is executed by a processor.

In (GILB 72), Gilbert and Chandler view the system of processes as follows:

> ...an individual process is approximated by an abstract process which consists of distinct portions or 'states'. A set of such states, together with a set of values of data variables, then approximates of configuration (or composite state of an entire system or process.
>
> The 'moves of individual processes from one state to the next are written as abstract partial rules. Partial rules for the different individual processes may then be combined to yield transition rules - moves from one composite state to a next - for the entire system of processes.

In (DAHL 66), Dahl and Hygaard in describing SIMULA, an ALGOL based simulation language, take the following view of process:

> In general a process has two aspects: it is a data carrier and it will execute actions.

In(DENN 66), a process is considered to be the status of a system, in (GILB 72) a sequence of such status values, and in (DAHL 66) as a generator of such sequences.

While Gilbert and Chandler (GILB 72) also consider "transition rules", it was Horning and Randell (HORN 73) who first combined all these notions in a single definition of process. Horning and Randell define a process as a triple (S,f,s) where S is a state space, f is an action function in that space, and siis a subset of S which defines the initial states of the process. By action function, they mean a relation on the Cartesian product of S and the set of all

possible actions. By actions, they mean assignments to variables in the state space.

While the definition of Horning and Randell is mathematically precise, it appears to be too general. It suffers from the fact that f may not even be computable and from its lack of relation to common process specifications (i.e. - algorithms, flowcharts, and programming languages). In this thesis, a definition like that of Horning and Randell isuused but f is replaced by a flowchart-like diagram similar to those disucssed by I. Nassi and B. Schneiderman (NASS 73), It is hoped that the resultant definition has both the mathematical precision and the practical significance to become a useful tool.

Once a definition of process is agreed upon, the concept pf parallel processing may be discussed. It is helpful to distinguish between two types of parallel processing; multiprocessing and multiprogramming. Multiprocessing is a simultaneous sharing of two or more portions of the same program by two or more processing units. Multiprogramming is the time and resource sharing of a computer system by two or more programs which reside simultaneously in primary memory.

There are two basic ways one may handle multiprocessing of a program; new programming language concepts may be introduced or parallelism may be automatically detected at the compiler level. AlGOL-68 (VANN 69) has incorporated parallelism concepts in its definitions. At the statement level, this is done by replacing ';' with ',' and at the procedure level by using a par symbol. At the assembly level, Dennis and Van Horn in (DENN 66) suggest the FORK-JOIN-QUIT combination:

> The basic primitive operation of parallel programming is implemented by the meta-instruction FORK W ; ... where W is a word name. A FORK meta-instruction indicates a new process at the instruction labeled W. The newly created branch process is part of the same computation as its creator or main process...
>
> A process that has completed a sequence of procedure steps is terminated by the meta-instruction QUIT after which the process no longer exists...We use the instruction JOIN T,W; which is essentially Conway's join instruction. Here T is the word name of the count to be decremented and W is the word name of an instruction word to be executed if the count becomes zero.

A number of attempts have already been made to automatically detect parallelism. Summaries of attempts to detect parallelism at the statement level may be found in (BAER 68). Bernstein (BERN 66) was the first to attack the problem of detecting parallelism at the interstatement level. His model may be simply stated as follows; suppose there are two statements in a programming language, P1 and P2, which were originally scheduled in sequence. The conditions under which P1 and P2 may be executed in parallel are as follows:

1) $I_1 \cap O_2 = \emptyset$

2) $I_2 \cap O_1 = \emptyset$

3) $O_1 \cap O_2 = \emptyset$

where I_i and O_1 represent, respectively, the input (those variables which appear only at the right of an assignment statement) and the output data sets of P_i.

Based on these conditions, systems that automatically detect parallelism have been written. Examples may be found in

(RUSS 69), (VOLA 70), (RAMA 69), and (BING 67). Detection of parallelism in DO loops is not quite as simple. Such analysis is carried out with some success by Bernstein (BERN 66), Russel (RUSS 69), and by Lamport (LAMP 74). The decomposition theory in this thesis suggests a method for automatic detection of parallelism. The concept of diagram as introduced in this thesis is closely related to a parallel programming language.

Normally, measurement and process are associated with computational complexity. Historically, the question of computational complexity grew from the origin of computability. One way to define computability is to say that a function is computable if it is possible to obtain it from a finite number of operations of composition, primitive recursion or application of the μ-operator to regular functions starting with the functions:

1) $S(x) = x + 1$

2) $N(x) = 0$

3) $U_i^n (x_1,\ldots,x_n) = x_i \qquad (1 \leq i \leq n)$.

Primitive recursion may be represented by the conditional expression notation, first introduced by McCarthy (MCCA 60). Conditional expressions will be written in the form:

$$(B_1 \rightarrow e_1,\ b_2 \rightarrow e_2, \ldots b_{n-1} \rightarrow e_{n-1},\ e_n)$$

where b_i denotes a Boolean expression. The value of the conditional expression is obtained by examining b_i's in turn from the left until one is found which is true; the value for the expression is e_i if b_i is true. If no true b_i's are found, the value for the expression is e_n. In this notation, the definition of the factorial function might be written:

$fac(n) = ((n+0) \rightarrow 1,\ n \cdot fac(n-1))$.

To put the preceding into historical perspective, Church (CHUR 36) in 1936 expressed the opinion that the concepts of λ-definable functions and recursive functions are both identifiable with the concept of computable function (Church's thesis). In the same paper, he showed that the decision problem for the predicate calculus is unsolvable. The same was proved by A.M. Turing (TURI 37) at about the same time. Turing introduced the concept of what is now known as Turing machine for his proof.

Once the concept of computability has been defined, then the question of how to characterize the complexity of computable functions arises. The most familiar ways of doing this are; by the number of steps needed to compute a function (HART 64, HART 65) and by the amount of memory needed for a computation (HART 65 b). A machine independent theory of the complexity of recursive functions is presented in (BLUM 67).

It should be noted that all results in complexity theory deal with classes of functions rather than measures for individual functions. Thus for example, the theory of Turing machine complexity classes may shed some light on the properties of languages that are suitable as programming languages (i.e. - languages that can be simply recognized) but it is of little use in extracting information about a particular language (other than the class to which it belongs).

Several interesting observations have been made about complexity classes. Every set accepted by a time or tape-bounded Turing machine is a recursive set. Furthermore, every recursive set is accepted by some tape-bounded Turing machine and also by some time-bounded Turing machine. Since no complexity class can contain all

recursive sets, there must be an infinite hierarchy of complexity classes. Lewis in (LEWI 71) makes the following observation:

> Quite often when pathological problems exist in complexity hierarchies, it has been shown that they exist only in the lower levels of the hierarchy...Conditions that occur in all but a finite number of places are accepted in automata theory as being desirable in most cases...
>
> But, in complexity hierarchies, the functions which are easiest to compute, and that are computed most often, occur at the bottom. These very functions are the ones computed in "real-life" and therefore are quite important.

In this thesis, measures involving processes are for the most part concerned with individual processes rather than classes of processes, hence computational complexity does not play a central role. However, when some of the results are generalized to results involving digraphs, a theory which has some parallel with complexity theory is developed. Rather than ordering the computable functions into complexity hierarchies, a theory for ordering members within a class of digraphs is developed.

Digraph theory serves as a productive tool in a number of areas in computer science. This stems from the fact that digraph theory deals with the structural properties of empirical systems. Such theory is bound to find applications in a discipline so full of structure as computer science.

Harary, Norman, and Cartwright (HARA 65) define a digraph as a net with no loops or parallel lines. A net has four primitive (undefined terms) and two axioms. The four primitives of nets (also of digraphs) are:

P_1 : a set V of elements called vertices.

P_2 : a set X of elements called lines.

P_3 : a function f whose domain is X and whose range is contained in V.

P_4 : a function s whose domain is X and whose range is contained in V.

The axioms for a net are:

A_1 : the set V is finite and not empty.

A_2 : the set X is finite.

Primitives P_3 and P_4 relate the lines to the points. They give the first and second point of each line.

In this thesis, a less restricted definition of digraph shall be used. A digraph will be a structure which satisfies P_1 through P_4 and the following axioms:

A_1 : the set V is not empty.

A_2 : there are no parallel lines (Two lines, x_1 and x_2, are parallel if $F(x_1) = f(x_2)$ and $s(x_1) = s(x_2)$)

Thus, a digraph may have an infinite number of points or lines and may contain loops. Digraphs will be used to determine sufficient conditions for the termination of processes. Also, a framework for measuring certain graph-theoretic properties will be developed.

Chapter 2

PROCESSES

In this chapter, the concept of process is introduced. Section One deals with the basic definitions and descriptive theorems. In Section Two, a parallel decomposition theory similar to that of Hartmanis and Stearns (HART 66) is developed. In section Three, the problem of nontrivial parallelism is addressed. It is shown that the best results obtainable are sufficient conditions for decomposition.

2.1 DEFINITIONS

As stated in Chapter One, the approach here is strongly influenced by the definition of process given in Horning and Randel (HORN 73).

Definition 2.1. A *state variable* is an elementary quantity which can assume certain well-defined values.

Definition 2.2. A set of labeled state variables constitutes a *state variable set*.

Definition 2.3. An assignment of values to all variables in a state variable set defines a state of the set.

Definition 2.4. The set of possible states for a given state variable set is the state space of that set.

Example 2.1. Consider the state variable set $V = \{a,b\}$ consisting of two variables labeled a and b whose values may be any natural number. If a is assigned the value 5 and b the value 7, this defines the state (a=5,b=7). The state space of this variable set is $S(V) = \{(a=m,b=n) \mid m>0,\ n>0\}$.

Example 2.2. The state variable set of a typical C.P.U. might include all registers and all memory locations. The state space for this set would be all possible combinations of values of these variables.

Definition 2.5. A computation in a state space is a sequence of states from that space.

Definition 2.6. The first element of a computation is called its initial state.

Definition 2.7. The last element of a finite computation is called its final state.

Example 2.3. Consider the state space of Example 1, the sequence $C_1 = \langle (a=2,b=1),(a=2,b=2),(a=2,b=3),(a=2,b=4) \rangle$ is a

finite computation for which the initial state is (a=2,b=1) and the final state is (a=2,b=4). An example of an infinite computation in the same state space is the sequence $C_2 = \langle (a=2,b=n) \quad n=1,2,3... \rangle$. Its initial state is (2,1) but it has no final state.

The concept of computation is essential to the definition of process. There are, of course, many ways in which a particular computation may be specified. The form in which most computations will be presented in this thesis is in terms of the transitions which occur between states.

Definition 2.8. An action in a state space is a finite set of assignments of values to some of the variables of its state variable set.

Definition 2.9. If a state is followed by an action, then the immediate successor of the state is the new state whose variables all have their old values except those which have new values assigned by the action.

Example 2.4. If (x=3,y=5) is followed by the action $\{y \leftarrow 4\}$, its immediate successor is (x=3,y=4).

Definition 2.10. The null action is the action which specifies no assignments.

In this thesis, actions will be represented in pictorial form by action boxes. The method used is an adaptation of a system of flowcharting developed by I. Nassi and B. Shneiderman (NASS 73). Their system was an alternative to conventional flowchart languages and designed to be more amenable to structured programs.

An action whose cardinality is one, such as the action of Example 2.4. would be represented like this:

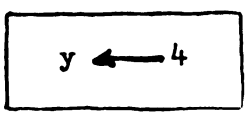


Fig 2.1

Actions whose cardinality is greater than one will be denoted by:

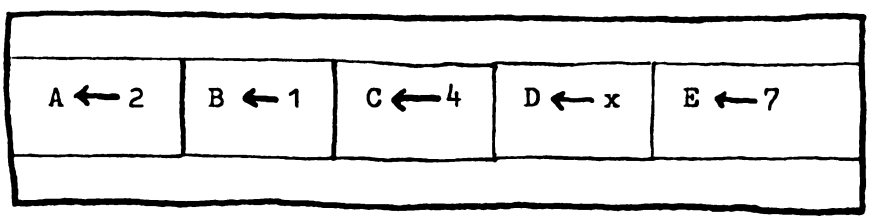


Fig 2.2

The assignments must not have variables left of an arrow in common. For example, an action may not contain both X ← Y + Z and X ← 7. All expressions right of an arrow are calculated and all assignments of an action are made simultaneously. Any reference to a state variable right of an arrow refers to its value.

If one of two basic digrams are to be performed, depending upon the value of some boolean expression, the If clause may be used:

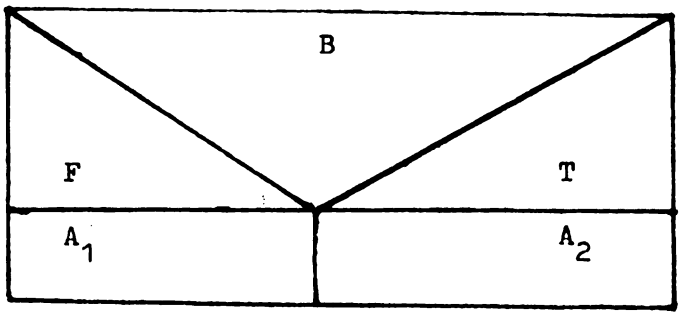


Fig 2.3

The central triangle contains a Boolean expression, the left and right triangles contains a T or an F to represent the possible outcomes and A_1 and A_2 are basic diagrams.

Example 2.5. If the state (a=1,b=2,c=3) is followed by:

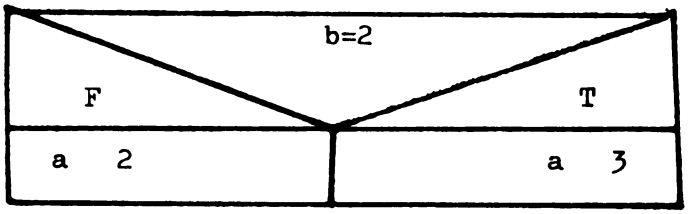


Fig 2.4

the result is the state (a=3,b=2,c=3).

To allow for iteration, the DO while clause may be used:

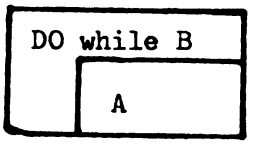


Fig 2.5

A is a basic diagram and B is a Boolean expression. The actions in the basic diagram are performed while B is true. A <u>basic diagram</u> consists of any of the symbols introduced so far, including the DO while clause, stacked one upon the other.

Before formally defining process, a few examples are in order.

<u>Example 2.6.</u> This example involves matrix multiplication. A and B are N×N matrices and the product C is an N×N matrix.

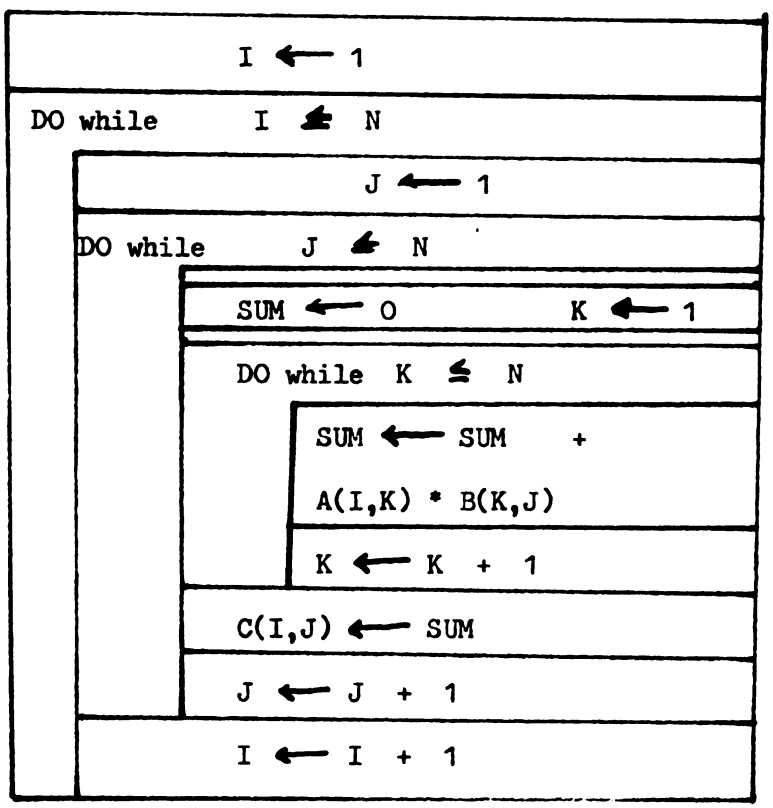


Fig 2.6

Example 2.7. This example shows how the concept of process may be used to model a continuous phenomenon.

