CAM 01: A precedence analyser

S. J. Waters

LSE, Houghton Street, London WC2A 2AE

This paper discusses the concepts of precedence networks and matrices as applied to large information processing systems. These concepts have been built into precedence analysers which are useful software aids for systems analysts, designers and programmers. CAM 01 is a standalone precedence analyser which requires little investment to exploit and is now available for field trials, demonstrations and practical use.

(Received June 1975)

This paper is further 'fall-out' from the CAM research project being carried out at the London School of Economics. This research is investigating computer-aided methods of developing computer-based information processing systems. The project, outlined by Waters (1972), is adequately financed by the Science Research Council.

The first phase of the CAM project studied current theory and practice, mainly in the highly-technical field of computer systems design. This research disproved some established but false techniques and suggested some new techniques; further, a manual design method was proposed which structures design decisions, and their alternative choices, with respect to design objectives to yield guidelines.

The current phase of the CAM project is attempting to prove the *technical* feasibility of computer-aided design along the lines suggested by Waters (1974c). A necessary first step has yielded a precedence analyser that automatically traces some of the relationships between elementary information items. This program, CAM 01, has useful applications at all stages of systems definition, design, implementation and maintenance and is fundamental to further research into computer-aided methods of systems development.

This paper discusses precedence networks, their associated precedence matrices, and some precedence analysers with suggested applications. Future developments to CAM 01 are also indicated.

Precedence networks

An information processing system can be viewed as a collection of procedures that transforms elementary information items into further elementary information items. Here, as in systems analysis, the emphasis is on information with procedures being sub-servient to the flow of information. This is opposite to the practice of many programmers who (possibly falsely?) emphasise procedures to the detriment of information flow.

Following Stamper (1973), an elementary information item (subsequently abbreviated as 'element') can be regarded as a sign in semiotics that:

1. Represents an entity (e.g. stock number)
2. Represents a property of an entity (e.g. quantity in stock)
3. Represents a relationship between entities (e.g. substitute stock number for an unavailable stock number), or
4. Represents an event (e.g. date that stock was last sold).

Once a 'real-world' entity, property, relationship or event has been denoted by an element then it can be manipulated within a logical model of the system.

Following Grindley (1966), an element of this model can either be 'given' to the system or be 'derived' by procedures in the system. Following Waters (1974a), an element can occur in input messages, data base, output messages or as a transient, strategic element. **Fig. 1** reconciles these views by analysing all possible occurrences of elements against the types of elements.

A procedure derives an element from other elements which are given or previously derived. Thus, a procedure generates an element from input elements that are termed 'first-order precedents' of the output element. For example, the decision table of **Fig. 2** derives an element from four first-order precedents which are used in decision-making and/or in action taking. As it stands, this particular procedure may be entirely programmed, assuming the factual first-order precedents are available to the computer; however, if the procedure were to include a management reduction in Christmas bonus due to a human decision on poor employee performance (based on punctuality, dress, non-cooperation, etc.), then it may not be entirely programmable. The contents of this paper are relevant to both types of procedure and therefore embrace human systems as well as computer systems; the reader can restrict his view to the latter, if appropriate.

The elements of a system can be represented by the nodes of a network where each branch denotes that its source element is a first-order precedent of its destination element: this is illustrated for the decision-table in Fig. 2, where a derived element is synonymous with the procedure that derives it. Such a directed graph is termed a 'precedence network'. Clearly, a total system may inter-relate many sub-systems in practice and therefore requires a large and complex precedence network, possibly containing thousands of elements.

