COMPUTER PROGRAMS AND GRAPH TRANSFORMATIONS

by

David C. Cooper

Carnegie Institute of Technology
Pittsburgh, Pennsylvania
September 19, 1966

This work was supported by the Advanced Research Projects Agency of the Office of the Secretary of Defense (SD-146).

Part I was presented at the Machine Intelligence Workshop held at Edinburgh University from 27th June to 1st July, 1966. The proceedings will be published by Oliver and Boyd Ltd., Edinburgh.

Part II was presented at the "International Seminar on Graph Theory" held in Rome, Italy from 5th July to 9th July, 1966, and will be published in "Actes des Journees d'Etudes sur la Theorie des Graphes" ICC, Dunod 1967.

ABSTRACT

Each part is complete in itself, which of course necessitates some overlap.

Part I describes the general approach and motivations of this research. It is assumed that with a computer program we may associate a directed graph (its "flow chart"). Several transformations of a directed graph are defined with the property that they "obviously" do not affect the meaning of the program. Two particular standard forms for graphs are described and results and conjectures given concerning the possibility of transforming graphs into these standard forms by using the defined graph transformations.

Part II gives the mathematical proofs of the theorems stated in Part I without proof and completely solves the problems concerning one of the defined standard forms.

TABLE OF CONTENTS

PART I

SOME TRANSFORMATIONS AND STANDARD FORMS OF GRAPHS, WITH APPLICATIONS TO COMPUTER PROGRAMS

Introduction: Programs and Directed Graphs

In Cooper (1966b) I reported on several attempts at proving theorems about computer programs, with the ultimate goal in mind of providing mathematical proofs that programs are correct rather than just testing them on some particular data sets. In the last section I commented that the proofs I obtained in Cooper (1966a) could be viewed as deep results about small programs but that what was needed were techniques for proving shallow results about large programs. By this latter I meant that the proofs did not depend too much on properties of the basic functions or commands used in the program but rather depend on the synthesis of a large number of trivial properties, the particular way this synthesis is performed being guided by the structure of the large program. Some of the other work reported on in Cooper (1966b), particularly the work of Evans (1965), has this flavour but are particular proofs about particular programs. Mention should also be made of the long proof of Balzer (1966) which proves that a particular finite automaton correctly performs its given task.

There are many programs whose control structure can be well shown by a directed graph - that is the flow chart of the program. The first goal of this research is to produce some kind of automatic scheme which will prove results about programs which only use conditional statements and assignment statements and we shall not consider those features of a programming language which imply some kind of control structure not immediately so representable; for example, the passing of parameters to subroutines, perhaps by the use of procedures and functions previously defined by the programmer. These are very important features of programming languages but they will not be further considered in this paper. Although the techniques of this paper may not be directly applicable in these circumstances some modification of the ideas used could well be useful; for example, the graphs could represent the relations between a set of mutual recursive function definitions and the transformatiors to be defined later could well be meaningful.

Having associated a graph with a program there may well be transformations which can be made on the graph which "do not affect" the program, later we shall amplify this remark. The question then arises as to whether the graph may be put into some standard form by a series of such transformations, this standard form being one in which it is easier to prove results about the program by taking advantage of special features of the standard form. The purpose of this chapter is to define several such transformations and two such standard forms, and to give results and conjectures about the possibility of transforming programs to the standard forms.

The particular way in which a directed graph is associated with a given program is not important. All that matters is that the graph transformations defined in the next section should represent meaningful transformations if performed on the programs with which the graphs are associated. However, in order to better illustrate the intention behind the transformations,we give a particular definition of a program and of the graph associated with a program. This definition will use a state vector approach; see McCarthy (1960). In this approach the current state of a computation is represented by the value of a vector, each component of which corresponds to one of the variables, machine locations, etc. (dependent on the particular programming language used). The effect of a basic command can then be described by a function whose argument is a state vector (the state of the machine before the command is obeyed) and whose value is also a state vector (the state of the machine after the command is obeyed). A sequence of commands corresponds to composition of the corresponding functions. A two-way test is represented by a predicate whose argument is a state vector.

Assume then that we have a domain D, a set of functions f_i which map D into D and a set of predicates p_i whose domains are D. A <u>program</u> is defined to be a directed graph with every arc of which is associated one of the f functions and one of the p predicates. It is a <u>legal program</u> if the graph has just one node (A, say) with no arcs leading to it, if it has just one node (Z, say) with no arcs leading from it, if every node is on a path which starts at A and ends at Z, and if for all nodes (except Z) the set of predicates on all arcs leading from the node is complete and mutually exclusive.

The intended interpretation is that if f_i and p_j are the function and predicate associated with some arc then f_i represents the effect of all the statements obeyed along that arc and p_j represents the condition that the arc is entered. A program then defines a partial function from D into D in an obvious manner (partial because the program may loop indefinitely).

It should again be emphasised that we are not concerned with this particular definition of a program, only with the ability to associate a graph with a program. As alternatives we could have associated the information in the program with the nodes rather than the arcs, or we could have taken some simple programming language and explicitly stated how to obtain the graph from the program.

The results to be obtained later could well have applications in other areas than computer programs. For example, Seshu and Reed (1961) describe the application of graphs to sequential machines and to systems of linear algebraic equations. All our transformations can be given interpretations in these areas, in fact the delete node transformation (T 5 of the next section) is already well known in these applications. The particular field of interest determines the transformations of interest; for example, later we shall reject a certain transformation (DT 1) as not being a useful one for computer programs. However, this transformation is of use in the areas of sequential machines and of linear equations.

Examples of Transformations

In this section we define several directed graph transformations of interest. In each case we give a description of the general case in English and illustrate the transformation with a particular case. The illustration of course only shows how the affected part of the graph is altered, the graph may have other nodes and arcs except that in all cases we assume that there are no further arcs which lead into or out of the nodes labelled N.

T 1 Stretch a node N by outputs.

Let O be some subset of the output nodes of N, this subset may be all the output nodes but must not be empty. Delete all arcs from N to nodes in O, add a new node N_1 and add arcs from N to N_1 and from N_1 to all nodes in O. (The output nodes of a node N are all those nodes P such that there is an arc NP, including N itself if there is an arc NN).

In the example O is the set $\{N,D,E\}$.

T 2 Stretch a node N by inputs.

Let I be some subset of the input nodes of N; this subset may be all the input nodes but must not be empty. Delete all arcs from nodes in I to N, add a new node N_1 and add arcs from N_1 to N and from all nodes in I to N_1. (The input nodes of a node are defined in an analogous way to the output nodes.)

In the example I is the set $\{A,B\}$.

T 3 Duplicate a node N.

Let I be some non-empty, proper subset of the input nodes of N, not including N itself if NN is an arc. Delete all arcs from nodes in I to N, add a new node N_1, add arcs from N_1 to all the output nodes of N (except N itself if NN is an arc) and add arcs from every node in I to N_1. If NN is an arc also add an arc N_1 N_1.

In the example I is the set $\{A,B\}$.

T 4 Delete an arc AN.

This transformation may only be made if there is no arc NN. Delete the arc AN and add arcs from A to all the output nodes of N. If A was the only input node of N then also delete node N and all arcs leading from N.

T 5 Delete a node N.

This transformation may only be made if there is no arc NN and is equivalent to applying a sequence of T 4 transformations in any order to all the arcs leading to N. Add all possible arcs from an input node of N to an output node of N and delete node N and all arcs leading to or from N.

In all these cases if the untransformed graph is the graph of some program P it is easy to find an equivalent program with the transformed graph as its associated graph. For example, in the diagram illustrating transformation T 1 if $p_{AN}(s)$ and $f_{AN}(s)$ are the predicate and function associated with arc AN on the untransformed graph (and similarly for the other arcs) then in the transformed graph on arc NN_1 we have the predicate $p_{ND}(s) \vee p_{NE}(s) \vee p_{NN}(s)$ and the identity function, N_1N, N_1D and N_1E have the same predicates and functions as NN, ND and NE respectively, and all other arcs have the same predicates and functions as before the transformation. Transformation T 4 represents the effect of anticipating the test at N at the previous node A so that on the transformed graph, for example, with the arc AC we associate the predicate $p_{AN}(s) \wedge p_{NC}(f_{AN}(s))$ and the function $f_{NC}(f_{AN}(s))$.

As two examples of transformation which we do not wish to allow consider the following:

DT 1 Delete a node N with a loop.

This is defined exactly as T 5 but we remove the restriction that there must be no arc NN.

DT 2 Join two arcs AB and CD.

Delete the arcs AB and CD, add a new node N and add arcs AN, NB, CN and ND.

In DT 1 with arc AC we would have to associate a predicate expressing the condition that when control finally comes out of the loop at N control passes to node C. This can be done, but in general requires an infinite disjunction, we assume this is not a useful concept to introduce. The aim in making the transformation is to better reveal the program's structure, we do not wish to produce a transformed program which introduces complex new concepts and makes our task of proving the desired theorem about the program much more difficult. Admittedly we are being vague here, the transformations T 1 to T 5 imply we do not mind complicating our programs by forming conjunctions and disjunctions of predicates and compositions of functions. Note that the situation arising in DT 1 would not be improved by adding loops on the transformed graph at A and B, or at new nodes internal to the arcs AC, AD, BC and BD.

We could allow the transformation DT 2 if an extra Boolean variable is defined and semantics added to the program to set this variable true along arc AN, false along arc CN. The predicate on arc NB would then be a test whether this variable is true, on arc ND whether it is false. In effect we are coding the path already taken on the graph into a variable. Again we assume that this is not a good thing to do.

In this chapter we define T 1 to T 5 as the only allowable transformations, and then prove or conjecture theorems concerning them. We would like to somehow characterize the allowable transformations and perhaps produce some more. A possible approach is to define a transformation of one graph into another by a definition of the following form:

$G \rightarrow G'$ if and only if $(p)(\exists p')[G=\text{graph}(p)\supset\{G'=\text{graph}(p')\wedge f(p,p')\}]$. This definition presupposes that we already have a definition of a program, of a function "graph" which produces the graph of a program and of a predicate f(p,p'). This predicate must express the condition that program p' is equivalent to program p, and moreover can be produced from it with some given rules which allow such things as function composition but does not allow anything corresponding to the infinite disjunction of DT 1 or the new Boolean variable of DT 2.