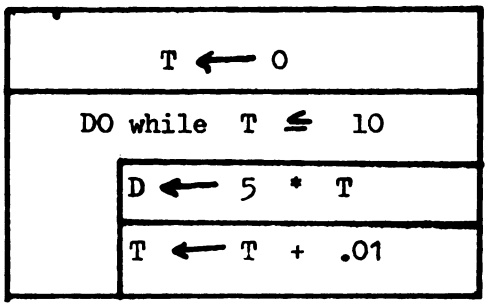


Fig 2.7

The state variables are D and T. T may be thought of as time and D as some instantaneous object whose 'snapshot' is being taken from T=0 through T=10 in increments of .01.

The formal definition of process in now given.

Definition 2.11. A process is a triple (S,d,I) where S is a state space, I is a subset of S, and d is a basic diagram.

Definition 2.12. The final (initial) states of a process are the final (initial) states of all of its computations.

Definition 2.13. A diagram is made up of basic diagrams stacked in parallel as follows:

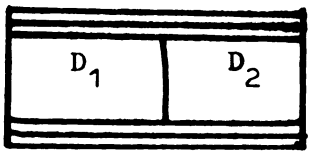


Fig 2.8

A diagram must be finite and the D_i must have no state variables in common.

Definition 2.14. A combined process is a triple (S,d,I) where S is a state space, I is a subset of S, and d is a diagram.

Given any basic diagram, labels may be associated with each action box. These numbers will be useful in proving a number of results about processes.

Definition 2.15. The label of an action is its position in its basic diagram (only actions are counted). Since each basic diagram has only a finite number of positions, it is possible to assign each when a number which will be thought of as its label.

Example 2.8. In the figure, there are 5 labels. The actions in the body of the DO while clause are counted but not the clause itself. The label of the action $G \leftarrow X + 3$ is three.

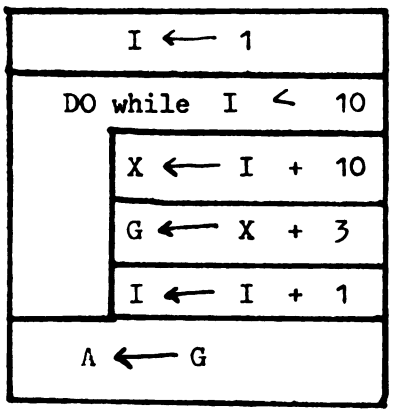


Fig 2.9

Definition 2.16. For any process (S,d,I) let I_i be the set of all states which can immediately precede actions labeled i. (If a process has n labels, I_{n+1} shall be used to denote its final states for sake of completeness.)

Theorem 2.1. It is possible to translate a process into a recursive function.

Proof

Each level n in the basic diagram of a process may be thought of as a function whose domain is I_n.

Suppose the basic diagram of the process is:

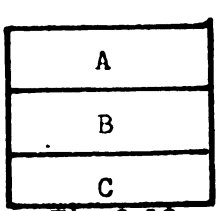


Fig 2.10

where A,B,C are actions. With each action can be associated a function a, b, c.

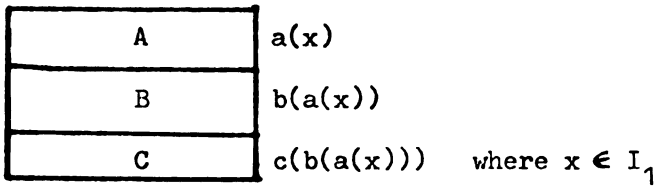


Fig 2.11

If clauses are handled in the following manner:

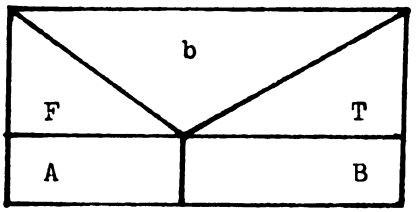


Fig 2.12

Let b(x) be the function associated with B, and a(x) the function associated with A. The function f associated with the If clause is f(x) = (b→b(x),a(x)).

Do while clauses are handled in the following manner:

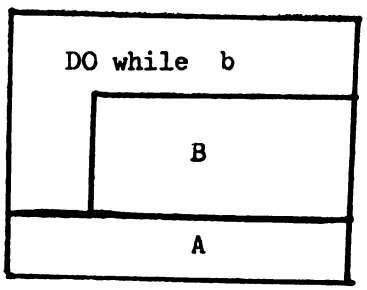


Fig 2.13

Suppose the basic diagram B has function e(x) and the next basic diagram A has function a(x), then the function for the entire basic diagram f(x) will be: $f(x) = s(e(x))$ where $s(x) = (b \rightarrow f(x), a(x))$. If there is no basic diagram following the DO while clause then $f(x) = s(e(x))$ where $s(x) = (b \rightarrow f(x),$ undefined).
This completes the proof.

Corollary. It is possible to translate a combined process into a recursive function.

Proof

Since the D_i have no state variables in common, partition S accordingly and construct the recursive function, f_i, for each separate process.

2.2. DETECTION OF PARALLELISM

In this section, a decomposition theory, along the lines of that of Hartmanis and Stearns (HART 66) for sequential machines, is developed for processes. A necessary and sufficient condition for this type of parallelism is presented.

Definition 2.17. The parallel connection of two processes $P_1 = (S_1, d_1, I_1)$ and $P_2 = (S_2, d_2, I_2)$ is the combined process $P = P_1 \| P_2 = (S_1 \times S_2, d, I_1 \times I_2)$ where d is the following diagram:

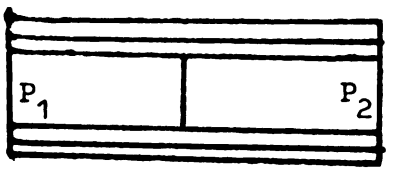


Fig 2.14

and $S_1 \cap S_2 = \emptyset$.

In order to study the parallel decomposition of processes, P = (S,d,I), it is necessary to associate with the process a successor function and with S N the concept of partition. (Where N is the set of labels of d).

Definition 2.18. The successor function of a process P=(s,d,I) is a function from S X N into S X N such that

$$f((x,n)) = \begin{cases} (y,e) \text{ if } x \{\text{the } n^{th} \text{ level action}\} = y \\ \text{and } x \in I_n \text{ where e is the next label.} \\ \text{undefined otherwise} \end{cases}$$

where x and y are in S, N is the set of levels of P, and n is in N. (For convenience, N will always contain one more label which will be associated with final states.)

Definition 2.19. A partition π on the Cartesian produce of the state space and the labels of a process (S,d,I) is said to have the substitution property iff $r \equiv t(\pi)$ implies $f(r) \equiv f(t)(\pi)$ for all r and t in S X N (where f is the successor function).

Definition 2.20. Let π be a partition with the substitution property on the Cartesian product of the state space and the labels of a process $P = (S,d,I)$. The π-image of P is the process: $P_\pi = (\pi, d_\pi, I_\pi)$ where $f_\pi(B_\pi) = B'_\pi$ iff there exists x in B_π such that $f(x) \subseteq B'_\pi$ (f_π is the successor function of P_π.)

The last definition is correct because π has the substitution property iff f maps blocks of π into blocks of π. (That is to say for B_π in π there exists a unique B'_π in π such that $f(B_\pi) \subseteq B'_\pi$.) P_π may be thought of as a process which does only part of the computation performed by P, since it only keeps track of which block of π contains a particular state of S.

Example 2.9. Consider the following process $P = (S,d,I)$ where S is the set of natural numbers, $f(x,1) = x+1$, $I = \{1\}$, and d is:

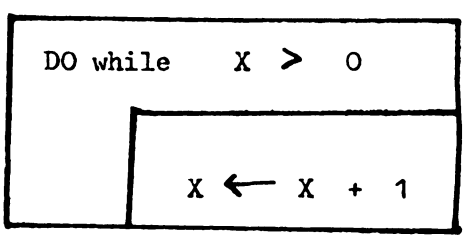


Fig 2.15

suppose $\pi = \{(B_0,1),(B_1,1)\}$ such that (x,1) is in $(B_0,1)$ if

x is even and (x,1) is in B_1 if x is odd. Clearly, π has S.P. (Take any two odd numbers x and y. f(x,1) is even and f(y,1) is even so $f(x,1) \equiv f(y,1)(\pi)$. The case for which both x and y are even follows in the same manner.) $P_\pi = (\pi, d_\pi, \{B_1\})$ where $f_\pi(B_0) = B_1$ and $f_\pi(B_1) = B_0$.

Both P and P_π produce a single infinite computation. P_π may be thought of as a process which only computes whether or not the number is odd or even, while P also computes the value of the number. This can also be thought of in terms of ignorance of a state of P. If π has S.P. on P, by knowing P_π, the block of π which contains a state of P is known, given a block of π, the block of π to which the successor function will lead can be computed. If a partition π does not have S.P., then this computation is not possible. So S.P. partitions define a sort of uncertainty about the state of P which does not spread as the machine operates. (e.g. -in the above example, it is known if the number is odd or even at any point in time.)

<u>Theorem 2.2.</u> If π_1 and π_2 are S.P. partitions onaa process P = (S,d,I) then so are the partitions $\pi_1 \cdot \pi_2$ $\pi_1 + \pi_2$

<u>Proof</u>

Suppose $r \equiv t(\pi_1)$ and $r \equiv t(\pi_2)$, then by definition of $\pi_1 \cdot \pi_2$

$$r \equiv t(\pi_1 \cdot \pi_2)$$

Now by the hypothesis of the theorem:

$$f(r) \equiv f(t)\ (\pi_1) \qquad \text{and}$$

$$f(r) \equiv f(t)\ (\pi_2)$$

but this implies $f(r) \equiv f(t)\ (\pi_1 \cdot \pi_2)$ by the definition of $\pi_1 \cdot \pi_2$ hence $\pi_1 \cdot \pi_2$ is an S.P. partition on P. To show $\pi_1 + \pi_2$ has S.P., note that $r \equiv t(\pi_1 + \pi_2)$ implies that there exists a chain $r = r_0, r_1, \ldots, r_m = t$ such that $r_j \equiv r_{j+1}(\pi_1)$ or $r_j \equiv r_{j+1}(\pi_2)$ where $j = 0,1,\ldots,m-1$.
This shall be used to show

$$f(r) \equiv f(t)\ (\pi_1 + \pi_2).$$

Since π_1 and π_2 have S.P., $f(r) \equiv f(r_1)(\pi_1)$ or

$$f(r_1) \equiv f(r)(\pi_2).$$

Since both π_1 and π_2 are finer than $\pi_1 = \pi_2$, it is evident that $f(r) \equiv f(r_1)\ (\pi_1 + \pi_2)$. In like manner:

$$f(r_1) \equiv f(r_2)\ (\pi_1 + \pi_2)$$

$$\vdots$$

$$f(r_{m-1}) \equiv f(t)\ (\pi_1 + \pi_2)$$

hence, $f(r) \equiv f(t)\ (\pi_1 + \pi_2)$

hence, $\pi_1 + \pi_2$ has S.P.

Theorem 2.3. The set of all S.P. partitions on the Cartesian product of the state space and the labels of a process P = (S,d,I) forms a lattice L_P, under the natural partition ordering. Furthermore, L_P contains the trivial partitions 0 and 1.

Proof

From the last theorem, the set of all S.P. partitions is closed under $\cdot$ and $+$, thus it forms a sublattice of the lattice of all partitions on S X N and hence is a lattice in the natural ordering

of partitions. The last statement is trivial.

Definition 2.21. If P_1 and P_2 are two processes, then the function α is said to be an assignment of P_1 into P_2 if:

i) α is a one to one mapping of $S_1 \times L_1$ into $S_2 \times L_2$ and

ii) $f_2(\alpha(x)) = \alpha(f_1(x))$ for all x in $S_1 \times L_1$.

Definition 2.22. Process P_2 is said to realize the state behavior of process P_1 iff there exists an assignment of P_2 into P_1.

Theorem 2.4. A process P.has a parallel decomposition of its state behavior iff there exist two S.P. partitions π_1 and π_2 on P such that $\pi_1 \cdot \pi_2 = 0$.

Proof

Let the state behavior of P be realized by $P_1 \| P_2$. Let α be the assignment such that $\alpha : S \times L \rightarrow (S_1 \times K_1) \times (S_2 \times L_2)$. (L, L_1, and L_2 are the labels of P, P_1, and P_2 respectively.) α defines two equivalence relations π_1 and π_2 on $S \times L$ as follows: $r \equiv t(\pi_1)$ iff $r_1 = t_1$ where $\alpha(r) = (r_1, r_2)$ and $\alpha(t) = (t_1, t_2)$

$r \equiv t(\pi_2)$ iff $r_2 = t_2$ where $\alpha(r) = (r_1, r_2)$ and $\alpha(t) = (t_1, t_2)$.

Since α is one-to-one:

$s \equiv t(\pi_1 \cdot \pi_2)$ implies $s = t$ and hence, $\pi_1 \cdot \pi_2 = 0$.

To see that π_1 and π_2 have S.P., note that if $r \equiv t(\pi_1)$, then $\alpha(r) = (r_1, r_2)$ and $\alpha(t) = (r_1, t_2)$ but this implies

that the first components of the next states are identical hence,

$$f(r) \cong f(t) \ (\pi_1).$$

The same argument shows π_2 also has S.P.

To show the converse, assume that π_1 and π_2 are nontrivial S.P. partitions on P such that $\pi_1 \cdot \pi_2 = 0$. To construct P_1 and P_2 let:

$P_1 = P_{\pi_1} = (\pi_1, d_{\pi_1}, I_{\pi_1})$

$P_2 = P_{\pi_2} = (\pi_2, d_{\pi_2}, I_{\pi_2})$.

Since $\pi_1 \cdot \pi_2 = 0$, each pair in $\pi_1 \times \pi_2$ determines a unique r in $S \times L$.

2.3. DETECTION OF NON-TRIVIAL PARALLELISM

In this section, a number of sufficient conditons for the parallel execution of the actions of a process will be given. The theory developed here differs from that of the previous section. Here the question of how may the actions of a diagram be rearranged so as to allow their parallel execution is considered. In the last section, the question of how may the process be broken into different portions each operating on some segment of the state space was solved. To illustrate the difference, consider the following two examples.

Example 2.10. $P = (S,d,I)$ where $S = \{(a,b,c) \mid a,b,c$ are natural numbers$\}$, $I = (1,1,1)$ and d is:

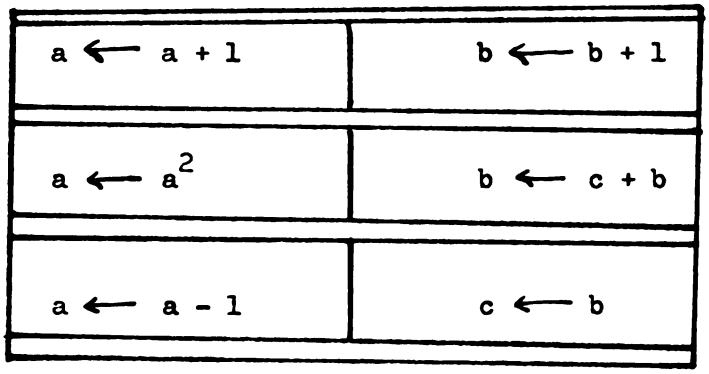


Fig 2.16

By the theory developed in the last section, P may be decomposed into the following processes $P_1 = (S_1, d_1, I_1)$ and $P_2 = (S_2, d_2, I_2)$

where $S_1 = \{(a) \mid a \text{ is a natural number}\}$

$S_2 = \{(b,c) \mid b \text{ and } c \text{ are natural numbers}\}$

$I_1 = (1)$ $I_2 = (1,1)$

d_1 is:

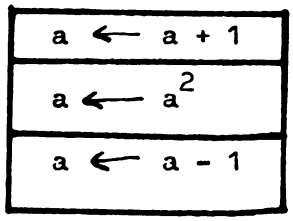


Fig 2.17

and d_2 is:

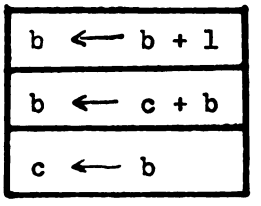


Fig 2.18

It may not be evident at first that these two processes result from the theory of the last section. To verify this, consider these two partitions:

$\pi_1 = \{B \mid (a_1,b_1,c_1)$ and (a_2,b_2,c_2) are in B iff $a_1 = a_2\}$
$\pi_2 = \{B \mid (a_1,b_1,c_1)$ and (a_2,b_2,c_2) are in B iff $b_1 = b_2$ and $c_1 = c_2\}$

Clearly $\pi_1 \cdot \pi_2 = 0$. It is evident that $P\pi_1 = P_1$ and $P\pi_2 = P_2$.

