Lecture Notes in
Control and
Information Sciences

Edited by A.V. Balakrishnan and M.Thoma

17

O. I. Franksen · P. Falster
F. J. Evans

Qualitative Aspects of Large Scale Systems

Developing Design Rules Using APL

Springer-Verlag
Berlin Heidelberg New York 1979

Series Editors
A. V. Balakrishnan · M. Thoma

Advisory Board
L. D. Davisson · A. G. J. MacFarlane · H. Kwakernaak
Ya. Z. Tsypkin · A. J. Viterbi

Authors
O. I. Franksen and P. Falster
Electric Power Engineering Department
The Technical University of Denmark

F. J. Evans
Department of Electrical & Electronic Engineering
Queen Mary College, University of London

ISBN 3-540-09609-4 Springer-Verlag Berlin Heidelberg NewYork
ISBN 0-387-09609-4 Springer-Verlag NewYork Heidelberg Berlin

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically those of translation, reprinting, re-use of illustrations, broadcasting, reproduction by photocopying machine or similar means, and storage in data banks.

Under § 54 of the German Copyright Law where copies are made for other than private use, a fee is payable to the publisher, the amount of the fee to be determined by agreement with the publisher.

© Springer-Verlag Berlin Heidelberg 1979
Printed in Germany

Printing and binding: Beltz Offsetdruck, Hemsbach/Bergstr.
2060/3020-543210

FOREWORD

This monograph has been given a rather broad general title, although a glance at the table of contents quickly shows that it is constructed around the central concepts of controllability and observability. If any excuse is needed for this, it is that we felt the work does represent an investigation in the true inductive spirit from the particular to the general. We also feel that the use of an interactive APL facility has allowed the work to proceed in the classical empirical manner, which demonstrates the role of a "computer laboratory" in modern systems analysis.

As Karl Popper has pointed out, theories can never be verified but only falsified, by experiment, but it is experiment, fertilised by intuitive and imaginative thoughts, that generate the theories.

We have also been greatly influenced by several other convictions. Firstly, that the application of a group theoretic approach to systems analysis should bring benefit, and we have attempted to show that the adoption of a type of Erlanger Program which has proved so successful elsewhere, is also of value here. Indeed, a fundamental advantage of such an approach is that it separates qualitative properties from quantitative and yet combines them into a consistent logical framework permeated by abstract symmetries.

Secondly, we feel there is a need to supplement the algebraic symbolism and associated reaching after complete mathematical rigour, by the exposure of more qualitative or structural properties. This can be a source of further theoretical development in itself. Thirdly, it is our opinion that the type of approach described here is far more fruitful in the production of prescriptive design rules than the more analytic algebraic approach so far proposed.

So, in conclusion, we feel that these lecture notes will interest those who like to learn from numerical illustrations and are interested also in the scaffolding, which so often is carefully demolished in many current publications.

Lyngby and London, February 1979

O.I. Franksen, P. Falster & F.J. Evans

ACKNOWLEDGEMENT

The authors wish to express their gratitude to The British Council for financial support enabling them to overcome the problem of geographical distance, and their indebtedness to Kommunedata I/S, the computer organization of the Danish Municipalities, for making available the excellent service of its APLSV system.

APL GLOSSARY

For convenience this glossary list the subset of build-in, socalled *primitive*, APL operations applied in this monograph. Many of these APL operations are introduced, illustrated, and explained in the course of the text. However, since the aim is to use APL as a tool, no attempt as such is made to actually introduce APL as a programming language or a mathematical notation. For this purpose the reader should consult other literature like the references given in the text or the programming manuals of the computer manufacturers.

For each of the APL operations of this glossary is given in tabular form: its name; APL notation; conventional mathematical notation (if it exists); a brief explanation; and a reference to an example in the text. In this connection it should be noted that the order of execution of an APL expression is *from right to left with no priority* among the operations. That is, parentheses are used in the conventional manner to introduce any desired priority order of execution. From an overall point of view the subset of APL operations listed here, may be subdivided into operations on the *shapes* of the arrays and operations on the *elements* of the arrays respectively. This classification is maintained in the glossary organizing in order the operations into two tables called Tables A1 and A2.

