THE UNIVERSITY OF MICHIGAN

COMPUTING RESEARCH LABORATORY[1]

PERFORMABILITY MODELS AND SOLUTIONS

David G. Furchtgott

CRL-TR-8-84

JANUARY 1984

Room 1079, East Engineering Building
Ann Arbor, Michigan 48109
USA
Tel: (313) 763-8000

[1]Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author.

PERFORMABILITY MODELS AND SOLUTIONS

by
David Grover Furchtgott

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer, Information and Control Engineering)
in The University of Michigan
1984

Doctoral Committee:

Professor John F. Meyer, Chairman
Professor Daniel E. Atkins
Professor Arch W. Naylor
Associate Professor Robert L. Smith
Professor Bernard P. Zeigler, Wayne State University

ABSTRACT

PERFORMABILITY MODELS AND SOLUTIONS

by

David Grover Furchtgott

Chairman: John F. Meyer

A principal goal of computing system evaluation is the measurement of the system's ability to perform. Measures such as performance, reliability, and effectiveness are often employed, but such metrics are often not suitable when evaluating systems in the increasingly important class of degradable systems. Among the measures proposed for such systems is "performability," which is simply the probability measure of the system performance variable. Classical performance and classical reliability are specialized cases of performability.

To be effective, performability evaluation requires tractable techniques of solution. This dissertation concerns the modeling, calculation, and use of a system's performability. Specifically, we examine two broad classes of performability models: (1) those wherein system performance assumes values in an arbitrary, finite set, and (2) those systems where performance is continuous and identified with "reward".

For the first class of models, a methodology for relating low-level behavior to high-level system performance is formalized. Behavior is characterized in terms of finite arrays of variables having finite domains. A calculus is developed for manipulating such arrays, and algorithms are described for deriving the set of low-level behaviors which result in each system performance level. The probability of each performance level (and hence the performability) is obtained by

calculating the probability of the corresponding set of behaviors. Also described and illustrated is METAPHOR, a computer package implementing these algorithms.

For the second class of performability models, a general method for determining the probability distribution function of the performance variable (i.e, the performability) is derived. The result is an intergal expression which can be solved either analytically or numerically. Examples of both types of solutions are given, and procedures implementing the numerical solution and included within METAPHOR are described.

TABLE OF CONTENTS

DICTIONARY OF PRINCIPAL SYMBOLS

Symbol	*Definition*	Page
0	index denoting the top level of a model hierarchy	48
0	least element of a lattice	105
$(A, \mathcal{B}, p)$	performability probability space	57
A	accomplishment set	39
A	accomplishment set	98
a	an element of the accomplishment set A	39
A	finite set of non-negative integers	96
a	weight of a cube	111
a	weight of an anticube	113
a_i	i^{th} accomplishment level	67
$[a_k\ C_{kh}]_{kh:p \times r}$	array representation of disjunctive normal form of the function f	116
A^N	lattice formed by the direct product of N lattices$A$	105

A'	tentative accomplishment set	44
ATC	air traffic control	85
B	$B \subseteq$ PERF	26
B	Boolean lattice	103
$\mathcal{B}$	set of measurable accomplishment levels	57
$\mathbf{B}_{2^r}$	free Boolean lattice on $r+1$ generators	104
$[b^g_{jknh}]_{g:N_{jk}}$	lattice exponent	127
$[b^g_{jknh}]_{jknhg:r \times s \times l \times p_i \times N_{jk}}$	array of lattice exponents	127
B_k	in the example of Section 3.3.9, a measurable set of accomplishment levels: $\{a \mid a > h \cdot k\}$	53
$[b^g_{kh}]_{g:N_j}$	binary vector denoting the subset C_{jk} of U_j	117
C	array of lattice exponents	116
$c(\mathbf{x})$	cube function	111
$[(C_{jk})]^c_{r \times s}$	complement of a cube function	124
(C_{jknh})	lattice exponent	127

$(C_{(jk)nh})$	the jk^{th} exponent of the h^{th} cube function having weight a_n	127
$[(C_{jknh})]_{jknh:r \times s \times l \times p_i}$	array of lattice exponents	127
$[C_{kh}]_{p \times r}$	array of lattice exponents	116
$c(\mathbf{x})^c$	complement of the exponent of a cube function $c(\mathbf{x})$	124
d	depth of a node within a tree	144
DOMAIN(f)	the domain of the function f	-
IMAGE(f)	the image of the function f	-
$d(\mathbf{x})$	anticube function	112
div	integer division	145
EXTEND	function that extends a tentative trajectory $u' \in U'$ so that the trajectory is a total function	46
E[X]	the expected (mean) value of the random variable X	-
$\mathfrak{E}$	basic model event space	58
$\mathfrak{E}^i$	i^{th}-level basic model event space	63
$\mathfrak{F}$	set of measurable trajectories U	57

$\mathbf{F}$	array representation of disjunctive normal form of the function f	116
F	disjunctive normal form of a discrete function f	115
f	integer function	96
$\bar{f}$	negation of a discrete function f	113
FCS	flight control system	85
$\mathcal{F}^i$	i^{th}-level trajectory event space	64
f_j	the j^{th} projection of f	98
$F_{Y\|U,k;\mathbf{x}^k}(\bullet \mid u_k)$	the distribution of Y for system S_k given the sequence $\mathbf{U}$	243
$f_{i\|j,j-1,\ldots,i+1,\lambda_{i,j}}$	the density of the sojourn time in state i, conditioned on the history of the process	243
G	conjunctive normal form of a discrete function f	115
g	system description function	26
h	in the example of Section 3.3.9, the length of the utilization period T	53
h	random variable relating the base probability space $(\Omega, \mathcal{E}, P)$ to the trajectory probability space $(A, , p)$	59
h^i	random variable relating the i^{th}-level base probability space $(\Omega^i, \mathcal{E}^i, P^i)$ to the i^{th}-level basic trajectory probability space	64

k	in the example of Section 3.3.9, the fraction of the utilization period	53
l	greatest element of a lattice	105
m	index denoting the bottom level of a model hierarchy; there are $m + 1$ levels in the hierarchy	48
a mod b	a modulus b, i.e., the remainder when a is divided by b	145
N	number of entries in the value vector of discrete function f	102
NAV	navigation	85
N_j	number of elements in U_j	117
P	trajectory set	90
p	vector of the number of cube functions having weight p_n	127
p	number of array products in a trajectory set	90
p	performability	57
PERF	performance set	25
P	basic model probability measure	58
$\mathbf{p} = [p_0\ p_1\ \cdots\ p_l]$	vector of the number of cube functions having weight p_n	127

$\mathbf{P}^c$	complement of an array product $\mathbf{P}$	94
$\mathbf{p}^{\gamma_0}$	array of the number of cube functions comprising γ_0	135
$[p_n^{\gamma_0}]$	array of the number of cube functions comprising γ_0	135
$\mathbf{p}^{\gamma_i}$	array of the number of cube functions comprising γ_i	138
$[p_n^{\gamma_i}]_{n:l}$	array of the number of cube functions comprising γ_i	138
$\mathbf{P}^i$	i^{th}-level basic probability measure	63
$\mathbf{p}^{\kappa_0}$	array of the number of cube functions comprising κ_0	134
$[p_n^{\kappa_0}]_{n:l}$	array of the number of cube functions comprising κ_0	134
$\mathbf{p}^{\xi_{j_0k_0\kappa_i}}$	array of the number of cube functions comprising $\xi_{j_0k_0\kappa_i}$	134
$[p_n^{\xi_{j_0k_0\kappa_i}}]_{n:N_{j_0k_0}^{i-1}}$	array of the number of cube functions comprising $\xi_{j_0k_0\kappa_i}$	134
$\mathbf{Pr}$	trajectory probability space probability measure	57
$\mathbf{P}^r$	distributive lattice	103
$\mathbf{P}^r$	free distributive lattice on $r+1$ generators	103

$\Pr^i$	i^{th}-level trajectory probability measure	64
p_{SIM}	in the example of Section 3.3.9, the performability of the simplex computer	53
p_{TMR}	in the example of Section 3.3.9, the performability of the triple modular redundant computer	53
p_{S_i}	performability distribution of system S_i	242
Q	state space of the base model	40
q	structure	26
Q_c^i	level-i composite state space	82
Q'	tentative state space	46
r	number of attributes (components or rows) of an array product	82
$\mathbb{R}$	set of real numbers	59
r_i	number of level-i composite components	83
$[R_{jk}^l]_{r \times s}$	array product (trajectory set)	90
$\mathbb{R}^n$	n-dimensional space of real numbers	59
R_{ij}	subset of Q_{ij} in an array product	89
s	number of columns (components or rows) of an array product	82

STRUC	structure set	25
SIM	in the example of Section 3.3.9, the simplex computer	53
S_i	a system with $i+1$ states which degrades in the state sequence $\mathbf{u}_i = (u_i, u_{i-1}, \ldots, u_0)$	242
OBSER	time base	25
t	observation time	26
T	parameter set of the base model	40
t	time $t \in T$	40
T_b^i	level-i basic state space	82
T_b^i	level-i basic utilization period	82
T_c^i	level-i composite utilization period	82
T^i	level-i parameter set	62
TMR	in the example of Section 3.3.9, the triple modular redundant computer	53
$(U\mathfrak{S}, \Pr)$	trajectory probability space	57
u	trajectory	26
u	trajectory in the trajectory space U	40

U	trajectory space	40
$\mathbf{u}_b^i$	basic trajectory at level-i	82
$\mathbf{U}_b^i$	level-i basic trajectory space	82
$\mathbf{u}_{bk}^i$	level-i basic component k	83
$\mathbf{u}_b^i(t_k)$	level-i basic observation at time t_k	83
$[\mathbf{u}_{c_k}^i(t_k)]_{k:\theta_i}$	component trajectory of the j^{th} basic component	85
$\mathbf{u}_{b_j,\,t_k}^i$	a basic variable	85
$\mathbf{u}_{b_j}^i$	component trajectory of the j^{th} basic component	85
$\mathbf{u}_{b_j}^i(t_k)$	level-i basic component j observed at time t_k	84
$\{\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_r\}$	trajectory set	88
$\mathbf{u}$	trajectory	88
$[u_{jk}]_{r \times \theta}$	trajectory	88
U_{CONS}	set of consistent trajectories	47
$\mathbf{u}_b^i$	basic trajectory	83
$\mathbf{u}_c^i$	composite trajectory	83

$\mathbf{u}_c^i$	composite trajectory at level-i	82
$\mathbf{U}_c^i$	level-i composite trajectory space	82
$\mathbf{u}_{ck}^i$	level-i composite component k	83
U_{INCONS}	set of inconsistent trajectories	47
$\mathbf{u}_c^i(t_k)$	level-i composite observation at time t_k	83
$\mathbf{u}_{c_j,\, t_k}^i$	a composite variable	85
$\mathbf{u}_{c_j}^i$	component trajectory of the j^{th} composite component	85
$\mathbf{u}_{c_j}^i(t_k)$	level-i composite component j observed at time t_k	84
$\mathbf{U}_b^i$	level-i basic trajectory space	98
$\overline{\mathbf{U}}_b^i$	level-i basic model trajectory space	99
$\overline{\mathbf{U}}_b^m$	basic model trajectory space	99
$\mathbf{U}_c^i$	level-i composite trajectory space	98
$\mathbf{U}^i$	level-i trajectory space	98
U_i	i^{th} Cartesian component of set U	97
U_{ij}	i^{th} Cartesian component of the j^{th} Cartesian component of the set U	97

$U^i_{b,jk}$	level-i basic trajectory space of component p during phase k	99
$U^i_{c,jk}$	level-i composite trajectory space of component p during phase k	99
$(U^i, \mathfrak{F}^i, \Pr^i)$	i^{th}-level trajectory probability space	64
U_i	level-i trajectory space	48
$\mathbf{U}^i$	level-i trajectory space	82
$\overline{U}^i$	the level-i trajectory space along with all the basic trajectory spaces of higher level models	50
$u^k_{i,j}$	behavior of component i during phase j at level of abstraction k	69
$[\ u^i_j(t_k)\ ]_{j:(r_i+\rho_i+1)}$	trajectory observation at time t_k	85
$\mathbf{u}^i(t_k)$	trajectory observation at time t_k	85
$[\mathbf{u}^i_j]_{j:(r_i+\rho_i+1)}$	matrix expansion of level-i trajectory	83
$[\ u^i_j(t_k)\ ]_{jk:c,b\times s_i}$	matrix expansion of level-i trajectory	83
u_{jk}	the jk^{th} entry of trajectory $\mathbf{u}$	88
$[\ u^i_{jk}\ ]_{jk:(r_i+\rho_i+1)\times s_i}$	matrix expansion of a level-i trajectory	84
u'	A partial low-level trajectory	45

U'	tentative trajectory space	45
u_ω^i	sample function of the process X_b^i (the i^{th}-level base model)	63
u_ω	sample function of the process X (the base model)	58
$\hat{v}_i$	all the remaining time in $[0, t]$ after accounting for the intervals $(v_n, v_{n-1}, \ldots, v_{i+1})$	212
$\tilde{v}_{i-1}$	all the remaining time in $[0, t]$ after accounting for the intervals $(v_n, v_{n-1}, \ldots, v_i)$	210
$\hat{\mathbf{v}}_{\mathbf{u},i}$	$\hat{\mathbf{v}}_{\mathbf{u}}$ modified by the index i	211
$\tilde{\mathbf{v}}_{\mathbf{u},i}$	$\tilde{\mathbf{v}}_{\mathbf{u}}$ modified by the index i	211
$\hat{\mathbf{v}}_{\mathbf{u}}$	$(v_n, \ldots, v_{i+1}, \hat{v}_i, \hat{v}_{i-1}, \ldots, \hat{v}_0)$	211
$\tilde{\mathbf{v}}_{\mathbf{u}}$	$(v_n, \ldots, v_i, \tilde{v}_{i-1}, \tilde{v}_{i-2}, \ldots, \tilde{v}_0)$	211
$\mathbf{W}$	vector of cube function weights	116
$[w_i \ [b_{jknh}^g]_{g:N_{jk}}]_{jknh:r \times s \times l \times p_i}$	array representation of a discrete function	128
w_k	weight of cube function k	116
$[w_k]_{k:p}$	vector of cube function weights	116
$\hat{w}_t^i$	w_t^i for $\hat{\mathbf{v}}$	212

$\tilde{w}_i$	w_t^i for $\tilde{\mathbf{v}}$	211
$\hat{\mathbf{w}}^u$	the vector of $\hat{w}_t^i$ corresponding to $\hat{\mathbf{v}}_u$	212
$\tilde{\mathbf{w}}^u$	the vector of $\tilde{w}_i$ corresponding to $\tilde{\mathbf{v}}_u$	211
X	base model	41
X	base model	58
$\{X^0, X^1, \cdots, X^m\}$	model hierarchy	65
X_b^i	level-i basic process	63
$X_b^i(\omega)$	sample function of the process X_b^i (the i^{th}-level base model)	63
$X_{b,t'}$	level-i basic random variable	63
X_c^i	level-i composite process	63
$X_c^i(\omega)$	sample function of the process X_c^i (the i^{th}-level composite model)	64
$X_{c,t'}$	level-i composite random variable	63
X^i	level-i model	62
X^i	level-i model	64
$x_j^{(C_j)}$	lattice exponentiation	108

ξ_i	the projection on i	86
$[x_{ij}]_{i,j:r \times s}$	an $(r+1) \times (s+1)$ dimensioned matrix	82
$[x_{ij}]_{r \times s}$	an $(r+1) \times (s+1)$ dimensioned matrix	82
$[x_i]_{s_i}$	an $s+1$ dimensioned vector	81
$[x_0, x_1, \ldots, x_s]$	an $s+1$ dimensioned vector	81
$[x_i]_{i:s}$	an $s+1$ dimensioned vector	81
X_t	random variable of the base model	41
(X, γ)	performability model	42
$X(\omega)$	sample function of the process X (the base model)	58
Y	system performance	59
Y_{SIM}	in the example of Section 3.3.9, the performance of the simplex computer	53
Y_{TMR}	in the example of Section 3.3.9, the performance of the triple modular redundant computer	54
Δ^{dh}	array representation of the h^{th} partial cube function at depth d	145
$[(\Delta^{dh}_{(jk)})]_{jk:(r_i+\rho_i+1) \times s_i}$	array representation of the h^{th} partial cube function at depth d	145

$(\Delta^{00}_{(jk)}) = U_{jk}$	full cube function	145
Φ	null array product	112
Φ	the null array	91
γ'	tentative capability function	45
γ	capability function	41
γ	capability function	57
γ	the capability function	100
Γ^0	array representation of γ_0	135
$[(\Gamma^0_{(jk)nh})]$	array representation of γ_0	135
Γ^i	array representation of γ_i	138
γ_i	the level-i based capability function	99
γ^{-1}	inverse of the capability function γ	42
$[\,\Gamma^{ie}\,]_{e:i}$	array representation of γ_i	138
γ_i	level-i-based capability function	50
γ_i^{-1}	inverse of the level-i-based capability function	51
γ''	function mapping an inconsistent trajectory to $\not\in$	47

$\mathbf{K}_0$	array representation of κ_0	134
$\left[K^0_{(jk)nh}\right]_{jknh:(r_i+\rho_0+1)\times s_0\times l\times p_n^{\kappa_0}}$	array representation of κ_0	134
$\mathbf{K}_i^{j_0k_0}$	array representation of $\xi_{j_0k_0}\kappa_i(\mathbf{u})$	134
κ_u^i	sample function of the process X_c^i (the i^{th}-level composite model)	64
λ	in the example of Section 3.3.9, the failure rate of each computer	53
λ_i	parameter for state u_i	242
$(\Omega, \mathfrak{E}, P)$	basic model probability space	58
Ω	basic model sample space	58
$(\Omega^i, \mathfrak{E}^i, P^i)$	i^{th}-level basic model probability space	63
Ω^i	i^{th}-level basic model sample space	63
ϕ	the empty set	91
ϕ	structure function	27
ϕ	structure function	30
ρ_i	number of level-i basic components	83

ξ_i:A→A_i	projection of A on i	86
$\xi_{jk}(\kappa_{i+1})$	j^{th} component function at observation k	87
$\xi_{jk}(\kappa_{i+1})$	projection of κ on jk	86
A^B	direct product of set A by set B	26
ϕ	place-holding state used to extend those trajectories $u' \in U'$ that are not defined over the entire set T	46
A^c	set complementation	-
$[f_u]_N$	value vector of the discrete function f	102
$[f(u)]_N$	value vector of the discrete function f	102
$a \underset{0}{>} b$	a can be any non-negative (or zero) number exceeding b	220
A^B	the direct product of set A by set B	-
$a \dot{-} b$	$a - b$ if $a \geq b$ and 0 otherwise	231
$A - B$	set difference, i.e., the set containing those elements of set A that are not in set B	-
*	the full set (or universe)	91
$\lvert T \rvert$	the size of the set T	59

$A \times B$	Cartesian product of sets A and B, i.e., $\{(a,b) \mid a \in A, b \in B\}$	64
$\bigtimes_{r=0}^{n} U_r$	Cartesian product: $U_0 \times U_1 \times \cdots \times U_n$	97
$\mathcal{A} \otimes \mathcal{B}$	smallest σ-algebra containing the product of the σ-algebras $\mathcal{A} \times \mathcal{B} = \{A \times B \mid A \in \mathcal{A}, B \in \mathcal{B}\}$	64
U	finite set of non-negative integers	96

LIST OF DEFINITIONS

In the text, defined terms are usually denoted by *italics*.

LIST OF TABLES

Table

LIST OF FIGURES

Figure

LIST OF KEY PROPOSITIONS

Proposition

LIST OF ALGORITHMS

Algorithm

LIST OF APPENDICES

Appendix

CHAPTER 1

INTRODUCTION

1.1. Problem Statement

A principal goal of computing system evaluation is the measurement of the system's ability to perform. Reflecting that ability are several descriptors commonly used to characterize figures of merit for a system, including performance [1]-[4], reliability [5]-[7], and effectiveness [8]-[10]. (See [11, Section 5] for a bibliography of literature about formal system evaluation published recently.) The characterizations listed are often studied separately, by stipulating that other descriptors are invariant. For example, reliability evaluations typically assume a system that has a constant performance rate. Similarly, performance evaluations often, though not always, suppose a system that never fails. A performance-reliability disjunction is suitable for systems whose ability to perform is either total or nonexistent when faults are present. However, to evaluate those classes of computing systems whose performance is "degradable," features of both performance and reliability must be blended. Recently, several combined performance and reliability measures for dealing with degradable systems have been suggested [12]-[28]. In particular, Meyer [29] has introduced a universal modeling framework called performability, which is well suited for the measurement of the combined performance-reliability.

To employ effectively such measures within system evaluations, we require proper generalizations of both discrete event-oriented models utilized in reliability evaluation (e.g., fault trees), and analytic models and solution methods employed in performance evaluation (e.g., Markov models).

The formal description of performability [29] delineates in broad terms the quantities that the analyst requires. The description also suggests a basic method for arriving at those quantities. However, to be effective, a framework such as performability requires tractable techniques of solution. As an analogy, the formal characterization of differential equations, although useful for *describing* physical phenomena, would not be nearly so useful to engineers without techniques for *solving* such equations. In the area of reliability, almost all the literature during the past twenty years has concerned not the *definition* and *nature* of reliability, but the *acquisition* and *application*, i.e., the modeling, calculation, and use of a system's reliability. The thrust of this dissertation concerns the modeling, calculation, and use of a system's performability.

Often, we shall speak of "solving" a performability model, or of the "solution" of a performability model. By "solving" a performability model is meant the following: Given a general performability model as discussed in Chapter 3 [FUNDAMENTAL CONCEPTS AND RESULTS], one cannot always directly calculate the system's performability. Sometimes, by using suitable manipulations, one can obtain from the original model a description that allows direct calculation of the performability. For our purposes, such derived descriptions will usually take the form of an analytic equation, an integral equation, or a matrix of such equations. The process of carrying out these manipulations on the model will be called ***solving***, while the results of solving will be called the ***solution***. "Solving" can also involve solving differential or integral equations, although in this dissertation, we shall often consider a model "solved" when an integral equation is obtained. ***Construction*** of a model refers to the process of obtaining a model–a model cannot be solved until it has been constructed. Sometimes, a model may be simultaneously constructed and solved. Indeed, the methodologies proposed in this dissertation allow some degree of simultaneous construction and solution. A solution can

be ***evaluated*** by evaluating the equations comprising the solution. Using the analogy presented in the preceding paragraph, the study of modeling systems by means of differential equations has almost identical terminology.

In summary, this dissertation addresses the questions of modeling a system to reflect its performability and solving that model.

1.2. Thesis

> *The thesis of this study is that tractable techniques for solving certain classes of performability models exist and that these techniques can be applied to the analysis of degradable computing systems.*

1.3. Research Objectives

Some introductory work (Ballance, Furchtgott, Meyer, and Wu) [30]-[33][1] has been done concerning performability models and solutions for certain restricted classes of systems. The objective of this research is to extend that initial activity by continuing the development of techniques for analyzing degradable computing system performability. In particular, we seek the development of methods for solving performability models. More specifically,

a) we shall generalize the analysis of performability models having discrete performance variables, and

b) we shall extend the development of methods for solving performability models having continuous performance variables.

With regard to a):

---

[1]This thesis addresses the specific problem of solving certain classes of performability models. (See section 1.2 [Thesis].) The above are major works of reference addressing the issue of solving specific performability models. More generally, the following list of available literature (excluding internal memoranda) more completely traces the development of performability at The University of Michigan (Ballance, Furchtgott, Meyer, Movagar, and Wu) (see Chapters 2 [BACKGROUND AND LITERATURE SURVEY] and 3 [FUNDAMENTAL CONCEPTS AND RESULTS]): [11], [29]-[64].

i) We shall describe a calculus for relating low-level structural behavior to high-level system performance.

ii) We shall propose algorithms and heuristics for efficiently performing the calculations associated with the calculus.

iii) We shall discuss a computer package we have written that implements those algorithms.

iv) Employing that computer package, we shall analyze example systems.

In the case of b):

i) We shall develop the problem more generally in the context of reward models and nonrecoverable processes [55], [59].

ii) We shall derive a general solution for the performability of a system modeled by a finite-state, acyclic, nonrecoverable process. This solution takes the form of an integral equation. The solution will be illustrated.

iii) We shall derive a recursive form of the solution of ii). In addition, we will present specific examples of the recursive solution.

iv) We shall discuss a computer package we have written that implements the solution.

v) Using the above tools, we shall analyze a nontrivial example.

vi) We shall begin consideration of still more general models.

These developments constitute a significant extension of performability theory, and more broadly, of performance and reliability theory for degradable systems.

Although not in the scope of this thesis, the analysis will support a methodology for determining optimal system configurations, phasing strategies, and component values for degradable systems, Finally, as an aside, among other issues which this thesis does not attempt to address are the measure-theoretic basis of performability (the reader is referred to

Meyer [29] and Wu [59], [60]), or developing new model types such as "extended" queueing networks or stochastic Petri nets (see, for example, Movagar [65] and Meyer [64], [66]).

1.4. Dissertation Organization

The remainder of this dissertation is organized as follows: The next chapter motivates in detail the research and presents a short review of the previous work in combining reliability and performance. Review is important to put the present work in perspective. Much previous work has been done in the general area of computer evaluation, but less has been done in the area of degradable computing system evaluation, and little has been done in the study of performability evaluation.

Chapter 3 [FUNDAMENTAL CONCEPTS AND RESULTS] presents an overview, in non-technical language, of the fundamental concepts and results of performability and of the dissertation. Chapter 4 [DISCRETE PERFORMANCE VARIABLE (DPV) METHODOLOGY] addresses evaluations in which the performance of the system can be described by using a finite number of values, and Chapter 5 [CONTINUOUS PEFORMANCE VARIABLE (CPV) METHODOLOGY] investigates the case in which a continuum of values is required. Finally, Chapter 6 [CONCLUSIONS] summarizes the thesis results and suggests future research.