This concept of information precedence networks was first introduced to the author during informal discussions with

	Occurrence of element				
Type of element	*Input messages*	*Input data base*	*Transient*	*Output data base*	*Output messages*
Given	√				
	√			√	
	√				√
	√			√	√
		√			
		√		√	
		√			√
		√		√	√
Derived			√		
				√	
					√
				√	√

Fig. 1 Analysis of all combinations of element occurrences

Decisions	*Rules*							
1. Is employee a member of the established staff?	√	√	√	√	×	×	×	×
2. Is employee's length of service > 10 years?	√	√	×	×	√	√	×	×
3. Is employee male?	√	×	√	×	√	×	√	×
Actions								
Christmas bonus =								
1. 1% of annual salary								×
2. 2% of annual salary				×		×	×	
3. 5% of annual salary		×	×		×			
4. 10% of annual salary	×							

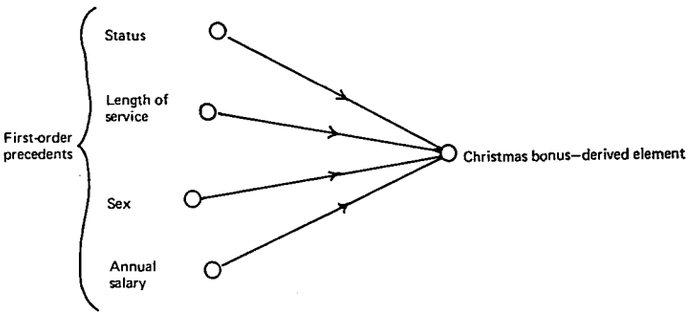


Fig. 2 Decision table procedure for deriving Christmas bonus

Professor P. A. Losty, now at Cranfield Institute of Technology. A large engineering company was attempting to manually examine such a precedence network for its total system but was meeting with some difficulty! An alternative approach is to input the precedence network into computer software that may use graph theoretic approaches to automatically analyse the network. Typical results output by such an analysis include:

1. *Full precedence list*
This identifies *all* precedent elements of each derived element. If element A is a first-order precedent of element B, which is itself a first-order precedent of element C, then element A is termed a 'second-order precedent' of element C; for example, if the procedure of Fig. 2 follows another procedure which derives length of service from date of joining and current date, then these latter two elements are second-order precedents of Christmas bonus; similarly for precedents of order three, four, five, etc. Thus, precedent elements of all orders are listed for each derived element.

2. *Full successor list*
This identifies all derived elements that have each element as a precedent, whatever the order of precedence. If element A is a nth order precedent of element B, then element B is termed a 'nth order successor' of element A; for example, in Fig. 2 Christmas bonus is a first-order successor of length of service. Thus, successor elements of all orders are listed for each element.

3. *Cycles list*
This identifies all elements that are contained in each cycle of the precedence network. If element A is a precedent of element B, which is itself a precedent of element C, which in turn is a precedent of element A, then elements A, B and C form a cycle or loop in the precedence network. Notice that elements A, B and C are each precedents of themselves in this case; further, cycles may span hundreds of elements and be deeply embedded in the precedence network.

4. *Independent sub-networks list*
This identifies all elements that are contained in each independent sub-network of the precedence network. Two sub-networks are independent if no element of one is a precedent or successor of any element of the other; thus, there are no branches between the elements of one sub-network and the elements of another.

5. *Similar elements list*
This identifies all those pairs of elements having similar precedents or successors to each other. For example, the following similarities can be listed:

5.1. *First-order precedents*
Elements A and B are similar if all the first-order precedents of one are also first-order precedents of the other.

5.2. *Given element precedents*
Elements A and B are similar if all the given elements that are precedents of one are also precedents of the other.

5.3. *First-order successors*
Elements A and B are similar if the first-order successors of one are also first-order successors of the other.

5.4. *Terminal element successors*
An element is termed 'terminal' if it has no successors; elements A and B are similar if all the terminal elements that are successors of one are also successors of the other.

Lists 5.2 and 5.4 would normally exclude element pairs where one element is a precedent of the other.

Thus, computer software may automatically analyse a large, complex precedence network to produce listings that are tedious and costly to provide by manual means. Also, the precedence network may be revised and re-analysed more efficiently by computer software than by manual means.

Clearly, network analysis has been exploited in other application areas. Burman (1972) and others discuss project control using PERT for critical path analysis. Langefors (1966) and others discuss production control using BOMP for parts explosion or resources 'implosion'. Thus, the intention here is to exploit similar techniques in the area of systems development to achieve 'loose-end analyses' and 'derivation cascades'.