Example 2.11. P = (S,d,I) where S = {(a,b,c) | a,b,c are natural numbers}, I = (1,1,1) and d is:

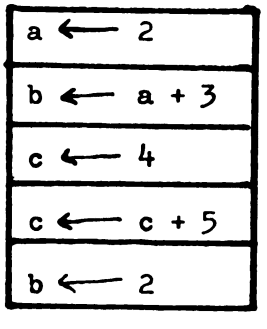


Fig 2.19

A process which does the same thing is $P' = (S,d,I)$ where d' is:

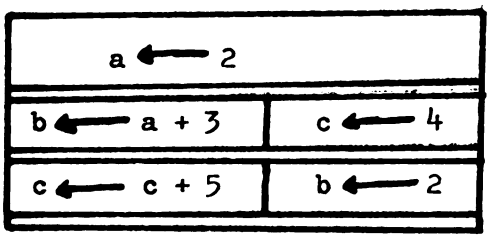


Fig 2.20

Attention will now be turned to situations in which two adjacent sections of a basic diagram may be performed in parallel, as was the case in the last example. Unlike the previous section, it is impossible to prove a theorem which completely characterizes this kind of phenomenon.

Theorem 2.5. There is no algorithm which can decide if any two sections of a diagram may be performed in parallel.

Proof

This assumption will be shown to imply a solution to the halting problem for an arbitrary Turing machine T. Consider the following class of diagrams:

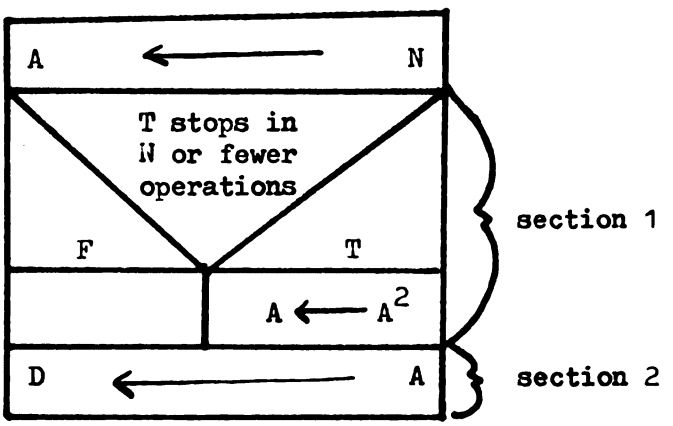


Fig 2.21

where N is an arbitrary integer stored on T's input tape. If T never halts, then for all input data N, section 1 takes the false branch and hence section 1 and section 2 may be performed in parallel. If T eventually stops, then there exists values of N for which section 1 takes the true branch. In this case, section 1 and section 2 must be performed sequentially. Thus, to see if Section 1 and section 2 can be performed in parallel, the halting problem must be solved.

The legality of the Boolean expression in section 1 of the basic diagram may be questioned. So far, it has been assumed that any Boolean expression must deal with only variables in the state space. In order for the last theorem to make sense, there must be a way for a process to simulate a Turing machine. Bernstein (BERN 66) has a similar proof but ignores this question. He makes no clear definition of process. At times he implies that it is a segment of statements from a program and at other times implies that it is a Turing machine.

Definition 2.23. $|K|$ is the cardinality of K.

Definition 2.24 $A(|I|, |K|)$ where I and K are finite sets, denotes $|I| \cdot |K|$ variables.

Theorem 2.6. For an arbitrary Turing machine T, there exists a process P which accepts an input string iff T does.

Given an arbitrary Turing machine T, a process shall be constructed which is equivalent to it. For Turing machine $T = (K, \Sigma, \Gamma, \delta, q_0, F)$ construct the following process $P = (S, d, I)$ where

$S = (C, D_c(|k|, |\Gamma|), D_e(|K|, |\Gamma|) D_w(|K|, |\Gamma|), W, E, T, J, A(1), A(2), \ldots)$

The values the variables may take on are:

1) C may take on any value from K or the special symbol U.

2) $D_c(-,-)$ may taken on any value from K or the special symbol U.

3) $D_e(-,-)$ may take on values from $\Gamma - \{B\}$ or the special symbol U.

4) $D_w(-,-)$ may take on values from $\{L,R,U\}$.

5) W may take on values from $\{L,R,U\}$.

6) E may taken on values from $\{L,R,U\}$.

7) T may take on values from the natural numbers.

8) J may take on values from $\Gamma - \{B\}$ or the special symbol U.

9) A(-) may taken on values from Γ.

The initial states I of the process are all states which have the following three properties:

1) C is q_0.

2) A(I) for I from 1 to N for some natural number N, is not B and the remaining A(-) are B.

3) $D_c(-,-), D_e(-,-)$, and $D_w(-,-)$ are assigned according to δ. If x and y are in K, z is in Γ, r is in $\Gamma - \{B\}$ and s is in $\{L,R\}$ then if $\delta(x,z) = (y,r,s)$, then

D_c

D_e

D_w

and

D_c

D_e

D_w

d is a

It is e

the proc

$D_c(x,z) = y$

$D_e(x,z) = r$

$D_w(x,z) = s$

and if $\delta(x,z)$ is undefined, then

$D_c(x,z) = U$

$D_e(x,z) = U$

$D_w(x,z) = U$

d is as follows:

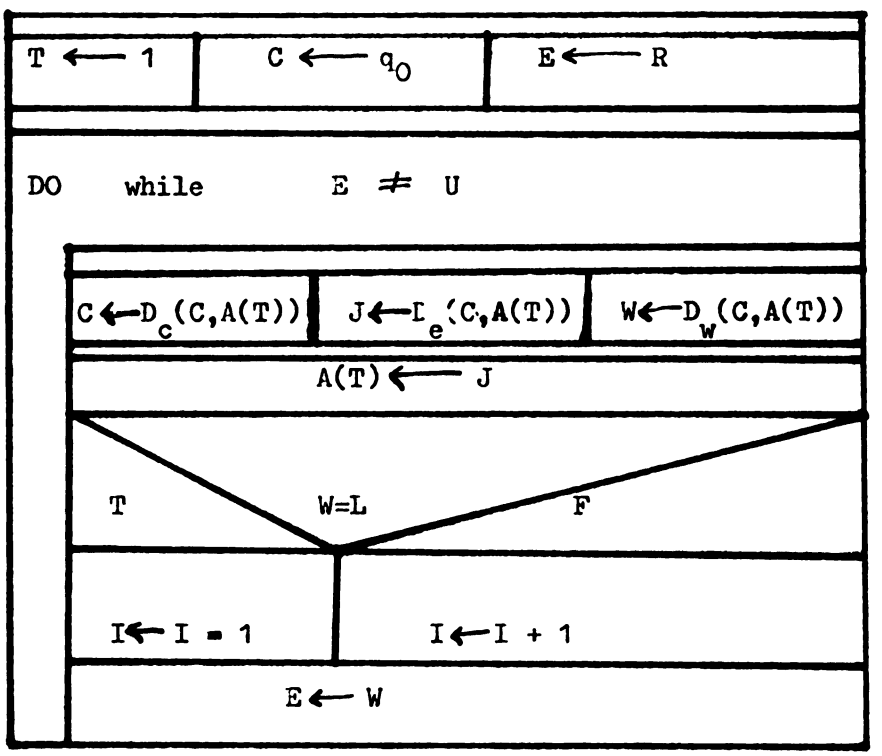


Fig 2.22

It is evident from the construction that the Turing machine and the process do the same thing.

Corollary The concepts of Turing machine and process are equivalent.

Proof

It follows from the above theorem and from Theorem 2.1.

Using the same techniques developed by Bernstein (BERN 66), it is possible to analyse parallelism in processes. The state variables of a state space may be used in two ways by an action. A state variable may only be referenced, in which case the state variable is left unchanged. On the other hand, a state variable may be changed. Clearly, if a state variable is on the left side of an arrow, it is changed and if it is on the right side of an arrow, it is referenced.

There are four different ways that a segment of a basic diagram may use a state variable.

1) The state variable is only referenced by the segment.
2) The state variable is only changed by the segment.
3) the first action involving this state variable is one in which it is changed. One of the succeeding actions references the state variable.
4) The first action involving this state variable is one in which it is changed. One of the succeeding actions references the state variable.

Let R_i, C_i, P_i, and K_i denote the sets of state variables falling into these divisions respectively for section i. It is evident that only a portion of the state variable set is modified

by a particular section i of a basic diagram. That portion is $C_i \cup P_i \cup K_i$. Also, the execution of section i depends only on that portion of the state variable set that is referenced by section i. That portion is $R_i \cup P_i \cup K_i$. An important distinction must be made between K_i in which the referenced state variable was computed in section i itself and R_i and P_i in which the referenced state variable has been assigned prior to section i. Consider the following basic diagram (without DO while clauses):

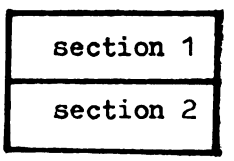


Fig 2.23

The question is when may the above diagram be translated into the following basic diagram:

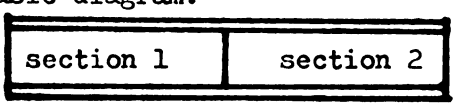


Fig 2.24

Example 2.12. Consider the following diagram:

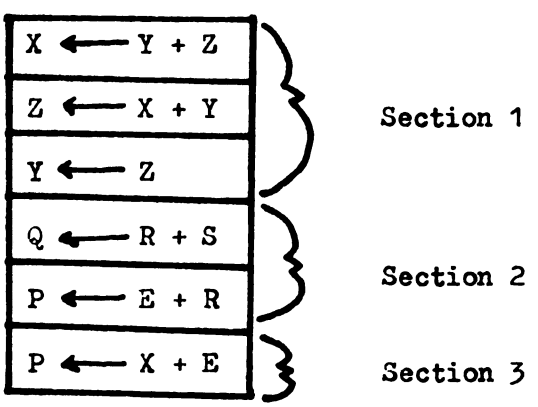


Fig 2.25

It may be translated into

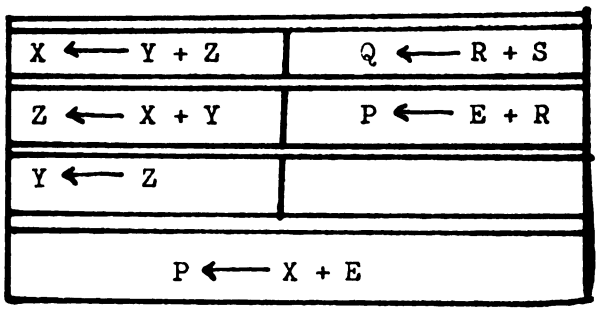


Fig 2.26

which is equivalent to:

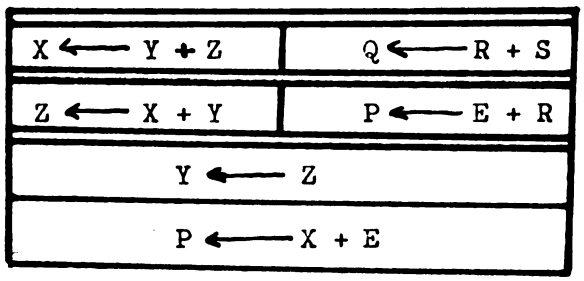


Fig 2.27

The above example leads to the formal definition of the parallel connection of two adjacent sections of a diagram.

Definition 2.25. The parallel connection of two adjacent sections of a basic diagram (without DO while clauses) exists if the new structure formed may be rewritten as a basic diagram in which the successive actions of each section are performed

at the same time.

Definition 2.26. Two processes are equivalent iff their initial states are the same and for any finite computation in one, the computation with the same initial state in the other has the same final state.

Two comments may now be made. First, the parallel connection of two adjacent sections of a basic diagram without DO while clauses may not even exist. Second, if it does exist, the new basic diagram may not yield an equivalent process.

Example 2.13. This is an example in which the parallel connection between two sections does not exist.

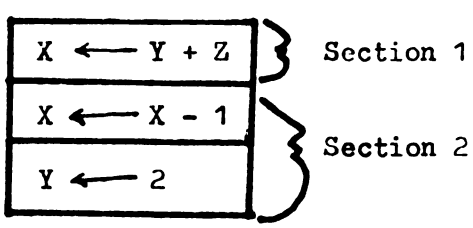


Fig 2.28

The new structure is:

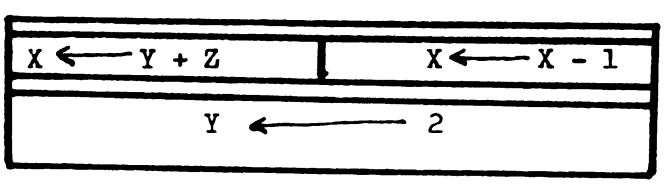


Fig 2.29

The new structure is not a basic diagram since the first action is illegal.

Example 2.14. This is an example of a basic diagram for which the new basic diagram does not yield an equivalent process. Let $P = (S,d,I)$ where $S = \{(a,b,c) \mid a,b, \text{ and } c \text{ are natural numbers}\}$

$I = (1,1,1)$

and d be:

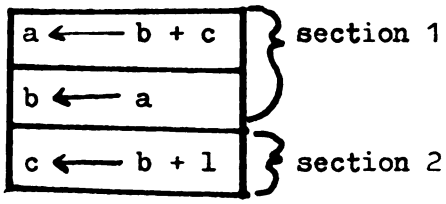


Fig 2.30

The parallel connection of section 1 and section 2 does exist and is d' :

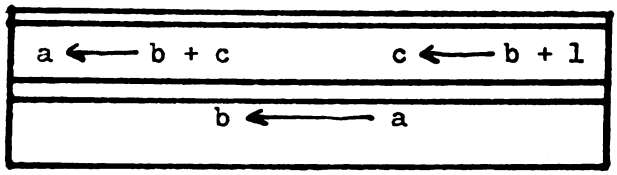


Fig 2.31

(S,d',I) is a process with a single computation whose final state is (2,2,2). (S,d,I) is a process with a single computation whose final state is (2,2,3).

It is evident from the last example that care must be taken in forming the parallel connection of two adjacent sections of a process. The new basic diagram generated by the parallel connection of two adjacent sections of a basic diagram must be only be a basic diagram but also must generate the same final states as the original.

Theorem 2.7. If $(R_2 \cup P_2 \cup K_2) \cap (C_1 \cup P_1 \cup K_1) = \emptyset$
and $(R_1 \cup P_1 \cup K_1) \cap (C_2 \cup P_2 \cup K_2) = \emptyset$
then two adjacent sections (without DO while clauses) of a process may be connected in parallel to yield an equivalent process.
(the subscript 1 refers to the first section, the subscript 2 refers to the second section, and R,C,P, and K are as before.)

Proof

In order that when section 1 references a state variable, it will be the same in each basic diagram, the following is required:

$$(R_1 \cup P_1) \cap (C_2 \cup P_2 \cup K_2) = \emptyset$$

That is to say, no state variables which are referenced in section 1 may be changed by section 2. To insure that section 2 does not change any state variables that section 1 is saving for later reference, the following is required:

$$K_1 \cap (C_2 \cup P_2 \cup K_2) = \emptyset$$

which means that no state variables which section 1 first changes and then later references may be changed by section 2. The two

above equations imply:

$$(K_1 \cup R_1 \cup P_1) \cap (C_2 \cup P_2 \cup K_2) = \emptyset$$

By like reasoning:

$$(K_2 \cup R_2 \cup P_2) \cap (C_1 \cup P_1 \cup K_1) = \emptyset$$

Bernstein (BERN 66) developes a similar theorem but he is forced to add another condition because of his use of branching. This leads to the conclusion that structured programming lends itself more readily to automatic detection of parallelism.

The case of DO while clauses will now be discussed. As DO while clauses were specifically excluded in the parallel connection definition, all parallelism will be considered from the viewpoint of detecting parallelism within the body of a DO while clause. If there are other DO while clauses within the body they may not be used in making a parallel connection. If the last statement of a DO while clause modifies a state variable in the Boolean expression it must also be excluded. (By the definition of DO while clause, this is the only place that such a modification may take place.)

If the DO while clause contains no indexed state variables, this analysis is adequate. If the DO while clause does contain indexed variables, one possible solution is to expand it so that each of the indexed variables is indexed by an element of the index set and then apply the above techniques. This is not only time consuming but is sometimes impossible. The approach taken in (LAMP 74) may be used but the results of Lamport are so restricted that they apply to few DO while clauses.

Chapter 3

SUFFICIENT CONDITIONS FOR POSSESSION OF AN INFINITE COMPUTATION

The appendix contains the terminology needed for the use of digraphs in this thesis. The digraph of a process P has as its vertices S X N, where S is the state space of P and N is its labels.

The directed edge (p,q) will be in the digraph if and only if f(p) = q where p and q are in S X N, and f is the successor function.

Definition 3.1. The reduced digraph of a process P is the digraph of P with all isolated vertices removed.