CHAPTER 2

BACKGROUND AND LITERATURE SURVEY

2.1. Introduction

To understand the need for combined measures such as performability, consider the two major techniques usually employed for computer system evaluation: performance analysis and reliability analysis. Classical performance analysis presumes a system and environment whose structure and properties for all time are either constant (e.g., an M/M/1 queueing system) or recurrent (for instance, a system continually cycling through failure and repair). Thus, most models assume that a system attains an average steady-state behavior and that if several "copies" of the system were built, all would have identical performance. Reliability analysis, on the other hand, typically deals with determining the probability that a system will perform "successfully," where the notion of "success" is usually based on the internal structure of the system. Such an evaluation is usually concerned not with how well the system performs its function over the utilization period, but with whether the system performs above a given threshold during the utilization period.

2.2. Degradable Systems

With the above meanings, performance and reliability analyses are not applicable separately when analyzing systems in the increasingly important class of ***degradable***

computing systems. A system is in the class if the total performance (or worth) of the system can vary from utilization to utilization because of changes in one of the following:

1) the structure of the system,
2) the internal state of the system, and
3) the environment.

The variations listed are generally modeled by stochastic processes, and the length of the utilization period will usually, but not necessarily, be finite.

Most of the degradable systems of interest to us are multi-processor computing systems used in applications requiring continuous availability (i.e., no downtime) during a finite interval of time. Examples of such applications include real-time control computers in aircraft, spacecraft, and nuclear power plants. Such systems can also be utilized in roles where availability is important but not critical–for example, in communications, manufacturing, business, banking, and airline reservation systems.

Typically, degradable multi-processor systems operate as follows: when a fault is detected and isolated in a hardware or software module, the module is discarded. The performance of the system may decrease since certain non-critical computations may be performed more slowly or not at all. The systems thus trade decreased computing power in exchange for additional availability.

Specific examples of degradable computers include the SIFT (Software Implemented Fault Tolerance) computer [67], [68] and the FTMP (Fault-Tolerant Multi Processor) computer [69], [70]. Other types of degradable systems include those in which performance, and, in certain cases, reliability, vary as a function of time, workload, and state (e.g., timesharing computing systems and distributed computing systems).

Degradable systems also appear in disciplines other than computer engineering. As illustrations:

(1) *A set of machines in a small factory.* As machines break down, the ability of the factory to produce a product will be decreased; the loss of key machines may bring the entire factory to a stop, while the loss of other machines may reduce throughput, increase

costs, etc. One concern is to maximize the factory's output while minimizing the number of key machines and hence the possibility of a complete factory shutdown.

(2) *An automobile with pneumatic tires.* As air leaks from the tires, the driving efficiency (in terms of miles per gallon of gasoline) of the car decreases. Performance/reliability tradeoffs may exist if a tire with better initial efficiency looses air faster than a tire with a lower initial efficiency. Note that the performance degradation is continuous rather than discrete as in the previous examples.

(3) *An athletic team.* The performance of the team degrades as the athletes grow tired or are injured. Injuries are more apt to occur to a tired athlete than a rested one. One could conceptualize such problems as maximizing the performance while minimizing the injuries of the team for the present game; injuries would be of concern since they would affect the team's performance in future games. Note that the performance degradation here is both discrete (an athlete is injured) and continuous (an athlete tires).

A common characteristic of degradable systems is that performance and reliability present conflicting requirements, i.e., for a fixed amount of resources, a configuration that increases system reliability generally decreases system performance, and vice versa. Thus, degradable system design exhibits needs similar to system synthesis in which performance is to be maximized under cost constraints (e.g., see Kachhal and Arora [71] and Trivedi and Sigmon [72]). For instance, we may wish to maximize performance under reliability constraints or maximize reliability under performance constraints. The ability to examine the performance/reliability tradeoffs is the key to good engineering design of degradable systems.

2.3. The Evaluation Procedure

Most evaluations of systems such as degradable computing systems use a basic, top-down, three-step procedure. Each of the three steps presents significant research problems. In this section, the steps of the procedure are identified, previous work by other authors in each of the three steps is discussed, and the results of this dissertation research are placed in the context of the evaluation procedure.

To evaluate a system, the measure to be used must be determined. In particular, the metric must be defined and its properties described. Once the metric has been formalized, techniques for evaluating specific systems must be delineated. For the systems of interest, namely degradable computing systems, these techniques typically consist of: first, describing how the structure or composition of the system affects the chosen metric (the description may

have stochastic components), and second, characterizing how the stochastic nature of the system affects the system's structure.

Based on those observations, the following procedure is fundamental to evaluating degradable systems:

1) Define the measure to be employed,
2) Specify how the measure reflects the structure of the system, and
3) Characterize the probabilistic nature of the system's structure.

Table 2.1, Table 2.2, and Table 2.3 summarize[1] work to date on each of the above steps. Table 2.1 presents a brief overview of many important evaluation measures and notes when they were first applied to computing systems. Table 2.2 gives an outline of the various techniques that have been proposed for relating the system structure to the evaluation measure. Finally, Table 2.3 presents the techniques proposed for describing the stochastic nature of the system. Both Tables 2.2 and 2.3 also show relevant software packages. Because of the amount of information conveyed, all three tables are necessarily succinct. Most of the pertinent work to date with which we are familiar has been included. As the entries come closer to the topic of this dissertation, the table should become more complete.

Sections 2.4 [A Taxonomy of Measures and Models of Degradable Systems (Table 1)], 2.5 [Description of the Structure of Degradable Systems (Table 2)], and 2.6 [Determining the Probability Measure of System Performance (Table 3)] discuss Tables 1, 2, and 3. Sections 2.7 [Structure-Based Models], 2.8 [Other Reliability Oriented Models], and 2.9 [Performance Oriented Models] treat various classes of models.

---

[1]For an extensive, though unannotated, bibliography of work in the area of formal computing system evaluation (for the period 1977-1981), see Chapter 5 of Meyer, Furchtgott, and Movagar [11].

Measure Classification	Measure Name	Model Components: Time Base OBSER	Model Components: Structure Set STRUC	Model Components: Performance Set PERF	System description function (See KEY below) $g : STRUC^{OBSER} \rightarrow PERF$	Probability measure of system performance (See KEY below)	References	First Application to Analyzing Computing Systems
Classical Performance Measures	**Performance measures** (e.g., throughput, response time, turnaround time.)	Any set	Any set	R	Performance measure (see measures at left)	***	Many. For an overview, see [2], [3], [73]	Many, including [74]-[76]
Classical Reliability Measures	**Reliability (also derived measures such as availability, mean time between failures [MTBF])**	$\{0\}$	$\{0,1\}^N$	$\{0,1\}$	structure function ϕ	$Prob[\phi(U) = 1]$	Many. For some early formal work, see von Neumann 1956 [77], Moore and Shannon 1956 [78], and Birnbaum, Esary, Saunders 1961 [79] For an overview, see Barlow and Proschan 1975 [7]	Many. See von Neumann 1956 [77], Moore and Shannon 1956 [78] For the reliability of degradable systems see fault-tolerant systems, see Bouricius Carter, and Schneider 1969 [80] and Bouricius, Carter, Jessep Schneider, and Wadia 1971 [81]
Structure-Based Measures ("Snap-shot")	**"Cannibalization" Reliability**	$\{0\}$	$\{0,1\}^N$	$\{0, 1, \ldots, M\}$	cannibalized structure function ϕ^T	$Prob[\phi^T(U) = 1]$	Hirsch, Meisner, and Boll 1968 [82]	Hirsch, Meisner, and Boll 1968 [82]
	Reliability of "Nondichotomic" structures	$\{0\}$	$[0, 1]^N$	$[0, 1]$	structure function ϕ	Prob $[\phi(U) \leq b]$, $b \in [0, 1]$	Postelnicu 1970 [83]	Postelnicu 1970 [83]
	Reliability of Multi-state Systems	$\{0\}$	$\{0, 1, \ldots, n\}^N$	$\{0,1\}$	structure function ϕ	$Prob[\phi(U) = 1]$	Murchland 1975 [84]	Murchland 1975 [84]
	Pseudo-reliability	$\{0\}$	$[0,1]^N$	$[0,1]^N$	relative performance of system i, γ_i and an implicit structure function $\|T\|E$ [relative performance of the system]$\|T\|$ Tillman, Lie, and Hwang 1976 [85]	Tillman, Lie, and Hwang 1976 [85]	***	
	Measures oriented towards degradable systems (measures derived from reliability, e.g., availability, mean time between crashes, average processing power, proportion of time spent in degraded	$\{0\}$	$\{0,1\}^N$	$\{0,1\}$	[structure function ϕ, not explicitly stated]	$Prob[\phi(U) = 1]$	Losq 1977 [16]	Losq 1977 [16]

Table 2.1 -- A brief overview of many important evaluation measures and when they were first applied to computing systems

Measure Classification	Measure Name	Model Components: Time Base OBSER	Model Components: Structure Set STRUC	Model Components: Performance Set PERF	System description function (See KEY below) $g: STRUC^{OBSER} \to PERF$	Probability measure of system performance (See KEY below)	References	First Application to Analyzing Computing Systems
Structure-Based Measures ("Snap-shot")	Operational Effeciency	$\{0\}$	$\{0,1,\ldots,N\}$	R	workpower of lumped state i ρ^i	$E[\rho]$	Troy 1977 [15]	Troy 1977 [15]
	Job-Related Reliability	$\{0\}$	$\{0,1\}^N$	$\{0,1\}$	[a job J_i related structure function, no name for the function given] $S(J_i)$	for job J_i, Prob [$U \in S(J_i)$]	Mine and Hatayama 1979 [18]	Mine and Hatayama 1979 [18]
	User-Oriented Reliability (Apparent Capacity and Expected Elapsed Time required to correctly execute a given program.)	$\{0\}$	$\{0,1\}$	$\{0,1\}$	[structure function ϕ, not explicitly stated ϕ is the identity function.]	Prob $[\phi(U) = 1]$	Castillo and Siewiorek 1980 [21]	Castillo and Siewiorek 1980 [21]
	Dependability	$\{0\}$	$\{0,1,\ldots,N\}$	$\{0,1,\ldots,n\}$	dependability levels D_i	Prob $[D_i]$	Laprie and Medhaffer-Kanoun 1980 [86]	Laprie and Medhaffer Kanoun 1980 [86]
	An expected performance measure	$\{0\}$	$\{0,1,\ldots,N\}$	R	performance of state i, P_i	$E[P]$	Chou and Abraham 1980 [20]	Chou and Abraham 1980 [20]
	Workload Dependent Reliability Measures	$\{0\}$	$\{0,1\}$	$\{0,1\}$	[structure function ϕ, not explicitly stated. ϕ is the identity function.]	Prob $[\phi(U) = 1]$	Castillo and Siewiorek 1981 [23], [87]	Castillo and Siewiorek 1981 [23], [87]
	Performance Reliability (also derived measures such as performance availability, response time reliability, response time availability, throughput reliability)	$\{0\}$	$\{0,1,\ldots,N\}$	R	normalized performance of state i, $\eta(i)$	Prob ["system performance" $(U) \leq b$], $b \in R$	Huslende 1981 [24]	Huslende 1981 [24]
Other Reliabilty-Oriented Measures	Phased Mission Reliabilty	$\{0,1,\ldots\}$	$\{0,1\}^N$	$\{0,1\}$	structure function ϕ	Prob [$\phi(U)$ eq 1]	Winokur and Goldstein 1969 [88]	Winokur and Goldstein 1969 [88]
	Computation-Based Reliability	$\{0,1,\ldots\}$	$\{0,1,\ldots,N\}$	$\{0,1\}$	tolerance relation σ	Prob [U is a σ-success]	Meyer 1976 [12]	Meyer 1976 [12]
	Reliability	$\{0\}$	$\{0,1\}^N$	$\{0,1\}$	[structure function ϕ, not explicitly stated]	Prob $[\phi(U) = 1]$	Borgerson and Freitas 1975 [13]	Borgerson and Freitas 1975 [13]

Table 2.1 – A brief overview of many important evaluation measures and when they were first applied to computing systems

Measure Classification	Measure Name	Model Components: Time Base OBSKR	Model Components: Structure Set STRUC	Model Components: Performance Set PERF	System description function (See KEY below) $g: STRUC^{OBSKR} \rightarrow PERF$	Probability measure of system performance (See KEY below)	References	First Application to Analyzing Computing Systems
Capability-Based Measures	**Performability (Discrete Performance Variable)**	$\{0, 1, \ldots, n\}$	$\{0, 1, \ldots, N\}$	$\{0, 1, \ldots, M\}$	capability function γ	Prob $[\gamma(U) = b]$, $b \in \{0, 1, \ldots, M\}$	Meyer, Ballance, Furchtgott, and Wu 1977 [36], Meyer 1978 [38]	Furchtgott 1977 [89]
	Performability (Continuous Performance Variable)	$[0,1]$	$\{0, 1, \ldots, n\}$	$\mathbb{R}$		Prob $[\gamma(U) \leq b]$, $b \in \mathbb{R}$		Meyer 1980 [51]
	Computation Capacity Based Measures (measures derived from reliability, e.g., computation reliability, mean computation before failure, computation threshold, computation availability, capacity treshold)	$[0,1]$	$\{0, 1, \ldots, N\}$	$\mathbb{R}$	Total capacity of the system $\sum_{i \in STRUC} U_i a_i$ where a_i is the computation capacity of state i	Prob["total capacity of the system"$(U) \leq b$], $b \in \mathbb{R}$	Beaudry 1977 [17], [90]	Beaudry 1977 [17], [90]
	Workload Model Performance Measures (expected system throughput, expected number of transitions lost, through availability, lost throughput)	$[0,1]$	$\{0, 1, \ldots, N\}$	$\mathbb{R}$	total reward $\sum_{i \in STRUC} U_i r_i$ where r_i is a reward rate, e.g., C_i, capacity of state i μ_i, throughput of state i	E[total reward]	Gay [91] Gay and Ketelson 1979 [19]	Gay [91] Gay and Ketelson 1979 [19]
Capability-Based Measures	**Expected Reward**	$[0,1]$	$\{0, 1, \ldots, N\}$	$\mathbb{R}$	total reward $\sum_{i \in STRUC} \sum_{j \in STRUC} \left[U_{ij} y_{ij} + N_{ij} b_{ij} \right]$ where y_{ij} is the yield (reward) rate is generated while state i is occupied and the next state is j, bonus reward b_{ij} is obtained when the transition is made. Solution techniques will also allow a discount rate α, though this is not discussed by De Souza [22]	E[total reward]	De Souza 1980 [22]	De Souza 1980 [22]

Table 2.1 – A brief overview of many important evaluation measures and when they were first applied to computing systems

Measure Classification	Measure Name	Model Components: Time Base OBSER	Model Components: Structure Set STRUC	Model Components: Performance Set PERF	System description function (See KEY below) $g: STRUC^{OBSER} \rightarrow PERF$	Probability measure of system performance (See KEY below)	References	First Application to Analyzing Computing Systems
	Probability of the Performance of an Assignment	[0,1]	$\{0,1,\ldots,N\}$	R	assignment $\sum_{i \in STRUC} U_i c_i$ where c_i is the productivity of state i.	Prob [assignment is performed during the total system life]	Cherkesov 1981 [92]	***
	Cost Index (Probability of dynaic failure, mean cost, variance cost, modified cost , mean modified cost, variance modified cost)	[0,1]	$\{0,1,\ldots,N\}$	R	system cost function $S(t) = \sum_{i \in TASKS} C_i(t)$ where TASKS is the set of tasks that the system must perform, and $C_i(t)$ is a cost function associated with the following: task i, the number of times a task is executed, the time required to execute the task, and any "hard deadlines" associated with the task.	probability distribution function of S and E[total system cost | system does not miss any hard deadlines]	Krishna and Shin 1982 [26]	Krishna and Shin 198 [26]
	Performance Related Dependability (life-cycle reliability, mean capacity cumulated before system down)	[0,1]	$\{0,1,\ldots,N\}$	R	capacity cumulated before system down $\sum_{i \in STRUC} U_i K_i$ where k_i is the performance index associated with state i.	E[capacity cumulated]	Arlat and Laprie 1983 [27]	Arlat and Laprie 1983 [27]
	Performance measures	[0,1]	$\{0,1,\ldots,N\}$	R	system performance $a(t)$ (viewed as a scalar valued random process)	Prob [$a(t) \in I_a$], I_a a (measurable) interval; and E [$a(t)$]	Gai and Adams 1983 [28]	Gai and Adams 1983 [28]

Table 2.1 -- A brief overview of many important evaluation measures and when they were first applied to computing systems

KEY: U is a random variable taking values in $\mathbf{STRUC}^{\mathbf{OBSER}}$

U_i = Time in state i

$$= \int_{\mathbf{OBSER}} I_{U(t)}(i)dt$$

where

$$U_{U(t)}(i) = \begin{cases} 1 & \text{if } U(t) = i \\ 0 & \text{otherwise} \end{cases}$$

U_{ij} = time in state i when the next state is j

$$= \int_{\mathbf{OBSER}} I_{U(t)}(i) I_{N_U(t)}(j)dt$$

where

$N_U(t)$ = next state that U enters after time t

$$= U(\tau),$$

$$\tau = \infty \left\{ w \in \mathbf{time} \mid U(w) \neq U(t) \text{ and } w > t \right\}$$

N_{ij} = number of transitions from state i to state j *during* **OBSER**

Table 2.1 – A brief overview of many important evaluation measures and when they were first applied to computing systems

Measure Name	Method of representing (obtaining) the function g or $g^{-1}(C)$, $C \subseteq B$ (See Table 1)				Software Tools	
	General Approach	Specific Technique	Common Assumptions	References	References	Program Name
Classical Performance	Algebraic	Queuing models	System reaches a steady state	Many. For an overview, see [2], [3], [73]	Many, including [75], [76]	•••
		Operational Analysis	System reaches a steady state	Denning and Buzen 1978 [93]	•••	•••
	Approximation	Simulation	•••	Many. For an overview, see [2], [3], [73]	Many	Many, e.g., SCERT, CASE, GPSS, SIMSCRIPT, SIMULA SIMPL/1
		Measurement	•••		•••	•••
Classical Reliability	Algebraic	Truth Tables, etc.	•••	•••	•••	
		Networks	Coherent structure	•••	•••	•••
		Fault-trees (event-trees)	Coherent structure	Developed by safety engineers at Bell Telephone Laboratories and The Boeing Company. See Haasl 1965 [94]	Lapp and Powers 1977 [95]	FTS
					Apostolakis, Salem, and Wu 1978 [96]	CAT
			Phasing and coherent structure	Esary and Ziehms 1975 [97]	•••	•••
			Non-coherent structure	Kumamoto and Henley 1978 [98]	•••	•••
		GO charts	Coherent structure	Gateley and Williams 1978 [99]	Gateley and Williams 1978 [99]	GO
"Cannibalization" Reliability	Algebraic	Truth Table	Coherent structure	Hirsch, Meisner, and Boll 1968 [82]	•••	•••
Reliability of "Nondichotomic" structures	Algebraic/ Approximation	Functional description of the structure function ϕ	Coherent structure	Postelnicu 1970 [83]	•••	•••
Reliability of Multistate Systems	Algebraic	Fault Trees (event trees)	Coherent structure	Caldarola 1978 [100]	•••	•••
Pseudo-reliability	Approximation	Measurement/Conjecture	•••	Tillman, Lie, and Hwang 1978 [85]	•••	•••
Measures oriented towards degradable systems	Algebraic	Delineation of failure states	•••	Losq 1977 [16]	•••	•••

Table 2.2 – An outline of the various techniques that have been proposed for relating the system strucute to the evaluation measure

Measure Name	Method of representing (obtaining) the function g or $g^{-1}(C)$, $C \subseteq B$ (See Table 1)				Software Tools	
	General Approach	Specific Technique	Common Assumptions	References	References	Program Nam
Operational Effeciency	Algebraic	Calculation of the work-powers ρ_i	•••	Troy 1977 [15]	•••	•••
Job-Related Reliability	Algebraic	Enumeration of the sets $S(J_i)$	Every $S(J_i)$ has a minimal element	Mine and Hatayama 1979 [18]	•••	•••
User-Oriented Reliability	Algebraic	The structure function ϕ is the identity function No structural description required	•••	Castillo and Siewiorek 1980 [21]	•••	•••
Dependability	Algebraic / Approximation	Delineation of the system states associated with each dependability level	•••	Laprie and Medhaffer-Kanound 1980 [86]	•••	•••
An expected performance measure	Algebraic	delineation of the performances P_i	•••	Chou and Abraham 1980 [20]	•••	•••
Workload Dependent Reliability Measures	Algebraic	The structure function ϕ is the identity function No structural description required	•••	Castillo and Siewiorek 1981 [23]	•••	•••
Performance Reliability	Algebraic	Delineation of the normalized performance of states i, η_i	•••	Huslende 1981 [24]	•••	•••
Computation-Based Reliability	Algebraic	Description of the tolerance relation σ	None explicitly given, example required coherent structure and finite utilization	Meyer 1976 [12]	•••	•••
Reliability	Algebraic	Description of the structure function ϕ		Coherent structure	Borgerson and Freitas 1975 [13]	•••
Phase Mission Reliability	Algebraic	Description of the structure function ϕ for each phase T_i $\#T_i$ Coherent structure function	Winokur and Goldstein 1969 [88]	•••	•••	

Table 2.2 – An outline of the various techniques that have been proposed for relating the system strucute to the evaluation measure

Measure Name	Method of representing (obtaining) the function g or $g^{-1}(C)$, $C \subseteq B$ (See Table 1)				Software Tools	
	General Approach	Specific Technique	Common Assumptions	References	References	Program Nam
Performability (Discrete Performance Variable)	Algebraic	Trajectory Sets	Phasing	Furchtgott 1977 [89]	Furchtgott 1979 [41], [46], Furchtgott 1981 [101]	METAPHOR
Performability (Continuous Performance Variable)	Algebraic	Graph of the inverse capability function γ^{-1}	Decaying Process with three states or less	Meyer 1981 [56]	•••	•••
		Functional description of the inverse capability function γ^{-1}	Decaying Process with finite number of states	Furchtgott 1981 [102]	Furchtgott 1983 [63]	METAPHOR
Computation Capacity Based Measures	Algebraic	Delineation of the computation capacity a_i of each state i	None explicitly given, example was monotonic, but this does not seem to be a restriction	Beaudry 1977 [17], [90]	•••	•••
Workload Model Performance Measures	Algebraic	Delineation of the capacities C_i	•••	Gay and Ketelson 1979 [19]	•••	•••
Expected Reward	Algebraic	Delineation of the yield rates v_{ij}, bonuses b_{ij}, and discount rate α	•••	De Souza 1980 [22]	•••	•••
Probability of the Performance of an Assignment	Algebraic	Laplace transform (two dimensional Laplace-Carson transform)	•••	Cherkesov 1981 [92]	•••	•••
Cost Index	Algebraic	Description of the cost functions C_i and any hard deadlines associated with each task i	•••	Krishna and Shin 1982 [26]	•••	•••
Performance Related Dependability	Algebraic/ Approximation	Description of the cost functions C_i and any hard deadlines associated with each task i	•••	Ariat and Laprie 1983 [27]	•••	•••
Performance Measures	Algebraic	Description of the performance measure $a(t)$	•••	Gai and Adams 1983 [28]	•••	•••

KEY:

••• = Either not applicable or the author is not aware of any suitable entries

Table 2.2 -- An outline of the various techniques that have been proposed for relating the system strucute to the evaluation measure

Measure Name	Method of calculating the probability measure of system performance, i.e., $\text{Prob}[g^{-1}(C)],\ C \subseteq B$ (See Tables 1 and 2)				Software Tools	
	General Approach	Specific Technique	Common Assumptions	References	References	Program Nam
Classical Performance	•••	•••	•••	•••	•••	•••
Classical Reliability	Algebraic	Direct integration of density functions over the time T	s-i, t-i	Many For some early formal work, see von Neumann 1956 [77], Moore and Shannon 1956 [78], and Birnbaum, Esary, Saunders 1961 [79] For an overview, see Barlow and Proschan 1975 [7]	•••	•••
		Truth Table (the sum of the probabilities of individual entries in the table of ϕ)	s-i, t-i	See, e.g., Woodcock 1968 [103]	Woodcock 1968 [103]	NOTED
		Tie Sets (the sum of the probabilities of each path set)	s-i, t-i	Kim, Case, and Ghare 1972 [104]	•••	•••
		Symbolic Manipulation	s-i, t-i	Parker and McCluskey 1975 [105]	•••	•••
		Graph Theoetical	s-i, t-i	Satyanarayana and Prabhakar 1978 [106]	•••	•••
		Synthesis	s-i, t-i	Woodcock 1968 [103]	•••	•••
	Approximation	Inclusion-Exclusion	s-i, t-i	See Barlow and Proschan 1975 [7]	•••	•••
		Monte-Carlo	•••	Hammersly and Handscomb 1964 [107]	Kongso 1972 [110]	RELY4
					Locks 1975 [111]	SPARCS
					WASH-1400 1975 [112]	SAMPLE
				See Fussell and Arendt [108] or McCormick [109] for descriptions of many of the software tools at right.	Matthews 1977 [113]	MOCARS
					Erdmann, et al. 1978 [114]	SPASM

Table 2.3 -- Techniques proposed for describing the stochastic nature of a system

Measure Name	Method of representing (obtaining) the function f or $f^{-1}(C)$, $C \subseteq B$ (See Table 1)				Software Tools	
	General Approach	Specific Technique	Common Assumptions	References	References	Program Name.
Classical Reliability	Algebraic	Cut Sets (1 - sum of the probabilities of each cut set)	s-i, t-j	Jensen and Bellmore 1969 [115] See Fussell and Arendt [108] or McCormick [109] for descriptions of many of the software tools at right.	Vesely and Narum 1970 [116]	PREP and KITT
					Blazek, et al. 1971 [117]	TASRA
					Semanderes 1971 [118]	ELRAFT
					Fussell, Henry, and Marshall 1974 [119]	MOCUS
					The Boeing Company [120]	ARMM
					Pande, Spector, and Chatterjee 1975 [121]	PREP and MICSUP
					Van Slyke and Griffing 1975 [122]	ALLCUTS
					Bjurman, et al. 1976 [123]	CARSRA
					Burdick, Marshall, and Wilson 1976 [124]	COMCAN
					Leverenze and Kirch 1976 [125]	WAM-BAM
					Platz and Olsen 1976 [126]	FAUNET
					Cate and Fussell 1977 [127]	BACFIRE
					Fussell, Rasmuson, and Wagner 1977 [128]	SUPERPOCUS
					Lambert and Gilman 1977 [129]	IMPORTANCE
					Olmos and Wolf 1977 [130]	PL-MOD
					Pelto and Purcell 1977 [131]	MFAULT
					Vesely and Goldberg 1977 [132]	FRANTIC
					Wheeler, et al. 1977 [133]	FAULTRAN
					Erdmann, Leverenz, and Kirch 1978 [134]	WAM-CUT
					Gateley and Williams 1978 [135]	GO
					Rasmuson, et al. 1978 [136]	COMCAN II
					Rasmuson and Marshall 1978 [137]	FATRAM
					Worrell and Stack 1978 [138]	SETS