Precedence matrices

Langefors (1966) pioneered the application of precedence networks to information processing systems. Although some aspects of this thought-provoking work are not practical, it does include a useful technique for representing the 'variable-length' precedence networks as 'fixed-length' precedence matrices. Some of the manipulations of precedence networks are then conveniently described in terms of (Boolean) matrix algebra.

A precedence network containing elements $1, 2, \ldots, N$ is represented by a first-order precedence matrix $(P_{ij})^1$ where and j each take values $1, 2, \ldots, N$; thus, $(P_{ij})^1$ is a square matrix of size $N \times N$ and contains one row and one column for each element. An entry P_{ij} of the first-order precedence matrix is given unit value if element i is a first-order precedent of element j, otherwise P_{ij} is given zero value. Fig. 3(a) illustrates a simple precedence network and Fig. 3(b) represents the corresponding first-order precedence matrix $(P_{ij})^1$. Clearly, given elements have zero columns, terminal elements have zero rows and the leading diagonal should be zero since no element is allowed to be a first-order precedent of itself (i.e. all $P_{ii} = 0$).

A nth order precedence matrix $(P_{ij})^n$ contains entries P_{ij} taking unit value if element i is a nth order precedent of element

j, otherwise taking zero value. **Fig. 4** illustrates that

$$(P_{ij})^2 = (P_{ij})^1 \times (P_{ij})^1$$
$$(P_{ij})^3 = (P_{ij})^2 \times (P_{ij})^1$$
$$\vdots$$
$$(P_{ij})^n = (P_{ij})^{n-1} \times (P_{ij})^1$$

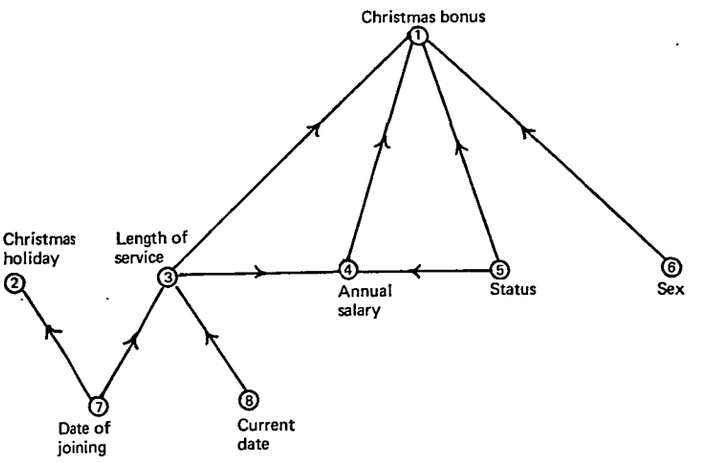
until the highest order of precedence is exceeded, using logical addition. Further, **Fig. 5** illustrates that the full precedence matrix (P_{ij}), containing unit entries for precedents of all orders, is given by

$$(P_{ij}) = \sum_{n=1}^{n=\infty} (P_{ij})^n$$

where the summation is performed by logical addition to yield a Boolean matrix and the summation can be terminated once the next $(P_{ij})^n = 0$.

The analyses of the previous section can be obtained from this full precedence matrix. The columns yield a full precedence list and the rows yield a full successor list; cycles are detected by the leading diagonal containing non-zero entries; independent sub-networks and similar elements are found by comparing the rows and columns.

Thus, precedence matrix notation is convenient for describing some of the manipulations of precedence networks. A fuller account is given by Langefors (1966) but the reader is warned that the design methods developed from precedence matrices are open to doubt; for example, some doubts are outlined by Waters (1974*b*). Further, the usefulness of precedence matrices as a description tool does not imply they should be used as an implementation technique.