Example 3.1. Consider the process P = (S,d,I) where

S = $\{(a,b,c) \mid$ a,b, and c are in $\{0,1,2\}\}$

I = $\{(1,1,1),(1,0,2)\}$, and d is:

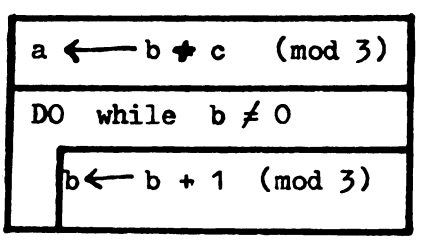


Fig 3.1

For this process the re uced digraph is:

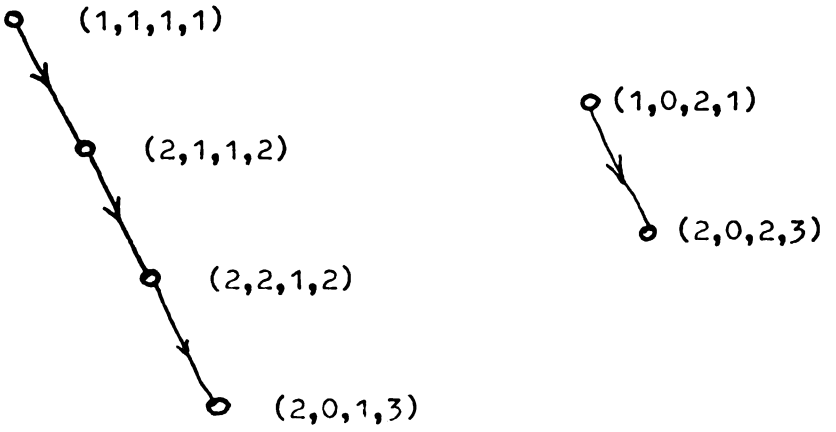


Fig 3.2

The vertex (2,0,2,3) and the vertex (2,0,1,3) are final states. Recall the convention mentioned earlier of labeling the final states with n + 1 as a level, where n is the number of levels in d.

In the above example, both the digraph and the reduced digraph of the process were finite. To obtain the digraph, the other points in the state space are merely added. In the next example, the reduced digraph is finite while the digraph is infinite.

Example 3.2. Consider the process P = (S,d,I) where

S = {(a,b,c) a,b and c are natural numbers}

I is {(1,1,1), (0,0,0), (0,0,1)} and d is:

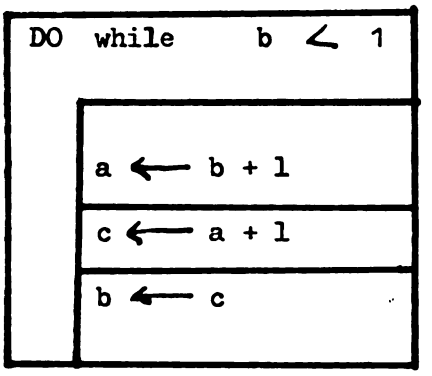


Fig 3.3

The digraphs of this process will have an infinite number of points but the reduced digraph is:

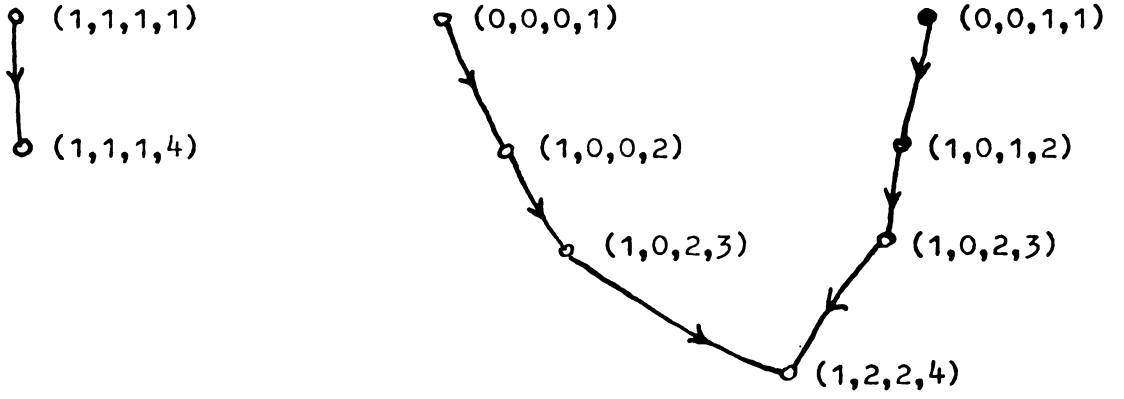


Fig 3.4

The above example is also of interest in that the computation whose initial state is (0,0,0) has the final state as the computation whose initial state is (0,0,1). This kind of joining of computations may occur at any point of a computation but from this point on both computations are the same. That is to say, once two computations join

they may not branch. This means that the outdegree of each vertex of a reduced process digraph is at most one.

Theorem 3.1. If a process P has a finite number of initial states and its reduced digraph is infinite, then P has at least one infinite computation.

Proof

Observe first that the outdegree of each vertex is at most one, by the way process is defined. Since the digraph is infinite, at least one of the initial vertices (states) has an infinite number of successors, call it s_1. Now since s_1 has an infinite number of successors and the outdegree of each vertex is at most one, s_1 must be the first vertex in an infinite path, thus P has an infinite computation.

Theorem 3.2. If a process P has no infinite computations and its reduced digraph is infinite, then P must have an infinite number of initial states.

Proof

This follows from the fact that the outdegree of each vertex is at most one.

If the reduced digraph C of a process P is finite, then the only way for P to have an infinite computation is for there to be a cycle in G.

Theorem 3.3. If the reduced digraph G of a process P is finite, then G has a cycle iff G has a walk of length greater than or equal to n, where n is the number of vertices in G.

Proof

The only way for a walk of length greater than n to exist is for one of the vertices to be repeated, hence there must be a cycle. Conversely, if G has a cycle, then the cycle need only be repeated until a walk longer than n has been produced.

Corollary If a computation of a process P is longer or equal to the number of vertices in the reduced digraph of P, then the computation is an infinite computation.

The above theorem is the basis for an efficient algorithm for determining if a process P with a finite reduced digraph has an infinite computation. The property of process digraphs which makes this possible is the outdegree of any vertex is at most one.

Algorithm Let n be the number of vertices in a reduced process digraph G and P = $\{v_1, \ldots, v_k\}$ be the set of initial vertices. (note that $k \leq n$)

Step 1 i $\longleftarrow$ 1

Step 2 c $\longleftarrow$ 1

Step 3 If v_i has a successor, go to step 4, otherwise go to step 6.

Step 4 $v_i \leftarrow$ successor of v_i ; $c \leftarrow c + 1$.

Step 5 If c is greater than or equal to n, go to step 7, otherwise go to step 8.

Step 6 $i \leftarrow i + 1$, if i is less than or equal to k, go to step 2, otherwise go to step 8.

Step 7 Halt, G has a cycle.

Step 8 Halt, G has no cycles.

Theorem 3.4. The algorithm must stop after checking $O(n^2)$ successors and there exist process digraphs for which this bound is reached for arbitrarily large n.

Proof

There are at most, n initial vertices. At worst, each will have n-1 successors, with the exception of the last which will have n. Therefore, the algorithm must stop after checking $O(n^2)$ successors. Towshow that this bound is reached for arbitrarily large n, suppose $n = 2r$. Consider the following digraph:

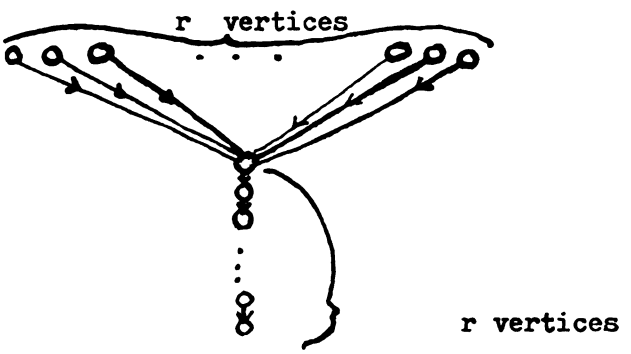


Fig 3.5

Clearly, the algorithm checks r^2 successors and r^2 is $n^2/4$. Since r was arbitrary, there exists a process digraph with an arbitrarily large number of vertices for which the bound holds.

Example 3.3. This example shows how the algorithm may be applied.

Let $S = \{(a,b,c) \mid a \in \{1,2,3\}, b \in \{1,2\}, c \in \{2,3\}\}$.

$I = \{(1,1,2), (3,2,3)\}$

and d be:

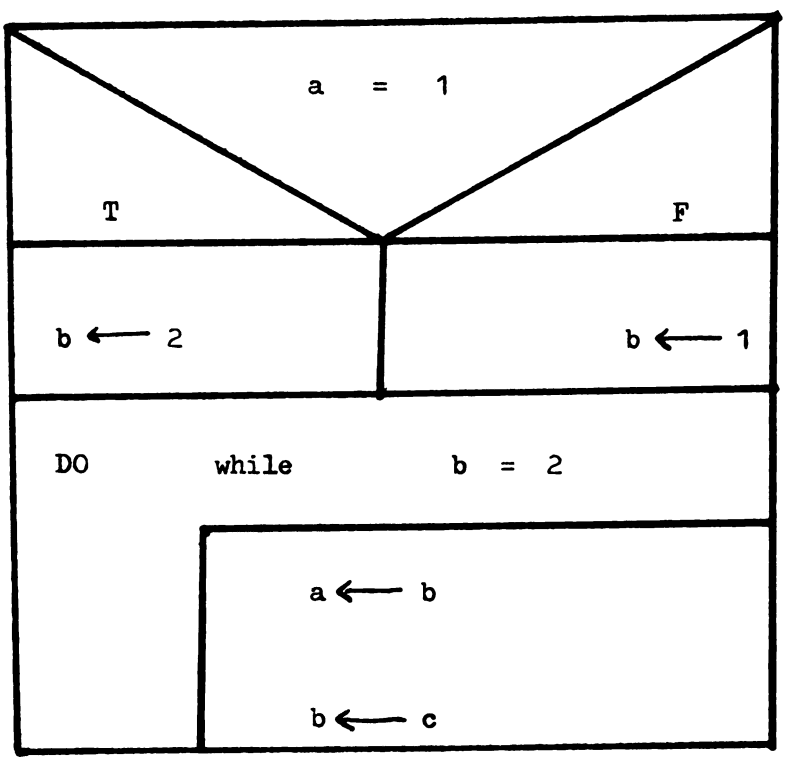


Fig 3.6

The reduced digraph is:

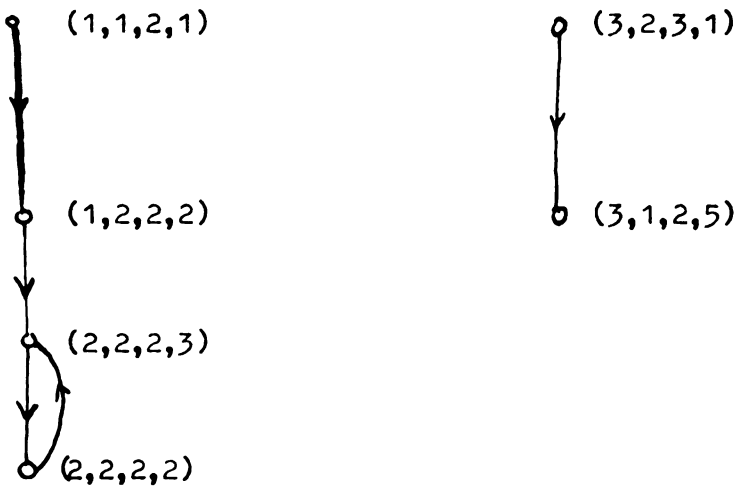


Fig 3.7

Applying Algorith 1: $v_1 = (1,1,2,1)$ $v_2 = (3,2,3,1)$ $k = 2$ $n = 6$

i	c	v_i	Step	
1		(1,1,2,1)	1	
1	1	(1,1,2,1)	2	
1	1	(1,1,2,1)	3	
1	2	(1,2,2,2)	4	
1	2	(1,2,2,2)	5	
1	2	(1,2,2,2)	3	
1	3	(2,2,2,3)	4	
1	3	(2,2,2,3)	5	
1	3	(2,2,2,3)	3	
1	4	(2,2,2,2)	4	
1	4	(2,2,2,2)	5	(to be continued)

i	c	v_i	Step continued
1	4	(2,2,2,2)	3
1	5	(2,2,2,3)	4
1	5	(2,2,2,3)	5
1	5	(2,2,2,3)	3
1	6	(2,2,2,2)	4
1	6	(2,2,2,2)	5
			7

Note that 5 successors were checked.

Definition 3.2. Let G = (V,E) be a reduced process digraph. The condensation of G.with respect to a partition of V, is the digraph whose points are the subsets of the partition and whose lines are determined by the following rule: there is a line from point S_i to point S_j to the new digraph iff in G there is at least one line from a point of S_i to one of S_j, for $i \neq j$. If $i = j$ and S_i is finite, there is a loop at S_i iff there is a cycle in S_i. If S_i is infinite, there is a loop at S_i (if there is a cycle in S_i or if there is an infinite path in S_i).

The concept of condensation may be used to extend the usefulness of the algorithm.

Example 3.4. Let S = $\{(a,b,c) \mid$ a,b, and c are natural numbers $\}$

I = $\{(1,1,1), (2,2,2), (3,3,3)\}$

and d be:

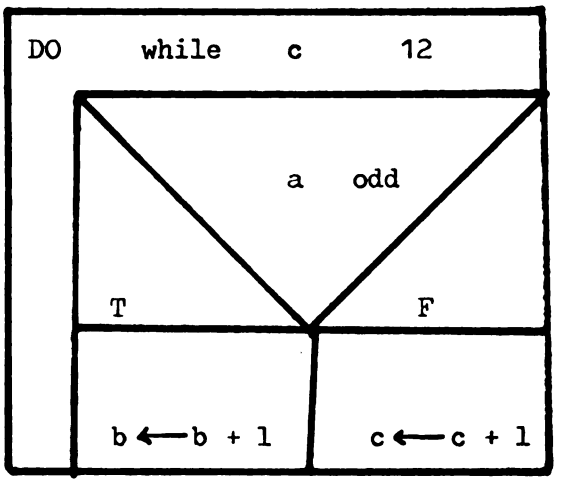


Fig 3.8

It is evident that the reduced digraph G = (V,E) of this process is infinite. Consider the following partition of V:

S_1 = $\{(a,b,c,n) \mid c > 12$ and a is even$\}$

S_2 = $\{(a,b,c,n) \mid c > 12$ and a is odd$\}$

S_3 = $\{(a,b,c,n) \mid c \leq 12$ and a is even$\}$

S_4 = $\{(a,b,c,n) \mid c$ 12 and a is odd$\}$

where n is in $\{1,2\}$

The condensation of G is as follows:

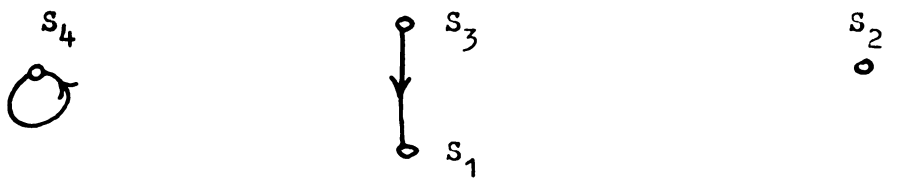


Fig 3.9

$I = \{S_4, S_3, S_2\}$

The initial vertices are just those vertices for which there exists an $x \in S_i$ such that x was an initial vertex of G.

If the algorithm is applied to the condensation of G, it halts in step 7. This leads to the conjecture that the process may have an infinite computation. The following example shows that such a conclusion cannot always be made.

Example 3.5. Let $S = \{(a,b,c) \mid a,b,$ and c are natural numbers$\}$

$I = \{(1,1,1), (2,2,2)\}$ and d be:

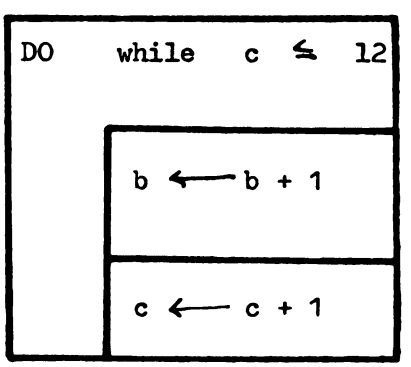


Fig 3.10

Consider the following partition of S X N:

$S_1 = \{(a,b,c,n) \mid c \text{ is even}\}$ $n \in \{1,2,3\}$

$S_2 = \{(a,b,c,n \mid c \text{ is odd}\}$ $n \in \{1,2,3\}$

The condensation of the reduced digraph of P is:

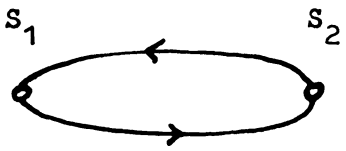


Fig 3.11

The algorithms halts in step 7 for the condensation but the reduced digraph is finite and cycle free.