Table 2.3 – Techniques proposed for describing the stochastic nature of a system

Measure Name	Method of representing (obtaining) the function f or $f^{-1}(C)$, $C \subseteq B$ (See Table 1)				Software Tools	
	General Approach	Specific Technique	Common Assumptions	References	References	Program Name
Classical Reliability	Algebraic	Integral-differential equations	s-i, t-i	Many	Roth 1967 [139]	REL
					Bouricius, et al. 1971 [81]	REL 70
					Fleming 1971 [140]	RELCOMP
					Mathur and Avizienis 1970 [141]	CARE
					Rennels and Avizienis 1973 [142]	RMS
					Stiffler, Bryant, and Guccione 1975 [143]	CARE-II
					Stiffler, Bryant, and Guccione 1979 [144]	CARE-III
		Markov models	s-i, t-i	Many	Ng and Avizienis 1978 [145]	ARIES 76
					Makam and Avizienis 1982 [146]	ARIES 81
		hybrid (markov model and simulation)	•••	Trivedi	Trivedi	HARP
		"Cannibalization" Reliability	Algebraic	Truth Table	s-i, t-i	Hirsch, Meisner, and Hol 1988 [82]
Reliability of "Nondichotomic" structures	Algebraic	Direct integration of the density functions over the structure function ϕ	s-i, t-i	Postelnicu 1970 [83]	•••	•••
Reliability of Multistate Systems	Algebraic	Direct probability calculation	s-i, t-i	Caldarola 1978 [100]	•••	•••
Pseudo-reliability	Algebraic	Direct probability calculation	s-i, t-i	Tillman, Lie, and Hwang 1976 [85]	•••	•••

Table 2.3 -- Techniques proposed for describing the stochastic nature of a system

Measure Name	Method of representing (obtaining) the function f or $f^{-1}(C)$, $C \subseteq B$ (See Table 1)				Software Tools	
	General Approach	Specific Technique	Common Assumptions	References	References	Program Name
Measures oriented towards degradable systems	Algebraic	Steady-state solution of a homogeneous, time-invariant Markov process	s-i, t-i	Losq 1977 [16]	•••	•••
Operational Efficiency	Algebraic	Direct integration of density functions over specified intervals of time	s-i, t-i	Troy 1977 [15]	•••	•••
Job-Related Reliability	Algebraic	Steady-state solution of a homogeneous, time-invariant Markov process	s-i, t-i	Mine and Hatayama 1979 [18]	•••	•••
User-Oriented Reliability	Algebraic	Direct integration of density functions over specified intervals of time	s-i	Castillo and Siewiorek 1980 [21]	•••	•••
Dependability	Algebraic	Markov models/ method of stages	s-i	Laprie and Medhaffer-Kanound 1980 [86]	Costes, et al 1981 [147]	SURF
An expected performance measure	Algebraic	Direct integration of density functions over specified intervals of time	s-i, t-i	Chou and Abraham 1980 [20]	•••	•••
Workload Dependent Reliability Measures	Algebraic	Direct integration of density functions over specified intervals of time	s-i	Castillo and Siewiorek 1981 [23]	•••	•••
Performance Reliability	Algebraic	Direct integration of density functions over specified intervals of time	s-i, t-i	Huslende 1981 [24]	•••	•••
Phased Mission Analysis	Algebraic	Direct probability calculation	s-i, t-i	Winokur and Goldstein 1960 [88]	•••	•••
Computation-Based Reliability	Algebraic	Direct integration of density functions over specified intervals of time	s-i, t-i	Meyer 1976 [12]	•••	•••
Reliabilty	Algebraic	Direct probabilty calculation	s-i, t-i	Borgerson and Freitas 1975 [13]	•••	•••

Table 2.3 -- Techniques proposed for describing the stochastic nature of a system

Measure Name	Method of representing (obtaining) the function f or $f^{-1}(C)$, $C \subset B$ (See Table 1)				Software Tools	
	General Approach	Specific Technique	Common Assumptions	References	References	Program Name
Performability (Discrete Performance Variable)	Algebraic	Trajectory Sets	s-i, t-i	Furchtgott 1977 [89]	Furchtgott 1979 [41], [46], Furchtgott 1981 [101]	METAPHOR
Performability (Continuous Performance Variable)	Algebraic	Integration of density function over the graph of the inverse capability function γ^{-1}	s-i	Meyer 1981 [58]	•••	•••
		Integration of density function over the Functional description of the inverse capability function γ^{-1}	•••	Furchtgott 1981 [102]	•••	•••
Computation Capacity Based Measures	Algebraic	Direct integration of density functions over specified intervals of time	s-i, t-i	Beaudry 1977 [17], [90]	•••	•••
Workload Model Performance Measures	Algebraic	Direct integration of density functions over specified intervals of time	s-i, t-i	Gay and Ketelson 1979 [19]	•••	•••
Expected Reward	Algebraic	Direct integration of density functions over specified intervals of time	s-i, t-i	De Souza 1980 [22]	•••	•••
Probability of the Performance of an Assignment	Algebraic	Inversion of a Laplace transform or evaluation of moments from a Laplace transform	s-i, t-i	Cherkesov 1981 [92]	•••	•••
Cost Index	Algebraic	Direct integration of density functions over specified intervals of time	s-i, t-i	Krishna and Shin 1982 [26]	•••	•••
Perfromance Related Dependability	Algebraic	Direct integration of density functions over specified intervals of time	s-i, t-i	Arlat and Laprie 1983 [27]	Costes, et al. 1981 [147]	SURF
Performance Measures	Algebraic	Direct probability calculation	s-i, t-i	Gai and Adams 1983 [28]	•••	•••

KEY:

••• = Either not applicable or the author is not aware of any suitable entries

s-i = failures of components are statistically independent

t-i = time-invariant statistical parameters

Table 2.3 -- Techniques proposed for describing the stochastic nature of a system

2.4. A Taxonomy of Measures and Models of Degradable Systems (Table 1)

2.4.1. Introduction

To evaluate degradable systems, investigators have been examining ways of appropriately combining performance and reliability issues within a single measure. Many measures discussed in the evaluation literature can be employed to describe degradable systems. However, because of the deficiencies of most of the general purpose measures, many researchers have suggested a range of special-purpose measures expressly to handle degradable systems. Examination of these measures reveals a distinctive organization common to all the measures. This section proposes a taxonomy of the measures and models used to analyze degradable systems. In addition to being useful for describing previous work in the area of degradable system evaluation, the taxonomy also provides a means of placing the work of this thesis in perspective with related research.

Measures and models are innately interrelated, and therefore separating the measure from the model is often difficult: to analyze a system using a particular measure, the analyst must employ a model that reflects the given measure. Typically, there is at least one class of models that embodies each measure. The usual nomenclature is to call the models by the same name as the measure, e.g., models reflecting the measure "reliability"are "reliability models." Since the models place real limitations on the applicability of the measures, the classification of measures propounded in this section are based on the models associated with the measures rather than on the most general possible definition of the measure. (Indeed, if each measure were defined to extreme generality, we would obtain a single, universal, but difficult to employ measure.) Tradeoffs exist for all the measures discussed. Typically, the qualities of

a) ease of interpretating, modeling, and solving, and
b) generality

are mutually conflicting, i.e., a measure that is advantageous regarding one of the above qualities may be poor with respect to the other.

The headings of Table 2.1 summarize the material treated in this section.

2.4.2. The Classifications (Column 1)

To date, the measures proposed to describe degradable systems can be classified according to four categories:

1) Classical performance measures
2) Classical reliability measures
3) Structure-based measures, and
4) Performance-oriented measures.

Column 1 of Table 2.1 lists these four categories.[2] For notational consistency, all countable sets in Table 2.1 have enumerations beginning at 0, while all real line segments are denoted by [0,1]. Also, distinction is made between such finite sets as $\{0,1\}$, $\{0,1\}^N$, $\{0,1,\ldots,n\}$, and $\{0,1,\ldots,n\}^N$. Although such sets are equivalent in the sense of being finite, they are different in the kinds of techniques they allow for modeling. The above categories are based on characteristics of the models that are used to obtain the measures. These characteristics will be amplified in Section 2.4.3 [Components of the Models (Columns 3-5)].

Briefly, Categories 1 and 2 consist of the ***classical performance measures*** and ***classical reliability measures***. These measures will not be discussed in depth within this dissertation. The amount of literature associated with these categories is huge (see, e.g., [2], [3], [73]), and, as argued in Section 2.1 [Introduction], these measures are generally not sufficient for describing degradable systems. They are included in Tables 2.1-2.3 for completeness and to provide the reader with a familiar basis for calibrating the tables.

---

[2]The second column lists specific measures. The next five columns (columns 3-7) deal with the the corresponding model, the eighth column provides references, and the ninth column notes information regarding the first application of the measure to analyzing computing systems. Columns 3-5 will be explored in more detail in Section 2.4.3 [Components of the Models (Columns 3-5)].

Category 3 measures are extensions of classical reliability measures. These measures are characterized by their corresponding models being based on a "structure function" (to be defined in Section 2.7 [Structure-Based Models]); hence measures in this category are called ***structure-based measures***. Finally, the fourth class contains those measures that share characteristics with classical performance measures. These measures are called ***performance-oriented measures***. Models used to describe these measures are typically multi-state markov reward models.

2.4.3. Components of the Models (Columns 3-5)

As mentioned above, the classification of measures is based largely on the characteristics of models utilized to describe the measures. This section discusses three major components of the models along with some relationships between the components. Based on these elements, the categories of Section 2.4.2 [The Classifications (Column 1)] can be more formally defined.

The components that appear in most of the models are the following sets:

1) the observation set *OBSER*
2) the structure set STRUC
3) the performance set PERF.

Call these sets the ***system description sets***. There are no intrinsic restrictions on the sets STRUC and PERF. *OBSER* has the restriction that it be a totally ordered set. Informally, the time base *OBSER* is a minimal set of observation points, i.e., times at which the state of the system must be sampled to determine the system's performance. For example, if the set *OBSER* is a singleton set, then the system performance can be determined from a single observation, that is, a "snapshot." On the other hand, if the size of *OBSER* is larger, then the system performance at any given time *cannot* be determined just by studying the system at that time; rather, the system performance can be determined only by examining the behavior of the system in the context of the system at all the times in *OBSER*.

The structure set STRUC is a set of descriptors of the physical realization of the system (for the purposes of this nomenclature, *STRUCT* includes the environment). Usually, STRUC is the set of ***states*** of the system. The performance set PERF is a set of descriptors of the behavior of the system.

2.4.4. The System Description Function (Columns 6-7)

The fundamental relation between *OBSER*, STRUC, and PERF is a function[3]

$$g:\text{STRUC}^{OBSER}\rightarrow\text{PERF} \tag{2.1}$$

Call g the ***system description function***. There are no inherent constraints on g other than that g must be a well-defined function. The interpretation of Eq. 2.1 is as follows: $u \in \text{STRUC}^{OBSER}$ is a function $u:OBSER\rightarrow\text{STRUC}$ that defines the structure $q \in \text{STRUC}$ of the system at observation time $t \in OBSER$, i.e., $u(t) = q$. The function u is called a ***trajectory***. The map g then assigns to each function u a value in the performance set. That is, $g(u) \in \text{PERF}$ is the performance associated with trajectory u.

Finally, there is a probability measure of system performance. Most of these measures are either probability distributions or expected values (means) of PERF (i.e., the system description function g interpreted as a random variable). The determination of this quantity is the goal when evaluating degradable systems.

Note that the probability measure can be dependent on time in a manner unrelated to the observation set *OBSER*. *OBSER* merely indicates the number of sample points required to determine the system performance, *not* the times at which those sample points occur. For instance, suppose $OBSER = \{0\}$. Then for $q \in \text{STRUC}$, $B \subseteq \text{PERF}$, $\text{Prob}[g(q) \in B]$ can

---

[3]The notation A^B denotes the *direct product* of the set A by set B, i.e., A^B is the set of all functions from B to A:

$$A^B = \{u \mid u:B\rightarrow A\}\ .$$

Sometimes the notation $B\rightarrow A$ is used in place of A^B.

depend on the specific time at which the observation takes place.

Several categories of system description functions g can now be defined. Let g be a system description function such that $|\ OBSER\ | = 1$, i.e.,

$$g\text{:STRUC}\rightarrow\text{PERF} . \tag{2.2}$$

Then g is a ***structure-based system description function.*** That is, if the system performance can be determined by a single sample of the system's structure, then the system description depends only on the system's current structure, and hence the name "structure-based."

As a simple example of a structure-based system description function, consider the following: Let $OBSER = \{0\}$, $\text{STRUC} = \{0,1,2,3\}$, $\text{PERF} = \{0,1\}$, and

$$g(q) = \begin{cases} 1 \text{ if } q \in \{1,2,3\} \\ 0 \text{ if } q = 0 \end{cases} \tag{2.3}$$

One interpretation for this example is that if $g(q) = 1$, then the system is operational. We can determine if the system is operational at a given time by simply observing what the structure (state) of the system is at that time.

If $|\ OBSER\ | > 1$, then g is ***non-structure based.*** Let g be a structure-based system description function such that $|\ \text{PERF}\ | = 2$ and $|\ \text{STRUC}\ | = 2^N$, $N < \infty$. Then g is a ***structure function.*** A structure function is usually denoted ϕ.

Let g be a system description function such that $|\ OBSER\ | > 1$. Then g is a ***capability-based system description function.*** With a system description function, the system performance requires some knowledge of the system's history. For instance, let $OBSER = [0, 1]$, $\text{STRUC} = \{0, 1, \ldots, n\}$, $\text{PERF} = \mathbb{R}$ (the real numbers), and

$$g(u(\ \cdot\ )) = \int_0^1 u(t)dt . \tag{2.4}$$

$u(\ \cdot\ )$ is a trajectory of the system, $u(t)$ is the system structure at time t, and $g(u(\ \cdot\ ))$ is the

accumulated value of the trajectory. $u(t)$ could be called the "reward rate" of the system at time t and $g(u(\cdot))$ would then be the reward accrued by the system during $[0, 1]$.

Finally, column 7 presents the probability measure of the sytem performance. Usually, this measure is either the probability distribution function of the performance or else the first moment of the performance. The remaining two columns present references to early work in the measure, as well as early work on applications to computing systems.

2.5. Description of the Structure of Degradable Systems (Table 2)

2.5.1. Introduction

As discussed in Section 2.4 [A Taxonomy of Measures and Models of Degradable Systems (Table 1)], most system measures that can be employed to evaluate degradable systems induce a system description function

$$g:\mathrm{STRUC}^{OBSER}\rightarrow\mathrm{PERF} \tag{2.1}$$

Two fundamental questions arise: For a given measure g,

1) How can one represent g?
2) How can one obtain g?

Table 2.2 encapsulates this information for the measures of Table 2.1. Also included are references to software tools useful for developing the system description function.

2.5.2. Representation and Acquisition of the Capability Function (Columns 2-5)

There are three general approaches for answering the above two questions: a direct measurement approach, an approximation approach, and an algebraic approach. The most accurate approach is the direct measurement. This technique is rarely used, however, because it is expensive and often infeasible. Most of the research to date has concerned algebraic and

approximation approaches. Approximation is useful if the algebraic underpinnings are well understood so that the effect of the approximation is known.

Column 3 discusses the specific technique used to determine the system description function, while column 4 states common assumptions made in order to apply that technique. Many of the techniques (especially those requiring the delineation of functions or reward rates) are general, allowing the technique to be applied to a wide class of problems. The corresponding reference for the method of obtaining the system description function is given in column 5.

2.6. Determining the Probability Measure of System Performance (Table 3)

2.6.1. Introduction

Once the the system description function has been obtained, some probability measure of that function must be calculated. Table 2.3 summarizes many of the methods of calculating the probability measure for the entries of Tables 2.1 and 2.3. In addition, some software tools for calculating the probability measure are given.

2.6.2. Calculating the Measure (Columns 2-5)

Columns 2-5 of Table 2.3 are identical to those of Table 2.2. The entries of column 5 refer to common assumptions regarding the parameters of the models. "t-i" indicates that the various components of the model generally have time invariant statistics, while "s-i" means that the components are usually statistically independent.

2.7. Structure-Based Models

Classical reliability analysis techniques (e.g., Birnbaum, Esary, and Saunders [79]) stress the determination of system failure or system success. The state of the system is represented

by a vector $\mathbf{X} = (X_1, \cdots, X_n) \in \{0,1\}^n$, where n is the number of components in the system and

$$X_i = \begin{cases} 1 & \text{if the } i^{th} \text{ component is functioning} \\ 0 & \text{if the } i^{th} \text{ component is failed .} \end{cases} \tag{2.5}$$

A ***structure function*** ϕ:$\{0,1\}^n \rightarrow \{0,1\}$ relates the system state to system success and system failure. Usually, focus is restricted to ***coherent*** (i.e., monotonic and nontrivial) ***structure functions***. Later work (e.g., Haasl [94] and Fussell, Powers, and Bennetts [148]) characterize coherent structure functions as fault trees or event trees. The probability of system success is called "reliability." Associating with each component a probability of success or failure, techniques for determining the reliability of a system have been developed. For complex systems, methods (based on minimal cut sets and minimal path sets) of computing bounds on the reliability have been found (Esary and Proschan) [149].

Other work has generalized the concept of structure functions to allow for a wider range of performance values. Hirsch, Meisner, and Boll [82] have studied degradable systems and have generalized the concept of structure function to allow any finite number of performance rates, called "levels of performance." Components of the system are either successful or failed; thus, the structure function is ϕ:$\{0,1\}^n \rightarrow \{0,1,\cdots,M\}$. The authors define a "cannibalization" to be a state transition function describing how the system is reconfigured when a fault occurs. Combining these two functions, Hirsch et al. obtain a "cannibalized structure function," depicting the system's performance rate for a given system state. Introducing a random process describing the failure of system components, the determination of the probability distribution of the system's performance rate at a given time is investigated. By assuming that the cannibalized structure function is coherent, the distribution provides a measure of the worst performance rate experienced by the system.

Postelnicu [83] has considered systems in which components, as well as the system itself, take on values of performance in the continuous range [0,1]. A generalized structure function is introduced: ϕ:$[0,1]^n \rightarrow [0,1]$. Bounds for the distribution of the system performance rate

are derived, as are limit distributions for iterated combinations.

Tillman, Lie, and Hwang [85] have proposed a measure called "pseudo-reliability" that is the system reliability weighted by the relative-performance. Pseudo-reliability provides a measure of the performance of a state along with the probability of achieving that state.

Another generalization has been phased mission analysis, where the reliability of the system is based on several observations (called ***phases***). In most of these studies, the system must perform at some minimum level during each phase in order to be successful. Winokur and Goldstein [88], Bricker [150], Esary and Ziehms [97], and Pedar and Sarma [25], [151], among others, have investigated phased mission reliability.

Although the theory for the class of reliability-based evaluation discussed above has been thoroughly developed, structure-based formulations are unsuitable characterizations of degradable systems. If the user is interested in the "worst case" behavior of the system and if the system has coherent properties, then analysis of the structure function is enough. However, as demonstrated in [29], if system quality is based on performance criteria rather than structural criteria, structure functions cannot reflect the system's ability to perform, because the structure function yields a "snapshot" of the system performance (i.e., the performance rate at the point of the evaluation), rather than the total system performance for the mission; the structure function does *not* yield the total system performance for the mission.

2.8. Other Reliability Oriented Models

Recognizing the problems inherent in simple extensions of classical reliability, researchers began exploring other methods of measuring a degradable system's ability to perform. Many of these measures are based on Markov processes, and most are reliability-oriented.

Meyer [12] has introduced the concept of "computation-based" reliability analysis, in which underlying criteria for success are based not on the system's internal state, but on the tasks the system must accomplish in the use environment. A formal "tolerance relation" on the set of all computations is used to specify "success," and the probability of all

computations being within tolerance is calculated. Thus, the system's structure can degrade without affecting the measure of the system's ability to perform.

Borgerson and Freitas [13] first investigated the specific needs of degradable systems by determining the probability of each of several performance levels of a degradable system over a finite interval. Other authors have used Markov models to evaluate degradable systems; for instance, Laprie [152], Ng and Avizienis [153], and Costes, Landrault, and Laprie, [154].

In terms of a computer system's structural state, Troy [15] has formulated the concept of "workpower" to quantify the amount of available processing capability, and has investigated the expected workpower ("operational efficiency") as a measure of the system's performance, provided the system does not fail. Thus, though no unified performance/reliability measure is presented, the desirability of considering the performance, as well as the reliability, of degradable systems is recognized.

Partitioning a degradable system into several independent "resources" and modeling each such resource as a Markov process, Losq [16] has examined performance/reliability measures oriented towards system availability, e.g., "availability," "mean time between crashes," "average processing power," and "proportion of time spent in degraded mode." Availability is defined in terms of a threshold on the structure (i.e., a minimum number of fault-free elements), and the optimization of availability is also investigated.

Although the models discussed in this section are somewhat more performance-oriented than the structure-based formulations, the measures are still primarily reliability-oriented. The study of degradable systems at this stage is more concerned with a system's survivability, regardless of its performance, than with its total contribution to the user.

2.9. Performance Oriented Models

As the methodology for evaluating degradable systems progressed, researchers began investigating ways of specifically incorporating performance attributes into the evaluation. These methods frequently use special Markov, semi-Markov, and Markov reward processes.

Beaudry [17] has developed an analysis method in which the amount of achieved computation defines success. By considering how the computation capacity of the system changes with time, one can build a transformed Markov model in which state transition rates are based on the expected amount of computation achieved in a state, as opposed to the expected amount of time sojourned in a state. From the transformed Markov model, one can determine quantities such as "mean computation before failure," "computation reliability," and "computation availability."

Mine and Hatayama [18] have considered systems in which various classes of jobs require the use of certain components within the system. Utilizing Markov models of both the system configuration and the job process, the "job-related reliability" (i.e., the probability of executing a job from a given job class) is examined.

Chou and Abraham [20] have used a discrete state, continuous time semi-Markov process to model a general n-processor shared-resource system. The processors are repairable and a steady-state performance for the system is calculated as a function of n, the number of processors. Assuming that the cost is proportional to n, performance-to-cost ratios are calculated, and the optimal number of processors is determined.

Recently, several researchers have focused on Markov reward processes (see Howard [155], for example) to model the total performance of degradable systems. Generally, the reward structure attaches a value to the sojourn time in each state, along with a bonus for entering each state. The reward rates are associated with performance; hence the total reward is a measure of the system's total performance. Techniques are known for obtaining certain results from such models. In particular, expected values are typically obtained.

Gay and Ketelson [19] have formulated Markov reward process-based "capacity" and "workload" models reflecting, respectively, the system's capability to do work, and the system's ability to satisfy the demands placed on it by the environment. The capacity model is similar to Beaudry's model, as are the derived effectiveness measures. Combining a capacity model (representing the reliability of the structure) with a birth-death model (represent-

ing the arrival and processing of jobs) yields a workload model, from which can be derived measures such as "expected system throughput" and "throughput availability" (or "relative throughput").

Employing a reward process model, Castillo and Siewiorek [21] have studied the measurement of the turnaround time of a computer job in a computing system having variable performance due to system workload and having fatal and nonfatal faults. Measures of "apparent capacity" and "expected elapsed time" (time required to execute a given program correctly) are obtained, reflecting the viewpoint of a user submitting jobs, as opposed to the outlook of the system's owner. Although the technique is not intended for degradable systems, the approach does account for reliability and variations in performance.

De Souza [22] has used a Markov reward process model to determine the expected operating costs (profits) of a computing system. In addition, a "cost reduction" measure is introduced to quantify the benefit of fault tolerance, and sensitivity analysis of the cost reduction is examined. Cherkesov [92] has introduced a solution for the probability of accumulating a minimum amount of reward during a finite interval.

Krishna and Shin [26] examine the probability distribution function, and more specifically the expected value, of the cost associated with a computer control system. Arlat and Laprie [27] consider a performance related reliability measure: mean capacity cumulated before system down. Gai and Adams [28] have discussed tradeoffs between "optimal" and "robust" response times.

A general modeling framework for modeling performance and reliability is discussed by Meyer [29]. At its center is a measure called "performability," which is the probability measure induced by a random variable called "system performance." The concept of a "capability function" relating low-level system behavior and structure to high-level interpretations of performance is also introduced. Because of the relevance of performability to the research of this thesis, the first few sections of the following chapter summarize the basic points of performability. Precise definitions or a theoretical construction should not be expected. For technical

details, refer to Meyer [29] or Wu [59], where discussion aims at providing the reader with an initial orientation to the problem area.

CHAPTER 3

FUNDAMENTAL CONCEPTS AND RESULTS

3.1. Introduction

The purpose of this chapter is to present the basic concepts of performability and the results of this dissertation. Underlying this dissertation is the concept of performability: how to obtain it and how to apply it. Section 3.2 [Motivation] examines the motivations in more detail. Section 3.3 [An Informal Introduction to Performability] summarizes the notion of performability in a non-technical manner. Some notes at the end of that section provide more detail for the interested reader. The development of performabilty given in Section 3.3 is somewhat different than given in previous expositions in that the treatment of Section 3.3 is more "constructive" in nature. Also presented in Section 3.3 [An Informal Introduction to Performability] are a procedure for constructing performability models and an example of a performability analysis demonstrating some of the advantages over moments which performability offers. Finally, in Sections 3.4 [Models Having Finite Performance Variables] and 3.5 [Models Having Continuous Performance Variables], the main results of the dissertation for finite and continuous accomplishment sets are reviewed.

3.2. Motivation

When a complex computing system is analyzed, the analyst must consider several perspectives, or viewpoints, of the system. One perspective is a high-level, user-oriented[1] view of system behavior during the period of utilization. This is the viewpoint that most users ultimately consider to be the most important. This perspective addresses questions such as "how does this system benefit me?" and "what does the system do for me?" Other system perspectives, lower-level viewpoints, are concerned with *how* the system actually achieves its high-level behavior. Of course, lower-level viewpoints are of great concern to the system designer and analyst, since only by studying the system in detail can it be accurately designed and analyzed.