Proper Cycle Free Form

Having defined the allowable graph transformations are there some particularly desirable standard forms into which we can try to transform a given graph? Two desirable forms will be defined, one in this section and one in the following section.

Most, if not all, of the questions we wish to answer about a computer program are easy to answer if there are no cycles in the program. The simpler the interconnections between the cycles the easier it should be to answer questions about whether a program performs correctly; it should therefore be profitable to make transformations aimed at simplifying the cycle structure of a graph.

The simplest (and also rarest) program is one which has no cycles. Next one might consider programs in which all the cycles are independent, i.e. no section of the program is in more than one cycle. This class of programs is more interesting but still very restrictive. However, experience indicates that a large number of interesting programs may be transformed by a series of transformations into a program with this independent loop structure.

In order to make these vague remarks more exact we define a proper cycle of a graph to be a cycle through two or more nodes and a proper cycle free (PCF) graph to be a graph with no proper cycles. A graph which is PCF clearly has the independent looping property, and it is also trivial to transform any graph with independent loops into PCF form. We therefore take PCF form as being a standard form into which we try to transform a given graph, and obtain necessary and sufficient conditions for a graph to be transformable into PCF form together with an algorithm for making the transformation if it is possible. For this purpose only transformation T 5, delete a node, turns out to be relevant. If a graph can be transformed to PCF form by a sequence of transformations, each of which is one of T 1 to T 5, then this may be done by a sequence of T 5 transformations alone.

Consider graph 1. By deleting node A we obtain graph 2 which is not PCF (and which in fact cannot be transformed to PCF form at all; see the theorem below). But if instead of deleting A we delete B and then C we obtain graph 3 which is PCF and "obviously much simpler" than graph 2. (The 2's on graph 3 indicate multiple arcs.) Graph 3 reveals the "basic structure" of graph 1 in a way which graph 2 does not. These graphs show that the order in which nodes are deleted is critical.

Define a graph to be loop connected if it contains two nodes, A and B say, and three cycles α, β and γ such that: 1) there is no one node which is on α, β and γ, 2) A is on α but not on β, 3) B is on β but not on α, and 4) both A and B are on γ. This is a generalization of part of graph 2 in which arcs have been replaced by paths. Then we have:

Theorem 1

A graph G can be transformed by a series of transformations, each of which is one of T 1 to T 5, to a graph which is PCF if and only if it is not loop connected.

In Cooper (1966c) only transformation T 5 is considered but it is easy to extend the results of that paper in such a way that theorem 1 is an immediate deduction from the theorems proved there.

Theorem 1 does not give an algorithm for performing the sequence of transformations. In order to do this define a delete node N transformation (T 5) to be a safe deletion if there is some node A (not N) of the untransformed graph with the property that all cycles of the graph which pass through node N also pass through node A. For example, in graph 1 B or C may be safely deleted but not A. Then the following theorem gives the required algorithm; it is an easy extension of one of the theorems in Cooper (1966c):

Theorem 2

Start with any graph G and perform safe deletions in any order until a graph G' is obtained on which no safe deletions may be performed. This process must terminate and either G' is PCF or there is no sequence of transformations, each of which is one of T 1 to T 5, which will transform G to a graph in PCF form.

Theorems 1 and 2 completely solve the relevant questions about transformations to PCF form. In the same paper an equivalent condition to loop connectedness is defined; this condition appears more complicated but simplifies the proofs. In addition a theorem is proved which shows that safe deletions is the widest possible subclass of deletions, i.e. if we make a deletion which is not a safe deletion we obtain a graph which is loop connected and hence not transformable to PCF form.

Block Form

The second standard form corresponds in a more natural manner than PCF form to the way a large number of programs are written; it roughly corresponds to programs in which all loops are properly nested. A recursive definition of block form is simply given: - a graph is defined to be in block form if it is of one of the forms B1, B2 or B3 (see the diagrams) where the shaded boxes are either arcs or subgraphs in block form.

It is not necessary to take exactly these forms; for example, instead of B2 we may allow any number of blocks in parallel and instead of B3 any number in series, obviously a graph of such form can easily be transformed into the standard block form. A more useful equivalent standard form is where instead of B2 and B3 we allow any graph made up of subgraphs, each in block form, and these subgraphs joined in any manner that does not create a cycle. Such a graph can be further transformed into the standard block form, but it may well be better not to carry out these further transformations as this usually involves a lot of duplicating of nodes. A third alternative is to replace B1 by B1a.

Very little progress has been made so far with proving results about transformation of graphs to block form, although a large number of small graphs have been investigated. A conjecture has been made as to the necessary and sufficient conditions for a graph to be transformable to block form and we hope soon to have an algorithm to effect the transformations. Any graph in PCF form can be transformed to block form; essentially it already is in block form according to the second alternative definition. However, consider graph 4, which is clearly loop connected and hence cannot be transformed into PCF form. First use T 1 (with N being A and O the set $\{B,Z\}$) and then use T 3 (N being A again and I the set $\{B\}$), this produces graph 5 and it is obvious how to put this into block form. Thus graph 4 is an example of a graph which can be transformed to block form but not to PCF form. As a more complex example, graph 6 may be transformed to block form; this is left as an exercise for the reader; a good start is to delete node A.

What graphs cannot be transformed to block form? Here we have no proven results but it seems almost certain that graph 7 cannot be so transformed (or any graph containing this as a subgraph). The situation is much more complex than for the PCF case; there a knowledge of just the cycles of a graph was sufficient to determine whether the graph could be transformed to PCF form, but graphs 4 and 7 have the same cycles and one can be transformed to block form but not the other. It seems that a generalization of graph 7 in which its arcs are replaced by paths (possibly intersecting each other) is the key situation which, if it occurs in a graph, makes the graph untransformable to block form. But examples show that if enough of the paths intersect then we are back to the position of being able to transform to block form.

By induction from examples tried we make the following definition:

A graph is loop connected with two exits if it contains three nodes, A, B and Z say, a cycle α through A, a cycle β through B, a path γ connecting A to B, a path δ connecting B to A, a path ϕ connecting A to Z and a path ψ connecting B to Z (see graph 8). Further there must be no node which is on all three members of any of the following triples:

(α,β,γ), (α,β,δ), (δ,α,ψ), (γ,β,ϕ), (ϕ,ψ,α) and (ϕ,ψ,β).

We conjecture the following:

Theorem 3

A graph can be transformed by a series of transformations T 1 to T 5 to a graph which is in block form if and only if it is not loop connected with two exits.

Böhm and Jacopini (1966) investigate the reduction of flow diagrams to standard forms, they show that corresponding to any flow diagram there is an equivalent one which is "decomposable into Π, Φ and Δ". This is precisely block form. Their algorithm for producing this equivalent flow diagram may be looked on as a sequence of graph transformations which, however, allow the transformation DT 2, which we have not allowed. Transformation DT 2 on programs was justified by the introduction of a new variable, rather than introducing a separate new variable everytime DT 2 is used Böhm and Jacopini introduce a single Boolean stack and add semantics to test the top of the stack, to delete the top of the stack and to add a new truth value to the top of the stack. This stack may be thought of as a code for the path through the flow diagram traversed so far. For our purposes we do not wish to allow this transformation, but it is an interesting result that if DT 2 is allowed then any graph may be transformed to block form.

References

Balzer, R.M. (1966). Studies Concerning Minimal Time Solutions to the Firing Squad Synchronization Problem. Ph.D. thesis. Pittsburgh: Carnegie Institute of Technology.

Böhm,C.&Jacopini,G.(1966).Flow Diagrams, Turing Machines and Languages with Only Two Formation Rules. Comm. ACM, 9, 366-371.

Cooper, D.C. (1966a). The Equivalence of Certain Computations. Computer J., 9, 45-52.

Cooper, D.C. (1966b). Mathematical Proofs About Computer Programs. To appear in Machine Intelligence 1, Editor Michie, D. Oliver and Boyd Ltd., Edinburgh.

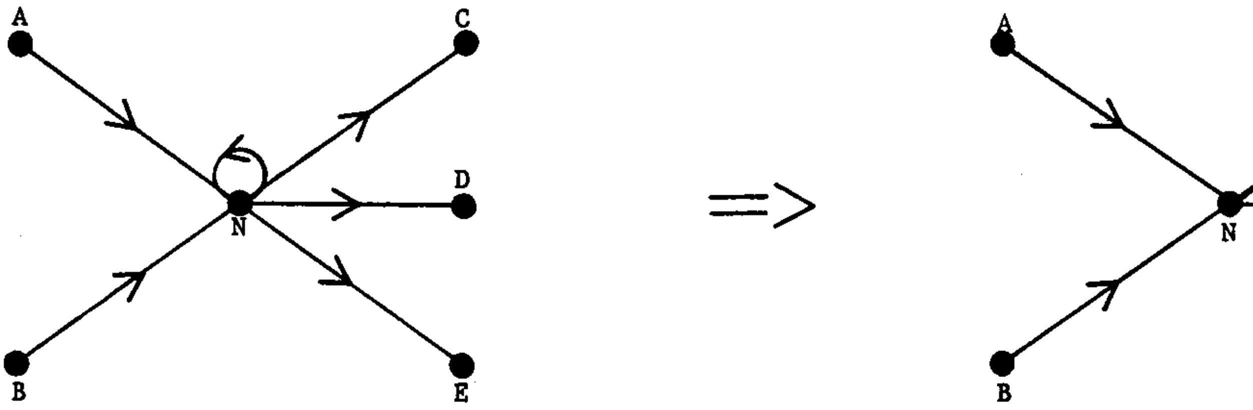
Cooper, D.C. (1966c). Reduction of Programs to a Standard Form by Graph Transformation. Presented at "International Seminar on Graph Theory and its Applications", Rome, Italy, July 5-9, 1966.*

Evans, A. (1965). Syntax Analysis by a Production Language, Ph.D. thesis. Pittsburgh: Carnegie Institute of Technology.