The last example shows that the condensation of a reduced digraph of a process may have a cycle while the reduced digraph is cycle free. The following theorem shows that if there is a cycle in the reduced digraph of a process, then any condensation must have a cycle. (In some cases, the cycle will be of length 1.)

Theorem 3.5. If the reduced digraph of a process has a cycle, then so does any of its condensations.

Proof

Let P = (S,d,I) be a process and G be its reduced digraph. Suppose G has a cycle: $r_1, r_2, \ldots r_n, r_1$ where the r_i are in S×N. Let S_1 be the partition of S×N, then for each r_i there exists an S_{r_i} such that r_i is in S_{r_i}. Consider the walk:

$$S_{r_1}, S_{r_2}, \ldots S_{r_n}, S_{r_1},$$

this walk contains a cycle in the condensation of G.

The reason that the converse of the above theorem does not hold becomes apparent when the proof of it is attempted. Suppose $S_1, S_2, \ldots S_n, S_1$ is a cycle in some condensation of a reduced process digraph. The definition of condensation implies that there is a

line between a point in S_1 say r_1 and a point in S_2 say r_2. It also guarantees that there is a line between a point in S_2 say r_2' and a point in S_3 say r_3. What it does not guarantee is that there is a path from r_2 to r_2'. It would seem natural then to say that if the strongly connected components of the reduced process digraph are chosen as the partitions, that the reduced digraph will have a cycle iff this condensation does. This is quite true but it is completely useless. If any of the strongly connected components have more than a single point then the reduced process digraph has a cycle, otherwise the partition is the trivial partition. The above arguments show that it is useless to try to find some special class of condensations for which the converse of the above theorem holds for all processes.

Definition 3.3. A process is said to terminate if it has no infinite computations.

Theorem 3.6. A process P terminates if there is a finite condensation of the reduced digraph of P which is cycle free.

Proof

If the reduced digraph of P is finite, this follows from the previous theorem. If the reduced digraph of P is infinite then at least one of the S_i contains an infinite number of vertices. Since there is no loop at any of the S_i which are infinite they do not contain cycles or infinite paths. Since there are no cycles in the

condensation itself there is no way for an infinite path to exist in the reduced digraph. This is evident from the fact that since the infinite S_i contain no infinite paths themselves, the only way for an infinite path to be in the reduced digraph would be for its points to alternate between the S_i. But there are only a finite number of S_i, this would cause a cycle.

Chapter 4

MEASURES OF PROCESS COST

In this chapter, the concept of measurement of the properties of a process is discussed. Three measures which appear to have some practical significance are suggested. In Section One, these measures are discussed at the computational level, while in Section Two, the concepts developed in Section One are extended to provide a single measure for a whole process.

4.1. COMPUTATIONAL COST

Definition 4.1. The *length* of a computation shall be the length of the equivalent path in the process digraph.

Definition 4.2. The *width* of a computation shall be the maximum number of state variables changed between any two adjacent vertices in the equivalent path in the process digraph.

Definition 4.3. The *work* of a computation shall be the number of assignments of values to state variables during a computation.

These three measures were chosen because there was a need for each in the development of an adequate theory of measurement of process. The notion of length corresponds most closely with the measure of time. The notion of width, in a practical sense, might be needed in considering the implementation of a parallel programming configuration as a process. Clearly, each computation of the process must not exceed a certain width (i.e. - the number of processors available). The concept of work is necessary in order to have some way of measuring how many changes in the state variable space a particular computation makes. The following example will be used to point out the interelation between these measures.

Example 4.1. This is an example of a process for which the length and the work both have the same numerical value for all computations. Let P = (S,d,I) where

S = $\{(A,B) \mid A \text{ and } B \text{ are natural numbers}\}$

I = $\{(1,1)\}$ and d is

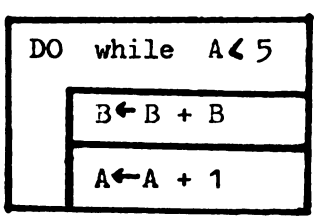


Fig 4.1

This process has a single computation:

$\langle (1,1),(1,2),(2,2),(2,3),(3,4),(3,8),(4,8),(4,16),(5,16) \rangle$

The length of the computation is 8 and its work is 8.

Example 4.2. This is an example of a process for which the numerical value of work is larger than the numerical value of time. It has the same state space and initial state as the previous example but with the following basic diagram:

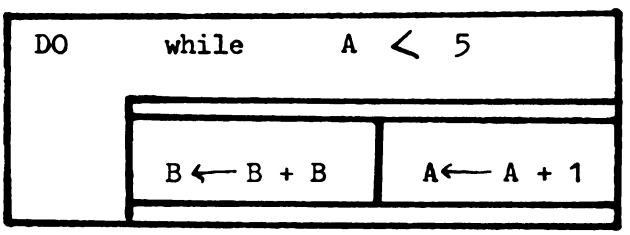


Fig 4.2

It has but one computation:

$$\langle (1,1),(2,2),(3,4),(4,8),(5,16) \rangle \ .$$

The length of the computation is 4 and its work is 8.

Example 4.3. This is an example of a process for which the numerical value of length is larger than the numerical value of work. It has the same state space and initial states as before but the following basic diagram:

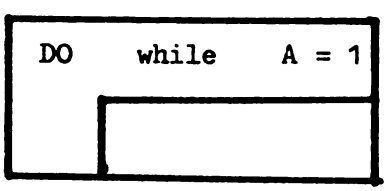


Fig 4.3

It has only one computation but it happens to be an infinite one:

$\langle (1,1),(1,1),\ldots \rangle$.

The length of this computation is infinite but its work is 0.

Example 4.4. This is an example of a process where the length and the work have the same numerical value for one computation but a different value for another. The state space is the same as the last example but the initial states are:

$(1,1)$ and $(1,2)$

and the basic diagram is:

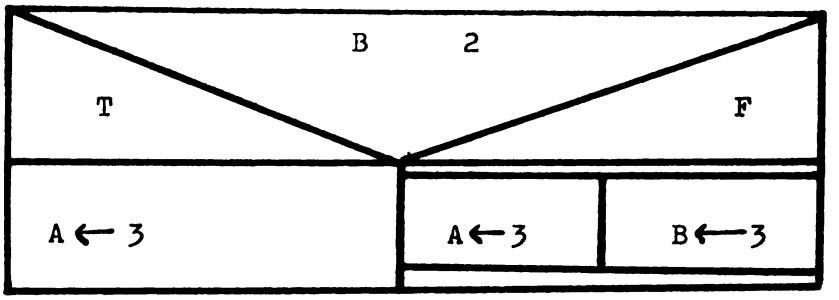


Fig 4.4

It has the computations:

$C_1 = \langle (1,1),(3,1) \rangle$ and

$C_2 = \langle (1,2),(3,3) \rangle$.

In the first computation, both length and work have a value of 1 while in the second, length has a value of 1 and work has a value of 2.

The above examples suggest a relationship between the

three measures for a typical computation. Example 4.3. is the lone exception and suggests that processes with null actions may be difficult to include in such a relationship.

Observation 4.1. In a process with no null actions, the numberical value of length is always less than or equal to the numerical value of work for any computation.

Proof

If no null actions are allowed, at least one state variable must be changed between each state of any computation.

Observation 4.2. In a process with a maximum width of n the work of any computation is less than or equal to n times its length.

Proof

At most, n changes are made per computation.

Observation 4.3. In a process with no null actions and no actions whose cardinality is greater than 1, the work and the length of each computation have the same numerical value.

Proof

It follows from the last two theorems.

The three previous definitions of measurable quantities give rise to some measures which are defined in terms of them.

Definition 4.4. The rate of a computation is its work divided by its length.

Definition 4.5. The capacity of a computation is its length times its width.

If both work and length are infinite, the rate is undefined. Since length is never zero, all other situations are defined. The capacity of a computation is always defined.

4.2. THE COST OF A PROCESS

Attention is now turned to measures of cost of the entire process rather than cost of a particular computation. The first measure considered shall be that of length. As an infinite computation would make the concept of average length meaningless, study of process cost shall be restricted to the cost of nice processes.

Definition 4.6. A nice process is a process whose reduced digraph is finite and cycle free.

With each nice process, may be associated the following sequence: $A = (a_1, a_2, \ldots)$ where a_i is the number of computations of length i. It follows that for each nice digraph, a_i will be zero for all but a finite number of a_i.

Definition 4.7. The average length μ of a nice process with

sequence $A = (a_1, a_2, \ldots)$ is:

$$\frac{\sum_{i=1}^{\infty} i\, a_i}{\sum_{i=1}^{\infty} a_i} .$$

Definition 4.8. The variance σ^2 of the length of a nice process with sequence $A = (a_1, a_2, \ldots)$ is:

$$\frac{\sum_{i=1}^{\infty} (i\, a_i - \mu a_i)^2}{\sum_{i=1}^{\infty} a_i} .$$

Both of these measures are of importance. For example, suppose that a compiler is modeled by a process in which the length of a computation is proportional to the compile time for a program, then a good compiler must not only have a low average compile time but must also have a reasonable variance. Both the average and the variance of a process digraph suggest a certain class of diagraph theoretic measures. These will be discussed in the next chapter.

With each nice process, associate the following sequence $B = (b_1, b_2, \ldots)$ where b_i is the number of computations with work equal to i. Clearly, since a nice digraph is finite and cycle free, b_i will be zero for all but a finite number of b_i.

Definition 4.9. The average work μ_w of a nice process with

sequence B = $(b_1, b_2, \ldots)$ is:

$$\frac{\sum_{i=1}^{\infty} i\, b_i}{\sum_{i=1}^{\infty} b_i}$$

Definition 4.10 The variance σ^2_w of the work of a nice process with sequence B = $(b_1, b_2, \ldots)$ is:

$$\frac{\sum_{i=1}^{\infty} (i\, b_i - \mu_w b_i)^2}{\sum_{i=1}^{\infty} b\, i}$$

With each nice process, associate the following sequence $C = (c_1, c_2, \ldots)$ where the c_i are the number of computations with width equal to i. Since a nice digraph is finite and the width of any computation must be finite, each c_i is finite and the c_i will be zero for all but a finite number of c_i.

Definition 4.11. The average μ_p width of a nice process with sequence $C = (c_1, c_2, \ldots)$ is:

$$\frac{\sum_{i=1}^{\infty} i\, c_i}{\sum_{i=1}^{\infty} c_i}$$

Definition 4.12. The variance σ^2_p of the width of a nice process with sequence $C = (c_1, c_2, \ldots)$ is:

$$\frac{\sum_{i=1}^{\infty} (i\, c_i - \mu_p c_i)^2}{\sum_{i=1}^{\infty} c_i}$$

Definition 4.13. The average rate r of a process is

$$\frac{\mu_w}{\mu} .$$

Definition 4.14. The average capacity c of a process is

$$\mu - \mu_p .$$

The following example illustrates these concepts.

Example 4.5. Consider the process P = (S,d,I) where

S = {(a,b,c)| a,b,c are in {0,1,2}}

I = {(1,1,1),(1,0,2)} and d is:

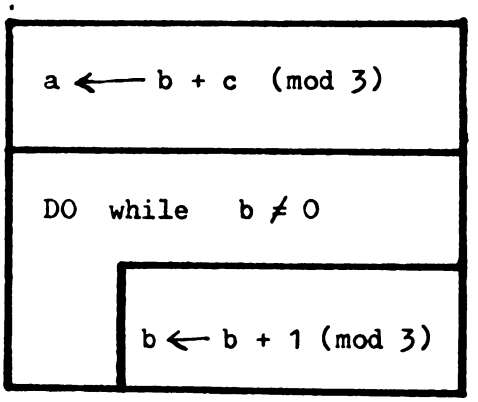


Fig 4.5

the reduced digraph is:

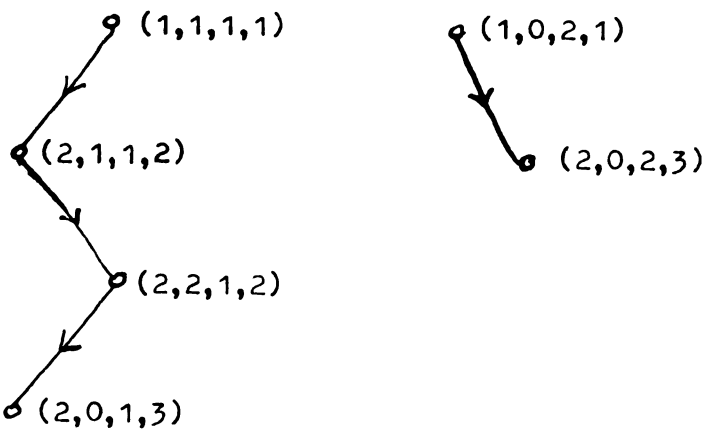


Fig 4.6

A = (1,0,1,0,0,...)

B = (1,0,1,0,0,...)

C = (2,0,...)

$$\mu = \frac{1+3}{2} = 2$$

$$\sigma^2 = \frac{(1-2)^2 + (3-2)^2}{2} = \frac{1+1}{2} = 1$$

$$\mu_w = \frac{1+3}{2} = 2$$

$$\sigma_w^2 = \frac{(1-2)^2 + (3-2)^2}{2} = 1$$

$$\mu_p = \frac{2}{2} = 1$$

$$\sigma_p^2 = \frac{2:1-2)^2}{2} = 0$$

$$r = \frac{\mu_w}{\mu} = \frac{2}{2} = 1$$

$$c = 2 \cdot 1 = 2$$

The above example suggests some relationships between the various measures of processes. The next few theorems point out some relationships which obtain to nice processes with no null actions.

Theorem 4.1. In a nice process with no null actions, μ is less than or equal to μ_w.

Proof

For a nice process $\sum_{i=1}^{\infty} a_i = \sum_{i=1}^{\infty} b_i$

because in each sequence, a computation is only counted once.

In view of observations 4.1:

$$\sum_{i=1}^{\infty} i\, a_i \leq \sum_{i=1}^{\infty} i\, b_i$$

for nice processes with no null actions, thus

$$\mu \leq \mu_w.$$

Theorem 4.2. In a nice process μ_w is less than or equal to n times μ where $n = \max \{c_1, c_2, \ldots\}$.

Proof

n is clearly the maximum width of the process, hence at most n changes are made per compuatation. Since:

$$\sum_{i=1}^{\infty} a_i = \sum_{i=1}^{\infty} b_i$$

it follows from observation 4.2 that

$$\mu_w \leq n \cdot \mu .$$

Observation 4.4. $1 \leq r \leq n$ for any nice process with no null actions, where $n = \max\{c_1, c_2, \ldots\}$.

Proof

From Theorem 4.1 and Theorem 4.2 it follows that

$$\mu \leq \mu_w \leq n \cdot \mu$$

Since $\mu \neq 0$ it follows that $1 \leq r \leq n$.

In the next chapter, these notions of measurement will be generalized to classes of digraphs. Just as the above measures may be used to order nice processes, the measures developed in the next chapter will be used to order members within a class of digraphs.

Chapter 5

A GENERAL THEORY OF DIGRAPH MEASUREMENT

In this chapter, a scheme is developed for assigning non-negative real numbers to digraphs. This scheme will later be used to obtain measures of certain digraph theoretic properties.

5.1. A DISCUSSION OF L AND NORMS ON L

Definition 5.1. Let L be the class of all infinite sequences of non-negative integers with only finitely many nonzero terms. Members of L will be denoted by capital letters from the beginning of the alphabet, with terms in the sequence being indicated by properly subscripted lower case letters. (e.g. $A = (a_1, a_2, \ldots)$)

Definition 5.2. Let N be the natural numbers.

Definition 5.3. Let I be the non-negative integers.

Definition 5.4. Let R be the positive reals.

Consider L and I, it is possible to define a multiplication of sequences in L by integers in I.

Definition 5.5. Let A be in L and i be in I, then iA = $(ia_1, ia_2, \ldots)$.

Definition 5.6. Let A and B be in L, then $A + B = (a_1, a_2, \ldots) + (b_1, b_2, \ldots) = (a_1+b_1, a_2+b_2, \ldots)$.

If iA is thought of as a type of scalar multiplication, the system considered here is almost a module. If falls down in two places; additive inverses in L and additive inverses in I. It is still possible to write $A-B = (a_1-b_1, a_2-b_2, \ldots)$.
but there is no guarantee of its existance.

Definition 5.7. A norm on L is a real-valued function d satisfying the following properties for all x in I and A,B, in L:

1) $d(A) = 0$ if $A = 0$, $d(A) > 0$ if $A \neq 0$.

2) $d(xA) = xd(A)$.

3) $d(A + B) \leq d(A) + d(B)$.

Definition 5.8. I_n is the sequence whose terms $i_m = 0$ if $n \neq m$ and $i_m = 1$ if $n = m$.

Examples of norms are now given.