(*a*) Precedence network

$(P_{ij})^1 =$

	1	2	3	4	5	6	7	8
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	1	0	0	1	0	0	0	0
4	1	0	0	0	0	0	0	0
5	1	0	0	1	0	0	0	0
6	1	0	0	0	0	0	0	0
7	0	1	1	0	0	0	0	0
8	0	0	1	0	0	0	0	0

(*b*) Corresponding first-order precedence matrix

Fig. 3 An eight-element precedence network represented as a first-order precedence matrix

Precedence analysers

A precedence analyser is a computer program that reads a first-order precedence network and automatically generates lists of full precedence, full succedence, cycles, independent sub-networks and similar elements (or some of these, at least). As such, a precedence analyser is therefore of fundamental importance to any computer-aided method that aims to take full advantage of the computer in all stages of systems definition, design, implementation and maintenance.

The significance of precedence networks has been duly recognised by several research and development groups who have included precedence analysers in their project software. These pioneers include:

1. The ISDOS group at Michigan University, USA. Teichroew and Sayani (1971) outline the aims of the ISDOS research and development project. One aspect is a Problem Statement Analyser (PSA) which derives a static model of a system from its Problem Statement Language (PSL) definition: this contains a precedence analyser.
2. The CASCADE group at the Technical University, Trondheim, Norway. Solvberg (1972) mentions a precedence analyser as an integral part of systems analysis software aids.
3. The DATAFLOW group at NCC Limited, Manchester, England. Boot (1969) discusses the DATAFLOW project which included a precedence analyser in systems definition software. Unfortunately, the project has since been terminated.

No doubt there are numerous other groups tackling these problems of precedence analysis, particularly in America and Scandinavia. Clearly, the wide influence of Langefors (1966) has had a profound impact within these latter countries and they are to be commended for tackling some of the fundamental problems of information processing in an effort to base the subject on a firmer, theoretical foundation. The difficulty is to recognise which problems are or are not solvable.

Although there are successful implementations of precedence analysers, some have failed to overcome severe limitations including restrictions on network size (e.g. less than 100 elements only), lengthy computer runs (e.g. hours) and high computing cost (e.g. hundreds of pounds per computer run). Further, some of the successful implementations are tightly embedded in large software projects involving, perhaps, hundreds of man-years effort; thus, a prospective user may be deterred from exploiting the precedence analyser because the total software project is unattractive.

The CAM research project, with few resources, has attempted to overcome these drawbacks by developing CAM-01 which is a stand-alone precedence analyser with an acceptable performance. CAM-01 is now available for field trials, demonstrations and practical use as a 1,000-statement FORTRAN IV program that runs on the University of London Computer Centre CDC configuration. CAM-01 analyses a precedence network of up to 10,000 elements, typically within seconds of CPU *and* disc transfer time (plus controllable input/output time). The computing costs are therefore trivial. Further, CAM 01 requires little extra documentation effort from a systems team to exploit it, merely a list of elements and their first-order precedents; some compilers and data base management systems generate such lists automatically. Notice that CAM 01 is one logical extension to the data dictionary schemes that are currently being developed.

Using a precedence analyser

A framework is now necessary to discuss the use of a precedence analyser such as CAM 01. This is provided by the typical systems approach (or life-cycle) of **Fig. 6.** Here, an information processing system is developed through major stages of definition, design, implementation and maintenance; each stage can advance to the next stage, retreat to a previous stage or abort.

Systems definition yields the information processing require-

$(P_{ij})^2 =$

	1	2	3	4	5	6	7	8
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	1	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	1	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0
7	1	0	0	1	0	0	0	0
8	1	0	0	1	0	0	0	0

$(P_{ij})^3 =$

	1	2	3	4	5	6	7	8
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0
7	1	0	0	0	0	0	0	0
8	1	0	0	0	0	0	0	0

$(P_{ij})^n = 0$ if $n > 3$

Fig. 4 Higher-order precedence matrices for the precedence network of Fig. 3

ments as a 'black box' which is aimed to meet the user organisation's needs. This 'black box' defines the input, data base and output information plus the procedures that transform the given information into derived information (where definable). The result is a total system specification that pays little regard to subsequent implementation strategy but concentrates on satisfying the user organisation's objectives; usually, this specification is achieved by analysing the logic of an existing system and synthesising the logic of an improved system.