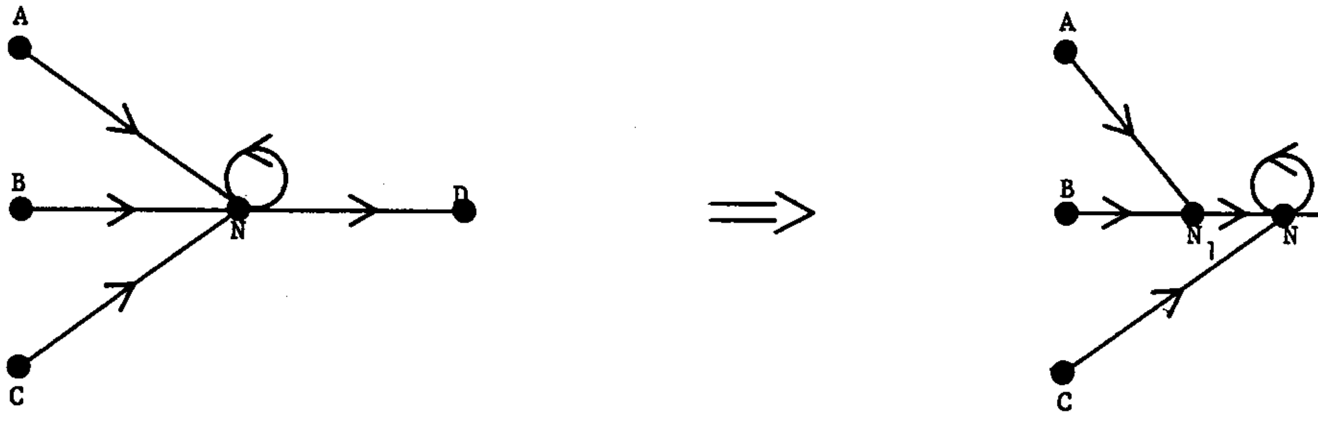
McCarthy, J. (1960). Recursive Functions of Symbolic Expressions and Their Computation by Machine. Comm. ACM 3, 184-195.

Seshu, S. & Reed,M.B. (1961). Linear Graphs and Electrical Networks. Addison-Wesley Publishing Co., Inc. Massachusetts and London.