Example 5.1. $d(A) = \text{Max}\{a_1, a_2, \ldots\}$

1) Clearly, $d(A) = 0$ if $A = 0$, $d(A) > 0$ if $A \neq 0$.

2) Clearly, $d(xA) = \text{Max}\{xa_1, sa_2, \ldots\} = x\text{Max}\{a_1, a_2, \ldots\} = xd(A)$.

3) $d(A+B) = \text{Max}\{a_1 + b_1, a_2 + b_2, \ldots\} \leq \text{Max}\{a_1 + \text{Max}\{b_1, b_2, \ldots\}$

$a_2 + \text{Max}\{b_1, b_2, \ldots\}, \ldots\} \leq \text{Max}\{b_1, b_2, \ldots\} + \text{Max}\{a_1, a_2, \ldots\}$

Example 5.2. $d(A) = \sum_{n=1}^{\infty} f(n)a_n$ where f is a function from N into R.

1) since $f(n) > 0$ for all n, $d(A) = 0$ if $A = 0$ and $d(A) > 0$ if $A \neq 0$.

2) $d(xA) = \sum_{n=1}^{\infty} f(n)xa_n = x \sum_{n=1}^{\infty} f(n)a_n = xd(A)$.

3) $d(A + B) = \sum_{n=1}^{\infty} f(n)(a_n + b_n) = \sum_{n=1}^{\infty} f(n)a_n + \sum_{n=1}^{\infty} f(n)b_n = d(A) + d(B)$.

Example 5.2. is very important since it is the only example of norm (as shown in the next theorem) for which condition three is an equality. Such a norm shall be called a strong norm.

Theorem 5.1. The only norms for which condition 3 is an equality are those of the form $d(A) = \sum_{n=1}^{\infty} f(n)a_n$ where f is a function from N into R.

Proof

Example 5.2. shows that d is a norm. Suppose d' is a norm for which condition 3 is also an equality. If $d'(I_1), d'(I_2), \ldots$ are known, $d'(A)$ can be found for any nonzero A. It is done in the following

manner: Let A be any nonzero member of L. Since there are only finitely many nonzero terms in A, there exists a last nonzero term, say a_n. Now $A = a_1I_1 + a_2I_2 \ldots + a_nI_n$. $d'(A) = d'(a_1I_1 + a_2I_2 \ldots + a_nI_n)$ and since condition 3 is an equality, then $d'(A) = d'(a_1I_1) + \ldots + d'(a_nI_n)$. By condition 2, $d'(A) = a_1d'(I_1) + \ldots + a_nd'(I_n)$. The proof is finished since $d'(I_n)$ must be greater than zero in order to satisfy condition 1.

5.2. THE IMPORTANCE OF STRONG NORMS

The importance of strong norms from a practical standpoint is somewhat obscured by the definition of norm given. In this section, the same results of section one will be obtained using standard definitions. These results represent a slight rearrangement of results presented by Norman and Roberts (NORM 72).

Suppose f is any function from N to R and A and B are in L; consider the metric d defined in the following manner: $d(A,B) = \sum_{n=1}^{\infty} f(n) \, |b_n - a_n|$. First it must be shown that d is a metric.

Lemma 5.1. (L,d) is a metric space.

Proof

1) symmetry: $d(A,B) = \sum_{n=1}^{\infty} f(n) \, |b_n - a_n|$

$$= \sum_{n=1}^{\infty} f(n) \, |a_n - b_n|$$

$$= d(B,A)$$

2) triangle inequality:

$$d(A,B) \quad d(B,C) = \sum_{n=1}^{\infty} f(n) \quad |b_n - a_n| + \sum_{n=1}^{\infty} f(n) \, |c_n - b_n|$$

$$= \sum_{n=1}^{\infty} f(n)(\, |b_n - a_n| + |c_n - b_n| \,)$$

$$\geq \sum_{n=1}^{\infty} f(n) \, |b_n - a_n + c_n - b_n|$$

$$= \sum_{n=1}^{\infty} f(n) \, |c_n - a_n| = d(A,C)$$

3) $d(A,B) \geq 0$ and $d(A,B) = 0$ iff $A = B$.

The first part is evident from the definition. If $A = B$, then $d(A,B) = \sum_{n=1}^{\infty} f(n) \, |a_n - a_n| = 0$. If $d(A,B) = 0$, then $\sum_{n=1}^{\infty} f(n) \, |b_n - a_n| = 0$, hence $b_n = a_n$ for all n because $f(n)$ is positive.

Before proving the next lemma, the notion of one

sequence being between two others must be defined. The Kemeny - Snell (KEME 62) definition shall be used.

Definition 5.9. Let A,B,C be in L, then B is between A and C if for all m, either $a_m \leq b_m \leq c_m$ or $a_m \geq b_m \geq c_m$. This shall be denoted by $[A,B,C]$.

Lemma 5.2. If $[A,B,C]$, then $d(A,C) = d(A,B) + d(B,C)$.

Proof

$$d(A,B) + d(B,C) = \sum_{n=1}^{\infty} f(n) |b_n - a_n| + \sum_{n=1}^{\infty} f(n) |c_n - b_n|$$

$$\sum_{n=1}^{\infty} f(N) (|b_n - a_n| + |c_n - b_n|)$$

Since $a_n \leq b_n \leq c_n$ or $c_n \leq b_n \leq a_n$ for each n, $(b_n - a_n)$ and $(c_n - b_n)$ have the same sign for each n, hence $d(A,B) + d(B,C) =$

$$\sum_{n=1}^{\infty} f(n) |b_n - a_n + c_n - b_n| = \sum_{n=1}^{\infty} f(n) |c_n - a_n| = d(A,C)$$

Lemma 5.3. $d(0,I_m) = f(m)$.

Proof

It is evident from the definition of d.

Lemma 5.4. $d(A + C, B + C) = d(A,B)$.

Proof

$$d(A+C,B+C) = \sum_{n=1}^{\infty} f(n) \left| (b_n\ c_n)-(A_n\ c_n) \right|$$

$$= \sum_{n=1}^{\infty} f(n) \left| (b_n-a_n) \right| = d(A,B)$$

Lemma 5.2. and Lemma 5.4. resemble certain axioms used by Kemeny-Snell (KEME 62) for distance between preference rankings and Lemma 5.3. insures that d(0,A) will be a strong norm.

A theorem which is analogous to theorem one is now proved. This theorem shows that the definition of d was in a sense, much more general than it appeared.

Theorem 5.2. For any function f from N into R, d is the only type of metric on L which satisfied Lemma 5.2., Lemma 5.3., and Lemma 5.4.

Proof

It must be shown that if there is such a metric, say d', that d' = d. First it will be shown that $d'(0,sI_m) = sf(m)$ for any non-negative integer s. This is done by induction on s. It is true for s = 0 because d' is a metric, which implies d'(0,0) = 0. Now if $s > 0$, then $[0,(s-1)I_m,sI_m]$ and hence by Lemma 5.2. $d'(0,sI_m) = d'(0,(s-1)I_m) + d'(s-1)I_m,sI_m)$. Now by Lemma 5.4, $d'((s-1)I_m,sI_m) = d'(0,I_m)$ which by Lemma 5.3. is f(m). It has now been established that $d'(0,sI_m) = d'(0,(s-1)I_m)$

+ f(m). Using the induction hypothesis, this becomes

$d'(0,sI_m) = (s-1)\ f(m) + f(m)$ hence $d'(0,sI_m) = sf(m)$

In order to complete the proof, define S_k to be that subset of L consisting of sequences in which only components one through k may differ from 0. It will be established that d' = d by induction of k. If k = 1 and A,B are in S_k, then $A = A_1I_1$ and $B = B_1I_1$. If $b_1 \neq a_1$, then B-A is in L and hence by Lemma 5.4, $d'(A,B) = d''(0,B-A) = d'(0,(b_1-a_1)I = f(1)|b_1-a_1|$ hence d' = d. If $b \neq a$, then S-B is in L and hence by Lemma 5.4. $d'(A,B) = d'(0,B-A) = d'(0,(b_1-a_1)I = f(1)\ b_1-a_1$ hence d' = d. If $b_1 \neq a_1$, the A-B is in L and hence by Lemma 5.4, $d'(A,B) = d'(B,A) = d'(0,A-B) = d'(0,(a_1-b_1)I_1) = f(1)|a_1-b_1|$ hence d' = d. Therefore d = d' for all A and B is S_1. To complete the induction, assume that d' = d for sequences in S_{k-1}. It must be shown that it holds for sequences in S_k. If A and B are is S_k define $C = (b_1,b_2,\ldots,b_{k-1},a_k,0,\ldots)$. Clearly, $[A,C,B]$ and hence by Lemma 5.2, $d'(A,B) = d'(A,C) + d'(C,B)$. But $d'(C,B) = f(k)|b_k-a_k|$ since either B-C or C-B is in L and $d'(C,B) = d'(0,B-C) = d'(0,(b_k-a_k)I_k)$ in the former case and $d'(C,B) = d'(C-B,0) = d'((a_k-b_k)I_k,0)$ in the latter. It is now evident that

$$d'(A,B) = \sum_{n=1}^{\infty} f(n)|b_n\ a_n|$$

this follows by the inductive assumption and Lemma 5.4., since A may be written as $A' + a_kI_k$ and C as $B' + a\ I$ where $A' = (A_1,\ldots a_{k-1},0\ldots)$ and $B' = (b_1,\ldots,b_{k-1},0,\ldots)$. d' = d since the induction is now complete.

5.3. ASSOCIATING NON-NEGATIVE REAL NUMBERS WITH DIGRAPHS

In this section, a method of associating non-negative extended real numbers with each digraph in a class of digraphs, Q, is developed. This number is assigned in such a way that it represents a mix of certain properties, each of which is represented by a sequence in L. To accomplish this, consider an n-tuple of functions $(h',\ldots h^n)$ from NXQ into I such tha for all G in Q and for all m in N, $h^i(m,G) = 0$ for all but finitely many m for $i = 1,\ldots,n$. These n functions associate each G in Q with n sequences in L. $S^{i=} = (h^i(1,G),h^i(2,G),\ldots)$.

Definition 5.10. $S = (S^1,S^2,\ldots S^n)$ is called the sequence tuple of G.

Suppose $d = (d^1,d^2,\ldots,d^n)$ is an n-tuple of norms on L and there is a subset D of L^n closed under sequence addition such that there is no digraph in Q with a sequence tuple not in D. Each digraph G in Q is associated with a non-negative extended real number by means of a function f(S,D). This number may be used to order the digraphs in Q. The ordering is accomplished in the natural manner.

Definition 5.11. Let G and H be in Q with sequence tuples $S_g = (S'g,\ldots,S^m{}_g)$ and $S_h = (S'_h,\ldots,S^n{}_h)$ respectively and let $d = (d_1,\ldots,d_n)$ be an n-tuple of norms on L, then $G \leq H$ iff $f(S_g,d) \leq f(S_h,d)$.

Theorem 5.3. If G and H are in Q and have the same sequence tuples then for all F in Q, $G \lesssim F$ iff $H \lesssim F$ and $F \lesssim G$ iff $F \lesssim H$.

Proof

It is evident from the definitions.

The notions defined so far may be used to obtain several partitions of Q. This is done by defining several equivalence relations.

1) S^i equivalence Let G and H be in Q, then GS^iH iff G and H have the same i^{th} sequence in their sequence tuple

2) S equivalence Let G and H be in Q, then GSH iff G and H have the same sequence tuple.

3) f equivalence Let G and H be in Q, then GFH iff $f(S_1,d) = f(S_2,d)$ where S_1 and S_2 are sequence tuples for G and H respectively, and d is an n=tuple of norms on L of the same dimension as S_1 and S_2.

Observation 5.1. The common refinement of the partitions generated by S^i equivalence is the partition generated by S equivalence.

f-equivalence will be used to resolve a slight problem with the ordering obtained in the previous definition. As it stands now, $\lesssim$ is not a partial ordering of Q unless f is 1-1 and for each G in Q, the sequence tuple is unique. However, it is

an easy matter to show that $\leq$ is a partial ordering of the equivalence classes of f. This is evident since the reason $\leq$ was not a partial ordering of Q in the first place was the antisymmetric property.

Definition 5.12. Let P be the collection of equivalence classes of f.

Theorem 5.4. $\leq$ is a partial ordering of P.

Proof

1) Clearly, if S is in P then $S \leq S$, since any two members of S have the same f value.

2) Let S and T be in P. If $S \leq T$ and $T \leq S$, then S and T must have the same f value, hence S = T.

3) Let X,Y,Z be in P. X Y implies the f value shared by members of X is less than or equal to the f value shared by the members of Y. $Y \leq Z$ implies that the f value shared by members of Y is less than or equal to the f value shared by members of Z. The above statements imply the f value shared by members of X is less than or equal to the f value shared by the members of Z, hence $X \leq Z$.

Theorem 5.5. $(P, \leq)$ is a lattice.

Proof

Let X and Y be in P. At least one of the following must hold:

1) $X \leq Y$ or 2) $Y \leq X$.

Suppose 1) holds, the l.u.b. (X,Y) = Y and g.l.b. (X,Y) = X.

Suppose 2) holds, then l.u.b. (X,Y) = X and g.l.b. (X,Y) = Y.

If both hold, then l.u.b. (X,Y) = g.l.b. (X,Y) = X = Y.

Theorem 5.6. $(P, \leq)$ has a zero if there is a digraph H in Q such that $f(S_h, d) = 0$.

Proof

Clearly, if $Z = \{G \in Q$ the f value of G is zero$\}$ then for all X in P l.u.b.(X,Z) = X.

Theorem 5.7. $(P, \leq)$ has an identity if there is a digraph in Q such that $f(S_h, d) = \infty$.

Proof

Clearly, if $I = \{G$ in $Q \mid$ the f value of G is $\infty\}$ for all X in P g.l.b.(I,X) = X.

Theorem 5.8. $(P, \leq)$ is a distributive lattice.

Proof

This is evident from the fact that P may be thought of as some subset of the extended real numbers with the natural order.

Theorem 5.9. Suppose there are digraphs H and G in Q such that $f(S_h, d) = \infty$ and $f(S_g, d) = 0$, then $(P, \leq)$

is complemented iff P consists of two classes Z and I defined as before.

Proof

Clearly, if P consists of only the two classes Z and I, then $(P, \leq)$ is complemented. Suppose there are more than two classes in P. This means that there is a class A such that the f value for A is not 0 or ∞. For this A there must be an A; such that g.l.b. (A,B') = Z. The only way this may happen is for A' to be Z. But l.u.b.(A,Z) = A not I. So $(P, \leq)$ is not complemented.

Chapter 6

APPLICATIONS OF THE GENERAL THEORY

6.1. BALANCE IN SIGNED DIGRAPHS

In this section, the methods of the previous chapter will be used to develop a measure of balance in signed digraphs. The method developed by Norman and Roberts (NORM 72) will be shown to be a special case of this measure.

Intuitively, any measure of balance in signed digraphs should involve some kind of ratio between balanced semicycles and semicycles whose sign is negative. As a consequence, the sequence tuples will be two-tuples and the function f will be a special kind of function.

Definition 6.1. $f(S,d)$ is called a ratio function if it is equal to $\frac{d^1(S^1)}{d^2(S^2)}$ or $\frac{d^2(S^2)}{d^1(S^1)}$ where

$S = (S^1,S^2)$ is a sequence tuple and

$d = (d^1,d^2)$ is a two=tuple of norms.

For the rest of this section, only ratio functions shall be considered. If a digraph has no semicycles, the question of

balance is not relevant. Consequently, the class of signed digraphs that can be ordered according to balance, consists of those signed digraphs with at least one semicycle.

Definition 6.2. Let Q be the class of all signed digraphs with at least one semicycle.

Definition 6.3. Let h^1, a function from N⨯Q into I, be defined as follows: $h^1(m,G)$ is the number of semicylces of length m + 2 whose sign is positive.

Definition 6.4. Let h^2, a function from N⨯Q into I, be defined as follows: $h^2(m,G)$ is the number of semicycles of length M + 2 whose sign is negative.

Definition 6.5. Let D = L⨯L-(0,0).

Lemma For each sequence tuple in D there is a signed digraph in Q with that sequence tuple.

Proof

Suppose (S^1,S^2) is an arbitrary sequence tuple in D. At least one of the S^i is not zero, since (0,0) is not in D. If S^1 is not zero, procede as follows: Start with a point x. For each S^1_i in S^1, construct s^1_i distinct positive cycles of length i+2 whose first and last point is x.

If S^2 is not zero procede as follows: Start with point x. For each

s^2_i in S^2, construct S^2_i distinct positive cycles of length $i+2$ whose first and last point is x.

It should be noted that the definition of L insures that a structure so constructed will be finite.