The high-level perspective should, if sufficiently high, have the advantages of being easily described and interpreted. The lower-level perspectives, if they are low enough, while not so easily described and interpretated, may give the system analyst or designer insight into the effects of the system's fundamental components and configurations. The lower-level perspectives may have additional characteristics not readily available at the high level, e.g., the existence of simple probabilistic descriptions of the system.

Since both high-level and lower-level views of the computing system are, after all, views of the *same* system, there must be some relationship between them. In particular, we assume that if enough information regarding low-level behavior is known, then high-level system behavior must *deterministically* correspond to specific low-level system behavior. If the correspondence is known, then a probabilistic characterization of high-level behavior can be based on a probabilistic characterization of low-level behavior. Furthermore, reasonable design decisions at the lower levels can be made based on knowledge of how changes in low-level behavior will affect high-level behavior.

---

[1] By *user*, we mean the person or entity that actually benefits from the employment of the system and for whom the system is being evaluated. For example, if the system is a computer to be used aboard a commercial aircraft, then the "user" is the air line company that owns the aircraft.

On the basis of the observations presented above, this dissertation can be said to have two primary goals. (See also Section 1.1 [Problem Statement].) The goals are: for specific classes of interesting and important computing systems (called ***degradable computing systems***, see Section 2.2 [Degradable Systems]), we desire

1) to develop methodologies for relating high-level and low-level behaviors, and

2) to apply these methodologies to the examination of design tradeoffs for degradable computing systems.

3.3. An Informal Introduction to Performability

3.3.1. Introduction

This section provides a brief summary of performabilty. The intent is to present only a short, non-technical motivation and synopsis. A few additional technical points are delimited by notes at the end of Section 3.3 [An Informal Introduction to Performability]. A formal definition of performabilty has been developed by Meyer and appears in [29]. A second formal definition appears in Wu [59, Chapter 2]. The reader interested in a complete exposition is referred to those two sources. The information in the notes is provided to connect this dissertation to the previous work, particularly [29] and [59].<NOTE 1>[2] However, understanding of the technical points appearing in those notes is not necessary to understand this dissertation. Other authors have also investigated various aspects of performabilty. These authors include Meyer and Ballance [37], Meyer [29], [33], [38], Furchtgott and Meyer [39], Meyer, Furchtgott, and Wu [32], [44], Meyer and Wu [43], [55], Hitt, Bridgman, and Robinson [156], Pedar [151], Pedar and Sarma [25], and Wu [59].

An important component of the two formal developments referred to above is the *initial* assumption of an underlying probability space describing the low-level behavior of the system. The rest of the framework is built bottom-up, based on the low-level probability space. The

[2] See the notes at the end of Section 3.3 [An Informal Introduction to Performability].

development in this chapter is different than the developments of [29] and [59], due to pedagogical considerations. The distinction between the presentations is important, since the one in this section supports either a top-down analysis or an analysis in which the high-level perspective has been made into an equivalent, but lower-level, model. In the development presented here, the introduction of the low-level probability space is deferred as long as possible. Until that time, the analysis is deterministic. Only when knowledge about the stochastic behavior of the low levels is required is the low-level probabilistic description inserted. The analysis then continues bottom-up to meet the deterministic analysis above.

The deferral of the probabilistic model affords the maximum possible deterministic simplification. In particular, the procedure allows the analyst to analyze the system structure and then insert many different probabilistic descriptions, without necessitating re-analysis of the system models.

We emphasize that the top-down approach presented here is equivalent to the bottom-up approaches of [29] and [59].

3.3.2. The Accomplishment Set

Consider first the high-level outlook. The user has a performance-oriented frame of reference, and so a high-level perspective should reflect accomplishment and achievement. Outcomes discernible by the user can usually be described by a single set (either countable or continuous) of values. This set A is called the ***accomplishment set***, or less often, the ***performance set***. An element a of A is called an ***accomplishment level*** (or ***performance level***). For example, in one of the simplest cases, $A = \{$ success, failure $\}$, and from the user's viewpoint, all utilizations of the system can be classified as having either accomplishment level "success" or accomplishment level "failure."

The user, being interested in performance, desires to know how well the system will behave. Often, however, the user cannot know in advance at which accomplishment level his system will perform, since random happenings (e.g., component failures, environmental fac-

tors) occur that may affect the system's performance.<NOTE 2> For instance, in certain applications the weather in which the system is used could influence the system's accomplishment level.

Because the user cannot predict system outcomes with certainty, he must rely on a probabilistic description of performance outcomes. One such probabilistic characterization is called ***performability.***<NOTE 3> A formal development of performability is presented by Meyer in [29], [38], [47]; earlier, less precise discussions of performability are given by Meyer in [34], [35], [157]. As a special case, when the accomplishment set $A = \{$ success, failure $\}$ or any other two-valued set, the probabilistic description is called ***reliability*** (see [7], for example). When the performance variable takes some value $a \in A$ with probability 1 (modeling steady-state mean values) then the description is usually called ***performance.***

3.3.3. The Trajectory Space

A second perspective in analyzing a computing system is a low-level, detailed, structural view of the components comprising the system, along with the behavior of the environment in which the system operates. In contrast to the single set A representing the high-level, performance outlook, the description of this structural perspective is usually much more complicated. In the most general case, the history over the entire utilization of each component and each environmental factor must be specified to characterize fully this perspective.<NOTE 4> Let U be the set of all possible low-level "histories" or outcomes of the system. These histories are represented by functions

$$u: T \rightarrow Q \tag{3.1}$$

from a set T called the ***parameter set*** to a set Q called the ***state space.*** Hence $U = Q^T$. Call U the ***trajectory space*** of the system, and call a specific history $u \in U$ a ***trajectory.***<NOTE 5> The usual interpretation of a trajectory u is that at time $t \in T$, the sys-

tem assumes state $u(t)$.[3]

There may be a stochastic description of the trajectory space U. This description takes the form of a stochastic process

$$X = \{X_t \mid t \in T\} \tag{3.2}$$

where $X_t: T \rightarrow Q$. Each trajectory $u \in U$ is a sample function of the process X. The process X is called the ***base model.***

3.3.4. The Capability Function γ

Every low-level outcome $u \in U$ results in a corresponding high-level performance $a \in A$. There is thus a function

$$\gamma: U \rightarrow A \tag{3.3}$$

relating trajectories to accomplishment levels. $\gamma(u) = a$ signifies that the low-level trajectory u will be interpreted by the user as having accomplishment level a. As an illustration, a particular behavioral occurrence u of the components and environment of a computing system may be interpreted by the user as a "success." In that case, $\gamma(u) =$ success. γ is called the ***capability function***[4] and is introduced by Meyer, Furchtgott, and Wu in [159] (see also [160], where the function is called ψ, and [89]). The capability function is an extension of the concept of a structure function (see Section 2.7 [Structure-Based Models]). The capability function plays an important role in performability analysis. The determination of the capability

---

[3]In the continuous performance variable methodology (Chapter 5 [CONTINUOUS PEFORMANCE VARIABLE (CPV) METHODOLOGY]), T is continuous and u is represented as a function over T. In the discrete performance variable methodology (Chapter 4 [DISCRETE PERFORMANCE VARIABLE (DPV) METHODOLOGY]), T is often finite and so a trajectory can be represented as a vector. In such cases, one often denotes trajectories in bold face type, e.g., **U** and **u**.

[4]The symbol γ was chosen to represent capability because both c (for capability) and γ are the third letters of their respective alphabets. c was not used because in the area of fault-tolerant computing, c is commonly employed to denote the important quantity "coverage"[158].

function and its inverse is, therefore, a significant component of this dissertation.

3.3.5. The Performability Model

A performability model is a two-tuple (X, γ). (See Section 3.3.3 [The Trajectory Space] for a description of X and Section 3.3.4 [The Capability Function γ] for a description of γ.)

3.3.6. Solving for the Performability

To obtain the probabilistic description of the accomplishment set A, the following two-step procedure can be employed.

Procedure 3.2: (Procedure for obtaining the performability of a system)

For each (measurable<NOTE 6>) set $B \subseteq A$ of accomplishment levels,

1) Determine

$$\gamma^{-1}(B) = \{u \mid \gamma(u) \in B\} \tag{3.4}$$

i.e., the set of all trajectories u that result in an accomplishment level in the set B, and

2) Determine the probability of the set of trajectories $\gamma^{-1}(B)$.

The above procedure<NOTE 7> has been delineated in [89] and [30]. It is also discussed in [32] and [29]. In simpler form, it is the procedure employed in classical reliability evaluation [7, chap. 2].

Step 1) addresses goal 1) of the previous section: "to develop methodologies for relating high-level and low-level behaviors." Hence, much of this dissertation deals with methodologies for obtaining $\gamma^{-1}(B)$. Step 2) will be treated only as is necessary for dealing with goal 2): "to apply these techniques to describe design methodologies for degradable computing systems." Briefly, step 2) consists of specifying the base model X (see Eq. 3.2), and then determining

the probabilities of the trajectories $\gamma^{-1}(B)$ (Eq. 3.4).

Two classes of accomplishment sets A are investigated in this dissertation: finite and continuous. Sections 3.4 [Models Having Finite Performance Variables] and 3.5 [Models Having Continuous Performance Variables] present an overview of the results of those investigations.

3.3.7. Constructing Performability Models

Procedure 3.2 of Section 3.3.6 [Solving for the Performability] specifies how to solve a performability model (see Section 3.3.5 [The Performability Model]). Implicit in that procedure is the existence of the model. In practice, however, the model must be constructed before it can be solved. Often, especially when A is continuous, we shall know γ *a priori* (and hence U and A) and X, i.e., we shall know the performability model. In other cases, particularly when A is discrete, we will initially not know either U or even A, and so clearly we cannot possess γ and X. Suppose that A, U, and γ are not initially specified and that A and U are finite; this section presents a procedure for constructing performability models. This constructive procedure has not been explicitly described before.

As discussed in Section 3.3.2 [The Accomplishment Set], the choice of the accomplishment set A is based on the user's viewpoint. The critical question regarding the accomplishment set is: "For a given total system, how are the accomplishment levels A chosen?" The accomplishment levels are selected by analyzing the concerns of the user and how those concerns might be influenced by the object system, which in the case of this dissertation, is a degradable computing system. If, however, accomplishment levels that are *not* influenced by the object system are chosen, then no significant harm is done, since the analysis will eventually discover this lack. Hence, the person choosing the accomplishment set need not be intimately familiar with the total system being analyzed. Indeed, the accomplishment set A can be chosen by someone other than the analyst, e.g., the user could select the set in which he is

interested.

The other major components of the performability model are the trajectory space U and the capability function γ relating U to A. Once the accomplishment set A is chosen, a critical question is: "For a given total system, how are the trajectory space U and the capability function γ chosen?" The answer is not as straightforward as the selection of the accomplishment set. If U and γ are initially not known, two points should be emphasized regarding their determination:

1) Rather than simply choosing a trajectory space, we shall *derive* a trajectory space, based on our knowledge of the total system, and
2) In the process of deriving the trajectory space, we shall *simultaneously* derive the inverse of the capability function, i.e., γ^{-1} (see Eq. 3.4).

The model construction procedure is stated below in general terms:

Procedure 3.3: (Procedure for constructing a performability model)

1) Characterize a "tentative" or "first pass" accomplishment set A' . A' is "tentative" because it may include accomplishment levels which are not possible. We assume that A' does contain all accomplishment levels in which the user is interested, i.e., that $A' \subseteq A$. A' will be refined to arrive at A. Call A' the ***tentative accomplishment set.***
   a) Choose the tentative accomplishment set A' .
   b) Decide which subsets of A' are desired to be measurable.<NOTE 8>
2) Characterize the accomplishment set A, the capability function γ, and the trajectory space U:
   a) Based on A' and the structure of the system, determine a "tentative" or "first pass" inverse capability function $(\gamma')^{-1}$ and a tentative trajectory space U' :
      i) For each $a \in A'$, determine those low-level behaviors that "correspond to" a.

Characterize behaviors as a function

$$u' : \text{DOMAIN}(u') \rightarrow \text{IMAGE}(u') \tag{3.5}$$

Denote this relation $(\gamma')^{-1}$. Call γ' the ***tentative capability function.***

We will not be specific about what is meant by "corresponds to." This is part of the modeler's art. Loosely, u' "corresponds to" a means that an occurrence of low-level behavior characterized by u' will be interpreted by the user as accomplishment a. See Section 3.3.8 [The Model Hierarchy] for a discussion of a hierarchy of models that can make the determination of the relation γ' easier.

γ' is a function from a set of functions $\{u'\}$ to A'. Observe that we make no restrictions on those functions. In particular, the domains of the various functions $\{u'\}$ can be different. In addition, γ' need not be onto, i.e., there can be accomplishment levels $a \in A'$ for which there no u' such that $\gamma'(u') = a$, and so $(\gamma')^{-1}(a) = \phi$. (Note also that no conditions are made requiring γ' to be measurable.)

ii) Set U' to the set of all behaviors that correspond to some accomplishment level $a \in A$, i.e.,

$$U' = \bigcup_{a \in A} \{u' \mid \gamma'(u') = a\}. \tag{3.6}$$

iii) Set A to the range of γ', i.e.,

$$A = \gamma'(U') = \{a \mid \gamma'(u') = a \text{ for some } u' \in U'\}. \tag{3.7}$$

Call U' the ***tentative trajectory space.***

b) Find U and γ^{-1} by augmenting U' and $(\gamma')^{-1}$:

i) Set T to the union of the domains of all the functions in U', i.e.,

$$T = \bigcup_{u' \in U'} \text{DOMAIN}(u'). \tag{3.8}$$

ii) Set Q' to the union of the images of all the functions in U', i.e.,

$$Q' = \bigcup_{u' \in U'} \text{IMAGE}(u'). \tag{3.9}$$

Call Q' the ***tentative state space.***

We wish to extend the functions in U' so that every function has the same domain and range. To do this, we will augment Q' with a special element $\not\in$ that will serve as a "placeholder." If a function $u' \in U'$ is not defined at some time $t \in T$, then $u'(t)$ will be redefined to be $\not\in$. The following two steps carry out this extension.

iii) Augment Q' with $\not\in$ to form Q, i.e.,

$$Q = Q' \cup \{\not\in\}. \tag{3.10}$$

iv) Set U to the set of all functions $u: T \rightarrow Q$, i.e.,

$$U = Q^T. \tag{3.11}$$

v) For each function $u' \in U'$, form a new function EXTEND(u') where

$$\text{EXTEND}(u')(t) = \begin{cases} u'(t) & \text{if } u'(t) \text{ is defined} \\ \not\in & \text{otherwise.} \end{cases} \tag{3.12}$$

Each function EXTEND(u') has domain T and range Q.

Sometimes, the domains of all the functions $u' \in U'$ will already be identical. In such cases, it is not necessary to augment Q' with $\not\in$ and so $Q = Q'$, $U = U'$, and EXTEND is the identity function.

vi) Set $U_{CONS} = \{\text{EXTEND}(u') \mid u' \in U'\}$. U_{CONS} contains all functions $u: T \rightarrow Q$ that are ***consistent*** in the sense that u can be related (via γ') to some accomplishment level.

vii) There may be some behaviors that are ***inconsistent***, i.e., some behaviors may have no corresponding accomplishment levels.[5] These are all the functions $u: T \rightarrow Q$ not in U_{CONS}. Call this set U_{INCONS}:

$$U_{INCONS} = U - U_{CONS} \tag{3.13}$$

viii) To any function $u \in U_{INCONS}$, assign an arbitrary accomplishment level $\gamma''(u)$.<NOTE 9>

ix) Set the capability function $\gamma: U \rightarrow A$ such that

$$\gamma(u) = \begin{cases} \gamma'(u) & \text{if } u \in U_{CONS} \\ \gamma''(u) & \text{if } u \in U_{INCONS} \end{cases} \tag{3.14}$$

Sometimes U_{INCONS} will be empty. In such cases, we do not need to make any arbitrary assignments, and so $\gamma = \gamma'$.

c) Based on the measurable sets of accomplishment levels and the capability function γ, choose the measurable sets of trajectories.<NOTE 10>

---

[5]Note that the analyst is completely free to state which functions are consistent and which are inconsistent. For example, inconsistent behaviors can be those that

1) are logically inconsistent, i.e., that, due to the structure of the system, are impossible regardless of the stochastic description of the the trajectory space (e.g., see Naylor and Meyer [161]), or,

2) if the analyst has prior knowledge of the eventual stochastic description of the trajectory space, are probabilistically "inconsistent," i.e, the probability of all such trajectories is zero.

3.3.8. The Model Hierarchy

3.3.8.1. Definition of a Model Hierarchy

As discussed in Section 3.3.4 [The Capability Function γ], the capability function γ (see Eq. 3.3) relates low-level trajectories to high-level accomplishment levels. The determination of the capability function and its inverse is an important step in the evaluation of a system's performability. [See step 1) of the procedure in Section 3.3 [An Informal Introduction to Performability].] However, directly obtaining γ^{-1} is difficult since

1) We do not initially know the trajectory space U, and
2) The "distance," in terms of how behavior of low-level components affects total system behavior, may be great.

To lessen the "distance," we introduce between U and A additional sets which facilitate determining γ^{-1} by allowing a gradual refinement of the relationship between U and A. Each such set describes a model called a ***level***. The collection of these intermediate models is a ***model hierarchy***.

The concept of applying such a model hierarchy to performability analysis is due to Meyer [34]. An early version of the techniques and methodology described in this section is presented by Furchtgott [89]. The notation is extended and the concept coupled to an underlying probability space in Meyer, Ballance, Furchtgott, and Wu [30]. That extension also appears in Meyer [29]. <NOTE 11>

If there are $m + 1$ levels in the hierarchy, then level-0 is the least detailed model, at the ***top-level model*** of the model. Level-m is the most detailed model, the ***bottom-level model***. The level-i model ($i = 0, 1, \ldots, m$) is represented by a set of trajectories U^i, called the ***level-i trajectory space***. <NOTE 12> (See Figure 3.1.)

At each level, the trajectory space U^i can be split into those components which can be composed in terms of the next lower level, and those which are basic to the level. The former is the ***level-i composite trajectory space*** U_c^i and the latter is the ***level-i basic trajectory space***

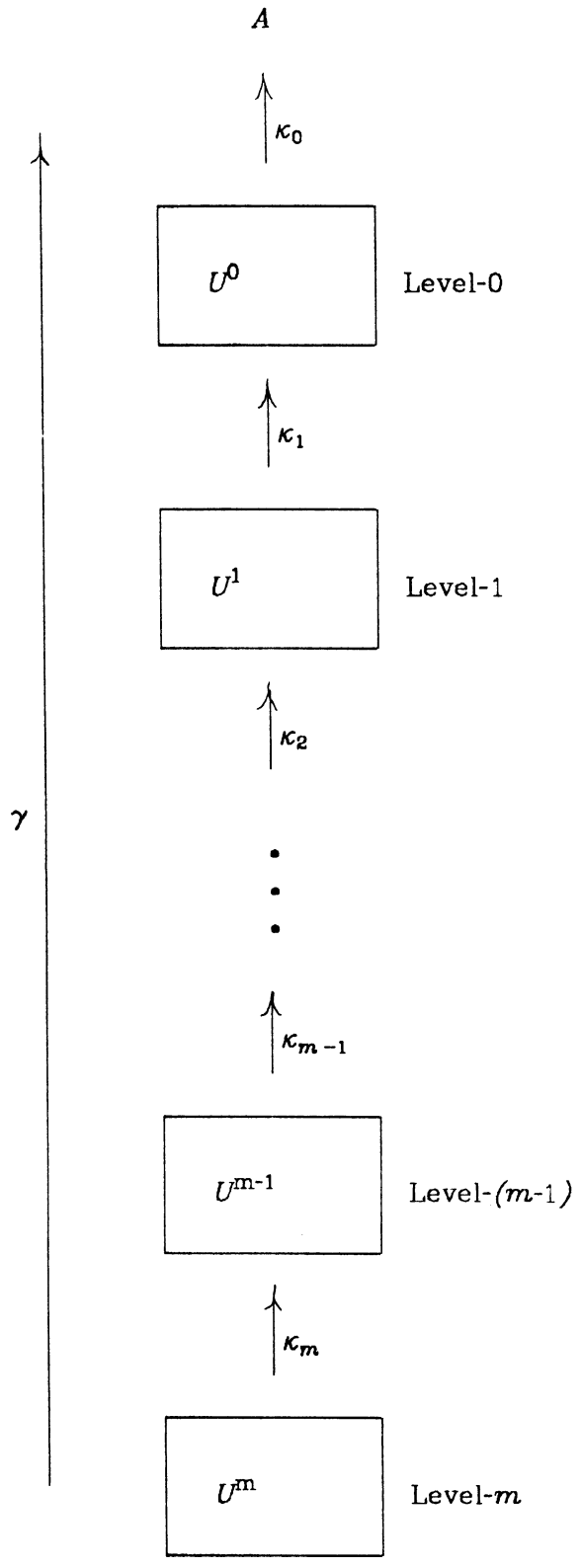


Fig. 3.1 The model hierarchy

U_b^i:

$$U^i = U_c^i \times U_b^i \ . \tag{3.15}$$

The level-$(i-1)$ $(i = 1, \ldots, m)$ composite trajectory space U_c^{i-1} is related to the level-i trajectory space U^i by a function called the ***i^{th}-interlevel translation***[6] κ_i:

$$\kappa_i: U^i \rightarrow U_c^{i-1} \ . \tag{3.16}$$

In the case $i = 0$,

$$\kappa_0: U^0 \rightarrow A \ . \tag{3.17}$$

Note that we can express the capability of a system in terms of higher level (less detailed) observation points than the bottom model. Specifically, let $\overline{U}^i$ denote the level-i trajectory space along with all the basic trajectory spaces of higher level models:

$$\overline{U}^i = U_b^0 \times U_b^1 \times \cdots \times U^i \tag{3.18}$$

where $\overline{U}^0 = U^0$ and $\overline{U}^m = U$. Define the ***level-i-based capability function***

$$\gamma_i: \overline{U}^i \rightarrow A \tag{3.19}$$

inductively as follows: if $i = 0$ and $u \in \overline{U}^0$, then

$$\gamma_0(u) = \kappa_0(u) \ , \tag{3.20}$$

and, for $i > 0$, if $(u, u') \in \overline{U}^i$ where $u \in U^i$, $u' \in U_b^0 \times U_b^1 \times \cdots \times U_b^{i-1}$, then

$$\gamma_i(u, u') = \gamma_{i-1}(\kappa_i(u), u') \ . \tag{3.21}$$

---

[6]The symbol κ for the interlevel translation was suggested by R. A. Ballance: each interlevel translation is actually a little "kappa-bility" function.

If $i = m$, then $\gamma_m = \gamma$.

3.3.8.2. Examples of a Model Hierarchy

A complete example of a model hierarchy can be complex. Two examples have been well documented in the literature [29], [32]. Though the examples are extremely relevant to this thesis, because of their availability and bulk, they have been omitted. The interested reader is referred to those other works.

An example by Furchtgott appears in [89]. The same example appears in Meyer, Ballance, Furchtgott, and Wu [30], and Meyer [29], [38]. A second example appears in Meyer, Ballance, Furchtgott, and Wu [31], Furchtgott and Meyer [39], and Meyer, Furchtgott, and Wu [32], [40], [44]; this latter example, in a much simplified form, is also the basis of the example by Pedar [151] and Pedar and Sarma [25]. A small example and two complex examples also appear in Hitt, Bridgman, and Robinson [156].

3.3.8.3. Constructing a Model Hierarchy

Consider the two-step procedure outlined in Section 3.3.6 [Solving for the Performability]. We can now state how the first step is carried out, i.e., how to determine, for a measurable $B \subseteq A$, $\gamma^{-1}(B)$. Beginning with the level-0-based capability function, we have

$$\gamma_0^{-1}(B) = \kappa_0^{-1}(B) , \tag{3.22}$$

and we proceed iteratively. Thus, if $\gamma_{i-1}^{-1}(B)$ has been determined, then

$$\gamma_i^{-1}(B) = \bigcup_{(u,u') \in \gamma_{i-1}^{-1}(B)} (\kappa_i^{-1}(u), u') \tag{3.23}$$

where $(\kappa_i^{-1}(u), u') = \{(v, u') \mid \kappa_i(v) = u\}$. When $\gamma_m^{-1}(B) = \gamma^{-1}(B)$ is reached, the procedure stops. To avoid manipulations of individual trajectories, actual implementations of the procedure utilize decompositions of $\gamma^{-1}(B)$ into characterizations of sets of trajectories,

i.e., subsets of the trajectory spaces. Section 3.4 [Models Having Finite Performance Variables] discusses such representations when the accomplishment set A is finite, while Section 3.5 [Models Having Continuous Performance Variables] discusses such representations when A is continuous.

The second step of Procedure 3.2 is to determine $\Pr(\gamma^{-1}(B))$. Again, implementations of the procedure seek to determine the probabilities of entire sets of trajectories in a single calculation, as opposed to dealing with single trajectories. Algorithms supporting the above procedure have been implemented in a computer programming package called METAPHOR [41], [46]. Section 4.4 [METAPHOR--A Performability Modeling and Evaluation Tool] discusses METAPHOR.

3.3.9. Difficulties of Employing Moments Rather than Performability

Often in performance analysis and occasionally in reliability analysis, moments are employed to describe a system's characteristics (e.g., mean throughput rate, mean response time, mean time between failure). (See, for example, Table 2.1.) Using moments is certainly attractive, since when compared to distributions, moments are often both easier to determine and simpler to interpret. However, the first few moments of the performance variable Y frequently do not yield enough information regarding the behavior of degradable systems. This deficiency has long been realized in reliability analysis [81]. We claim that for the evaluation of degradable systems, the performability distribution is the desired measure. This is not an obvious statement and requires justification.

We shall show by example that the system having the best ability to perform is generally not indicated by the moments of the random variable denoting performance. The steps of the formal procedure of Section 3.3.7 [Constructing Performability Models] will not be explicitly detailed in this example since they are straightforward and would obscure the focus

of this section.