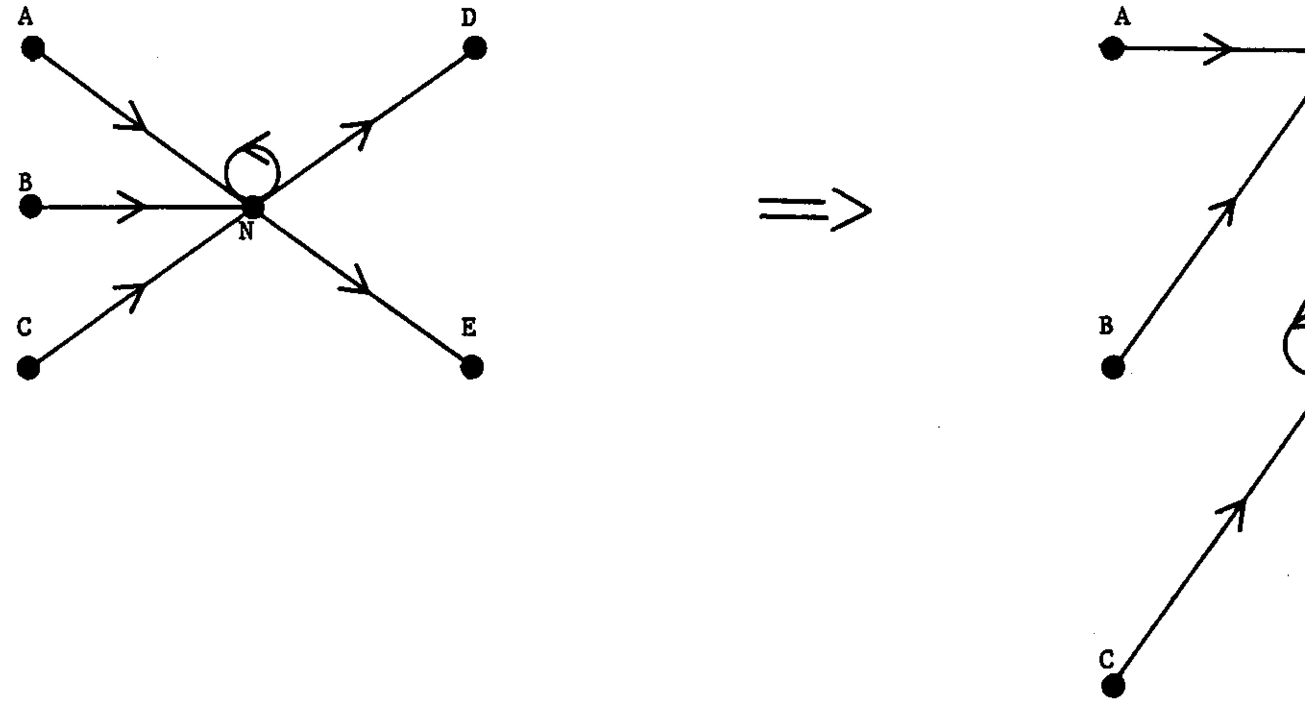
* Part II of this report



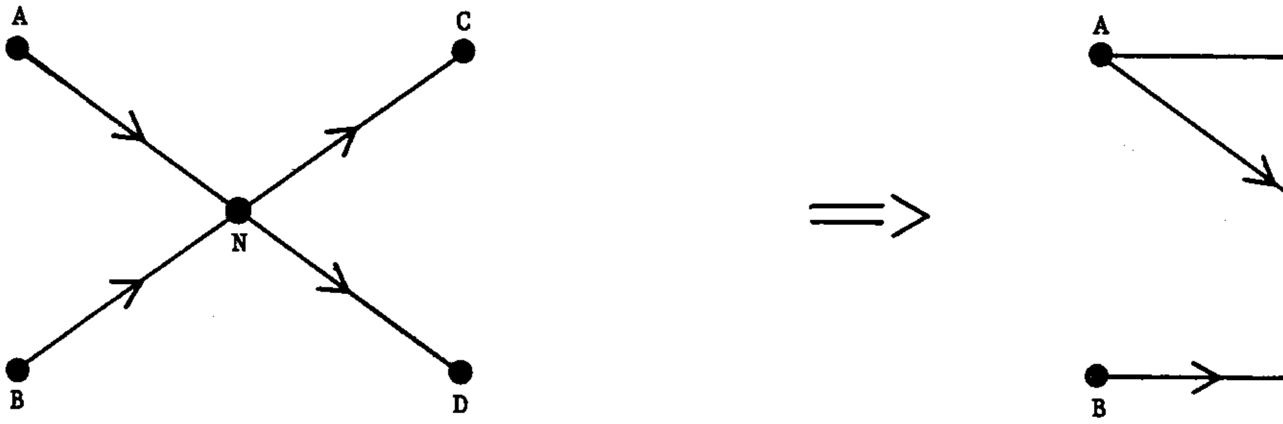
TRANSFORMATION T1



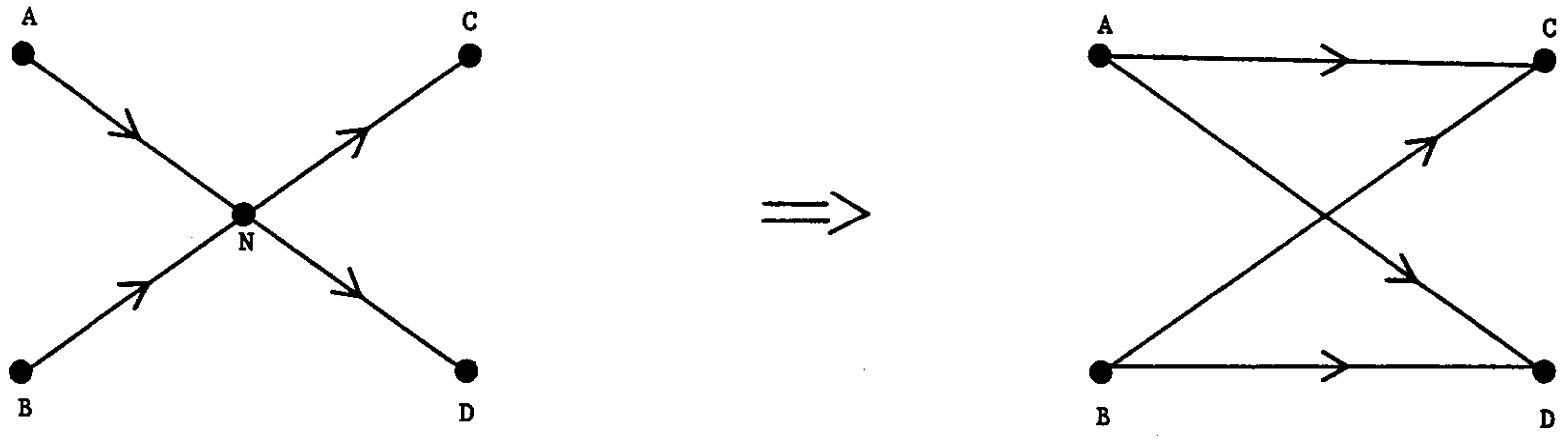
TRANSFORMATION T2



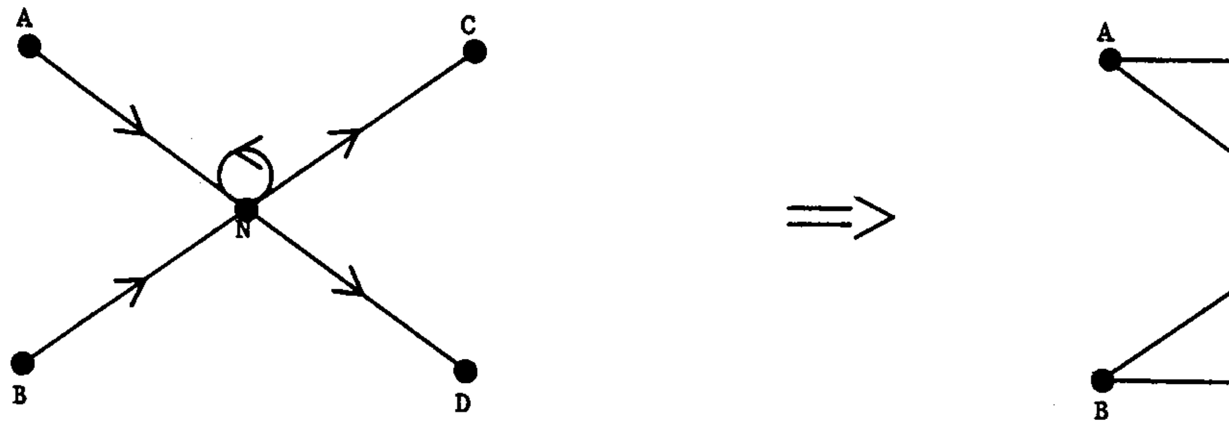
TRANSFORMATION T3



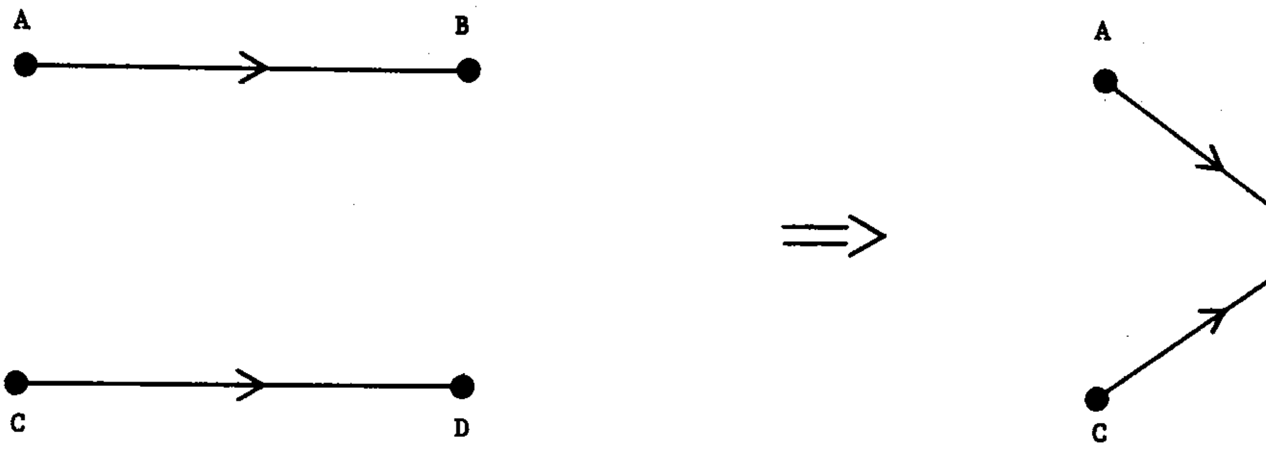
TRANSFORMATION T4



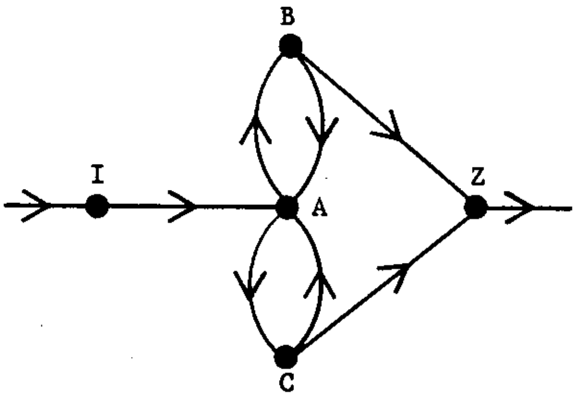
TRANSFORMATION T5



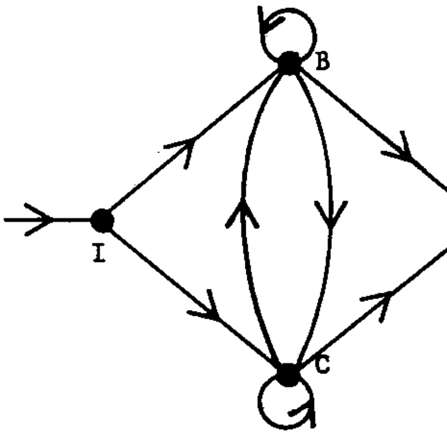
TRANSFORMATION DT1



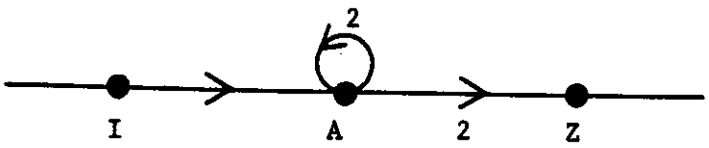
TRANSFORMATION DT2



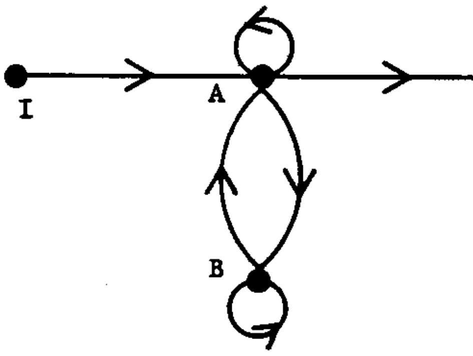
GRAPH 1



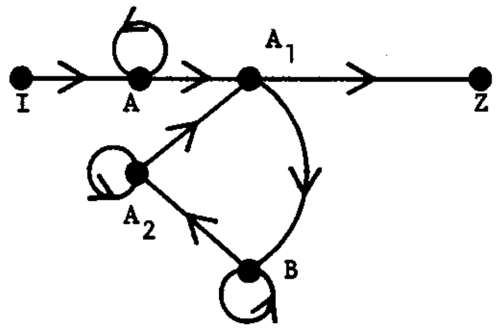
GRAPH 2



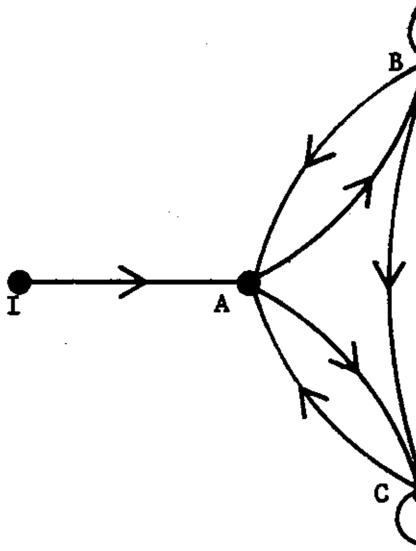
GRAPH 3



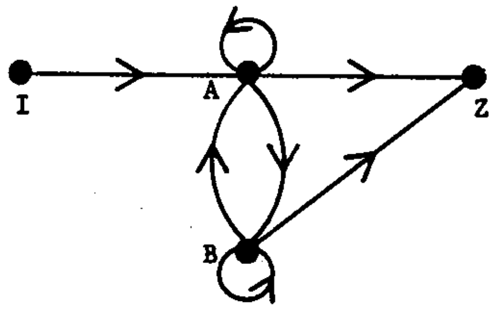
GRAPH 4



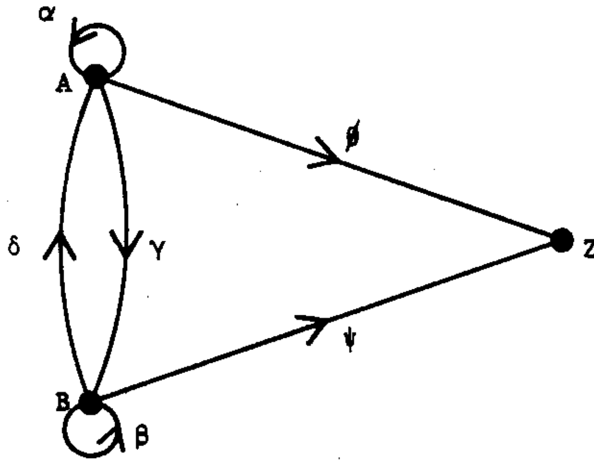
GRAPH 5



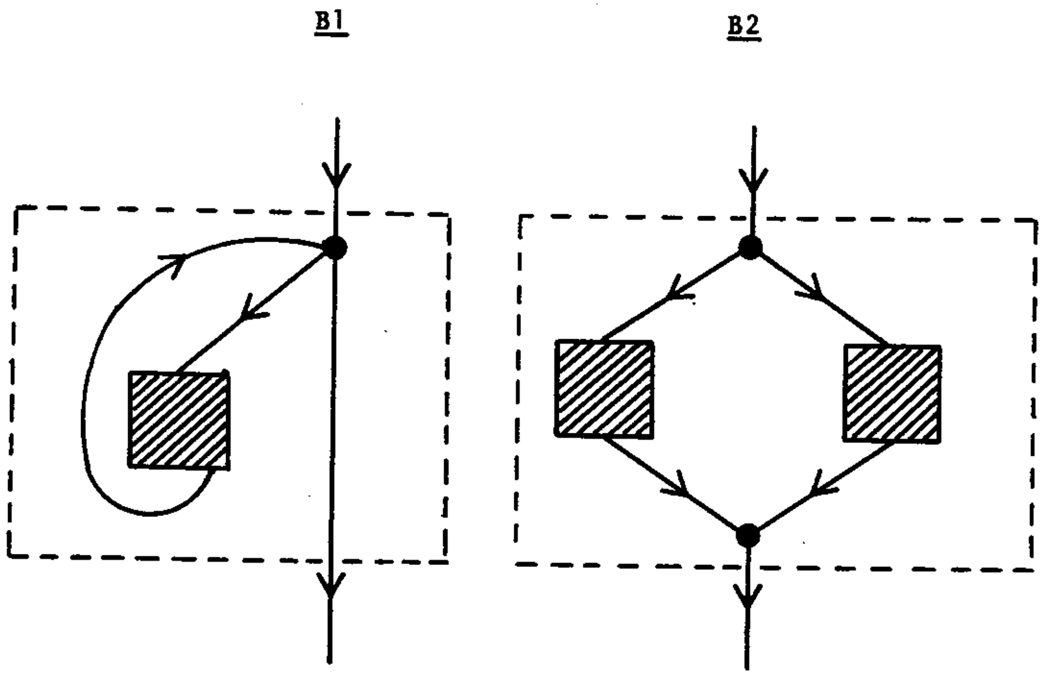
GRAPH

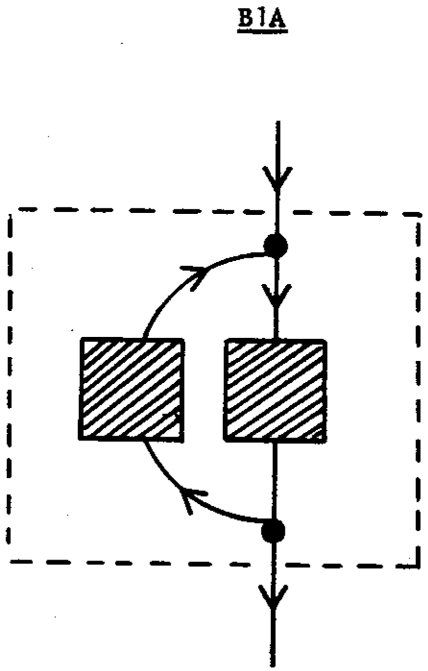


GRAPH 7



GRAPH 8





BLOCK FORM DEFINITIONS

ALTERNATE TO B1

PART II

REDUCTION OF PROGRAMS TO A STANDARD FORM BY GRAPH TRANSFORMATION

INTRODUCTION: Motivation for the theorems proved

Recently there have been several attempts at producing proofs about computer programs, see for example [1] and the other papers referred to therein. The work reported in this paper is motivated by the idea that a knowledge of the underlying structure of a specific program could be of great help in proofs concerning that program. For example the simplest (and also rarest) program is one which does not have any loops in it. Next one might consider programs in which all loops are independent, i.e. no section of the program is part of more than one loop. This class of programs is more interesting, but still very restrictive. However, rules may be set up for transforming a program into an equivalent program which could have a different looping structure. In fact experience indicates that a large number of interesting programs may be transformed by a series of trivial transformations into a program with this independent loop structure. This paper investigates one such transformation and shows how it may be used to transform a program to the desired form, if that is possible.

So far, we have been vague; we should define what we mean by a program, by a transformation, by loops and independent looping, and by equivalence of programs. We take a slightly different view. Assume that, whatever definition of a program is used, a directed graph may be associated in some way with a program, i.e. there is a total function whose domain is the class of all programs and whose range is the class of all directed graphs. Assume also that there is an equivalence relation defined between programs. Then any transformation from graph G to graph G' which is considered should have the property that, for any program with G as its graph, an equivalent program can be mechanically produced with G' as its graph. Further discussion and clarification of these remarks will be found in [2].

In this paper we consider one particular graph transformation which should satisfy this last criterion for most definitions of a program and of equivalence of programs; further such transformations will be found in [2]. This transformation consists of deleting a node from a graph by connecting all its input nodes to all its output nodes (for example Graph 1 may be transformed into Graph 2 by deleting node N, also see the more rigorous definition below). The justification for allowing this transformation is that whatever program has Graph 1 as part of its flowchart, the test at node N amy be anticipated at nodes A and B and direct connections made to nodes C and D as shown. With arc AC will be associated whatever assignment statements were made on arc AN, followed by those on arc NC, and so on. The new test associated with arc AC in Graph 2 will be the conjunction of the condition in Graph 1 for control to go from A to N (there may be other arcs out from A) and the condition for control to go from N to C; there might have to be substitutions made in this latter test to allow for the effect of changes in going from A to N. The node N may not be deleted if there is an arc from N to N; for example neither B nor C may be deleted from Graph 4. If this were to be allowed, the new test associated with AC (Graph 2) would have to be the condition that, after control goes from A to N (Graph 1) and after control finally comes out of the loop around N, control then goes to C. We assume it is not useful to accept this as a basic test in a program although it would, of course, be perfectly possible to do so.

DEFINITIONS

A graph is a set of nodes together with a set Γ of ordered pairs of these nodes. Each member of Γ is an arc. Note that only what are usually called directed graphs are considered in this paper, so we omit the adjective "directed".

The first member of an arc is its initial node, the second member, its final node.