Systems design partitions the total system into human and computer systems and their manual and automatic procedures are designed with due regard to available resources. The eight information flows between these two systems and their 'outside world' (see Fig. 6) are also designed. The results are human and computer system specifications that respectively define manual procedures with associated paperwork and computer programs with associated files.

Systems implementation builds the human and computer systems (with their respective, initial data bases) and brings them to satisfactory operational performance. The results are procedure manuals and computer programs with supporting documentation and training.

Systems maintenance monitors the human and computer systems and evaluates them against their objectives. The previous stages of definition, design and implementation are re-entered to revise the systems as and when necessary.

This systems approach can benefit from using a precedence analyser as follows:

1. *Systems definition*

1.1. *Full precedence list*

This helps to check that a system is completely defined. The list highlights given elements, which should be obtained from input messages or data base, and undefined elements, which are used in derivations but are not themselves specified.

1.2. *Full successor list*

This helps to check that a system is completely defined. The list highlights terminal elements, which should be used as output via messages or possibly data base. Alternatively, any unused elements may be spurious to the system and therefore may be eliminated; for example, if an output message (and its elements) are no longer required from the system, then any resulting obsolete elements can be traced.

1.3. *Cycles list*

This helps to check that a system is consistently defined because cycles should not normally occur.

1.4. *Independent sub-networks list*

This helps to partition a large system into smaller sub-systems which can be developed independently from each other. Alternatively, the list may indicate that a system is incompletely defined because relationships should in fact exist between the sub-networks.

1.5. *Similar elements list*

This can help to rationalise a systems definition. Similarities may be exploited by eliminating elements and redefining procedures to achieve a simpler systems definition.

2. *Systems design*

2.1. *Full precedence list*

This helps when grouping procedures, for example into clerical departments or computer programs. The list helps to define the

$(P_{ij}) =$

	1	2	3	4	5	6	7	8
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	1	0	0	1	0	0	0	0
4	1	0	0	0	0	0	0	0
5	1	0	0	1	0	0	0	0
6	1	0	0	0	0	0	0	0
7	1	1	1	1	0	0	0	0
8	1	0	1	1	0	0	0	0

Fig. 5 Full precedence matrix for the precedence network of Fig. 3

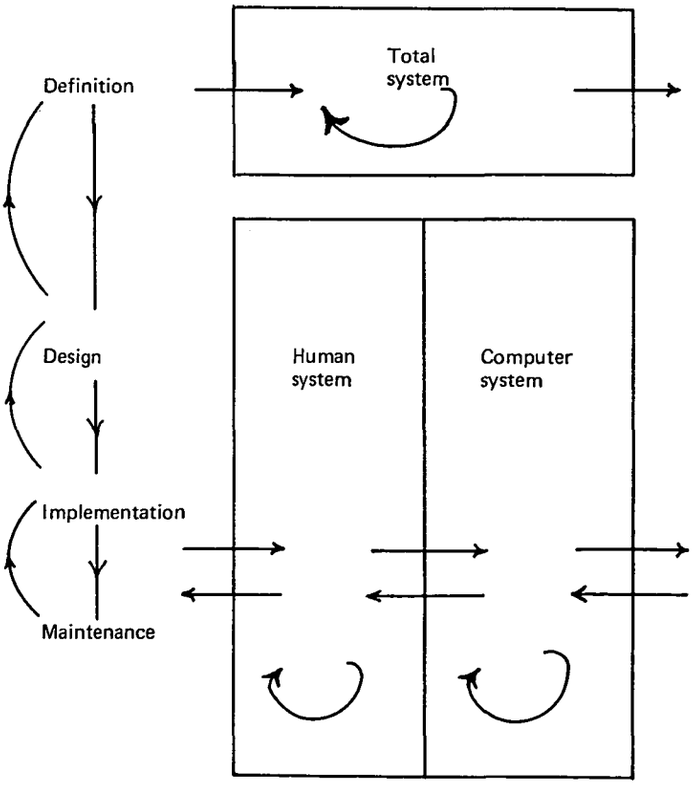


Fig. 6 Major stages in a typical systems approach

elements that must flow between groupings and the sequence in which these groupings can operate. Further, critical elements in output messages can be examined to see which input message elements are also therefore critical; for example, which elements must be highly accurate or must be available on time.