Example 6.1. Suppose $(S^1,S^2) = ((0,1,0,0\ldots),(2,1,0,\ldots))$ then the resultant graph would be:

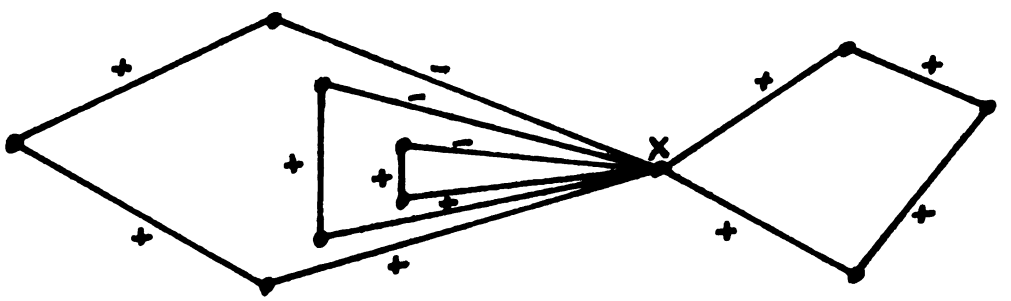


Fig 6.1.

The relation between two two-tuples of norms which induce the same order on Q is dealt with in the next theorem.

Theorem 6.1. Suppose $d_1 = (d^0_1, d^1_1)$ and $d_2 = (d^0_2, d^1_2)$ are two two-tuples of strong norms on L which induce the same order of Q and f is a ration function, then there are real numbers p and q such that $d^0_1(A) = pd^0_2(A)$ for all A in L and $d^1_1(A) = qd^1_2(A)$ for all A in L.

Proof

Now $d^0_1(A) = \sum_{n=1}^{\infty} f^0_1(n)\, a_n$

$$d^1_1(A) = \sum_{n=1}^{\infty} f^1_1(n)\, a_n$$

$$d^0_2(A) = \sum_{n=1}^{\infty} f^0_2(n)\, a_n$$

$$d^1_2(A) = \sum_{n=1}^{\infty} f^1_2(n)\, a_n$$

where f^0_1, f^1_1, f^0_2, f^1_2 are functions from N into R since d^0_1, d^1_1, d^0_2, d^1_2 are all strong norms.

In order to prove the theorem, it will be shown that $f^0_1(n) = pf^0_2(n)$ and $f^1_1(n) = qf^1_2(n)$ for all n. Suppose n is given. For any k and t, it is a simple matter to construct a signed digraph with sequence tuple (kI_n, tI_1). Also construct a digraph H with sequence tuple (I_1, I_1). Now $G \leq H$ iff

$$\frac{d^0_1(kI_n)}{d^1_1(tI_1)} \leq \frac{d_1(I_1)}{d^1_1(I_1)} \quad \text{iff} \quad \frac{kf^0_1(n)}{tf^1_1(1)} \leq \frac{f^0_1(1)}{f^1_1(1)}$$

$$\text{iff} \quad \frac{f^0_1(n)}{f^0_1(n)} \leq \frac{t}{k}$$

But since (d^0_1, d^1_1) and (d^0_2, d^1_2) induce the same order on Q it is evident that $G \leq H$ iff $\dfrac{f^0_2(n)}{f^0_2(1)} \leq \dfrac{t}{k}$

Since this happens for all t and k, it is clear that:

$$\frac{f^0_1(n)}{f^0_1(1)} = \frac{f^0_2(n)}{f^0_2(1)}$$

hence $f^0_1(n) = \left[f^0_1(1)/f^0_2(1)\right] \quad f^0_2(n)$.

A similar argument shows:

$$f^1_1(n) = \left[f^1_1(1)/f^1_2(1)\right] \; f^1_2(n)$$

In view of this last theorem, it is evident that the order derived fro the digraphs in Q is quite dependent on the choice of $d\ (d^0,d^1)$. From Theorem One of the previous chapter, it is evident that $d^0(A) = \sum_{n=1}^{\infty} g^0(n)\, a_n$ and $d^1(A) = \sum_{n=1}^{\infty} g^1(n)\, a_n$ where g^0 and g^1 are functions from N into R. This being the case, d^0 and d^1 may be described in terms of g^0 and g^1. It would seem reasonable to choose both g^0 and g^1 as some kind of decreasing functions, since intuitively it would seem that the shorter the semicycle, the more weight it should be given in determining balance.

The order developed in Norman and Roberts (NORM 72) is a special case of the techniques developed here. If d^0 and d^1 are identical, this order is obtained.

6.2. COST OF RETRIEVAL FROM A BINARY TREE

Palmer, Rahimi, and Robinson (PALM) in a paper dealing

with the efficiency of binary tree storage, develop a measure for the average and variance of the number of comparisons needed to retrieve one item from a store of n items. In this section, their results will be shown to be related to the class of measures derived in the previous chapters.

Definition 6.6. Let B be the class of all binary forests.

For each G in B, associate the following sequence: $A = (a_1, a_2, \ldots)$ where a_i is the number of paths on length i. As binary forests are finite, it follows that a_i will be zero for all but a finite number f a_i.

Definition 6.7. The average length μ of a binary forest with sequence $A = (a_1, a_2, \ldots)$ is

$$\frac{\sum_{i=1}^{\infty} i\, a_i}{\sum_{i=1}^{\infty} a_i} .$$

Definition 6.8. The variance σ^2 of a binary forest with sequence $A = (a_1, a_2, \ldots)$ is

$$\frac{\sum_{i=1}^{\infty} (i\, a_i - \mu a_i)^2}{\sum_{i=1}^{\infty} a_i} .$$

Both these measures reduce to the results given by Palmer, Rahimi, and Robinson (PALM 74) when the particular forest in question is the forest made up on all possible binary stores of n items. Palmer, Rahimi, and Robinson (PALM 74) were able to express their results in a manner which showed that the variance approaches $7 - 2/3\pi^2$ for large n. This raises a rather interesting question. Can techniques similar to theirs be used in other instances of graph-theoretic measurement?

Another way in which the theory developed in the last chapter may be applied to the results derived by Palmer, Rahini, and Robinson (PALM 74) is to let B be the class of all binary forests in which each tree has exactly n nodes. The theory then provides a means of ordering the various forests according to average length and variance. This ordering might provide help in devising a scheme for merging representatives from the various classes.

Example 6.2. Suppose B is the class of all binary forests in which each tree has four vertices. Suppose further that it is known that G_1 comes from the following forest:

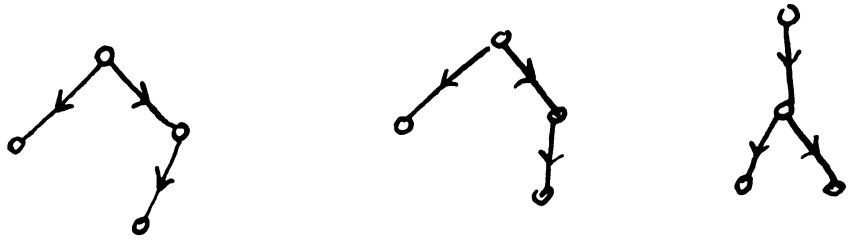
F_1 

Fig 6.2.

and that G_2 comes from the following forest:

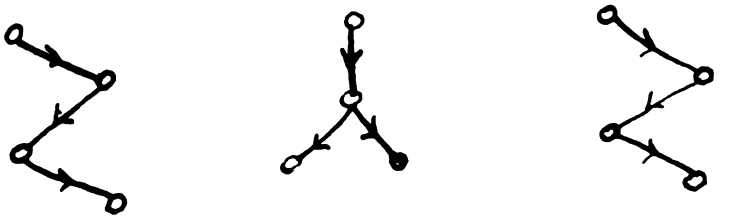
F_2 

Fig 6.3.

$$\mu(F_1) = \frac{\sum_{i=1}^{\infty} i\, a_i}{\sum_{i=1}^{\infty} a_i}$$

where A = (5,4,0,...0) hence $\mu(F_1) = \frac{5 + 8}{10} = 1.3$

$$\mu(F_2) = \frac{\sum_{i=1}^{\infty} i\, a_i}{\sum_{i=1}^{\infty} a_i}$$

where A = (3,4,2,0,0,...) hence $\mu(F_2) = \frac{3 + 8 + 6}{10} = 1.7$

μ could be used to make the decision as to how a tree known to be from F_1 might be merged with a tree known to be from F_2.
A more sophisticated scheme might also make use of variance.

6.3. A MEASURE OF TRANSITIVITY

The question of deciding how transitive a particular finite asymmetric digraph is, is of great importance in trying to measure the reliability of a particular judge. Until now, no satisfactory quantitative measure has been developed for this. The theory of the last chapter lends itself readily to this problem. The only restriction made upon the finite asymmetric digraphs is that they contain at least one path of length two.

Definition 6.9. Let Q be the class of all finite asymmetric digraphs with at least one path of length two.

Definition 6.10. If G is in Q and $S \subseteq V(G)$, then $\langle s \rangle$ is a maximal induced weakly connected intransitive subgraph iff it is weakly connected and intransitive and there is no point v in V(S)-S such that $\langle S \cup \{v\} \rangle$ is weakly connected and intransitive.

Example 6.3. Consider the following digraph:

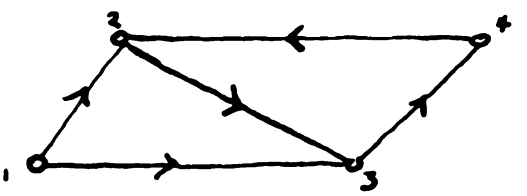


Fig 6.4.

let $S = \{2,3,4\}$ then S is the following digraph:

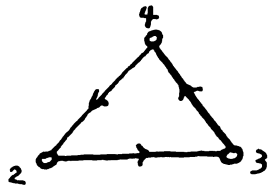


Fig 6.5.

Now $\langle S \rangle$ is clearly intransitive and weakly connected. It is also maximal, since if $\{1\}$ is added, the resulting digraph $\langle S \cup \{1\} \rangle$ is not intransitive. For all digraphs G in Q and for all m in N, let h(m,G) be the number of maximal induced weakly connected intransitive subgraphs having m + 2 points.

<u>Definition 6.11.</u> If G is in Q and $S \subseteq V(G)$, the $\langle S \rangle$ is the <u>maximal induced weakly connected transitive subgraph</u> iff it is weakly connected and transitive and there is no point v in V(G) - S such that $\langle S \cup \{v\} \rangle$ is weakly connected and transitive.

<u>Example 6.4.</u> Consider the digraph of the previous example. Let $S = \{1,2,3\}$ then $\langle S \rangle$ is the following digraph:

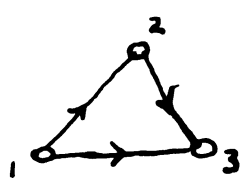


Fig 6.6.

Now $\langle S \rangle$ is clearly transitive and weakly connected. It is also maximal since if $\{4\}$ is added the resulting digraph, $\langle S \cup \{4\} \rangle$, is not transitive. For all graphs G in Q and for all m in N, let h(m,G) be the number of maximal induced weakly connected transitive subgraphs having m + 2 points.

Now let D = L$\times$ L - (0,0), there is no digraph in Q such that its sequence pair does not belong to D. It is evident that for each sequence pair $(S,\overline{S})$ in D there is a digraph G in Q such that $(S,\overline{S})$ is the sequence pair of G. This is proved in the following theorems.

Theorem 6.2. Every member of L is the transitivity (intransitivity) sequence of some member of Q. (S denotes the transitivity sequence and $\overline{S}$ the intransitivity sequence).

Proof

Let $S = (a_1,a_2,\ldots)$ be the sequence in L, for each nonzero term a_i make a_i copies of the transitive tournament on i + 2 points. Since there are only finitely many nonzero terms, the resulting structure is a digraph. S will be its transitivity sequence. Let $\overline{S} = (b_1,b_2,\ldots)$ be a sequence in L, for each nonzero term b_i make b_i copies of the cycle in i + 2 points. Since there are only finitely many nonzero terms, the resulting structure is a digraph. $\overline{S}$ will be its intransitivity sequence.

Corollary Each pair $(S,\bar{S})$ in D is the sequence pair of some digraph in Q.

Proof

Combine the constructions in the proof of Theorem 2 and note that (0,0) is not in D.

Let f be any ratio function and d and d' two strong norms on L Q may be ordered by its f-values. By the results in the previous chapter, the following theorems are evident. Let G_1 and G_2 be in Q, then $G_1 F G_2$ iff $f(S_1,\bar{S}_1,d,d') = f(S_2,\bar{S}_2,d,d')$ where $(S_1,\bar{S}_1)$ and $(S_2,\bar{S}_2)$ are the sequence pairs of G_1 and G_2 respectively. Let W be the collection of equivalence classes of F.

Theorem 6.3. $(W, \preceq)$ is a distributive lattice.

Theorem 6.4. $(W, \preceq)$ is not complemented.

In view of the last chapter, it is evident that the order derived for digraphs in Q is dependent on the choice of d and d'. From the last chapter, it is evident that

$$d(S) = \sum_{n=1}^{\infty} g(n)s_n \qquad \text{and} \qquad d'(S) = \sum_{n=1}^{\infty} g'(n)s_n$$

where g and g' are functions from N into R. This being the case, d and d' may be described in terms of g and g'. It would seem reasonable to choose both g and g' as some kind of increasing function. Perhaps, $g(m) = m^p$, for all m. For example, suppose it is desirable to have the order depend more upon the transitivity than the intransitivity of a digraph, then the following definitions could be made:

$$d(A) = \sum_{n=1}^{\infty} n^2 a_n \qquad \text{and} \qquad d'(A) = \sum_{n=1}^{\infty} n\, a_n \quad .$$

The function in this case would be $f(S,\bar{S},d,d') = \dfrac{d(S)}{d'(\bar{S})}$.

6.4. A MEASURE OF SYMMETRY

Another property of digraphs which might be of some interest to a psychologist of sociologist is that of symmetry.

Example 6.5. Suppose there are 5 couples in an apartment complex. A psychologist might ask each couple to list their friends. The psychologist might represent their responses by the following digraph:

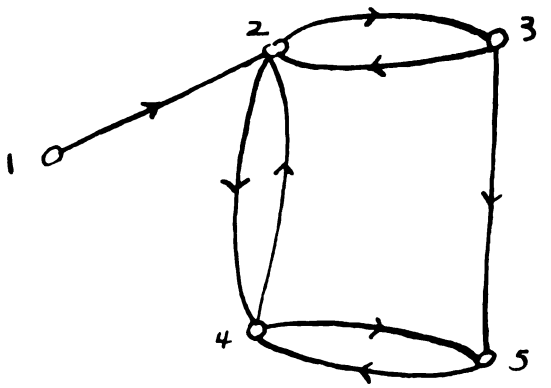


Fig 6.7.

A natural question to ask would be how symmetric is the digraph? Symmetry in this case, would indicate a friendship was mutual.

Definition 6.12. Let P be the class of all finite digraphs with a least one edge.

Definition 6.13. If G is in P and $S \subseteq V(G)$, then $\langle S \rangle$ is a maximal induced weakly connected symmetric subgraph iff it is weakly connected and symmetric and there is no point v in V(G) - S such that $\langle S \cup \{v\} \rangle$ is weakly connected and symmetric.

Example 6.6. Consider the following digraph:

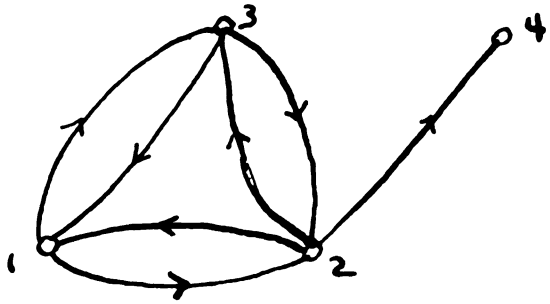


Fig 6.8

Let $S = \{1,2,3\}$ then $\langle S \rangle$ is the following digraph:

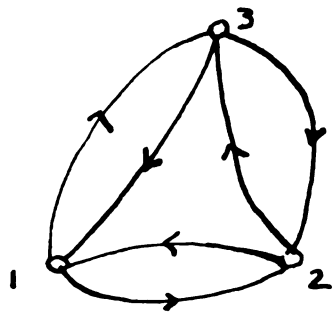


Fig 6.9.

Now $\langle S \rangle$ is clearly symmetric and weakly connected. It is also maximal since if $\{4\}$ is added, the resulting digraph is not symmetric. For all digraphs G in P and for all m in N, let h(m,G) be the number of maximal induced weakly connected symmetric subgraphs having m + 1 points.

Definition 6.14. If G is in P and $S \subseteq V(G)$, then $\langle S \rangle$ is a maximal induced weakly connected asymmetric subgraph iff it is weakly connected and asymmetric and there is no point v in V(G) - S such that $\langle S \cup \{v\} \rangle$ is weakly connected and asymmetric.

Definition 6.15. For all digraphs G in P and for all m in N, let $\bar{h}$ (m,G) be the number of maximal induced weakly connected asymmetric subgraphs having m + 1 points.