A path is a sequence of arcs $(u_1, u_2, \ldots, u_n)$ such that the final node of u_i is the initial node of u_{i+1} (for $i = 1, 2, \ldots, n-1$). It is a cycle if the final node of u_n is the initial node of u_1. It meets those nodes which are initial or final nodes of its arcs. Any node met by a path or cycle is said to be on the path or cycle. The initial node of a path is the initial node of u_1, and the final node is the final node of u_n. A subpath of a path $(u_1, u_2, \ldots, u_n)$ is a path $(u_i, u_{i+1}, \ldots, u_j)$ with $1 \le i \le j \le n$.

A path or cycle is elementary if it does not meet the same node more than once (except for the initial and final nodes in the case of a cycle).

A proper cycle is a cycle which meets two or more distinct nodes.

A proper cycle free (PCF) graph is a graph with no proper cycles.

The input nodes of a node N are those nodes A such that (A,N) is an arc and the output nodes of N are those nodes Z such that (N,Z) is an arc.

A graph is loop-connected (LC) if it contains two nodes, A and B, and three elementary cycles α, β, and γ such that:

1) There is no one node which is on α, β, and γ;

2) α meets A but does not meet B;

3) β meets B but does not meet A;

4) γ meets both A and B.

(For example, Graph 4 is loop-connected via nodes B and C.)

By theorem 11 if the word elementary is omitted we have an equivalent definition.

A graph is loop-connected(k) (LC(k)) where k is an integer ≥ 2 if there are k nodes $A_1, A_2, \ldots, A_k$; k elementary cycles $\gamma_1, \gamma_2, \ldots, \gamma_k$ and an elementary cycle δ such that:

Restriction A) There is no one node which is on δ and on all the γ_i;

Restriction B) For $1 \le i \le k$ the cycle γ_i meets all of $A_1, A_2, \ldots, A_k$ except A_i;

Restriction C) δ meets all of $A_1, A_2, \ldots, A_k$.

A graph is Kloop-connected (KLC) if there is a k such that it is LC(k). Obviously, LC is LC(2) so that LC implies KLC; theorem 12 also shows the converse, that KLC implies LC.

The node N can be deleted from graph G giving graph G' (written $G \xrightarrow{D_N} G'$) if and only if there is no arc (N,N). G' is obtained from G by adding arcs joining all input nodes of N to all output nodes of N and then removing N and all arcs with N as initial or final node. If a node A is both an input node and an output node of N, then an arc (A,A) is added.

Examples of node deletions are 1) delete node N from Graph 1 to give Graph 2, and 2) delete node A from Graph 3 to give Graph 4.

The node N can be safely deleted from graph G giving graph G' (written $G \xrightarrow{SD_N} G'$) if and only if it can be deleted and also there is some node A of graph G (not N) with the property that all elementary cycles of G which meet N also meet A. G' is obtained from G exactly as for deletion.

Particular cases in which N can be safely deleted are: if there is no cycle meeting N, or if only one elementary cycle meets N (proper cycle), or if N is the initial node of only one arc (not N,N)), or if N is the final node of only one arc (not (N,N)).

THE MAIN THEOREMS PROVED

Consider Graph 3. By deleting A, we obtain Graph 4 which is not PCF. But if instead of deleting A we delete B and then C, we obtain Graph 5 which is PCF and "obviously much simpler" than Graph 4. Graph 5 reveals the "basic structure" of Graph 3, but this basic structure is not revealed in Graph 4.

This example shows that the order in which nodes are deleted is critical. Graph 3 may be transformed into PCF form, but if we delete A first we obtain Graph 4 which cannot be transformed into PCF form. This raises two questions: what are the necessary and sufficient conditions for a graph to be transformable by a series of deletions into a graph which is PCF, and given a graph which can be transformed in what order should the deletions be performed?

The answer to the first question is that a graph can be transformed into PCF form if and only if it is not LC. However although LC is a simpler condition to understand than KLC, the proofs using KLC are much simpler than those we first obtained using LC. We therefore use KLC throughout and prove separately (theorem 12) that a graph is LC if and only if it is KLC. Alternative definitions of KLC are suggested by theorems 4 and 9.

Now consider the process which starts with any graph G and performs safe deletions in any order until a graph G' is obtained on which no safe deletions may be performed.

If G is not KLC, then the following two theorems prove that this algorithm must terminte with G' in PCF form (the algorithm must terminate as each safe deletion reduces by one the number of nodes on the graph).

Theorem 3: If G is not KLC and $G \xrightarrow{SD_N} G'$, then G' is not KLC.

Theorem 5: If G is not KLC and if no node of G can be safely deleted, then G is PCF.

If G is KLC, then the following two theorems prove that it cannot be transformed to PCF form.

Theorem 6: If G is KLC and $G \xrightarrow{D_N} G'$, then G' is KLC.