Consider the following example: suppose we are designing a computing system whose user is concerned with system uptime during a finite utilization period $T = [0, h]$. Specifically, the user desires a system that will likely be operational for at least some fraction of the total time, i.e., $h \cdot k$, where $k \in [0, 1]$. If the system is up for either more or less time than $h \cdot k$, the user places no value on the difference of time. We consider using either a single (simplex) computer (call this system SIM) or three computers in a triple modular redundant (TMR)[7] configuration (system TMR). The computers fail permanently with an exponential distribution having rate λ, and we assume the voter cannot fail. The user is concerned with the amount of system uptime, and so we let the accomplishment set $A = [0, h]$ represent the amount of system uptime. Note that the accomplishment set is continuous in this example. The performance Y takes values in the accomplishment set A, and we are interested in the measurable set

$$B_k = \{a \in A \mid a > h \cdot k\} . \tag{3.24}$$

The following results are easy to derive:

$$p_{\mathrm{SIM}}(B_k) = e^{-\lambda h \cdot k}, \quad k \in [0, 1] \tag{3.25}$$

$$p_{\mathrm{TMR}}(B_k) = 3e^{-2\lambda h \cdot k} - 2e^{-3\lambda h \cdot k}, \quad k \in [0, 1] \tag{3.26}$$

$$\mathrm{E}[Y_{\mathrm{SIM}}] = \frac{1 - e^{-\lambda h}}{\lambda} \tag{3.27}$$

---

[7] TMR [77] (triple modular redundancy) is a method of increasing system reliability at the expense of incorporating extra components into the system. Three identical units provide their output to a voter; the result of the majority of the units is taken to be the result of the system. Hence, the system can tolerate the failure of a single module (and, in the case of compensating failures, the failure of two modules).

$$E[Y_{TMR}] = \frac{2e^{-3\lambda h}}{3\lambda} - \frac{3e^{-2\lambda h}}{2\lambda} + \frac{5}{6\lambda} \quad . \tag{3.28}$$

$p(B_k)$ has the interpretation "the probability that the system is up for at least a fraction k of the utilization period, i.e., the interval availability is k"

Given values for λ, h, and k, we can choose either system SIM or system TMR by one of the following two criteria:

1) the system with the highest expected system uptime (availability) $E[Y]$, or

2) the system with the highest probability of having a fraction of uptime greater than k, i.e., the highest performability value $p_S(B_k)$.

Let $\lambda = 10^{-3}$ and $h = \frac{\ln 4}{\lambda} = 1386$. Then $E[Y_{SIM}] = E[Y_{TMR}] = 750$ and the first moment criterion favors neither system. Figure 3.2 shows $p_{TMR}(B_k)$ and $p_{SIM}(B_k)$ as functions of k. We see that

$$\text{for } k < \frac{-\ln 0.5}{\lambda h} = 0.5 \,, \quad p_{TMR}(B_k) > p_{SIM}(B_k)$$

$$\text{for } k = 0.5 \,, \quad p_{SIM}(B_k) = p_{TMR}(B_k)$$

$$\text{for } k > 0.5 \,, \quad p_{SIM}(B_k) > p_{TMR}(B_k) \quad .$$

Hence, if using the performability criterion, we choose either TMR or SIM, depending on the value of k. For example, if $k = .8$, then we choose SIM, while if $k = .1$, then we choose TMR. Thus, for the above values of λ and h, the two criteria differ.

Since $p_S(B_k)$ is a measure of the probability of the system having the desired uptime, and that probability is the user's interest, the more reasonable choice is the answer provided by $p_S(B_k)$. First moments, in this case, do not indicate the proper system. Thus, we see that moments generally do not suffice even for non-degradable systems.

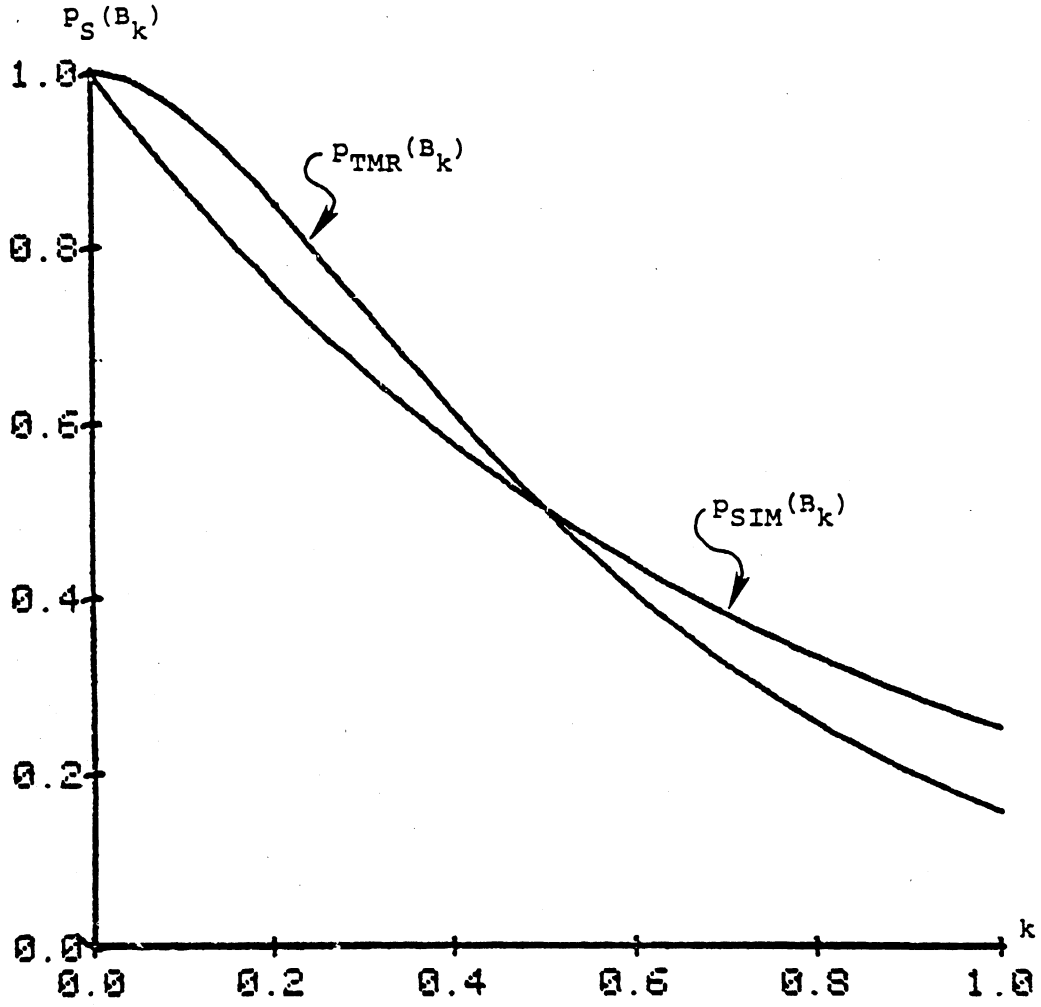


Fig. 3.2 $p_{TMR}(B_k)$ and $p_{SIM}(B_k)$ as functions of k

3.3.10. Notes for Section 3.3: An Informal Introduction to Performability

Note 1:

The information provided in the notes of this section is strongly based on the development of Section 2 of [29]. The work of [29], while precisely defining performability, does not address the issue of constructing performability models. In particular, the definition makes use of quantities that are assumed to be known; [29] does not concern the acquisition of those quantities. The emphasis in this set of notes is on constructive techniques. Specifically, the innovation presented in these notes is a procedure for constructing a performability model. (See Section 3.3.7 [Constructing Performability Models] for the construction procedure.) The following features of [59] have also been adopted here:

1) The development of the capability function γ as a random variable, and

2) The introduction of a random variable h relating the basic underlying probability space to the intermediate trajectory-based probability space.

Among the details in [59] not discussed in this thesis is discussion of how the co-ordinate probability space $(U,\mathfrak{F},Pr)$ is constructed knowing the finite-dimensional distributions of the base model X. Two innovations assist the construction of the performability model:

1) The explicit construction of a probability space $(A,\mathcal{B},p)$ that includes the accomplishment set and the performability, and

2) The *initial* assumption of the set $\mathcal{B}$ of measurable sets of accomplishment levels.

In these notes, important foundations will be established for the structures and relations employed in the evaluation of the capability function γ. The existence of various probability spaces underlying the total system will be postulated and the stochastic processes supporting "trajectory spaces" will be characterized. Recognition of these quantities is important, particularly to understand the stochastic nature of the models used and to differentiate between

"trajectories" and the random processes that define them.

Note 2:

To describe the stochastic behavior of the system and its environment, we assume the existence of a probability space $(A,\mathcal{B},p)$, where A is the ***sample space***, $\mathcal{B}$ is the set of ***events***, i.e., the measurable subsets of A, and $p:\mathcal{B}\rightarrow[0,1]$ is the ***probability measure***. (See [162], for example, for a discussion of probability spaces.)

Note 3:

The performability p is the probability measure of the above probability space $(A,\mathcal{B},p)$.

Note 4:

We assume there is a probability space $(U,\mathcal{F},\Pr)$ describing the low-level behavior of the system. The mapping

$$\gamma:U\rightarrow A \tag{3.29}$$

is a random variable if both of the following conditions hold for all $B\in\mathcal{B}$:

$$\begin{aligned} &a)\ \gamma^{-1}(B)=\{u\mid\gamma(u)\in B\}\in\mathcal{F}\\ &b)\ p(B)=\Pr(\gamma^{-1}(B))\,. \end{aligned} \tag{3.30}$$

γ is called the ***capability function***. (See Section 3.3.4 [The Capability Function γ].)

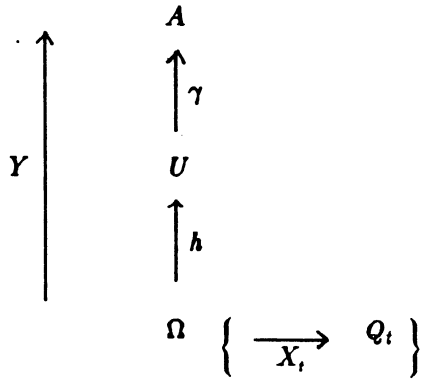
γ^{-1} is a set mapping, taking sets of A into sets of U. If γ is measurable, then γ^{-1} is measurable. We shall be constructing γ by first constructing γ^{-1}. Therefore, if γ^{-1} is a measurable set mapping, we are concerned with whether γ is measurable, or indeed, if γ even exists. Unfortunately, γ need not exist [e.g., if $a_1\neq a_2$, $\gamma^{-1}(a_1)=u$ and $\gamma^{-1}(a_2)=u$], and, if γ exists, it may not be measurable (e.g., see [163, p. 318]). Of course, we require that γ both exist and be measurable. We shall not pursue general conditions which insure these

conditions; rather, the classes of γ^{-1} considered in this dissertation will always induce γ which exist and are measurable.

The first class of γ^{-1} which we consider (see Chapter 4 [DISCRETE PERFORMANCE VARIABLE (DPV) METHODOLOGY]) will be from a countable set A to a countable set U. No restrictions will be placed on γ^{-1} except that no $u \in U$ can be in two preimages, i.e., if $u \in \gamma^{-1}(a_1)$ and $u \in \gamma^{-1}(a_2)$ then $a_1 = a_2$. Clearly, $\mathfrak{B}$ and $\tilde{\mathfrak{F}}$ are countable; so γ exists and is measurable.

The second class of γ^{-1} discussed (see Chapter 5 [CONTINUOUS PEFORMANCE VARIABLE (CPV) METHODOLOGY]) will be from $A = \mathbb{R}$ to $U = \mathbb{R}^n$. However, γ will be given *a priori* and will exist and be measurable.

The set U is the set of sample functions of a certain stochastic process. Thus, there is yet another probability space underlying U. The following diagram illustrates the relationships of the various sample spaces and random variables to be discussed:



(3.31)

We assume there is another probability space $(\Omega, \mathfrak{E}, P)$, a state space Q of the total system, and a stochastic process

$$X = \{X_t \mid t \in T\} \tag{3.32}$$

where $X_t: \Omega \to Q$. X is called the ***base model*** of the system. A sample function $X(\omega)$ is referred to as a ***state trajectory*** $u_\omega = \{u_\omega(t) \mid t \in T\}$, where $u_\omega: T \to Q$ and $u_\omega(t) = X_t(\omega)$. "State trajectory" is a term derived from the theory of modeling. In the context of stochastic

processes, the term "sample function" is typically used.

There is then a random variable $h:\Omega \rightarrow U$ defined as

$$h(\omega) = u_\omega \,. \tag{3.33}$$

The set U is then the set of all sample functions of $\{X_t\}$, i.e.,

$$U = \{u_\omega \,|\, \omega \in \Omega\} \,. \tag{3.34}$$

In practice, the underlying space $(\Omega, \mathfrak{S}, \mathrm{P})$ is unknown, and the base model X is usually described in terms of its finite-dimensional distributions. If T is continuous, there are then some measure-theoretical considerations in relating the space $(U, \mathfrak{F}, \mathrm{Pr})$ to the base model X. These difficulties arise because U is not countable (being the $|T|$-dimensional direct product of Q), while we have only finite-dimensional distributions and countable unions and intersections with which to work. These issues are addressed by Wu [59]. In particular, we shall require that X be a "separable" (see [164, p. 41]) process. h is called the ***system performance***: $Y = \gamma \cdot h$.

Note 5:

Note that the trajectory space U as informally defined above is *not* exactly the same trajectory space U as defined in Eq. 3.34. Eq. 3.34 restricts U to measurable "histories" only.

Note 6:

We desire to find the performability p. We know the accomplishment set A. To speak of the measure p, we must know which sets $\mathcal{B}$ are measurable. Therefore, we shall require that the analyst specify $\mathcal{B}$. In this dissertation, we shall consider only two classes of accomplishment sets A: finite and $\mathbb{R}^n$, $n < \infty$. The following event spaces $\mathcal{B}$ will be used:

a) If A is finite, then any subset of A will be measurable, i.e.,

$$\mathcal{B} = \{B \mid B \subseteq A\} \tag{3.35}$$

b) If A is $\mathbb{R}^n$, then $\mathcal{B}$ will be the n-dimensional Borel σ-algebra (see, for example, [164, Section 1.2].)

In the remainder of this dissertation, when we write "$\gamma^{-1}(B)$," we mean "$\gamma^{-1}(B)$ for $B \in \mathcal{B}$."

Note 7:

Knowing $(A,\mathcal{B})$, we wish to determine p. To do so, we:

1) (deterministically) construct γ^{-1} and so obtain $(U,\mathcal{F})$. This requires a top down analysis. Then,

2) we obtain the state set Q and index set T from U, describe the process $\{X_t\}$, and derive the measure Pr. From this information, we obtain the performability as

$$\begin{aligned} p(B) &= \Pr(\{u \mid \gamma(u) \in B\}) \\ &= \mathrm{P}(\{\omega \mid Y(\omega) \in B\}) . \end{aligned} \tag{3.36}$$

Note 8:

This is the set $\mathcal{B}$. See Section 3.3.2 [The Accomplishment Set].

Note 9:

See step c) below for restrictions on γ' ' .

Note 10:

Set $\mathcal{F}$ to the smallest σ-algebra that contains $\bigcup_{B \in \mathcal{B}} \gamma^{-1}(B)$. $\mathcal{F}$ must contain U_{CONS} since, when probabilities are later determined, the probability of a consistent trajectory must be one, i.e.,

$$\begin{aligned} \Pr(U_{\text{CONS}}) &= 1 \\ \Pr(U_{\text{INCONS}}) &= 0 . \end{aligned} \tag{3.37}$$

This condition sets restrictions on the function γ' ' of step 1.b.viii).

Note 11:

As in Section 3.3 [An Informal Introduction to Performability], notes will be employed in this section to present slightly more technical details. The treatment given in the notes of this section is somewhat different from the treatments of [29], [30] in that:

1) the underlying probability space is explicitly decomposed into separate probability spaces at each level,

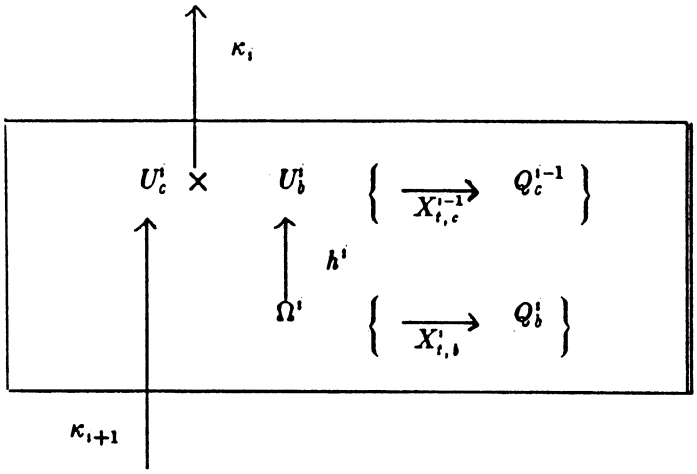
2) the interlevel translations κ_i are presented as random variables, and

3) the random variables h' are introduced to relate the basic level-i probability space to the level-i trajectory-based probability space.

Note 12:

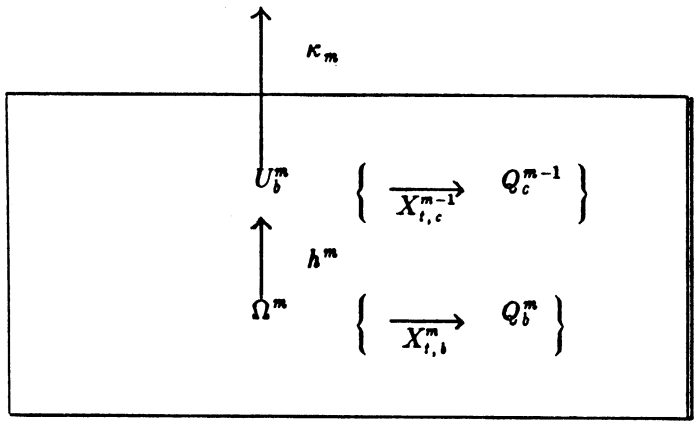
In Figure 1, the box representing level-i ($i = 1, 2, \ldots, m-1$) contains sample spaces and random variables with the following relationships, to be explained below (compare with

the diagram of Eq. 3.31):



(3.38)

Level-0 is the same as above, except that Q^{i-1} is replaced with A. Level-m is simply:



(3.39)

We assume that at level-i there is a state space Q^i. Corresponding to each level-i there is also a stochastic process

$$X^i = \{X_t^i \mid t \in T^i\} \tag{3.40}$$

where X_t^i is to be defined below. X^i is called the ***level-i model***. $T^i \subseteq T$ is called the ***level-i parameter set*** and Q_b^i is the ***level-i state space***. If the cardinality of T^i is finite, we refer to the interval between two adjoining times t_1 and t_2 as a ***level-i phase*** if

1) $t_1, t_2 \in T^i$, $t_1 < t_2$, and

2) there is no $t_3 \in T^i$ such that $t_1 < t_3 < t_2$.

Q^i can usually be decomposed

$$Q^i = Q^i_c \times Q^i_b \tag{3.41}$$

where Q^i_c is the ***composite state set*** and Q^i_b is the ***basic state set*** (at level-i). Note in Eq. 3.38 that Q^i_c is denoted in the next lower box, i.e., at level-(i+1). The composite state set contains those states in Q^i that are uniquely determined by the state behavior at level-(i+1), in a manner to be made clear below. Q^i_c is thus a composite of lower-level states. Basic state information newly introduced at level-i is contained in the basic state set Q^i_b. Thus, states in Q^i_b represent information not conveyed by states in Q^{i+1}. If the level-i model has no composite (or no basic) part, then the basic (or alternatively composite) part is deleted, i.e., $Q^i = Q^i_b$ (or $Q^i = Q^i_c$). Note that the bottom level (level-m) cannot have a composite part, and so $Q^m = Q^m_b$.

We view X^i as a pair of stochastic processes

$$X^i_c = \{X^i_{c,t} \mid t \in T^i_c\} \tag{3.42}$$

$$X^i_b = \{X^i_{b,t} \mid t \in T^i_b\} \tag{3.43}$$

called the ***level-i composite process*** and ***level-i basic process***, respectively. Here $T^i_c \subseteq T^i$, $T^i_b \subseteq T^i$, $X^i_{c,t} \in Q^i_c$, and $X^i_{b,t} \in Q^i_b$. Because the index sets T^i_c and T^i_b may not be the same, we extend the state spaces Q^i_b and Q^i_c with the fictious state ϕ so that if $t \notin T^i_c$, then $X^i_{c,t}$ is defined to be ϕ, (or, if $t \notin T^i_b$, then $X^i_{b,t}$ is defined to be ϕ) for all $\omega \in \Omega$.

The i^{th}-level basic process is defined over the probability space $(\Omega^i, \mathcal{E}^i, P^i)$. Following the development in Section 3.3.3 [The Trajectory Space], $X^i_{b,t}: \Omega^i \rightarrow Q^i_b$. A sample function $X^i_b(\omega)$ is a ***level-i basic state trajectory*** $u^i_\omega = \{u^i_\omega(t) \mid t \in T^i_b\}$, where $u^i_\omega: T^i_b \rightarrow Q^i_b$ and

$u^i_\omega(t) = X_{b,t}(\omega)$. There is a random variable $h^i: \Omega^i \to U^i_b$ defined as

$$h^i(\omega) = u^i_\omega \tag{3.44}$$

and the U^i_b is then the set of sample functions of $\{X^i_{b,t}\}$, i.e.,

$$U^i_b = \{u^i_\omega \mid \omega \in \Omega^i\} . \tag{3.45}$$

The development of the i^{th}-level composite process is slightly different. At level-i, there is a probability space $(U^i, \mathfrak{F}^i, \Pr^i)$ (see below). Again following the development in Section 3.3.3 [The Trajectory Space], $X^i_{c,t}: U^{i+1} \to Q^i_c$. A sample function $X^i_c(u)$ is a ***level-i composite state trajectory*** $\kappa^{i+1}_u = \{\kappa^{i+1}_u(t) \mid t \in T^{i+1}_c\}$, where $\kappa^{i+1}_u: T^{i+1}_c \to Q^i_c$ and $\kappa^{i+1}_u(t) = X^{i-1}_{c,t}(u)$. There is a random variable $\kappa^i: U^i \to U^{i-1}_c$ $(i = 1, 2, \ldots, m;\ \kappa^0: U^0 \to A)$ defined as

$$\kappa^i(u) = \kappa^i_u . \tag{3.46}$$

The random variable κ^i is called the i^{th} ***interlevel translation***. The set U^i_c is then the set of sample functions of $\{X^i_{c,t}\}$, i.e.,

$$U^i_c = \{\kappa^{i-1}_u \mid u \in U^{i-1}\} . \tag{3.47}$$

Now we define

$$X^i = (X^i_c, X^i_b) \tag{3.48}$$

and the probability space

$$(U^i, \mathfrak{F}^i, \Pr^i) = (U^i_c \times U^i_b, \mathfrak{F}^{i+1} \otimes \mathfrak{E}^i, \Pr^{i+1} \times \mathrm{P}^i) \tag{3.49}$$

where $\mathfrak{F}^{i+1} \otimes \mathfrak{E}^i$ is the smallest σ-algebra that contains $\mathfrak{F}^{i+1} \times \mathfrak{E}^i = \{F \times E \mid F \in \mathfrak{F}^{i+1}, E \in \mathfrak{E}^i\}$. The ***level-i trajectory space*** U^i is $U^i_c \times U^i_b$.

The capability function γ is then simply the composition of all the κ_i, i.e.,

$$\gamma = \kappa_0 \cdot \kappa_1 \cdot \cdots \cdot \kappa_{m-1} \cdot \kappa_m \ . \tag{3.50}$$

Of course, if either no composite or no basic part at level-i is present at level-i, then the above representations of X^i and U^i are understood to be the appropriate single component versions. The collection of level-i models $\{X^0, X^1, \cdots, X^m\}$ is a ***model hierarchy*** if the following conditions hold:

1) $X^m = X_b^m$, that is, the bottom model is comprised only of a basic process.

2) $X = \{X_t \mid t \in T\}$, where

$$X_t = (X_{b,t}^m, X_{b,t}^{m-1}, \cdots, X_{b,t}^0); \tag{3.51}$$

$$Q = Q_b^m \times Q_b^{m-1} \times \cdots \times Q_b^0; \text{ and} \tag{3.52}$$

$$U = U_b^m \times U_b^{m-1} \times \cdots \times U_b^0 \ . \tag{3.53}$$

3) For each level-i, there is an ***interlevel translation*** κ_i, where

$$\begin{aligned} &\kappa_0 : U_c^0 \times U_b^0 \to A \\ &\kappa_i : U_c^i \times U_b^i \to U_b^{i-1}, \quad (0<i<m) \\ &\kappa_m : U_b^m \to U_b^{m-1} \end{aligned} \tag{3.54}$$

such that for $u = (u_m, u_{m-1}, \cdots, u_0) \in U$ with $u_i \in U_b^i$

$$\gamma(u) = \kappa_0(\kappa_1(\cdots \kappa_{m-1}(\kappa_m(u_m), u_{m-1}), \cdots), u_0) \ . \tag{3.55}$$

The probability space underlying the base model $\{X_t\}$ is the product space

$$(\Omega, \mathcal{E}, P) = (\Omega^0 \times \Omega^1 \times \cdots \times \Omega^m, \tag{3.56}$$
$$\mathcal{E}^0 \otimes \mathcal{E}^1 \otimes \cdots \otimes \mathcal{E}^m, P^0 \times P^1 \times \cdots \times P^m) .$$

3.4. Models Having Finite Performance Variables

Much of the work done to date on performability modeling has concerned the case where the performance variable Y assumes a countable (finite or countably infinite) number of values. The previous work includes evaluations of real-time control computers in a finite-length mission [25], [29]-[32], [36], [39], [44], [89], [151], [156]. To perform large scale evaluations of the type reported on above, machinery for doing much of the mechanical work automatically must be specified and implemented. Part of this dissertation discusses notation, algorithms, and a software package for doing discrete performance variable performability evaluation. This section presents an overview of the results.