Now let $D = L \times L - (0,0)$. There is no digraph in P such that its sequence pair does not belong to D. For each sequence pair $(S,\overline{S})$ in D there is a digraph G in P such that $(S,\overline{S})$ is the sequence pair of G. This is evident from the following theorem:

Theorem 6.5. Every member of L is the symmetry (asymmetry) sequence of some digraph (S denotes the symmetry sequence and $\overline{S}$ is the asymmetry sequence.

Proof

Let $S = (s_1, s_2, \ldots)$ be a sequence in L, for each nonzero term s_i make s_i copies of the digraph for which $E(G) = V(G) \times V(G)$, and for which $|V(G)| = i + 1$. Since there are only finitely many nonzero terms, the resulting structure is a digraph. S will be its symmetry sequence. Its asymmetry sequence will be 0. Let $\overline{S} = (\overline{s}_1, \overline{s}_2, \ldots)$ be a sequence in L, for each nonzero term $\overline{s}_i$ make $\overline{s}_i$ copies of a tournament having $i + 1$ points. Since there are only finitely many nonzero terms, the resulting structure is a digraph. S will be its asymmetry sequence. Its symmetry sequence will be 0.

Corollary. Each pair $(S,\overline{S})$ in D is the sequence pair of some digraph in P.

Proof

Combine the constructions in the proof of the above theorem and note

that (0,0) is not in D.

Let f be any ratio function and d and d' be strong norms on L. P may be ordered by its f-values. By the results in the last chapter, the following theorems are evident. Let G_1 and G_2 be in P, then $G_1 F G_2$ if and only if

$$f(S_1,\bar{S}_1,d,d') = f(S_2,\bar{S}_2,d,d')$$

where $(S,\bar{S}_1)$ and $(S_2\bar{S}_2)$ are the sequence pairs of G_1 and G_2 respectively. Let W be the collection of equivalence classes of F.

Theorem 6.6. $(W, \leq)$ is a distributive lattice.

Theorem 6.7. $(W, \leq)$ is not complemented.

In view of the theory in the last chapter, it is evident that the order derived for the digraphs in P is very dependent on the choice of d and d'. The theory developed in the last chapter implies:

$$d(S) = \sum_{n=1}^{\infty} g(n)s_n \quad \text{and} \quad d'(S) = \sum_{n=1}^{\infty} g'(n)s_n$$

where g and g' are functions from N into R. This being the case, d and d' can be described in terms of g and g'. It would seem reasonable to choose g and g' as some kind of increasing functions.

For example, suppose it is desirable to have the order depend more on symmetry than asymmetry, then the following definition could be made:

$$d(A) = \sum_{n=1}^{\infty} n^2 a_n \qquad \text{and} \qquad d'(A) = \sum_{n=1}^{\infty} n\, a_n \quad .$$

The function f in this case would be

$$f(S,\bar{S},d,d') = \frac{d(S)}{d(S)} \quad .$$

Chapter 7

SUMMARY AND FUTURE RESEARCH

7.1 SUMMARY

Chapter One lays the foundation for the development of the thesis. A review of current literature shows the need for a definition of process which is both mathematically precise and practically significant. The need for measurement in a process and the relationship between this measurement and digraph theory is also suggested.

In Chapter Two, the concept of process is formally defined. The decomposition theory of Hartmanix and Stearns (HART 66) is extended to process and the problem of nontrivial parallelism is also addressed. It is shown that in the case of nontrivial parallelism, the best results obtainable are sufficient conditions for decomposition.

In Chapter Three, a digraph is associated with each process. This digraph is used to obtain sufficient conditions for possession of an infinite computation.

In Chapter Four, the concept of measurement of the properties of a process is discussed. Three measures which appear

to have some practical significance are suggested; the notion of length which corresponds to time, the notion of width which may be thought of as the number of processors needed, and the notion of work which measures how many changes are made in the state variable space.

In chapter Five, the measurement techniques of Chapter Four are generalized to digraphs. Chapter Six shows that two previously defined measures are special cases of the theory in Chapter Five and develops two new measures which may prove useful to social scientists.

7.2. FUTURE RESEARCH

The concept of indexed variable and its relationship to the DO while clause is an important topic for future research. It is evident that the results discussed in (LAMP 74) may be shown to hold for processes. Hopefully, some results which are less trivial are obtainable.

There is also a need for the study of the relationship between DO while clauses and If clauses within a process. For example, the basic diagram:

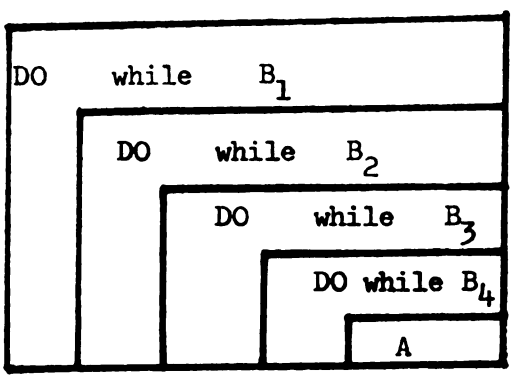


Fig 7.1

may be replaced by the diagram:

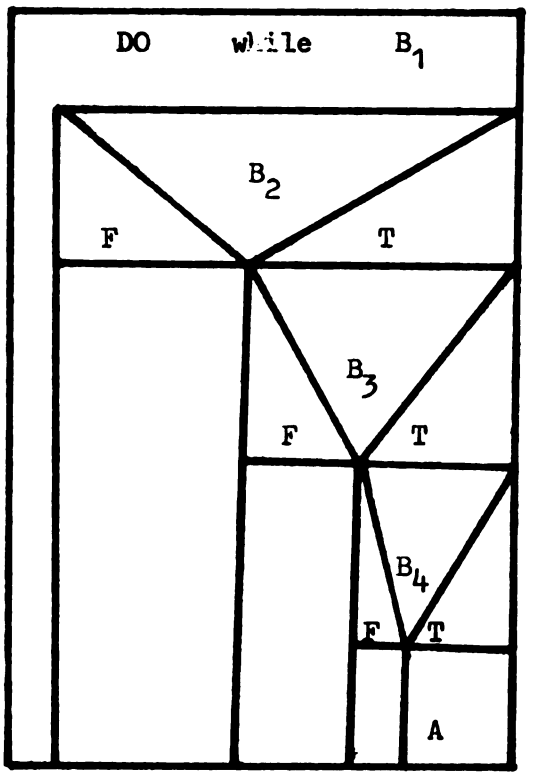


Fig 7.2

provided assignments to the state variables in B_2, B_3, and B_4 only occur in the last action of the basic diagram A. The task of finding similar relationships in a more complicated nesting structure is an open question. A reasonable conjecture to make at this point is that any nesting of DO while clauses can be replaced by one DO while clause and an appropriate number of If clauses. A concept of nesting depth of an action may be introduced. This concept is similar to the concept of star heights. The previous conjecture implies that if one is allowed complete freedom in Boolean expressions, any process can be rewritten as an equivalent

process with nesting depth one. Perhaps if the use of logical operators in the Boolean expressions is restricted in some way, a process which cannot be rewritten as an equivalent process with a lower nesting depth can be exhibited for any depth n.

The concept of process is defined in this thesis is deterministic. Although intuitively one would want a computer to behave deterministically, it is sometimes convenient to introduce nondeterminism. Some notation for the introduction of nondeterminism into basic diagrams might be developed.

Applications of the measurement theory developed in Chapter Five are another area for future research. Almost any binary concept may now be measured in a more sophisticated manner.

APPENDIX

APPENDIX

As every digraph is a relation, digraphs may be characterized in terms of relations.

Definition 1. A digraph is symmetric (asymmetric) if it is a symmetric (asymmetric) relation.

Definition 2. A digraph is reflexive (irreflexive) if it is a reflexive (irreflexive) relation.

Definition 3. A digraph is transitive (intransitive) if it is a transitive (intransitive) relation.

Definition 4. A digraph is complete if it is a complete relation.

Definition 5. A graph is a symmetric irreflexive finite digraph.

Definition 6. A tournament is a complete assymmetric irreflexive finite digraph.

In addition to the above definitions, a number of definitions which deal with lines and vertices of a digraph are needed.

Definition 7. The outdegree of vertex v is the number of lines from v.(i.e. - lines x for which f(x) = v)

Definition 8. The indegree of vertex v is the number of lines to v.(i.e. - lines x for which s(x) = v.)

Definition 9. A vertex u is adjacent to a vertex v if there is a line x such that f(x) = u and s(x) = v.

Definition 10. A vertex u is adjacent from a vertex v if there is a line x such that f(x) = v and s(x) = u.

Definition 11. A vertex is isolated if both its indegree and outdegree are zero.

Much of digraph theory deals with the concepts of joining and reaching. The following definitions make these concepts more precise.

Definition 12. A semiwalk joining v_1 and v_n is a collection of vertices $v_1, v_2, \ldots, v_n$ together with one from each pair of lines v_1v_2 or v_2v_1, v_2v_3 or v_3v_2, $\ldots, v_{n-1}v_n$ or v_nv_{n-1}.

Definition 13. A semipath is a semiwalk in which the points are distinct.

Definition 14. A walk joining v_1 and v_n is a collection of vertices $v_1, v_2, \ldots, v_n$ together with the lines $v_1v_2, v_2v_3, \ldots, v_{n-1}v_n$.

Definition 15. A path is a walk in which the points are distinct.

Definition 16. If there is a path from u to v then v is reachable from u.

Definition 17. A digraph is strongly connected if any two vertices are mutually reachable.

Definition 18. A digraph is unilaterally connected if for any two

vertices at least one is reachable from the other.

Definition 19. A digraph is weakly connected if every two vertices are joined by a semipath.

Definition 20. A digraph is disconnected if it is not even weak.

Definition 21. A digraph is rooted if it has a vertex u such that every other vertex is reachable from u.

Definition 22. A cycle (semicycle) is a path (semipath) with the same beginning and end vertex.

Definition 23. A tree is a rooted irreflexive finite digraph with no semicycles.
(This definition corresponds to what computer scientists call a tree rather than what graph theorists call a tree.)

Definition 24. A binary tree is a tree in which the outdegree of each vertex is at most two.

Definition 25. A labeled digraph is a digraph in which each line has associated with it a symbol.

Definition 26. A signed digraph is a labeled digraph in which the symbols are and - .

Another concept that will be of importance later in this thesis is that of subgraph of a digraph.

Definition 27. A digraph D' is a subgraph of a digraph D if its vertices and lines are vertices and lines of D.

Definition 28. Let S be some subset of the vertices of a digraph D, then the induced subgraph $\langle S \rangle$

is the subgraph of D whose vertices are S and in which there is a line between any two points in S iff there is a line between these points in D.

LIST OF REFERENCES

LIST OF REFERENCES

(BAER 68) BAER,J.L.;AND D.P.BOVET. "Compilations of arithmetic expressions for parallel computations." Proc. IFIP Congress 1968, North-Holland, Amsterdam, The Netherlands, 340-346.

(BERN 66) BERNSTEIN,A.J. "Analysis of programs for parallel processing." IEEE Trans. Electron. Comput,EC-15,(Oct.1966), 757-762.

(BING 67) BINGHAM,H.W.;D.A.FISHER;AND E.W.REIGEL. "Automatic detection of parallelism in computer programs." TR-ECON-02463-4, Burroughs TR-67-4, AD 622 274,(Nov. 1967).

(BLUM 67) BLUM,M. "A machine independent theory of the complexity of recursive functions." JAMC, 14, (1967(,322-336.

(BRIN 70) BRINCH,HANSEN,P. "The nucleus of a multiprogramming system." Comm. ACM 13, 4, (1970), 238-241,250.

(CHUR 36) CHURCH,A. "An unsolvable problem of elementary number theory." Amer.J. Math.,58, (1936), 345-363.

(DAHL 66) DAHL,O.J.; AN NYGAARD,K. "Simula, an ALGOL-based simulation language." Comm. ACM 9,9, (1966), 671-678.

(DIJK)68) DIJKSTRA,E.W. "Co-operating sequential processes." Programming languages, F. Genuys(Ed.),Academic Press, New York, (1968),43-112.

(GILB 72) GILBERT,P.;AND W.J. CHANDLER. "Interference between communicating parallel processes." Comm.ACM 15,6,(June 1972), 427-437.

(GONZ 70) GOLZALEZ,M.J.;AND C.V.RAMAMORTHY. "Recognition and representation of

parallel processable streams in computer programs." Parallel Processor Systems, Technologies and Applications. L.C. Hobbs, et al (Eds.), Spartan Books, New York, (1970), 335-374.

HARA 65) HARARY,F.;R.Z. NORMAN;AND D. CARTWRIGHT. Structural Models: An Introduction to the theory of Directed Graphs,=Wiley, New York, (1965).

(HART 64) HARTMANIS,J.; AND R.E. STEARNS. "Computational complexity of recursive sequences.: Proceedings of the Fifth Annual Symposium on Switching circuit Theory and Logical Design. Princeton, New Jersey, (1964), 82-90.

(HART 65) HARTMANIS.J.;AND R.E. STEARNS. "On the computational complexity of Algorithms." Trans.Amer.Math.Soc,,117,(1965), 285-306.

(HART 65_b) HARTMANIS.J.;P.M. LEWIS II;AND R.E. STEARNS. "Hierarchies of memory limited computations." IEEE Conference Record on Switching Circuit Theory and Logical Design, Ann Arbor, Michigan, (1965), 179-190.

(HART 66) HARTMANIS,J.; AND R.E. STEARNS. Algebraic Structure Theory of Sequential Machines, Prentice-Hall,Inc.,Englewood Cliffs, New Jersey, (1966).

(HORN 73) HORNING,J.J.; AND B. RANDELL."Process structuring." ACM Computing Surveys, 5, no. 1(march 1973), 5-29.

(KEME 62) KEMENY,J.G.; AND J.L. SNELL. Mathematical Models in the Social Sciences, Chapter 2, Blaisdell, New York, (1962).

(KNUT 66) KNUTH, D. "Additional comments on a problem in concurrent programming control." Comm. ACM 9, 5, (May 1966).

LAMP 74) LAMPORT, L. "The parallel execution of DO loops." Comm. ACM 17, 2, (Feb. 1974).

(LEHM 66) 83-93.

(LEHM 66) LEHMAN, M. "A survey of problems and preliminary results concerning parallel processors." Proc. IEEE, 54, (Dec. 1966), 1889-1901.

(LEWI 71) LEWIS,F.D. "The enumerability and invariance of complexity classes." J. of Comp. and Sys, Sciences 5, 3, (June 1971), 286-303.

(MANN 70) MANNA, Z. "Termination of programs represented as interpreted graphs." Proc. AFIPS 1970 Spring Joint Computer Conf., AFIPS press, Montvale, New Jersey, 83-89.

(MCCA 60) MCCARTHY,J. "Recursive functions of symbolic expressions and their computation by machine.: Commun. Assoc. Computing Machinery, 3, no. 4, (1960) 184.

(NASS 73) NASSI,I.; AND B. SHNEDERMAN. "Flowchart techniques for structured programming. SIGPLAN Notices, (August 1973), 12-26.

(NORM 72) NORMAN, . .; AND F.S. ROBERTS. " A derivation of a measure of relative balance for social structures and a characterization of extensive rationsystems." J. of Math. Psych. 9 (1972). 66-91.

(PALM 74) PALMER,E.M.;M. RAHIMI; AND R.W. ROBINSON. "Efficiency of a binary comparison storage technique." (to appear).

(RAMA 69) RAMAMOORTHY,C.V.; AND M.J. G ZALEZ. "A survey of techniques for recognition parallel processable streams in computer programs. programs.: Proc. AFIPS 1969 Fall Joint Computer Conf., AFIPS press, Montvale, N.J. 1-15.

(RUSS 69) RUSSELL,E.C. " Automatic program analysis.: PhD Dissertation, Dept. of Electrical Engineerying, Univ. of Calif. Los Angeles, Calif., 1969.

(SALT 66) SALTZER,J.H. "Traffic control in a multiplexed computer system" PhD Dissertation, Massachusetts Institute of Technology, 1966.

(TURI 37) TURING,A.M. "On computable numbers, with an application to the entscheidungs-problem." Proc. London Mathematical Society 2, 42, (1936-37)

(TURS 68) TURSKI,W.,M. "SODA - adual activity operating system.: Computer J. 11, 2, (1968), 148-156.

(VANW 69) VAN WIJNGAARDEN,A.;B.J. MAILLOUX, J.E.L.PECK; AND C.H.A.KOSTER. "Report on the algorithmic language ALGOL 68." Mathematisch Centrum. M R 101, Amsterdam, (Oct. 1969).

(VOLA 70) VOLANSKY,S.A. "Graph model analysis and implementation of computational sequences.: PhD Dissertation, Dept. of Electrical Engineering, Univ. of California, Los Angeles, Calif., 1970.