Theorem 7: If G is KLC, then G is not PCF.

The following theorem proves that safe deletions is the widest class of deletions which could be used in the process.

Theorem 8: If $G \xrightarrow{D_N} G'$ and the deletion is not a safe deletion, then G' is KLC.

Finally the following theorem is proved, although it is not needed in order to obtain the previous results.

Theorem 12: G is LC if and only if G is KLC.

A NOTE ON NOTATION

G and G' always name graphs, other Roman letters with or without subscripts name nodes. A small Greek letter with or without a subscript names either a path or a single node or may be empty; node names and Greek letters may be concatenated to make new path names.

For example if α and β are paths then $AB\alpha\beta C$ names the path which is the sequence of arcs which starts with (A,B) and then follows in order the arc joining B to the initial node of α, the arcs of α, the arc joining the final node of α to the initial node of β, the arcs of β and the arc joining the final node of β to C. If α were a single node, D say, then the path referred to would be the path $ABD\beta C$ and if α were empty the path would be $AB\beta C$.

If α is a path of some graph G and if G contains an arc joining the final node of α to the initial node of α then the cycle consisting of the arcs of α and then this arc is referred to as the cycle α*. In particular $AB\alpha$* is the cycle $AB\alpha A$, i.e. a * may be thought of as naming the initial node of the path in which it occurs.

Theorem 1

If $G \xrightarrow{D_N} G'$ and if all the elementary cycles of G are:

$$N\alpha_1 N,\ N\alpha_2 N,\ \ldots,\ N\alpha_p N,\ \beta_1,\ \beta_2,\ \ldots,\ \beta_q$$

where no β meets N, then all the elementary cycles of G' are:

$$\alpha_1*,\ \alpha_2*,\ \ldots,\ \alpha_p*,\ \beta_1,\ \beta_2,\ \ldots,\ \beta_q$$

and also all those cycles of the following form which are elementary:

$$\alpha_{i_1}\alpha_{i_2}\ \cdots\ \alpha_{i_m}* \quad .$$

Proof

By definition of deletion these are all cycles of G', and clearly they are elementary. We must also prove the converse, that any elementary cycle of G' must take one of the above forms.

Consider any elementary cycle of G', if this is also a cycle of G it must be a β. If it is not, this can only be because it contains one or more arcs which were added to G when N was deleted; let this cycle be of the form $\gamma_1\gamma_2 \ldots \gamma_n*$ where each γ is a path on G but the arc joining the final node of γ_i to the initial node of γ_{i+1} (and of γ_n to γ_1) are all arcs in G' but not in G. Then it is clear that $N\gamma_1 N\gamma_2 \ldots N\gamma_n N$ is a cycle of G. Further each $N\gamma_i N$ must be an elementary cycle of G i.e. these γ's are α's and we have our result.

Theorem 2

If $G \overset{SD_N}{\rightarrow} G'$ and if all the elementary cycles of G are:

$$N\alpha_1 N,\ N\alpha_2 N,\ \ldots,\ N\alpha_p N,\ \beta_1,\ \beta_2,\ \ldots,\ \beta_q$$

where no β meets N, then all the elementary cycles of G' are:

$$\alpha_1*,\ \alpha_2*,\ \ldots,\ \alpha_p*,\ \beta_1,\ \beta_2,\ \ldots,\ \beta_q\ .$$

Proof

As we have a safe deletion of N there must be some node which is on all the α's. There can therefore be no cycle of the form $\alpha_{i_1}\alpha_{i_2} \ldots \alpha_{i_m}*$ $(m \geq 2)$ which is elementary and the result follows from theorem 1.

Theorem 3

If G is not KLC and if $G \overset{SD_N}{\rightarrow} G'$ then G' is not KLC.

Proof

Define the α's and the β's as in theorem 2, so that $\alpha_1*, \alpha_2*, \ldots, \alpha_p*$, $\beta_1, \beta_2, \ldots, \beta_q$ are all the elementary cycles of G'.

Now assume G' is KLC, we shall derive a contradiction. There are in G' then k nodes $A_1, A_2, \ldots, A_k$ on a cycle δ' and k cycles $\gamma_1', \gamma_2', \ldots, \gamma_k'$ subject to restrictions A), B) and C) of the definition of LC(k). Each γ_i' and also δ' must be either of the form α_j* or of the form β_j. Define γ_i to be $N\alpha_j N$ in the former case and γ_i' in the latter case and similarly δ.

Then G is LC(k) by nodes $A_1, A_2, \ldots, A_k$, cycles $\gamma_1, \gamma_2, \ldots, \gamma_k$ and cycle δ. All these are nodes and cycles of G by the way we defined them. The addition of node N to some of the cycles cannot affect restrictions B) and C). The only way restriction A) could be violated is if N has become a node common to the k+1 cycles i.e. if δ' and all γ' were of the form α_i^* . But as the transformation was a safe deletion all the α's have some node E in common and this node would then be common to all the cycles γ_i' and δ' . But these cycles were a set for the LC(k) condition of G' and cannot have a common node.

We have therefore proved that G is LC(k) i.e. is KLC and this is a contradiction.

Theorem 4

A graph which has an elementary proper cycle δ and also n (≥ 2) elementary cycles $\gamma_1, \gamma_2, \ldots, \gamma_n$ with the property that each γ_i meets at least one node of δ and that there is no common node to the n+1 cycles $\delta, \gamma_1, \gamma_2, \ldots, \gamma_n$, is KLC.

The converse of this theorem is trivial, so here we have a possible alternative definition of KLC.

Proof (by induction on n)

Case n = 2

Let A be any common node of δ and γ_1 (there is at least one by the conditions of the theorem). Then A cannot be on γ_2 or it would then be common to δ, γ_1 and γ_2. Similarly let B be any node on δ and γ_2, B cannot be on γ_1. Further there is no common node to δ, γ_1 and γ_2 and we see therefore that G is LC (i.e. LC(2)) and is therefore KLC.

Assume theorem true for n-1 (≥ 2)

Consider a graph G with a cycle δ and n cycles $\gamma_1, \gamma_2, \ldots, \gamma_n$

satisfying the conditions of the theorem. If some n-1 of the γ's have no common node which is also on δ we can use the induction hypothesis to prove that G is KLC. We may therefore assume that any n-1 of the γ's have a node in common which is on δ. For $1 \leq i \leq n$ let A_i be a node which is on δ and on all γ_j $(j \neq i)$. Then A_i cannot be on γ_i, otherwise it would be on δ and all the γ's and further the A's are all distinct nodes for the same reason. Now δ meets all the A_i, each γ_i meets all the A's except A_i and there is no common node to δ and all the γ's. Therefore by the definition G is LC(n) i.e. is KLC.

Theorem 5

If G is not KLC and if no node of G can be safely deleted, then G is PCF.

Proof

We prove the following which is logically equivalent to theorem 5:

If G is not PCF and no node of G can be safely deleted then G is KLC.

Let δ be any proper cycle of G and let its nodes be $A_1, A_2, \ldots, A_n$. For any ordered pair (A_i, A_j) with $i \neq j$ there must be an elementary cycle meeting A_i but not A_j, otherwise A_i could be safely deleted from G. Call this cycle $\gamma_{i,j}$. Then δ and all these γ's meet the requirements of theorem 4 and therefore G is KLC.

Theorem 6

If G is KLC and $G \overset{D_N}{\rightarrow} G'$ then G' is KLC.

Proof

As G is KLC there are k nodes $A_1, A_2, \ldots, A_k$; k elementary cycles $\gamma_1{*}, \gamma_2{*}, \ldots, \gamma_k{*}$ and an elementary cycle $\delta{*}$ subject to restriction A), B), and C).

Case I: N, the deleted node, is not one of the A's.

Then A_1, A_2, ..., A_k are nodes of G' and by the definition of deletion G' contains elementary cycles $\gamma_1'*$, $\gamma_2'*$, ..., $\gamma_k'*$ and $\delta'*$, where γ_1', γ_2', ..., γ_k' and δ' are obtained by omitting node N wherever it occurs in γ_1, γ_2, ..., γ_k and δ respectively. This does not affect restrictions A), B), and C) and so G' is KLC.

Case II: N is an A and $k \geq 3$.

Assume without loss of generality that N is A_k. Define $\delta'*$, $\gamma_1'*$, $\gamma_2'*$, ..., $\gamma_k'*$ as before. Then if δ', γ_1', γ_2', ..., γ_{k-1}' do not have a common node G' is LC(k-1). (Use nodes A_1, A_2, ..., A_{k-1}.) If they do have a common node, E say, then because of restriction A) γ_k' cannot meet E. Therefore G' is LC(k). (Use nodes A_1,A_2, ..., A_{k-1}, E and cycles $\delta'*$,$\gamma_1'*$,γ_2',...,$\gamma_k'*$.)

Case III: N is an A and $k = 2$.

The proof of case II holds unless δ' and γ_1' do not have a common node. Assume then that $\delta*$ is $A_2\delta_1A_1\delta_2A_2$ and γ_1* is $A_2F\epsilon A_2$ where no node of $F\epsilon$ is on $\delta_1A_1\delta_2$. The in graph G' (obtained from G by deleting A_2) the nodes A_1 and F and the elementary cycles $A_1\delta_2\delta_1A_1$, $F\epsilon F$ and $F\epsilon\delta_1A_1\delta_2F$ show that G' is LC.

Theorem 7

If G is KLC then G is not PCF.

Proof

This is trivial, as by the definition of KLC G must contain a proper cycle.

Theorem 8

If $G \xrightarrow{D_N} G'$ and the deletion is not a safe deletion, then G' is KLC.