When the performance variable Y is discrete, the performability methodology will be referred to as the "discrete performance variable methodology," or ***DPV methodology***. Analogously, if Y is continuous, then we refer to the "continuous performance variable methodology," or ***CPV methodology***. In this dissertation, if the performance variable is discrete, then it is finite. We shall not deal with countably infinite accomplishment sets in this treatise. Section 3.5 [Models Having Continuous Performance Variables] discusses the case where Y is

continuous.

3.4.1. Basic Concepts

3.4.1.1. The Accomplishment Set

Let the accomplishment set contain $n < \infty$ accomplishment levels, i.e.,

$$A = \{a_0, a_1, \ldots, a_{n-1}\}. \tag{3.57}$$

No assumption will be made regarding the relative importance of the a_i to the user. That is, for any i and j, a_i will not be considered to have more or less value than a_j. The finite accomplishment set induces the type of performability evaluation that comes closest to traditional reliability evaluation. Indeed, as mentioned in Section 3.3.2 [The Accomplishment Set], reliability evaluation requires the two-element set A = { successful, fail }.

As an example of a finite accomplishment set, consider the following scenario suggested by Furchtgott [89], that also appears in Meyer, Ballance, Furchtgott, and Wu [30], [36], Meyer, Furchtgott, and Wu [31], Meyer [29], [38], Furchtgott and Meyer [39], Meyer, Furchtgott, and Wu [32], [44], Pedar [151], and Pedar and Sarma [25]. An airline company has an aircraft with a degradable computing system. The company is concerned with the period of utilization, consisting of, say, a flight of 10 hours duration. The airline is interested in safety, passenger convenience (specifically, avoiding diversion to an alternate landing site), and fuel consumption. Five levels of accomplishment are recognized:

a_0: safe, no diversion to an alternate landing site, and low fuel consumption

a_1: safe, no diversion, and high fuel consumption

a_2: safe, diversion, and low fuel consumption

a_3: safe, diversion, and high fuel consumption

a_4: unsafe.

Clearly, it is difficult to assign an ordering to the above mentioned accomplishment levels. For instance, unless one is familiar with airline operations, it not clear whether the airline would prefer accomplishment a_1 to a_2. Because no ordering of the set A is presumed, any reliance on a concept similar to coherence (see Section Barlow and Proschan [7]) is precluded.

3.4.1.2. The Trajectory Space

As mentioned in Section 3.3.3 [The Trajectory Space], the set of all low-level system behaviors is characterized by the trajectory space U. We shall use the assumptions and notation suggested by Furchtgott [89] that were expanded in [30], [36]. Those concepts also appear in [29], [31], [32], [38], [39], [44].

We assume that the trajectory space U (see Section 3.3.3 [The Trajectory Space]) is also finite. In particular, the parameter set T and the state space Q are both finite. The number of trajectories hence is $|Q|^{|T|}$. Since we assume that the connections between high-level and low-level behaviors are deterministic [see Step 1) of the procedure in Section 3.3.6 [Solving for the Performability]], we do not need to consider the underlying stochastic description of U. Only when we wish to consider the probabilities of accomplishment levels will we need to deal with the base model X (Eq. 3.2) [see Step 2) of the procedure in Section 3.3.6 [Solving for the Performability]].

U describes the possible behaviors of the low-level components during certain intervals called "phases." Each component during each phase will have a certain behavior, and the behavior will be assigned a value from some finite set. The phases associated with each component need not be the same. Also, as seen in Section 3.3.8 [The Model Hierarchy], it is convenient to group certain components together, especially those that have similar "levels of abstraction," or briefly, "levels." Again, only a finite number of levels is allowed.

To describe a trajectory $u \in U$, we use a vector of matrices whose sizes are finite but

not necessarily identical:

$$u = \left([u^0_{i,j}],\ [u^1_{i,j}],\ \ldots,\ [u^m_{i,j}] \right), \tag{3.58}$$

where the element $u^k_{i,j}$ denotes the behavior of component i during phase j at level of abstraction k. Thus, each matrix represents a level of abstraction, each row of each matrix represents a component, and each column represents a phase. The set U^k_{ij} is the set of all possible u^k_{ij}.

For example, consider the aircraft example presented above. In evaluating the computing system, we may wish to take into account the weather that the flight encounters. The weather may then be denoted a system component. Further, we may wish to consider the behavior of the weather only during the period of time during which the aircraft lands. We assume that the weather affects only the various operations of the aircraft, e.g., landing and navigation, and not the functioning of the computer itself. The level of abstraction of the weather (which is higher than that of the computer) might then be called the "operational level." Thus, we would speak of the component "weather" during the phase "landing" at the level "operational level." The set of possible values might be

$$U^{\text{operational level}}_{\text{weather, landing}} = \{\text{Cat III, non-Cat III}\} \tag{3.59}$$

where Cat III is a category of bad weather requiring certain types of instrumentation for landing. For conciseness, we will usually symbolically encode names of components, phases, levels, etc. into numbers. Eq. 3.59 may then be written

$$U^1_{2,3} = \{0, 1\}. \tag{3.60}$$

When used, the encoding will be made clear.

3.4.2.3. Trajectory Sets and Their Representations

When the performability model of a system involves a finite number of trajectories, one can theoretically write out every trajectory. However, in practice, there may be a large number (say, billions, trillions, or more) of such trajectories, and so dealing with each trajectory separately is intractable.

Instead, we shall deal with sets of trajectories as single entities. These entities are called ***trajectory sets***. A single trajectory set may denote from as few as zero trajectories to as many as are in the entire trajectory space. Trajectory sets and a representation of them were first described in the context of performability modeling by Furchtgott [89], and are also described in Meyer, Ballance, Furchtgott, and Wu [30]. A calculus for manipulating trajectory sets is presented by Furchtgott [89] and also appears in Meyer, Ballance, Furchtgott, and Wu [30].

Section 4.2.3 [Alternative Representations of Trajectory Sets] discusses various representations of trajectories and trajectory sets. In particular, the representation of [89] and [30] is discussed. This representation of trajectory sets is a variation of the lattice representation of discrete functions. (See Davio, Deschamps, and Thayse [165].) The lattice representation is also briefly reviewed in Section 4.2.3.2 [Representations Using Lattice Expressions] and is then employed in Section 4.3 [Calculation of Trajectory Sets] to describe algorithms for calculating trajectory sets for a performability model.

3.4.3. Algorithms for Calculating Trajectory Sets

Suppose we have specified the structure of a model hierarchy (see Section 3.3.8 [The Model Hierarchy]), i.e., the accomplishment level A, the models $(X^0, X^1, \ldots, X^m)$, and the interlevel translations $\kappa_0, \kappa_1, \ldots, \kappa_m$. Then we wish to solve the model by calculating the inverse capability function γ^{-1} (see Eq. 3.4).

Computationally tractable algorithms or heuristics are required to perform these calculations. In addition, errors in the specification of the κ_i can be made by the analyst. Hence, further algorithms for checking the integrity of the κ_i are useful. To save additional computational costs, heuristics for reducing the number of trajectory sets are also of value.

Algorithms and heuristics for the above purposes are introduced in Section 4.3 [Calculation of Trajectory Sets]. Let a be an accomplishment level in A. The algorithms include:

1) Based on Eq. 3.23, an algorithm for iteratively calculating γ^{-1} given the κ_i^{-1}.

2) A heuristic, recursive, tree-based algorithm for calculating $\gamma_i^{-1}(a)$ (see Eq. 3.19) given γ_{i-1}^{-1} and κ_i^{-1}. The leaves of the tree represent the trajectory sets of $\gamma_i^{-1}(a)$ and the branches denote certain intermediate trajectory sets. The algorithm recursively constructs the tree.

3) A heuristic algorithm for reducing the number of trajectory sets. The algorithm iteratively attempts to combine pairs of trajectory sets into single trajectory sets.

4) An algorithm for

   a) determining whether the specification of κ_i^{-1} is complete, i.e., if for every $u \in \overline{U}^{i-1}$ (see Eq. 3.18), whether $\kappa_i^{-1}(u)$ has been specified, and

   b) if the specification is incomplete, determining which values of $\overline{U}^{i-1}$ have not been specified.

Embedded in the above algorithms are other algorithms that implement the operations of the trajectory set calculus of [31], [89]. The purpose of describing the above-mentioned algorithms and heuristics is to automate, to the greatest extent possible, those tasks of performability modeling that are mechanical, laborious, and error-prone. Hence, the emphasis is on issues concerning computer implementation of the algorithms. We are particularly concerned with computational efficiency, where "efficiency" relates to both

1) the amount of space required to represent the functions γ and κ_i, and

2) the amount of time required to determine the representations of those functions.

Much of the theory developed to date concerning discrete functions deals with optimizing the amount of space required to represent such functions. That is because, in most applications (e.g., the design of switching circuits using switching functions), discrete functions do not need to be calculated repeatedly. Hence, the cost of the time required to compute a spatially optimal representation is slight, compared to the cost of the space itself. The current theory thus concerns topics such as concise representations and optimal coverings. However, our applications concern extensive manipulation of discrete functions, where the cost of the space (i.e., computer memory) to represent the function is not great. On the other hand, the computing time necessary to modify a function's representation is relatively expensive. We must specify how much time is appropriately spent reducing the size of the function's representation before we proceed with other computations involving that function.

Such issues have been addressed in the study of exact reliability evaluation of fault trees [166]-[169] and reliability networks [170]-[174]. However, most of those investigations concern problems such as determining a disjoint sum-of-products representation, knowing the set of prime implicants. The problem we face is one of constructing a disjoint sum-of-products representation of γ^{-1} directly from the interlevel translations κ_i.

The tradeoff between computing time and representation space is not straightforward. For example, reducing the size of a function's representation can serve to reduce the time required to determine other representations. Thus, the above-mentioned two efficiency components, space and time, are not totally conflicting, since, by reducing the size of a function, we could also reduce its related computational time. Nevertheless, finding a function's optimal spatial representation can require significantly more time than finding some sub-optimal spatial representation. Further, storage in contemporary computer system memories is not expensive compared to system time. Therefore, it appears that the most efficient method of computing γ is to invest some time in reducing the size of given functions, and not to spend a large amount of time determining optimal representations. We explore the above

issues in Section 4.3 [Calculation of Trajectory Sets].

3.4.4. METAPHOR--A Performability Modeling and Evaluation Tool

The algorithms and heuristics necessary to calculate the trajectory sets of a performability model are discussed in Section 4.3 [Calculation of Trajectory Sets]. To be useful, the algorithms must be implemented as computer programs. Therefore, we have incorporated the algorithms into the software package METAPHOR (**M**ichigan **E**valua**T**tion **A**id for **P**erp**HOR**mability). METAPHOR was originally envisioned as a tool to be used at all stages of performability analysis, from the definition of model levels and interlevel translations (κ'), to the calculation of trajectory sets, to the evaluation of the probabilities of those sets. Also, because of the design nature of constructing the model hierarchy, METAPHOR was intended to be an interactive facility. Owing to the large quantities of data that are sometimes necessary to input, METAPHOR can also be run in batch mode.

The first version [41], [46] implements the probability evaluation of trajectory sets. The second and third versions implement the other steps. (In addition, the third version handles continuous performance variable evaluations as well as discrete performance variable ones.) The structure and use of METAPHOR are briefly discussed in Section 4.4 [METAPHOR--A Performability Modeling and Evaluation Tool].

The primary data structure employed in the algorithms of Section 4.3 [Calculation of Trajectory Sets] is the array, and so the first two versions of the package are written in APL [175], a computer language with extensive array manipulation capabilities and compact notation. These APL versions have been written under the Michigan Terminal System (MTS). Version 2 was substantially completed, but because of accessibility considerations, METAPHOR has been ported to UNIX[8]. Unfortunately, there is currently no UNIX APL suffi-

---

[8]UNIX is a trademark of Bell Laboratories.

ciently powerful to run the APL version of METAPHOR. Therefore, version 3 is written in C [176], a powerful, efficient language with flexible data structures and modern control flow. The third version also contains facilities for solving continuous performance variable performability models and has a menu-based user interface.

METAPHOR is large. Not including the help facilities, METAPHOR contains approximately 150 functions; version 2 consists of 4500 lines of (commented) APL code (2700 lines uncommented) and version 3 has about 12,000 lines of (commented) C. Figure 3.3 shows the general layout of METAPHOR.

3.4.5. Examples

As mentioned in Section 3.3.8.2 [Examples of a Model Hierarchy], examples of model hierarchies can be complex; complete evaluations can be even more complex. One example of a performability model and evaluation appears in Furchtgott [89], Meyer, Ballance, Furchtgott, and Wu [30], and Meyer [29], [38]. A second example, a performability evaluation of the SIFT computer [67], [68] appears in Meyer, Ballance, Furchtgott, and Wu [31], Furchtgott and Meyer [39], and Meyer, Furchtgott, and Wu [32], [40], [44]. These studies generated the basic developments in the discrete performance methodology and also demonstrated the feasibility of performing such analyses. Also, a small example and two complex examples of performability evaluations appear in Hitt, Bridgman, and Robinson [156].

Section 3.4.5 [Examples] examines further developments in the DPV methodology that generally extend its scope of applicability. In particular, a less restricted notion of "phasing" is introduced. A non-trivial example is presented in Section 3.4.5 [Examples] to illustrate the methodology.

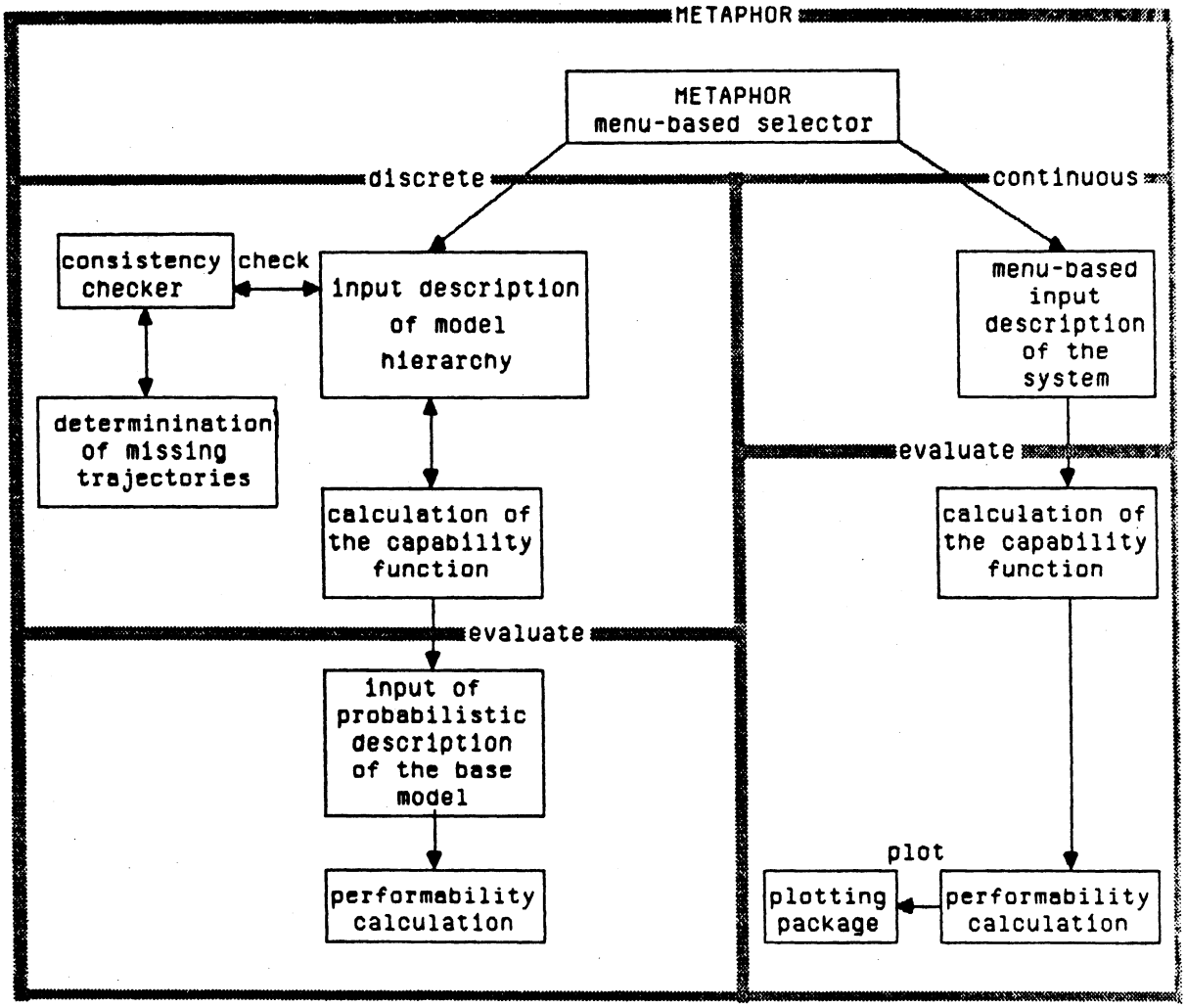


Figure 3.3 -- Block diagram for METAPHOR

3.5. Models Having Continuous Performance Variables

The models employed when the performance variable is continuous (the CPV methodology) are similar to those employed in performance evaluation (e.g., Markovian queueing models) but are extended to account for variations in structure that are due to faults. The initial work on the CPV methodology is by Meyer [51]. That investigation examines a degradable buffer/multi-processor system with 2 processors, where the performance variable Y was taken to be the "normalized average throughput rate" of the system. Later, a more systematic approach was developed and documented by Meyer [33], [53], [56] that models the system with N processors.

Among the fundamental results of [33], [53], [56] is an innovative approach for obtaining closed-form solutions relative to the above important class of performability models. This approach is summarized on page 24 of [53] by a five-step algorithm. In Chapter 5 [CONTINUOUS PEFORMANCE VARIABLE (CPV) METHODOLOGY],

i) the above problem is developed more generally in the context of reward models and nonrecoverable processes [55],

ii) an integral solution for the class of systems that can be modeled by a finite-state, acyclic, nonrecoverable process is derived,

iii) a recursive solution is derived from the solution of ii),

iv) a software package for the calculation of CPV performability is discussed,

v) a non-Markovian, multiprocessor/air conditioner example is investigated,

vi) several additional examples are investigated, and

vii) consideration of the closed-form solutions of still more general models is begun.

This section presents an overview of these results. First, Section 3.5.1 [Basic Concepts] introduces the basic concepts of the CPV methodology and contrasts the CPV and DPV methodologies. Section 3.5.2 [Reward Models and Nonrecoverable Processes] briefly discusses reward models and nonrecoverable processes while Section 3.5.3 [Solution of Finite-State

Nonrecoverable Processes] overviews the solution obtained for finite-state, acyclic, nonrecoverable processes.

3.5.1. Basic Concepts

3.5.1.1. The Accomplishment Set

Let the performance variable Y be continuous, specifically, the image of Y is a continuum such as the real numbers $\mathbb{R}$ or the n-dimensional space $\mathbb{R}^n$. As an illustration of the applicability of a continuous performance variable, consider the example presented by Meyer [33] in which the accomplishment of a computing system is simply the normalized throughput rate[9] of the system. The accomplishment set is the interval [0,1]. Another illustration is the example of Section 3.3.9 [Difficulties of Employing Moments Rather than Performability], where the accomplishment is the amount of system uptime, and so $A = [0, h]$. Still other instances of interpretations of continuous performance variables include performance-based measures such as response time, turnaround time, processor utilization, waiting time, and length of busy period, as well as cost-based measures such as cost-per-mission.

We assume that unlike the finite performance variable, the continuous performance variable has a partial ordering $\leq$. (In fact, in this dissertation, we generally assume A is isomorphic to $\mathbb{R}^n$, $n < \infty$; see Note 6 of Section 3.3 [An Informal Introduction to Performability].) The partial ordering provides the ability to say that one outcome may be "better" than another; hence, we can speak of the set of outcomes that are "better" than some threshold.

3.5.1.2. The Trajectory Space

---

[9]The *throughput rate* is the number of jobs processed per unit of time. The *normalized throughput rate* is the achieved throughput rate divided by the maximum throughput rate.

Generally, in the CPV methodology, we assume the trajectory space U (see Section 3.3.3 [The Trajectory Space]) is uncountable. If U were countable, then the image of the capability function $\gamma: U \rightarrow A$ (Eq. 3.3) would be countable and a countable accomplishment set IMAGE(γ) could be substituted for A.

The elements of the trajectory space U are functions

$$u: T \rightarrow Q \tag{3.1}$$

where T is the parameter set and Q is the state space. As a precondition for U to be uncountable, either T or Q must be uncountable. We generally assume that T is uncountable. No restrictions will be placed on the state space Q, although in this dissertation, Q will usually be finite. Of course, T could be countable and Q could be uncountable and still satisfy the criterion that U be uncountable, but we do not consider such cases in this research.

As an illustration of a trajectory space used to support a continuous accomplishment set, consider the simplex (SIM) computer of the example of section 3.3.9 [Difficulties of Employing Moments Rather than Performability]. The state space is the two-element set [0,1], where 1 denotes that the computer is operational and 0 denotes the computer has failed.

The parameter set T could be chosen to be either $[0, h]$ or $[0, \infty)$ since the behavior of the system after time h is irrelevant. Let us choose $[0, \infty)$ to make the stochastic description easier. The trajectory space U is the set of all functions $u: T \rightarrow Q$, i.e., Q^T.

The system has no repair, and so once the system enters state 0, state 1 cannot be entered again. Hence, the consistent trajectory space (see Section 3.3.7 [Constructing Performability Models]) U_{CONS} is the set of all monotonically non-increasing functions from T to Q, i.e.,

$$U_{CONS} = \left\{ u: T \rightarrow Q \mid u(t) \geq u(t') \text{ for all } t, t' \in T \text{ such that } t \leq t' \ . \right\}. \tag{3.61}$$

The set of inconsistent trajectories U_{INCONS} contains all other functions $T \rightarrow Q$, i.e.,

$$U_{\text{INCONS}} = \{u : T \rightarrow Q \mid u \notin U_{\text{CONS}}\} . \tag{3.62}$$

3.5.2. Reward Models and Nonrecoverable Processes

A common feature of user-oriented behavioral descriptions of computing systems is the presence of various "operational modes." Typically, such modes reflect different rates at which the system accumulates (or dissipates) reward, where the reward is a measure of user satisfaction. Reward rates can often be identified with aspects of system performance such as productivity, responsiveness, and utilization, or, at a higher level, with broader measures such as economic benefit. "Operational mode" is a behavioral concept, not a structural one, and hence operational modes are not to be confused with the structural or physical state of the system. There is frequently a strong correspondence between physical states and operational modes; for example, the present state of a system may induce the present mode of the system. However, in other cases, other factors, such as the system's state trajectory, may affect the system's mode.

Meyer and Wu [55] have introduced a user-oriented "operational model" that is based on the variations in rates from mode to mode. In Section 5.2 [Reward Models], we review a special case of operational model called "reward model."

3.5.3. Solution of Finite-State Nonrecoverable Processes

Among the fundamental results of [53], [56] is an innovative approach for obtaining closed-form solutions relative to an important class of performability models. This approach is summarized on page 24 of [53] by the algorithm that is recapitulated in Section 5.3.3 [The Approach]. However, although the algorithm delineates in broad terms the basic method for arriving at solutions, the algorithm does not suggest specific techniques for actually implementing the prescribed steps. In particular, the set of trajectories $\gamma_1^{-1}(B)$ (Eq. 3.4) must be

characterized. Thus, the computational example presented in [33], [53], [56] was derived in a relatively *ad hoc* manner; effectively, the solution was based on a graphical argument. This type of approach becomes more difficult when the number of states in a trajectory is four, and becomes intractable when the number of such states grows to five or more.

Section 5.3.8 [Solution of Finite State Acyclic Reward Models] presents an integral solution for the class of systems that can be modeled by a finite-state, acyclic, nonrecoverable process. The crux of the solution is the characterization of the regions C_y. If specific state transition distributions are given, the integrations can be performed and the performability determined. The stochastic nature of the underlying process is unrestricted, and in particular, the process can be non-Markovian.

CHAPTER 4

DISCRETE PERFORMANCE VARIABLE (DPV) METHODOLOGY

4.1. Trajectory Sets: Basic Notation and Operations

This section discusses a concise notation for describing trajectory sets (see Section 3.3.3 [The Trajectory Space]), along with a primitive calculus for manipulating trajectory sets. This calculus is useful for manual manipulation of trajectory sets. Most of the concepts in this section appear in Furchtgott [89]. They are also discussed in Meyer, Ballance, Furchtgott, and Wu [30]. Most of the early performability evaluations (e.g., Meyer [29] and Meyer, Furchtgott, and Wu [32]) were carried out using the ideas, notation, and calculus described in this section.

A more formal development appears in Section 4.2 [Discrete Functions and the Representation of Trajectory Sets]. The discussion in this section provides a straightforward introduction to Section 4.2 [Discrete Functions and the Representation of Trajectory Sets].

4.1.1. Notation and Terminology

The discrete variable methodology makes extensive use of arrays. The indexing of vectors, matrices, and arrays in this chapter will always begin at 0. An $s + 1$ dimensioned vector $[x_0, x_1, \ldots, x_s]$ will frequently be denoted by $[x_i]_s$, or if the index i must be explicitly indicated, by $[x_i]_{i:s}$ An $(r + 1) \times (s + 1)$ dimensioned matrix will be similarly denoted by

$[x_{ij}]_{r \times s}$ or $[x_{ij}]_{i,j:r \times s}$, i.e.,

$$[x_{ij}]_{r \times s} = [x_{ij}]_{i,j:r \times s} = \begin{bmatrix} x_{00} & x_{01} & \cdots & x_{0n} \\ x_{10} & x_{11} & \cdots & x_{1n} \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ x_{r0} & x_{r1} & \cdots & x_{rs} \end{bmatrix} . \tag{4.1}$$

Higher dimensioned arrays and arrays of vectors, etc., make similar use of the same notation.