2.2. *Full successor list*
This can be used to design data base amendments. Given an incorrect value of an input message element then the list highlights all those data base elements whose values are also incorrect as a result. Thus, composite amendments can be arranged to overcome such errors. Generally, the list can help trace the effects of errors and, similarly, delays.

2.3. *Similar elements list*
This helps when grouping elements, for example into messages and records, and also when grouping procedures, for example into clerical departments or computer programs.

3. *Systems implementation*
The full precedence and successor lists can help to plan the implementation where phasing is necessary. For example, if one part of the system must be built before another (for testing or operational purposes), then these lists indicate which procedures generate the output elements from each part of the system.

4. *Systems maintenance*
Maintained systems often 'grow like Topsy' and include numerous inefficiencies, such as obsolete elements, even though they may be working reasonably satisfactorily. These lists can be used as above to help rationalise such a system.

Thus, a precedence analyser can be widely used by systems analysts, designers and programmers; it is particularly useful for co-ordinating the work of the many systems people involved in a large project. One aim of the CAM 01 research is to discover other possible uses for this tool.

It is recognised that a major difficulty is the general acceptance of such a tool.

Conclusion

This paper has presented CAM 01, a precedence analyser capable of producing lists of full precedence, full succedence, cycles, independent sub-networks and similar elements. The data required from a systems team is merely a list of elements with their first order precedents. CAM 01 is proven, fast and cheap to use.

CAM 01 results from research into the problems of analysing the precedences of large information processing systems. Some applications of the software have been discussed and users will no doubt find other ways of exploiting this powerful tool. A specification is available for prospective users and the author would be pleased to discuss this research with any interested organisations.

The CAM research project is now attempting to introduce structure and identification properties into the precedence analyser. Structuring allows information elements to be grouped into sets (e.g. records, messages, files, sub-schema, schema, etc.) and processing procedures to be grouped into systems (e.g. subroutines, modules, programs, applications, etc.) Identifying allows occurrences of information elements and sets to be pin-pointed in the sense of Grindley (1966). Both of these properties are seen as necessary steps towards the computer-aided design approach of Waters (1974c) and as useful additions to the basic precedence analyser, CAM 01.

Finally, the author wishes to acknowledge the assistance of his colleagues in the LSE Systems Research Group, particularly Mr. S. Paramasamy who was responsible for programming CAM 01, and Mrs. C. L. Warner, a former student of Birkbeck College who presented an M.Sc thesis on this subject.

References

BOOT, R. (1969). *DATAFLOW Internal Reports*, NCC Limited, Manchester
BURMAN, P J. (1972). *Precedence Networks*, McGraw-Hill.
GRINDLEY, C. B. B. (1966). Systematics, *The Computer Journal*, Vol. 9, No. 2, p. 124
LANGEFORS, B. (1966). *Theoretical Analysis of Information Systems*, Studentlitteratur, Sweden.
SOLVBERG, A. (1972). Formal Systems Description in Information Systems Design, Proceedings of NCC Conference on *Approaches to Systems Design.*
STAMPER, R. K. (1972). *Information*, Batsford.
TEICHROEW, D., and SAYANI, H. (1971). Automation of System Building, *Datamation*, 15 August.
WARNER, C. L. (1974). Computer Assisted Design: The Use of Precedence Matrices in Systems Definition, M.Sc. Thesis, London University.
WATERS, S. J. (1972). A Survey of CAM and its Publications, Proceedings of NCC Conference on *Approaches to Systems Design.*
WATERS, S. J. (1974a). *Introduction to Computer Systems Design*, NCC Limited, Manchester.
WATERS, S. J. (1974b). Methodology of Designing Computer Systems of Files and Programs, Ph.D Thesis, London University.
WATERS, S. J. (1974c). Computer-Aided Methodology of Computer Systems Design, *The Computer Journal*, Vol. 17, No. 3, p. 211.