Proof

Suppose all the elementary cycles meeting N of graph G are $N\gamma_1 N$, $N\gamma_2 N$, ..., $N\gamma_n N$. Then as the transformation is not a safe deletion there is no common node to the γ's and $n \geq 2$. By the definition of deletion G' contains elementary cycles γ_1^*, γ_2^*, ..., γ_n^* such that, for all i and j, $\gamma_i\gamma_j^*$ is a cycle (not necessarily elementary). Theorem 8 can therefore be deduced from the following lemma:

Lemma

If a graph G contains elementary cycles γ_1^*, γ_2^*, ..., γ_n^* where $n \geq 2$, if there is no node which is on all these cycles and if, for all i and j, $\gamma_i\gamma_j^*$ is a cycle of G, then G is KLC.

Proof (by induction on n)

Case $n = 2$

There is no node common to γ_1 and γ_2, therefore the first node of each and the elementary cycles $\gamma_1\gamma_2^*$, γ_1^* and γ_2^* show that G is LC and therefore KLC.

Assume true for $n-1$ (≥ 2)

Consider a graph G with elementary cycles γ_1^*, γ_2^*, ..., γ_n^* satisfying the conditions of the lemma. If some n-1 of these cycles have no common node then we use the induction hypothesis to prove that G is KLC. We may therefore assume that any n-1 of these cycles have a common node. For $2 \leq i \leq n$ let A_i be a node which is on all γ_j ($j \neq i$). Now consider the nodes A_2, A_3, ..., A_n. A_i cannot be on γ_i, otherwise it would be on all the γ's and further, the A's are all distinct for the same reason. The cycle γ_1^* meets all these nodes, and for $2 \leq i \leq n$ γ_i^* meets all the A's except A_i. There is no common node to the γ's and so G is LC(n-1).

Theorem 9

If G is KLC then G has an elementary proper cycle δ and also n ($n \geq 2$) elementary cycles of the form $\delta_i\lambda_i{}^*$ ($1 \leq i \leq n$) where each δ_i is a subpath or a node of cycle δ, there is no common node to $\delta_1, \delta_2, \ldots, \delta_n$ and no λ meets a node which is on δ.

The converse of this theorem is true by theorem 4, and so again we have a possible alternative definition of KLC.

Proof

As G is KLC there are k nodes $A_1, A_2, \ldots, A_k$; k elementary cycles $\gamma_1, \gamma_2, \ldots, \gamma_k$ and an elementary cycle δ satisfying restrictions A), B), and C). The cycle γ_i must be of the form

$$\delta_{i1}\lambda_{i1}\delta_{i2}\lambda_{i2} \cdots \delta_{iq}\lambda_{iq}{}^*$$

where each δ_{ij} is either a subpath of δ or is a node of δ and each λ_{ij} has no node on δ.

Define δ_{iab} so that $\delta_{ia}\delta_{iab}\delta_{ib}$ is an elementary subpath of δ. Consider the q cycles:

$$\delta_{i2}\delta_{i21}\delta_{i1}\lambda_{i1}{}^*$$
$$\delta_{i3}\delta_{i32}\delta_{i2}\lambda_{i2}{}^*$$
$$\vdots$$
$$\delta_{i1}\delta_{i1q}\delta_{iq}\lambda_{iq}{}^*$$

(if $q = 1$ just consider the single cycle $\delta_{i1}\lambda_{i1}{}^*$).

Do this for all i with $1 \leq i \leq k$ and take all these cycles to be the $\delta_i\lambda_i{}^*$ of the theorem, they are clearly elementary. It only remains to prove that all these cycles cannot have a common node which is also on δ.

Suppose there was such a node, A_Q. Now A_Q may or may not be one of $A_1, A_2, \ldots, A_k$, but in any case A_Q cannot be on all the γ's, assume A_Q

is not on γ_i. Then as A_Q is on all the cycles listed above none of δ_{i21}, δ_{i32}, ..., δ_{i1q} can be empty and A_Q must be in them all, as these are the only portions of the cycles not on γ_i. But there can be no such node by the definition of δ_{iab}. (This is left as an exercise for the reader. The δ_{ij} are subpaths of δ, consider the ordering imposed on them if A_Q is on all the δ_{iab}.)

<u>Theorem 10</u>

If a graph G has an elementary proper cycle δ and also n ($n \geq 2$) elementary cycles of the form $\delta_i\lambda_i{*}$ ($1 \leq i \leq n$) where each δ_i is a subpath or a node of cycle δ, if there is no common node to δ_1, δ_2, ..., δ_n and if no λ_i meets a node of δ, then G is loop-connected (i.e. LC(2)).

<u>Proof</u> (by induction on n)

<u>Case n = 2</u>

Take A to be any node on δ_1, B any node on δ_2. Then nodes A and B and cycles δ, $\delta_1\lambda_1{*}$ and $\delta_2\lambda_2{*}$ show that G is LC.

<u>Assume true for n-1 ($\geq$ 2)</u>

Let G be a graph with cycles δ, $\delta_1\lambda_1{*}$, $\delta_2\lambda_2{*}$, ..., $\delta_n\lambda_n{*}$ satisfying the conditions of the theorem. Then if some n-1 of δ_1, δ_2, ..., δ_n have no common node we can use the induction hypothesis to prove that G is LC. We may therefore assume that any n-1 of the δ_i have a common node. For $1 \leq i \leq n$ let A_i be a node on all δ_j ($j \neq i$). Then A_i cannot be on δ_i, otherwise it would be on all the δ's and further the A's are all distinct nodes for the same reason.

Without loss of generality we can assume A_1, A_2, ..., A_n are in order around the cycle δ. Now consider the cycle $\delta_i\lambda_i{*}$. The δ_i meets all the A's except A_i, the graph must therefore take the form of graph 6.

(What appear to be arcs in this graph may be paths i.e. there may be other nodes on them.)

Define β_i and γ_i to be either elementary subpaths of δ or single nodes of δor empty so that the cycle $\delta_i\lambda_i$* is the cycle A_{i+1} (round δ to) $A_{i-1}\beta_i\lambda_i\gamma_iA_{i+1}$, where we define A_{n+1} to be A_1 and A_0 to be A_n.

There can be no node which is on a β and a γ, for suppose there is an i such that a node N is on γ_i and on β_{i+1} which is the only possibility. Then N is on δ_i (as γ_i is either a subpath or a node of δ_i), N is on δ_{i+1} (as β_{i+1} is either a subpath or a node of δ_{i+1}) and also N must be on all the other δ's (as they all include the complete path from A_i to A_{i+1}). Therefore N is on all δ's, but they do not have a common node.

We now can find the cycles involved in a loop connection, but need to consider three cases.

Case 1: n is even

Consider the following cycles (the reader should draw a picture):

I δ

II $A_2\beta_3\lambda_3\gamma_3A_4\beta_5\lambda_5\gamma_5A_6 \cdots A_n\beta_1\lambda_1\gamma_1A_2$

III $A_1\beta_2\lambda_2\gamma_2A_3\beta_4\lambda_4\gamma_4A_5 \cdots A_{n-1}\beta_n\lambda_n\gamma_nA_1$

Then I meets A_1 and A_2, II meets A_2 but not A_1 and III meets A_1 but not A_2. Also they cannot have a common node; no λ meets a node of δ and no β or γ meets any other β or γ. Therefore G is LC by definition.

Case 2: n is odd and λ_1 meets no λ_q with q even

Consider the following cycles (again a picture will help to visualize the situation):

I $\quad A_1(\text{round } \delta \text{ to})A_2\beta_3\lambda_3\gamma_3A_4\ \beta_5\lambda_5\gamma_5A_6 \cdots A_{n-1}\beta_n\lambda_n\gamma_nA_1$

II $\quad A_n(\text{round } \delta \text{ to})A_1\beta_2\lambda_2\gamma_2A_3\ \beta_4\lambda_4\gamma_4A_5 \cdots A_{n-2}\ \beta_{n-1}\lambda_{n-1}\gamma_{n-1}A_n$

III $\quad A_n\beta_1\lambda_1\gamma_1A_2$ (round δ through A_3, A_4, A_5 ... to) A_n .

Then I meets A_1 and A_2, II meets A_1 but not A_2 and III meets A_2 but not A_1. The only nodes on δ which both I and II meet are on the path $\gamma_nA_1\beta_2$ and III meets no node of this path. The only λ in III is λ_1 and we have postulated this meets no λ_q where q is even, therefore II cannot meet any node on λ_1 and so we have proved that I, II and III can have no common node. Therefore G is LC.

<u>Case 3: n is odd and λ_1 meets λ_q where q is even</u>

Suppose W is a node which is on λ_1 and also on λ_q, say λ_1 is the path $\lambda_1'W\lambda_1''$ and λ_q is the path $\lambda_q'W\lambda_q''$.

Consider the cycles:

I $\quad \delta$

II $\quad A_n\beta_1\lambda_1'W\lambda_q''\gamma_qA_{q+1}\beta_{q+2}\lambda_{q+2}\gamma_{q+2}A_{q+3} \cdots A_n$

III $\quad A_1$ (round δ to) $A_2\beta_3\lambda_3\gamma_3A_4\beta_5\lambda_5\gamma_5A_6 \cdots A_1$.

Then I meets A_n and A_1, II meets A_n but not A_1 and III meets A_1 but not A_n, the only common node to I, II and III must be on δ but it is easy to see that II and III do not have a common node on δ. Therefore G is LC.

<u>Notes on the proof of these three cases</u>

It is not immediately clear that all the cycles produced are legitimate cycles for small values of n e.g. case 1 for n = 2 or 3. Remembering however that $n > 2$ (as we have already dealt with the case n = 2) if n is odd it is at least 3 and if even at least 4. This is sufficient to make all the cycles valid. Also the definition of LC stated that the cycles α, β and γ were all to be elementary, many of the cycles used above need not be elementary as there was no restriction stating that two of the λ's could not have a common node. We therefore need the following theorem.

Theorem 11

A graph G which has two nodes, A and B, and three cycles α, β and γ such that all the conditions for LC, except the restriction that α, β and γ be elementary, are satisfied is LC.