Briefly reviewing Section 3.3.8 [The Model Hierarchy], a ***composite*** (alternatively, ***basic***) ***trajectory at level-i*** is a function[10] $\mathbf{U}_c^i$: $T_c^i \rightarrow Q_c^i$ ($\mathbf{U}_b^i$: $T_b^i \rightarrow Q_b^i$), where $\mathbf{U}_c^i(t) = X_{c,t}^i = X_{c,t}^i(\omega)$ ($\mathbf{U}_b^i(t) = X_{b,t}^i = X_{b,t}^i(\omega)$) for some $\omega \in \Omega$. T_c^i (T_b^i) is the ***level-i composite (basic) utilization period*** while Q_c^i (Q_b^i) is the ***level-i composite (basic) state space.*** The ***level-i composite (basic) trajectory space*** is the set $\mathbf{U}_c^i = \{\mathbf{u}_c^i\} = \{\mathbf{u}_{c,\omega}^i \mid \omega \in \Omega\}$ ($\mathbf{U}_b^i = \{\mathbf{u}_b^i\} = \{\mathbf{u}_{b,\omega}^i \mid \omega \in \Omega\}$). The ***level-i trajectory space*** is thus $\mathbf{U}^i = \mathbf{U}_c^i \times \mathbf{U}_b^i = \{(\mathbf{u}_{c,\omega}^i, \mathbf{u}_{b,\omega}^i) \mid \omega \in \Omega\}$.

The random processes X_c^i and X_b^i generally describe several system ***components***, i.e., features such as hardware subsystems or behavioral functions that are identifiable and helpful in portraying the system. As noted in Section 3.3.8 [The Model Hierarchy], Q_c^i and Q_b^i can be coordinatized; the projection of $X_{c,t}^i$ ($X_{b,t}^i$) on a particular coordinate is called a ***composite (basic) variable.*** For the trajectories used, two coordinates are employed. One coordinate is the particular component being observed, while the other coordinate is the observation time.

A level-i trajectory $\mathbf{u}^i = \mathbf{u}_c^i \times \mathbf{u}_b^i \in \mathbf{U}^i = \mathbf{U}_c^i \times \mathbf{U}_b^i$ is first written as a column array:

$$\mathbf{u}^i = \begin{bmatrix} \mathbf{u}_c^i \\ - - - \\ \mathbf{u}_b^i \end{bmatrix} \tag{4.2}$$

---

[1] In this chapter, trajectories u are denoted by boldface type to emphasize their vector-like qualities.

where $\mathbf{u}_c^i$ is the ***composite trajectory*** and $\mathbf{u}_b^i$ is the ***basic trajectory***. If the number of observations at level i is $s_i < \infty$, expansion along the time coordinate y ields the representation

$$\mathbf{u}^i = \begin{bmatrix} [\mathbf{u}_c^i(t_k)]_{k:s_i} \\ - - - - - - - \\ [\mathbf{u}_b^i(t_k)]_{k:s_i} \end{bmatrix} = \begin{bmatrix} \mathbf{u}_c^i(t_0) & \mathbf{u}_c^i(t_1) & \cdots & \mathbf{u}_c^i(t_{s_i}) \\ - - - - - - - - - - - - - - - \\ \mathbf{u}_b^i(t_0) & \mathbf{u}_b^i(t_1) & \cdots & \mathbf{u}_b^i(t_{s_i}) \end{bmatrix} = [\, u_j^i(t_k)\,]_{jk:c,\, b \times s_i} \tag{4.3}$$

where $T^i = \{t_0, t_1, \ldots, t_{s_i}\}$. If the composite and basic components are respectively $\mathbf{u}_{c_0}^i, \mathbf{u}_{c_1}^i, \ldots, \mathbf{u}_{c_{r_i}}^i$, and $\mathbf{u}_{b_0}^i, \mathbf{u}_{b_1}^i, \ldots, \mathbf{u}_{b_{\rho_i}}^i$, then expansion along the component coordinate yields:

$$\mathbf{u}^i = \begin{bmatrix} [\mathbf{u}_{c_j}^i]_{j:r_i} \\ - - - \\ [\mathbf{u}_{b_j}^i]_{j:\rho_i} \end{bmatrix} = \begin{bmatrix} \mathbf{u}_{c_0}^i \\ \mathbf{u}_{c_1}^i \\ \cdot \\ \cdot \\ \cdot \\ \mathbf{u}_{c_{r_i}}^i \\ - - - \\ \mathbf{u}_{b_0}^i \\ \mathbf{u}_{b_1}^i \\ \cdot \\ \cdot \\ \cdot \\ \mathbf{u}_{b_{\rho_i}}^i \end{bmatrix} = [\mathbf{u}_j^i]_{j:(r_i + \rho_i + 1)} \,. \tag{4.4}$$

Thus, along both coordinates, the expanded representation is:

$$\mathbf{u}^i = \begin{bmatrix} [\mathbf{u}^i_{c_j}(t_k)]_{jk:r_i \times s_i} \\ - - - \\ [\mathbf{u}^i_{b_j}(t_k)]_{jk:\rho_i \times s_i} \end{bmatrix}$$

$$= \left[\begin{array}{cccc} u^i_{c_0}(t_0) & u^i_{c_0}(t_1) & \cdots & u^i_{c_0}(t_{s_i}) \\ u^i_{c_1}(t_0) & u^i_{c_1}(t_1) & \cdots & u^i_{c_1}(t_{s_i}) \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ u^i_{c_{r_i}}(t_0) & u^i_{c_{r_i}}(t_1) & \cdots & u^i_{c_{r_i}}(t_{s_i}) \\ \hline u^i_{c_0}(t_0) & u^i_{c_0}(t_1) & \cdots & u^i_{c_0}(t_{s_i}) \\ u^i_{c_1}(t_0) & u^i_{c_1}(t_1) & \cdots & u^i_{c_1}(t_{s_i}) \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ u^i_{c_{\rho_i}}(t_0) & u^i_{c_{\rho_i}}(t_1) & \cdots & u^i_{c_{\rho_i}}(t_{s_i}) \end{array}\right] \tag{4.5}$$

$$= [\, u^i_{jk} \,]_{jk:(r_i + \rho_i + 1) \times s_i} \, .$$

A projection along a single time coordinate is referred to as a ***trajectory observation***. Similarly, a projection along a single component will be called a ***component trajectory***. For

instance,

$$\mathbf{u}^i(t_k) = \begin{bmatrix} [\mathbf{u}^i_{c_j}(t_k)]_{j:r_i} \\ - - - \\ [\mathbf{u}^i_{b_j}(t_k)]_{j:\rho_i} \end{bmatrix} = \begin{bmatrix} \mathbf{u}^i_{c_0}(t_k) \\ \mathbf{u}^i_{c_1}(t_k) \\ \cdot \\ \cdot \\ \cdot \\ \mathbf{u}^i_{c_{r_i}}(t_k) \\ - - - - - - \\ \mathbf{u}^i_{b_0}(t_k) \\ \mathbf{u}^i_{b_1}(t_k) \\ \cdot \\ \cdot \\ \cdot \\ \mathbf{u}^i_{b_{\rho_i}}(t_k) \end{bmatrix} = [\, \mathbf{u}^i_j(t_k) \,]_{j:(r_i+\rho_i+1)} \,. \tag{4.6}$$

is a trajectory observation at time t_k, while

$$\mathbf{u}^i_{c_k} = [\mathbf{u}^i_{c_k}(t_k)]_{k:s_i} = \begin{bmatrix} \mathbf{u}^i_{c_j}(t_0) & \mathbf{u}^i_{c_j}(t_1) & \cdots & \mathbf{u}^i_{c_j}(t_{s_i}) \end{bmatrix} \tag{4.7}$$

is a component trajectory of the j^{th} composite component. The interval between the k^{th} and $(k+1)^{\text{th}}$ sample is called the $(k+1)^{\text{th}}$ ***phase***.

To conform with the trajectory set notation of Section 4.1.2 [A Calculus of Trajectory Sets], we shall usually write a composite (basic) variable $\mathbf{u}^i_{c_j}(t_k)$ as $\mathbf{u}^i_{c_j,\,t_k}$ ($\mathbf{u}^i_{b_j}(t_k)$ as $\mathbf{u}^i_{b_j,\,t_k}$).

As an example to clarify this notation and illustrate its use, consider the following. Suppose, at hierarchy level 2, a model with two composite components [flight control system (FCS) and navigation (NAV)] and a single basic component [air traffic control (ATC)] has been constructed. Consider, a trajectory

$$\mathbf{u}^2 = \begin{bmatrix} u^2_{c_0,t_0} & u^2_{c_0,t_1} & u^2_{c_0,t_2} \\ u^2_{c_1,t_0} & u^2_{c_1,t_1} & u^2_{c_1,t_2} \\ - - - - - - - - - \\ u^2_{b_0,t_0} & u^2_{b_0,t_1} & u^2_{b_0,t_2} \end{bmatrix} \begin{matrix} \text{FCS} \\ \text{NAV} \\ \\ \text{ATC} \end{matrix} \tag{4.8}$$

Here, the utilization period involves three samples $T = \{t_0, t_1, t_2\}$. To simplify notation, we shall not always write the level number with every variable of every trajectory when the meaning is clear. Then with the obvious correspondence $c_0 = 1$, $c_1 = 2$, $b_0 = 3$, $t_0 = 1$, $t_1 = 2$, and $t_2 = 3$, Eq. 4.8 is written

$$\mathbf{u}^2 = \begin{bmatrix} u_{00} & u_{01} & u_{02} \\ u_{10} & u_{11} & u_{12} \\ - & - & - \\ u_{20} & u_{21} & u_{22} \end{bmatrix} \begin{matrix} \text{FCS} \\ \text{NAV} \\ \\ \text{ATC} \end{matrix} . \tag{4.9}$$

Here the composite trajectory is

$$\mathbf{u}_c^2 = \begin{bmatrix} u_{00} & u_{01} & u_{02} \\ u_{10} & u_{11} & u_{12} \end{bmatrix} \begin{matrix} \text{FCS} \\ \text{NAV} \end{matrix} \tag{4.10}$$

while the basic trajectory is

$$\mathbf{u}_b^2 = \begin{bmatrix} u_{20} & u_{21} & u_{22} \end{bmatrix} \text{ ATC} , \tag{4.11}$$

where, for example, $\mathbf{u}_{12}$ is the state of the navigation system at the third observation time.

Finally, if $\mathbf{A}$ is the matrix $[a_{jk}]_{r \times s}$, the projection function[2] $\xi_{jk}(\mathbf{A}) = a_{jk}$ is frequently composed with an interlevel translation, e.g., $\xi_{jk}(\kappa_{i+1})$. This is done to extract the particular portion of the function range that is of interest. As an illustration, consider the level-i composite model having r components

$$\mathbf{U}_c^i = \mathbf{U}_{c_0}^i \times \mathbf{U}_{c_1}^i \times \cdots \times \mathbf{U}_{c_{r_i}}^i \tag{4.12}$$

$$= [\mathbf{U}_{c_j}^i]_{j:r} , \tag{4.13}$$

where the $\mathbf{U}_{c_j}^i$ can be further coordinatized along time, $\mathbf{U}_{c_j}^i(t_k) = U_{c_j, t_k}^i$, where $t_k \in T^i$.

---

[2]Let $\mathbf{A} = \underset{i \in I}{\times} A_i$ be a Cartesian product indexed by the set I. The ***projection of A on i*** is the function $\xi_i : \mathbf{A} \rightarrow A_i$ where for $\mathbf{a} \in \mathbf{A}$ and $a_i \in A_i$, $\xi_i(\mathbf{a}) = a_i$.

That is,

$$\mathbf{U}_c^i = [U_{c_j,t_k}^i]_{jk:r_i \times s_i} . \tag{4.14}$$

Using the projection function, $\mathbf{U}_{c_j}^i(t_k)$ will be written $\xi_{jk}\mathbf{U}_c^i$. The interlevel translation from level-i to level-$(i-1)$ (assuming level-i exists and $i > 0$) is (see Eqs. 3.15-3.16)

$$\kappa_i : \mathbf{U}^i \rightarrow \mathbf{U}_c^i \tag{4.15}$$

where $\mathbf{U}^i = \mathbf{U}_c{}^i \times \mathbf{U}_b{}^i$. To select the function mapping $\mathbf{U}^i$ into $\mathbf{U}_{c_j}^{i-1}(t_k) = \xi_{jk}\mathbf{U}_c^{i-1}$, we write

$$\xi_{jk}\kappa_i : \mathbf{U}^i \rightarrow \xi_{jk}\mathbf{U}_c^{i-1} . \tag{4.16}$$

We shall refer to $\xi_{jk}\kappa_i$ as the j^{th} ***component interlevel translation at observation k.***

4.1.2. A Calculus of Trajectory Sets

It is convenient now to introduce a calculus that is of great use in determining the γ-induced trajectory sets, i.e., γ^{-1}. This calculus can be used to simplify the upper level models of the hierarchy before any lower level models are examined. Also, the calculus is used to assimilate lower levels as they are developed. After the lowest level of the hierarchy has been operated upon, the result is γ^{-1}. Derivation of γ^{-1} is important because, using the techniques of Section 3.3.6 [Solving for the Performability] on γ^{-1}, performability calculations for the system can be effected.

Manipulation of sets of trajectories is necessary to derive the preimage sets of γ. However, handling such sets can be awkward because of their size. Therefore, we have investigated techniques of manually operating with sets of trajectories in a convenient and compact manner.

A set of trajectories will be called a ***trajectory set.*** We first introduce a simple

representation of a trajectory set. Consider the trajectory

$$\mathbf{u} = [u_{jk}]_{r \times s} = \begin{bmatrix} u_{00} & u_{01} & \cdots & u_{0s} \\ u_{10} & u_{11} & \cdots & u_{1s} \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ u_{r0} & u_{r1} & \cdots & u_{rs} \end{bmatrix} \tag{4.17}$$

where each u_{jk} can assume values in a set of states Q_{jk} . Each u_{jk} is a "variable." For example, we may have

$$\mathbf{u} = [u_{jk}]_{1 \times 1} = \begin{bmatrix} u_{00} & u_{01} \\ u_{10} & u_{11} \end{bmatrix} \tag{4.18}$$

where $u_{00} \in \mathbf{Q}_{00} = \{1, 2, 3\}$, $u_{01} \in \mathbf{Q}_{01} = \{0, 2, 4\}$, $u_{10} \in \mathbf{Q}_{10} = \{-1, -2\}$, and $u_{11} \in \mathbf{Q}_{11} = \{a, b, c\}$. Suppose we have two trajectories $\mathbf{u}_1$ and $\mathbf{u}_2$ such that $\mathbf{u}_1$ and $\mathbf{u}_2$ are equal variable-by-variable except for a single variable. That is,

$$\mathbf{u}_0 = \begin{bmatrix} u_{00} & \cdots & u_{0n} \\ \cdots & u_{jk} & \cdots \\ u_{r0} & \cdots & u_{rs} \end{bmatrix} \tag{4.19}$$

$$u_1 = \begin{bmatrix} u_{00} & \cdots & u_{0n} \\ \cdots & u_{jk}' & \cdots \\ u_{r0} & \cdots & u_{rs} \end{bmatrix} \tag{4.20}$$

where $u_{jk} \neq u_{jk}'$. We then write trajectory set $\{\mathbf{u}_1, \mathbf{u}_2\}$ as

$$\{\mathbf{u}_0, \mathbf{u}_1\} = \left\{ \begin{bmatrix} u_{00} & \cdots & u_{0n} \\ \cdots & u_{jk} & \cdots \\ u_{r0} & \cdots & u_{rs} \end{bmatrix}, \begin{bmatrix} u_{00} & \cdots & u_{0n} \\ \cdots & u_{jk}' & \cdots \\ u_{r0} & \cdots & u_{rs} \end{bmatrix} \right\} \tag{4.21}$$

$$= \begin{bmatrix} \{u_{00}\} & \cdots & \{u_{0n}\} \\ \cdots & \{u_{jk}, u_{jk}'\} & \cdots \\ \{u_{r0}\} & \cdots & \{u_{rs}\} \end{bmatrix} \tag{4.22}$$

This representation is called an ***array product***. Of course, all array products are trajectory sets. Frequently, when the trajectory in which we are interested can be written as an array product, we shall use the term trajectory set and array product synonymously. Note that the concept is similar to that of a cross product. Of course, the idea can be generalized:

$$\begin{bmatrix} R_{00} & R_{01} & \cdots & R_{0n} \\ R_{10} & R_{11} & \cdots & R_{1n} \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ R_{r0} & R_{r1} & \cdots & R_{rs} \end{bmatrix} = \left\{ \begin{bmatrix} u_{00} & u_{01} & \cdots & u_{0n} \\ u_{10} & u_{11} & \cdots & u_{1n} \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ u_{r0} & u_{r1} & \cdots & u_{rs} \end{bmatrix} \middle| \; u_{jk} \in R_{jk} \subseteq Q_{jk} \right\} . \tag{4.23}$$

As an illustration, suppose that

$$\mathbf{u}_0 = \begin{bmatrix} 1 & 2 \\ -2 & a \end{bmatrix} \quad \mathbf{u}_1 = \begin{bmatrix} 3 & 2 \\ -2 & a \end{bmatrix} \tag{4.24}$$

$$\mathbf{u}_2 = \begin{bmatrix} 1 & 2 \\ -2 & b \end{bmatrix} \quad \mathbf{u}_3 = \begin{bmatrix} 3 & 2 \\ -2 & b \end{bmatrix} , \tag{4.25}$$

then

$$\{\mathbf{u}_0, \mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3\} = \begin{bmatrix} \{1,3\} & \{2\} \\ \{-2\} & \{a,b\} \end{bmatrix} . \tag{4.26}$$

Because the use of array products has been so widespread in our work with trajectory sets, we have adopted the simplifying convention of writing array product elements that are singleton sets as elements without set brackets. Thus Eq. 4.26 can be written

$$\{\mathbf{u}_0, \mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3\} = \begin{bmatrix} \{1,3\} & 2 \\ -2 & \{a,b\} \end{bmatrix} \tag{4.27}$$

No confusion should result since context will make clear whether an object is an array product (and hence a set) or a single trajectory. Furthermore, this convention makes array products easier to read by cutting down on the number of brackets through which a reader must

wade.

Often a single array product cannot by itself represent all the trajectories within a single set. In that instance, the union of several array products must be employed to represent the trajectory set. Thus, for the general case, we write a trajectory set as the union of p array products $\mathbf{P}_i$:

$$\{\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_r\} = \mathbf{P}_0 \cup \mathbf{P}_1 \cup \cdots \cup \mathbf{P}_p \tag{4.28}$$

$$= [R_{jk}^0]_{r \times s} \cup [R_{jk}^1]_{r \times s} \cup \cdots \cup [R_{jk}^p]_{r \times s} \tag{4.29}$$

$$= \begin{bmatrix} R_{00}^0 & R_{01}^0 & \cdots & R_{0n}^0 \\ R_{10}^0 & R_{11}^0 & \cdots & R_{1n}^s \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ R_{r0}^0 & R_{r1}^0 & \cdots & R_{rs}^0 \end{bmatrix} \cup \begin{bmatrix} R_{00}^1 & R_{01}^1 & \cdots & R_{0n}^1 \\ R_{10}^1 & R_{11}^1 & \cdots & R_{1n}^s \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ R_{r0}^1 & R_{r1}^1 & \cdots & R_{rs}^1 \end{bmatrix} \cup \cdots \cup \begin{bmatrix} R_{00}^p & R_{01}^p & \cdots & R_{0n}^p \\ R_{10}^p & R_{11}^p & \cdots & R_{1n}^p \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ R_{r0}^p & R_{r1}^p & \cdots & R_{rs}^p \end{bmatrix} \tag{4.30}$$

$$= \left\{ \begin{bmatrix} u_{00}^l & u_{01}^l & \cdots & u_{0n}^l \\ u_{10}^l & u_{11}^l & \cdots & u_{1n}^l \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ u_{r0}^l & u_{r1}^l & \cdots & u_{rs}^l \end{bmatrix} \middle| \begin{array}{c} l \in \{0, 1, \ldots, p\}, \\ u_{jk}^l \in R_{jk}^l \subseteq Q_{jk} \end{array} \right\} \tag{4.31}$$

For example, in addition to $\mathbf{u}_0$, $\mathbf{u}_1$, $\mathbf{u}_2$, and $\mathbf{u}_3$ of Eq 4.24 above, let

$$\mathbf{u}_4 = \begin{bmatrix} 3 & 0 \\ -2 & a \end{bmatrix} \quad \mathbf{u}_5 = \begin{bmatrix} 3 & 0 \\ -2 & b \end{bmatrix} \tag{4.32}$$

Then

$$\{u_0, u_1, u_2, u_3, u_4, u_5\} = \begin{bmatrix} \{1,3\} & 2 \\ -2 & \{a,b\} \end{bmatrix} \cup \begin{bmatrix} 3 & 0 \\ -2 & \{a,b\} \end{bmatrix} . \tag{4.33}$$

In passing, note that this representation is not unique, e.g., the set in Eq. 4.33 above can also be written

$$\{u_0, u_1, u_2, u_3, u_4, u_5,\} = \begin{bmatrix} 3 & \{0,2\} \\ -2 & \{a,b\} \end{bmatrix} \cup \begin{bmatrix} 1 & 2 \\ -2 & \{a,b\} \end{bmatrix} . \tag{4.34}$$

A canonical form can be easily defined. For instance, the elements of the arrays can all be required to be singleton sets. Under this constraint, the representations of a given trajectory set are identical up to the ordering of the arrays. However, such canonical forms are of little practical value since the number of arrays quickly becomes large.

Two special sets should be mentioned. One is the ***empty set*** (or null set) ϕ, the set containing no elements. The other is the ***full set*** (or universe) * that represents the set containing all elements "of interest." For trajectory sets, this is the set of all possible states a variable can assume. For instance, if

$$u_6 = \begin{bmatrix} 1 & 2 \\ -1 & a \end{bmatrix} , \tag{4.35}$$

then

$$\{u_0, u_6\} = \begin{bmatrix} 1 & 2 \\ * & a \end{bmatrix} . \tag{4.36}$$

Another frequently used quantity is the ***null array*** Φ. This is defined to be any array product that contains the empty set ϕ as an element. As an instance,

$$\Phi = \begin{bmatrix} \{1,2\} & \phi \\ * & \{a,b\} \end{bmatrix} . \tag{4.37}$$

A second symbol, $\not\phi$, having functional properties identical to *, is sometimes employed; see

Section 3.3.7 [Constructing Performability Models].

We now define the operation of intersection on the class of array products. The intersection $\cap$ of two array products $\mathbf{P}_0$ and $\mathbf{P}_1$ is the element-by-element intersection of the two arrays. $\mathbf{P}_0$ and $\mathbf{P}_1$ must have the same dimensions.

$$\mathbf{P}_0 \cap \mathbf{P}_1 = [R^0_{jk}]_{r \times s} \cap [R^1_{jk}]_{r \times s} \tag{4.38}$$

$$= \begin{bmatrix} R^0_{00} & R^0_{01} & \cdots & R^0_{0n} \\ R^0_{10} & R^0_{11} & \cdots & R^0_{1n} \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ R^0_{r0} & R^0_{r1} & \cdots & R^0_{rs} \end{bmatrix} \cap \begin{bmatrix} R^1_{00} & R^1_{01} & \cdots & R^1_{0n} \\ R^1_{10} & R^1_{11} & \cdots & R^1_{1n} \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ R^1_{r0} & R^1_{r1} & \cdots & R^1_{rs} \end{bmatrix} \tag{4.39}$$

$$= \begin{bmatrix} R^0_{00} \cap R^1_{00} & R^0_{01} \cap R^1_{01} & \cdots & R^0_{0n} \cap R^1_{0n} \\ R^0_{10} \cap R^1_{10} & R^0_{11} \cap R^1_{11} & \cdots & R^0_{1n} \cap R^1_{1n} \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ R^0_{r0} \cap R^1_{r0} & R^0_{r1} \cap R^1_{r1} & \cdots & R^0_{rs} \cap R^1_{rs} \end{bmatrix} \tag{4.40}$$

$$= [R^0_{jk} \cap R^1_{jk}]_{r \times s} \; . \tag{4.41}$$

The following table defines the element intersection $R^0_{jk} \cap R^1_{jk}$:

	a_0	ϕ	*	¢
a_1	$a_0 \cap a_1$	ϕ	a_1	a_1
ϕ	ϕ	ϕ	ϕ	ϕ
*	a_0	ϕ	*	*
¢	a_0	ϕ	*	¢

where a_0 and a_1 are any sets and $a_0 \cap a_1$ is standard set intersection. Thus,

$$\begin{bmatrix} \{1,2\} & \phi \\ * & \{a,b\} \end{bmatrix} \cap \begin{bmatrix} \{1,3\} & \{0,2\} \\ -2 & ¢ \end{bmatrix} = \begin{bmatrix} \{1,2\} \cap \{1,3\} & \phi \cap \{0,2\} \\ * \cap -2 & \{a,b\} \cap ¢ \end{bmatrix} \tag{4.42}$$

$$= \begin{bmatrix} 1 & \phi \\ -2 & \{a,b\} \end{bmatrix} \tag{4.43}$$

Array product intersection is distributive over set union. For instance:

$$\begin{bmatrix} \{1,2\} & \phi \\ * & \{a,b\} \end{bmatrix} \cap \left(\begin{bmatrix} \{1,2\} & 0 \\ -2 & \{b,c\} \end{bmatrix} \cup \begin{bmatrix} \{1,3\} & \{0,2\} \\ -2 & ¢ \end{bmatrix} \right) \tag{4.44}$$

$$\begin{aligned} = & \left(\begin{bmatrix} \{1,2\} & \phi \\ * & \{a,b\} \end{bmatrix} \cap \begin{bmatrix} \{1,2\} & 0 \\ -2 & \{b,c\} \end{bmatrix} \right) \\ & \cup \left(\begin{bmatrix} \{1,2\} & \phi \\ * & \{a,b\} \end{bmatrix} \cap \begin{bmatrix} \{1,3\} & \{0,2\} \\ -2 & ¢ \end{bmatrix} \right) \end{aligned} \tag{4.45}$$

$$= \begin{bmatrix} \{1,2\} & \phi \\ -2 & b \end{bmatrix} \cap \begin{bmatrix} 1 & \phi \\ -2 & \{b,c\} \end{bmatrix} \tag{4.46}$$

$$= \Phi \cap \Phi \tag{4.47}$$

$$= \Phi \tag{4.48}$$

The complement $\mathbf{P}^c$ of an array product $\mathbf{P}$ is the set of all arrays not represented by $\mathbf{P}$. This can be found as follows:

$$\mathbf{P}^c = \begin{bmatrix} R_{00} & \cdots & R_{0n} \\ R_{10} & \cdots & R_{1n} \\ \cdot & & \cdot \\ \cdot & & \cdot \\ \cdot & & \cdot \\ R_{r0} & \cdots & R_{rs} \end{bmatrix}^c \tag{4.49}$$

$$= \begin{bmatrix} R_{00}^c & \cdots & * \\ * & \cdots & * \\ \cdot & & \cdot \\ \cdot & & \cdot \\ \cdot & & \cdot \\ * & \cdots & * \end{bmatrix} \cup \begin{bmatrix} * & \cdots & * \\ R_{01}^c & \cdots & * \\ \cdot & & \cdot \\ \cdot & & \cdot \\ \cdot & & \cdot \\ * & \cdots & * \end{bmatrix} \cup \cdots \cup \begin{bmatrix} * & \cdots & R_{0n}^c \\ \cdot & & \cdot \\ \cdot & & \cdot \\ \cdot & & \cdot \\ * & \cdots & * \end{bmatrix}$$

$$\cup \cdots \cup \begin{bmatrix} * & \cdots & * \\ * & \cdots & * \\ \cdot & & \cdot \\ \cdot & & \cdot \\ \cdot & & \cdot \\ R_{r0}^c & \cdots & * \end{bmatrix} \cup \cdots \cup \begin{bmatrix} * & \cdots & * \\ * & \cdots & * \\ \cdot & & \cdot \\ \cdot & & \cdot \\ \cdot & & \cdot \\ * & \cdots & R_{rs}^c \end{bmatrix} \tag{4.50}$$

$$= \bigcup_{a=0}^{r} \bigcup_{b=0}^{s} \begin{bmatrix} \hat{R}_{00} & \cdots & \hat{R}_{0n} \\ \cdots & \hat{R}_{jk} & \cdots \\ \hat{R}_{r0} & \cdots & \hat{R}_{rs} \end{bmatrix} \tag{4.51}$$

where

$$\hat{R}_{jk} = \begin{cases} R_{jk}^c & \text{if } a = j,\ b = k, \\ * & \text{otherwise .} \end{cases} \tag{4.52}$$

$R_{jk} \subseteq Q_{jk}$, and $R_{jk}^c = \{q \mid q \in Q_{jk} \text{ and } q \notin R_{jk}\}$. Also, $*^c = \phi$, $\phi^c = *$, and $\notin^c = \phi$.