NAME	APL SIGN	MATH. SIGN	EXPLANATION	EXAMPLE
ORIGIN	⎕IO←1		Indexing beginning with 1 (or 0) as specified	
SIZE	⍴A		Dimension or size of array A. ⍴⍴A gives the *rank* of A.	(I.11) (I.27)
RESHAPE	V⍴A		Reshapes right argument, array A, to the size of the left argument, vector (or scalar) V.	*ASSIGN* page 62
INDEX	V[I] M[I;J] M[I;]	V_i $M_{i,j}$ $M_{i.}$	Indexing of a vector V, a matrix M, and a row in matrix M.	(I.46)
COORDINATE	...[I]A		Following a primitive APL-operation (or function) this expression indicates that the operation is applied along the I'th coordinate axis of array A.	(I.68) (I.14a)
MONADIC TRANSPOSE	⍉A	A^t	Generalized transpose reversing the order of coordinate axes of array A. Note, that APL cannot distinguish between row and column vectors.	(I.18)

Table A1. Structural Operations Based on Index-Sets

NAME	APL SIGN	MATH. SIGN	EXPLANATION	EXAMPLE
DYADIC TRANSPOSE	*I*⍉*A*		A permutational transposition of the coordinate axes of array *A* specified by index vector *I*. Equating two indices in vector *I* produces a diagonal hyperplane of array *A*.	(I.14b) (I.26)
CATENATE	*A*,[*I*]*B*		Joining arrays *A* and *B* along an *existing* coordinate axis *I*, maintaining the maximal given rank.	(I.14a)
LAMINATE	*A*,[*K*]*B*		Joining arrays *A* and *B* along a *new* coordinate axis *K* (non-integer), increasing the rank by 1.	(I.14a)
TAKE	*V*↑*A*		*V* is an integer or a vector of integers. If *V*[*I*] is positive (negative), take the first (the last) *V*[*I*] components of array *A* along its I'th coordinate axis.	*PLUSΔMPLY* page 9
COMPRESS	*V*/[*I*]*A*		*V* is a Boolean vector of the size of dimension *I* of array *A*. The components along dimension *I* of *A* corresponding to unit elements in *V*, are selected.	*PLUSΔMPLY* page 9

Table A1 continued

NAME	APL SIGN	MATH. SIGN	EXPLANATION	EXAMPLE
ASSIGNMENT	*R*←*A*	R=A	Variable *R* defined equal to *A*.	(I.20)
OUTPUT	⎕←*A*		Assign the value of *A* to ⎕, i.e. print the value of *A*.	(I.24)
	A		Abbreviated version of the latter above.	(I.13)
MULTIPLE ASSIGNMENTS	⎕←*R*←*A*		Printing variable *R* defined equal to *A*.	(I.24)
ORDER OF EXECUTION			Composite expressions are evaluated from right to left with no priority order among the operations	
PARENTHESES	(....)	(....)	Parentheses are used conventionally to introduce a priority order in the evaluation of computer expressions	(I.11)
ADDITION	*A*⊕*B*	A⊕B	Scalar "addition" generalized to the element-by-element operation ⊕ of conforming arrays *A* and *B*	(I.49)
REDUCTION	⊕/[*I*]*A*	$\sum_{i=1}^{N} A_{..i..}$	Scalar "summation" generalized to ⊕ reduction along the *I*'th coordinate axis of array *A*	(I.26) (I.68)
SCAN	⊕\[*I*]*A*		Scalar "accumulation" generalized to a ⊕ scan along the *I*'th coordinate axis of array *A*. Applied to a vector the result is a vector of the same size with the *K*'th element equal to the ⊕ reduction over the first *K* elements.	<u>*ASSIGN*</u> page 62

Table A2. Constituent Operations Based on Scalar Elements

NAME	APL SIGN	MATH. SIGN	EXPLANATION	EXAMPLE
OUTER PRODUCT	*A*∘.⊗*B*	$C_{mnpq} = A_{mn}B_{pq}$	Tensor outer product, evaluated as the "product" of each element in array *A* with each element in array *B*.	*PLUSΔMPLY* page 9
INNER PRODUCT	*A*⊗.⊗*B*	AB or A×B	Tensor inner product generalizing the conventional matrix product to conforming arrays *A* and *B*.	(I.26) (I.67)
INVERSE	⌹*M*	M^{-1}	The inverse of matrix M or, if it does not exist, a *DOMAIN ERROR* message	(I.27)
TENSOR CONTRACTION	+/[*K*]*I*⍉*A*	A^k_k	Equating two indices *K* in the index-set *I* followed by a plus-reduction (summation) along the equated index *K*.	(I.20) (I.26)
INDEX GENERATOR	⍳*N*		Applied to a non-negative integer N a vector of the first N integers is produced counting from origin ⎕*IO*.	*PLUSΔMPLY* page 9
GRADE UP (DOWN)	⍋*V* (⍒*V*)		The index permutation vector that would sort the elements of vector V in ascending (descending) order.	(III.11)

Table A2 continued

TABLE OF CONTENTS

page