Proof

Let α be the cycle $A\alpha'A$, β be $B\beta'B$ and γ be $A\gamma'_1B\gamma'_2A$. Then we may assume that α', β', γ'_1 and γ'_2 are each either an elementary path or a node or empty, and that none of α', β', γ'_1 and γ'_2 meet A or B, otherwise it is easy to find, by omitting nodes, new α', β', γ'_1 and γ'_2 with these properties. There remains the possibility that γ'_1 and γ'_2 have a common node or nodes.

Suppose N is such a node and write γ as $A\gamma'_{11}N\gamma'_{12}B\gamma'_{21}N\gamma'_{22}A$, where no node on γ'_{11} is also on $\gamma'_{21}N\gamma'_{22}$. Now N cannot be on both α' and β', assume it is not on α', a similar proof will hold if it is not on β' . Consider nodes A and N and cycles $A\gamma'_{11}N\gamma'_{22}A$, $N\gamma'_{12}B\gamma'_{21}N$ and $A\alpha'A$. If the second of these is not elementary replace it by that part of itself which is an elementary cycle through N. Then if these cycles do not have a common node they show that G is LC.

If the cycles do have a common node P then consider the nodes B and P and the cycles $N\gamma'_{12}B\gamma'_{21}N$, $A\alpha'A$ and $B\beta'B$. The first two both meet P and there can be no node on all three cycles as they are each part of the original three cycles. The first may not be elementary but it must have fewer repeated nodes than $A\gamma'_1B\gamma'_2A$ and so we proceed by induction to prove that G is LC.

Theorem 12

A graph is LC if and only if it is KLC.

Proof

If a graph is KLC then theorem 9 and 10 shows it is LC. If a graph is LC it is LC(2) and hence KLC.

ACKNOWLEDGEMENTS

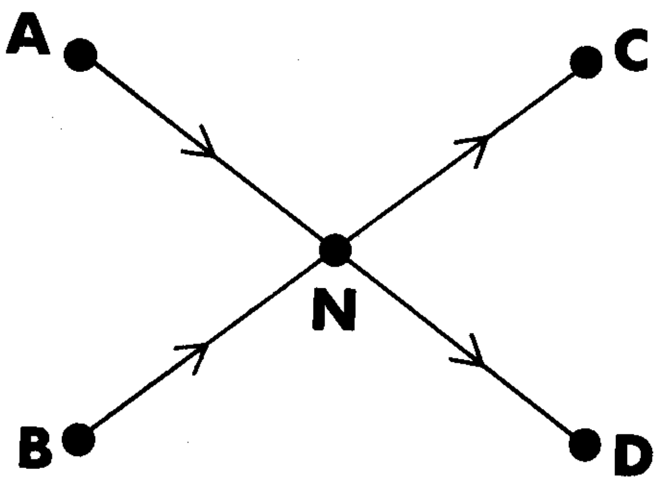
The concept of LC(k) and many of the above proofs are due to Z. Manna. I myself obtained a complete set of theorems and proofs based directly on LC, but the above presentation is simpler.

REFERENCES

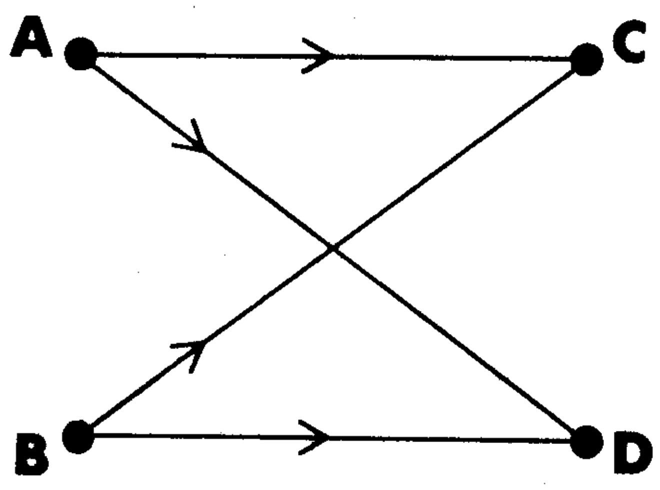
[1] Cooper, D. C., "Mathematical proofs about computer programs". Machine Intelligence 1, ed. D. Michie. Oliver and Boyd Ltd. To appear October 1966.

[2] Cooper, D. C., "Theoretical results concerning programs regarded as directed graphs". To be presented at Second Machine Intelligence Conference, Edinburgh, June 27 - July 1, 1966.*

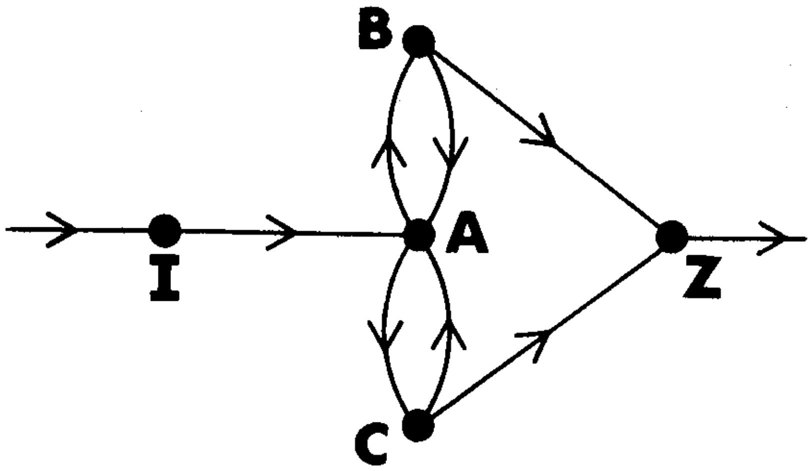
* Part I of this report



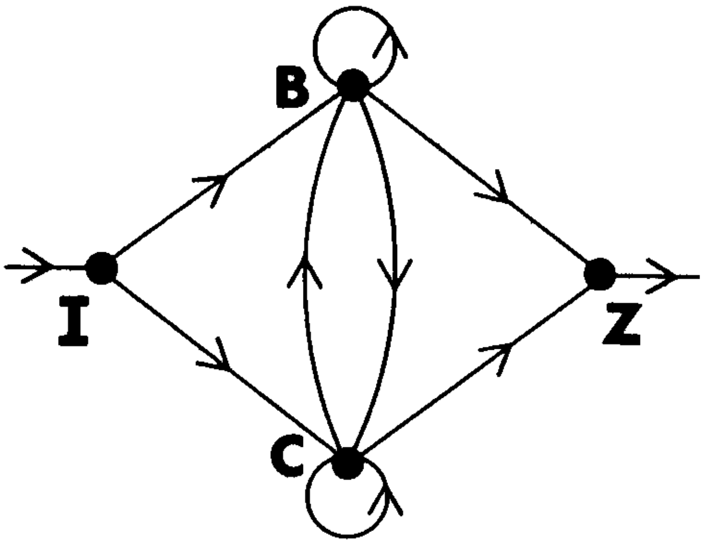
GRAPH 1



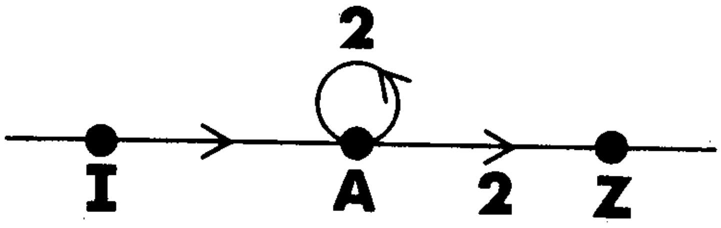
GRAPH 2



GRAPH 3

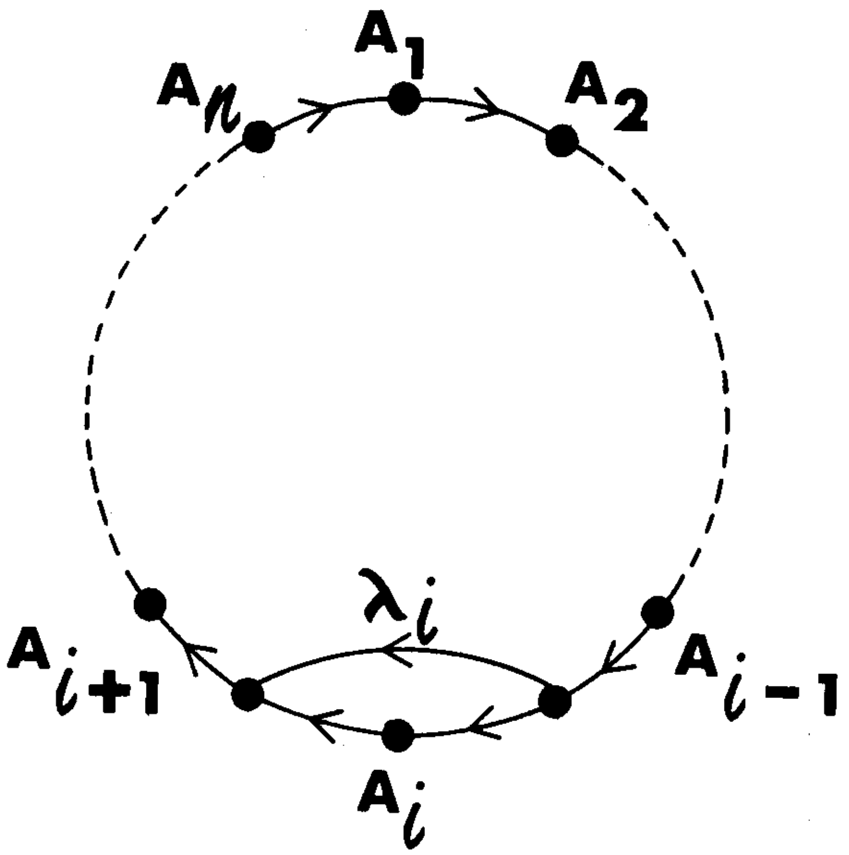


GRAPH 4



2 indicates a multiple arc

GRAPH 5



GRAPH 6