To determine the complement of a trajectory set, De Morgan's Law can be used. Suppose V is a trajectory set composed of p array products. Then

$$V^c = (\mathbf{P}_1 \cup \mathbf{P}_2 \cup \cdots \cup \mathbf{P}_p)^c \tag{4.53}$$

$$= \mathbf{P}_1^c \cap \mathbf{P}_2^c \cap \cdots \cap \mathbf{P}_p^c \ . \tag{4.54}$$

As an example,

$$\left(\begin{bmatrix} \{1,3\} & 2 \\ -2 & \{a,b\} \end{bmatrix} \cup \begin{bmatrix} \{1,2\} & 2 \\ -1 & * \end{bmatrix} \right)^c = \begin{bmatrix} \{1,3\} & 2 \\ -2 & \{a,b\} \end{bmatrix}^c \cap \begin{bmatrix} \{1,2\} & 2 \\ -1 & * \end{bmatrix}^c \tag{4.55}$$

$$\begin{aligned} &= \left(\begin{bmatrix} 2 & * \\ * & * \end{bmatrix} \cup \begin{bmatrix} * & \{0,4\} \\ * & * \end{bmatrix} \cup \begin{bmatrix} * & * \\ -1 & * \end{bmatrix} \cup \begin{bmatrix} * & * \\ * & c \end{bmatrix} \right) \\ &\cap \left(\begin{bmatrix} 3 & * \\ * & * \end{bmatrix} \cup \begin{bmatrix} * & \{0,4\} \\ * & * \end{bmatrix} \cup \begin{bmatrix} * & * \\ -2 & * \end{bmatrix} \cup \begin{bmatrix} * & * \\ * & \phi \end{bmatrix} \right) \end{aligned} \tag{4.56}$$

$$\begin{aligned} &= \begin{bmatrix} 2 & \{0,4\} \\ * & * \end{bmatrix} \cup \begin{bmatrix} 2 & * \\ -2 & * \end{bmatrix} \cup \begin{bmatrix} 3 & \{0,4\} \\ * & * \end{bmatrix} \cup \begin{bmatrix} * & \{0,4\} \\ * & * \end{bmatrix} \\ &\cup \begin{bmatrix} * & \{0,4\} \\ -2 & * \end{bmatrix} \cup \begin{bmatrix} 3 & * \\ -1 & * \end{bmatrix} \cup \begin{bmatrix} * & \{0,4\} \\ -1 & * \end{bmatrix} \cup \begin{bmatrix} 3 & * \\ * & c \end{bmatrix} \\ &\cup \begin{bmatrix} * & \{0,4\} \\ * & c \end{bmatrix} \cup \begin{bmatrix} * & * \\ -2 & c \end{bmatrix} . \end{aligned} \tag{4.57}$$

We have found, however, that the evaluation of performability ($\Pr(\gamma^{-1}(a))$) is often simpler if the γ-induced trajectory sets ($\gamma^{-1}(a)$) are represented as the union of disjoint sets.

The set of Eq. 4.55 above, for instance, can be written

$$\left(\begin{bmatrix} \{1,3\} & 2 \\ -2 & \{a,b\} \end{bmatrix} \cup \begin{bmatrix} \{1,2\} & 2 \\ -1 & * \end{bmatrix} \right)^c \tag{4.58}$$

$$= \begin{bmatrix} * & \{0,4\} \\ * & * \end{bmatrix} \cup \begin{bmatrix} 2 & 2 \\ -2 & * \end{bmatrix} \cup \begin{bmatrix} 3 & 2 \\ -1 & * \end{bmatrix} \cup \begin{bmatrix} \{1,3\} & 2 \\ -2 & c \end{bmatrix} .$$

Representing a trajectory set as a union of disjoint array products has analogies with representing a Boolean function in disjunctive form. Thus, we see the possibility of employing a generalized version of Roth's cubical calculus (see [177] for instance) to handle sets other than $\{0, 1\}$ and so manipulate trajectory sets. The next section discusses such an approach.

4.2. Discrete Functions and the Representation of Trajectory Sets

Many of the algorithms implemented in METAPHOR manipulate trajectory sets. These algorithms are described in Section 4.3.3 [Algorithms]. To state the algorithms concisely, we require a compact notation for denoting trajectory sets and the discrete functions which relate them. This section discusses discrete functions and the representation of trajectory sets. Much of the notation and development of this section is strongly influenced by Chapters 2, 3, and 8 of Davio, Deschamps, and Thayse [165].

4.2.1. Discrete Functions

A function

$$f: U \rightarrow A \tag{4.59}$$

is a ***discrete function*** when U and A are finite, nonempty sets. Function f is an ***integer function*** when the elements of U and A are non-negative integers. For our applications, we will usually deal with sets that do **not** contain integers. However, the representation of these

sets within METAPHOR is based on integers. Hence, though METAPHOR internally deals exclusively with integer functions, much of the theory below is based on discrete functions. A set U will often be decomposed into a Cartesian product

$$U = U_0 \times U_1 \times \cdots \times U_r = \bigtimes_{j=0}^{r} U_j . \tag{4.60}$$

In such cases, U will usually be written in boldface type (**U**) to emphasize its vector nature. **U** will often be further decomposed using two indexes:

$$\begin{aligned} \mathbf{U} = {} & U_{00} \times U_{01} \times \cdots \times U_{0s} \times U_{10} \times U_{11} \times \\ & \cdots \times U_{1s} \times \cdots \times U_{r0} \times U_{r1} \times \cdots \times U_{rs} \end{aligned} \tag{4.61}$$

$$= \bigtimes_{j=0}^{r} \bigtimes_{k=0}^{s} U_{jk} . \tag{4.62}$$

One may visualize such a product **U** as a "matrix"

$$\mathbf{U} = [U_{jk}]_{r \times s} = \begin{bmatrix} U_{00} & U_{01} & \cdots & U_{0s} \\ U_{10} & U_{01} & \cdots & U_{1s} \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ U_{r0} & U_{r1} & \cdots & U_{rs} \end{bmatrix} . \tag{4.63}$$

Indeed, the description of the "trajectory set calculus" of [30] and of Section 4.1.2 [A Calculus of Trajectory Sets] utilizes such a matrix construction.

We shall deal with functions

$$f: \mathbf{U} \rightarrow \bigtimes_{j=0}^{r} A_j \tag{4.64}$$

whose domains are Cartesian products. Such functions are referred to as ***general discrete***

functions. Such functions can be decomposed into a set of $r+1$ discrete functions

$$f_j: \mathbf{U} \rightarrow A_j, \quad 0 \leq k \leq r \, . \tag{4.65}$$

The function f_j is the $\mathbf{j^{th}}$ ***projection of*** f and is often written $\xi_j f$ (see Section 4.1 [Trajectory Sets: Basic Notation and Operations]).

Since the domain $\mathbf{U}$ or range A of a discrete function f are finite, one can describe the function by either exhaustively enumerating all the values $f(\mathbf{u}) \in A$ for all $\mathbf{u} \in \mathbf{U}$, or alternatively, by considering each $a \in A$ and enumerating all values $\mathbf{u} \in \mathbf{U}$ for which $f(\mathbf{u}) = a$, i.e., by enumerating $f^{-1}(a)$. Both approaches will be used. Section 4.2.2 [Discrete Functions and Capability Functions] below deals more fully with representations of discrete functions. First, let us review the motivation for our studying discrete functions.

4.2.2. Discrete Functions and Capability Functions

In the framework of discrete performability evaluations, the sets of concern are a set A (accomplishment set) of objects called ***accomplishment levels*** and sets U (trajectory spaces) of objects called ***trajectories***. The trajectory spaces will usually be written as vectors or matrices, and to emphasize this, U will usually be written in boldface type. Several trajectory spaces $\mathbf{U}_b^0$, $\mathbf{U}_b^1, \ldots, \mathbf{U}_b^m$ and $\mathbf{U}_c^0$, $\mathbf{U}_c^1, \ldots, \mathbf{U}_c^m$ may be of concern. $\mathbf{U}_c^i$ ($\mathbf{U}_b^i$) is the ***level-i composite (basic) trajectory space*** (see Section 3.3.8 [The Model Hierarchy]). The Cartesian product $\mathbf{U}^i = \mathbf{U}_c^i \times \mathbf{U}_b^i$ is the ***level-i trajectory space***. Regarding Cartesian products in this section, make the convention that if a trajectory space $\mathbf{U}$ is empty ($\mathbf{U}_c^m$ is always ϕ and, for $0 \leq i \leq m$, $\mathbf{U}_b^i$ may or may not be empty), then any reference to $\mathbf{U}$ in a Cartesian product is deleted. Thus, if either $\mathbf{U}_c^i$ or $\mathbf{U}_b^i$ is empty, then $\mathbf{U}^i$ is the other, non-empty, set.

The following conditions hold:

1) $\mathbf{U}_c^m = \phi$

2) For all $0 \leq i \leq m$, $\mathbf{U}_c^i \neq \phi$ and $\mathbf{U}_b^i$ may or may not be empty. The product of all basic

trajectory spaces of level-i and below

$$\overline{\mathbf{U}}_b^i = \mathbf{U}_b^0 \times \mathbf{U}_b^1 \times \cdots \times \mathbf{U}_b^i \tag{4.66}$$

$$= \bigtimes_{j=0}^{i} \mathbf{U}_b^j \tag{4.67}$$

is the ***level-i basic model trajectory space*** (Section 3.3.8 [The Model Hierarchy]). $\overline{\mathbf{U}}_b^m$ is called the ***basic model trajectory space.***

Note that $\overline{\mathbf{U}}_b^i$ can be defined recursively as follows:

$$\begin{aligned} \overline{\mathbf{U}}_b^0 &= \mathbf{U}_b^0 \\ \overline{\mathbf{U}}_b^i &= \mathbf{U}_b^i \times \overline{\mathbf{U}}_b^{i-1}, \quad m \leq i < 0 . \end{aligned} \tag{4.68}$$

Any composite (basic) trajectory space $\mathbf{U}_c^i$ ($\mathbf{U}_b^i$) may be decomposed

$$\mathbf{U}_c^i = \bigtimes_{j=0}^{m} \bigtimes_{k=0}^{n} U_{c,jk}^i \tag{4.69}$$

$$\left(\mathbf{U}_b^i = \bigtimes_{j=0}^{m} \bigtimes_{k=0}^{n} U_{b,jk}^i \right) . \tag{4.70}$$

The values of indexes i and j delineate ***components*** and ***phases***, respectively. The set $U_{c,jk}^i$ ($U_{c,jk}^i$) is the ***level-i composite (basic) trajectory space of component j during phase k.***

Two types of functions interest us. The first type consists of functions of the form

$$\gamma_i : \mathbf{U}^i \times \overline{\mathbf{U}}_b^{i-1} \rightarrow A , \quad 0 \leq i \leq m . \tag{4.71}$$

That is, γ_i is a function whose domain is the Cartesian product of the level-i trajectory space with all non-empty basic trajectory spaces between levels 0 and $i-1$, and whose codomain is A. γ_i the ***level-i based capability function.*** The function γ_m is written γ and is called the

capability function.

The second type of function is of the form

$$\kappa_i: \mathbf{U}^i \rightarrow \mathbf{U}_c^{i-1}, 1 \leq i < m . \tag{4.72}$$

κ_i is the ***level-i interlevel translation*** and maps level-i trajectories into level-$(i-1)$ composite trajectories. κ_0 is identified with γ_0. Given a level-$(i-1)^{\text{th}}$ component j and a level-$(i-1)^{\text{th}}$ phase k, the projection

$$\xi_{jk}\kappa_i: \mathbf{U}^i \rightarrow U_{c,jk}^{i-1} \tag{4.73}$$

is the function mapping level-i trajectories into the level-$(i-1)$ jk^{th} composite trajectory space.

The relation between the level-i capability function γ_i and the interlevel translations $\kappa_0, \ldots, \kappa_i$ for levels 0 through i is as follows: Let

$$\mathbf{u}'_i \in \overline{\mathbf{U}}_b^i = \bigtimes_{j=0}^{i} \mathbf{U}_b^j . \tag{4.74}$$

That is $\mathbf{u}'_i = (\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_i)$ where, for $i \geq j \geq 0$, $\mathbf{u}_j \in \mathbf{U}_b^j$. Then

$$\gamma_i(\overline{\mathbf{u}}_i) = \kappa_0(\cdots \kappa_{i-1}(\kappa_i(\mathbf{u}_i), \mathbf{u}_{i-1}), \cdots, \mathbf{u}_0) . \tag{4.75}$$

Recursively, for

$$\mathbf{u}'_i \in \overline{\mathbf{U}}_b^i = \mathbf{U}_b^i \times \overline{\mathbf{U}}_b^{i-1} , \tag{4.76}$$

$\mathbf{u}'_i = (\mathbf{u}'_{i-1}, \mathbf{u}_i)$ where $\mathbf{u}'_1 \in \overline{\mathbf{U}}_b^{i-1}$, and

$$\gamma_i(\mathbf{u}'_i) = \gamma_{i-1}(\kappa_i(\mathbf{u}_i), \mathbf{u}'_{i-1}) . \tag{4.77}$$

Thus, if the κ_i are known for all $0 \leq i \leq m$, then one can recursively specify the γ_i. This specification is performed by listing for each $a \in A$, the values $\gamma^{-1}(a)$. Hence, for $a \in A$, (see

Eq. 3.23)

$$\gamma_0^{-1}(a) = \kappa_0^{-1}(a)$$

$$\gamma_i^{-1}(a) = \bigcup_{(u'_{i-1}, u_i) \in \gamma_{i-1}^{-1}(a)} (\kappa_i^{-1}(u_i), u'_{i-1}) \ . \tag{4.78}$$

A tractable implementation of the above equation is the reason for the development of the trajectory manipulation algorithms of METAPHOR. (See the discussion of Section 3.4.3 [Algorithms for Calculating Trajectory Sets].)

4.2.3. Alternative Representations of Trajectory Sets

We wish to be able to represent or denote specific discrete functions using notation more compact and general than trajectory sets (Section 4.1 [Trajectory Sets: Basic Notation and Operations]). With such machinery, we can discuss the algorithms used to manipulate discrete functions and we can prove some useful properties, e.g., that trajectory sets can represent all discrete functions. Many methods of representing a discrete function are known. Section 4.2.3.1 [Some Tabular Representations] discusses some basic graphical descriptions and Section 4.2.3.2 [Representations Using Lattice Expressions] describes and algebraic representation using the lattice operations disjunction and conjunction. The former method gives insight into discrete functions, while the latter is the basis for the algorithms in METAPHOR. Other representation techniques, such as those using the ring operations ring sum and ring product and those techniques using polynomials over a Galois field (see Davio, Deschampes, and Thayse [165] for a discussion of these topics) will not be discussed here.

Consider the representation of a discrete function

$$f: U \rightarrow A \tag{4.79}$$

where $|U| = N + 1$ and $|A| = l + 1$. There exists a one-to-one mapping (enumeration) of U to the integer sequence $0, 1, \ldots, N$. Assume, without loss of generality, that

$U = \{0, 1, \ldots, N\}$. Similarly, take $A = \{0, 1, \ldots, l\}$. As a special case, if $U = \{0, 1\}^M$ for some $M < \infty$ and $A = \{0, 1\}$, then $f:\{0, 1\}^M \rightarrow \{0, 1\}$ is a ***switching function***. The properties and applications of switching functions are well understood (see, e.g., Miller [177], Kohavi [178], or Preparata and Yeh [179]).

4.2.3.1. Some Tabular Representations

The simplest representations of discrete function are the tabular representations. Of these, the most elementary is a vector that enumerates the entire function; the vector is called the ***value vector of f***

$$[f(0)\, f(1) \cdots f(N)] = [f(u)]_N \text{ or } [f_u]_N, \quad u = 0, 1, \ldots, N . \tag{4.80}$$

The number of such functions is l^N. If {U} is a product[3]

$$\mathbf{U} = \bigtimes_{j=0}^{d} U_j \tag{4.81}$$

where the number of entries is

$$N = \prod_{j=0}^{r} \left| U_j \right| , \tag{4.82}$$

then the function f can still be expressed as a value vector

$$[f(\mathbf{u})]_N \text{ or } [f_{\mathbf{u}}]_N, \quad \mathbf{u} \in \bigtimes_{j=0}^{r} U_j . \tag{4.83}$$

---

[3]The development of the material in this section usually employs a single dimensional product $\mathbf{U} = \bigtimes_{j=0}^{r} U_j$ (see [165], for example). However, in the context of METAPHOR, U_j is itself a one dimensional quantity: $\mathbf{U} = \bigtimes_{j=0}^{r} \bigtimes_{k=0}^{s} U_{jk}$. More generally, products of such two dimensional trajectory spaces are often of concern, i.e., $\mathbf{U} = \bigtimes_{i=0}^{m} \left[\bigtimes_{j=0}^{r_j} \bigtimes_{k=0}^{s_j} U_{jk} \right]$. In the following exposition of the theory, the single dimension notation will be employed because it is easier to grasp. However, our applications will usually require more complex dimensioning.

There are $(l + 1)^{N+1}$ such functions. The order in which the f_u appear in the vector $[f_u]_N$ is arbitrary so long as the order follows some well specified convention. Convenient orderings include lexicographical order and Gray code order. $[f_u]_N$ can be written in tabular form. If the ordering of the u is lexicographical, such a table is called a ***Veitch chart*** [180]; if the ordering is the Gray code, the table is called a ***Karnaugh chart*** [181]. For our purposes, none of the above techniques are appropriate for direct computer implementation. Instead, we shall employ an algebraic approach based on the lattice structure of discrete functions.

4.2.3.2. Representations Using Lattice Expressions

4.2.3.2.1. Lattice Polynomials

The concepts of cubical representation and cubical notation for a switching function [177] will now be extended to discrete functions of r variables. This extension is not straightforward, however. To understand why, consider the lattice[4] theoretic basis of switching theory. In the more general development of lattice theory, (see Grätzer [182], for instance), one usually 1) defines the concept of r-ary lattice polynomials,[5] 2) shows that, for any distributive lattice A over which the variables take values, there is an obvious equivalence relation among the polynomials,[6] and 3) shows that the equivalence classes form a distributive lattice $\mathbf{P}^r$.[7] This lattice has interesting properties, but does not denote all the functions from A^r to A (much less from $\mathbf{U}^A$), a feature we must have.

Suppose **B** is a Boolean lattice.[8] Then, using the same approach as above, one 1) defines

---

[4] All lattices discussed in this chapter are presumed to be finite.

[5] Informally, a ***lattice polynomial*** is an expression (well-formed formula, see Section 4.2.3.2.4 [Lattice Expressions]) using $\wedge$, $\vee$, variables $x_0, x_1, \ldots, x_{r-1}$, and parenthesis. For instance, x_0, $(x_0 \wedge x_1)$, and $(x_1 \vee x_3) \wedge (x_2 \wedge x_0)$ are lattice polynomials

[6] Two polynomials are ***equivalent*** if they represent the same function.

[7] $\mathbf{P}^r$ is the free distributive lattice on $r+1$ generators. $\mathbf{P}_r$ is finite.

[8] A ***Boolean lattice*** is a distributive and complemented lattice A ***complemented lattice*** is one in which every element has at least one ***complement***, i.e., for each element a, there is an $\bar{a}$ such that $a \wedge \bar{a}$ is the least element and $a \vee \bar{a}$ is the greatest element. A finite, distributed lattice **B** is complemented (and hence Boolean) if and only if **B** contains 2^r elements, $r \geq 0$.

the concept of "r-ary Boolean polynomials,"[9] 2) shows that for the Boolean lattice **B** there is an equivalence relation among the polynomials, and 3) shows that the equivalence classes form a Boolean lattice $\mathbf{B}_{2^r}$.[10] Further, if **B** is the two element Boolean lattice $\{0, 1\}$, then every discrete function $f:\{0, 1\}^r \rightarrow \{0, 1\}$ is represented by one of the elements of $\mathbf{B}_{2^r}$. Hence, one can understand the reason for the use of boolean lattices in switching algebra, namely, one can compactly denote and manipulate switching functions on the lattice $\mathbf{B}_{2^r}$. However, if **B** is an arbitrary Boolean lattice, $\mathbf{B} \neq \{0, 1\}$, then it can be easily shown that there are functions $f:\mathbf{B}^r \rightarrow \mathbf{B}$ which are not representable by any boolean polynomial in $\mathbf{B}_{2^r}$. Since we wish to manipulate discrete functions, we clearly cannot use either lattice polynomials or Boolean polynomials and so another method of representing and transforming discrete functions must be employed.

4.2.3.2.2. The Lattice of Discrete Functions

Our approach is direct: Rather than constructing a distributive lattice of some special subset of functions in which we are interested, we shall instead construct a distributive lattice A^N of all the discrete functions. (Recall [Eq. 4.82]

$$N = \prod_{j=0}^{r} |U_j| \tag{4.82}$$

is the cardinality of **U**.) Of course, A^N will be much larger and more unwieldly than either $\mathbf{P}^r$ or $\mathbf{B}_{2^r}$, but A_N will contain all the functions that we need. (See MacLane and Birkhoff [183], Theorem 13, Chapter 14, for a succint development of the lattice of discrete functions.)

Take

$$f: \bigtimes_{j=0}^{r} U_j \rightarrow A\ , \tag{4.84}$$

---

[9] In a manner similar to lattice polynomials, ***boolean polynomials*** (or ***boolean expressions***) are written with the additional operators $\bar{}$, 0, and 1, e.g., x_0, $\overline{(x_0 \wedge x_1)}$, $1 \vee x_0$. The functions described by these polynomials are called ***boolean functions in n variables over*** **B**.

[10] $\mathbf{B}_{2^r}$ is the free Boolean lattice on $r+1$ generators. $\mathbf{B}_{2^r}$ is finite.

where the range $A = \{0, 1, \ldots, l\}$, and define a lattice $\{A, \vee, \wedge, 0, l\}$, where 0 is the least element, l is the greatest element, and the operations disjunction (join) $\vee$ and conjunction (meet) $\wedge$ are defined as follows: for $a, b \in A$, $a \wedge b = \min(a, b)$ and $a \vee b = \max(a, b)$. The lattice $\{A, \vee, \wedge, 0, l-1\}$ is hence a ***chain***, and importantly, distributive. Now consider the direct product[11] of N lattices $\{A, \vee, \wedge, 0, l\}$. This is again a lattice and is $\{A^N, \vee_N, \wedge_N, 0^N, l^N\}$, where $A^N = \bigtimes_{j=0}^{N} A$. The lattice A^N is distributive since the direct product of distributive lattices is distributive (see, e.g., MacLane and Birkhoff [183], p. 496). However, A^N is not necessarily a complemented lattice, and it is this lack which keeps A^N from necessarily being Boolean. Given two functions $[f_u]_N$ and $[g_u]_N$, their conjunction and disjunction are the componentwise extensions

$$[(f \wedge g)_u]_N = [f_u \wedge g_u]_N \tag{4.85}$$

$$[(f \vee g)_u]_N = [f_u \vee g_u]_N . \tag{4.86}$$

Also, for $a \in A$,

$$[(a \wedge f)_u]_N = [a_u \wedge f_u]_N \tag{4.87}$$

$$[(a \vee f)_u]_N = [a_u \vee f_u]_N . \tag{4.88}$$

A point $\mathbf{a}$ of the lattice A^N corresponds to the function $f_{\mathbf{a}}: \mathbf{U} \to A$ described by the value vector $[a_j]_N$. Thus, the set of discrete functions $\{f: \mathbf{U} \to A\}$ is isomorphic to the product lattice $\{A^N, \vee_N, \wedge_N, 0^N, l^N\}$. Note that A^N can be large; for instance, if $r = 3$, $\left|U_j\right| = 3$, and $|A| = 3$, then $\left|A^N\right| = 3^{3 \times 3 \times 3} = 3^{27}$, or approximately 7×10^{12} functions. The Hasse diagram of the set of functions $\{f: \{0, 1\}^2 \to \{0, 1\}\}$ ($2^{2 \times 2} = 16$ elements) is shown in Figure 4.1, and the Hasse diagram of the set of functions $\{f: \{0, 1, 2\} \to \{0, 1, 2\}\}$ ($3^3 = 27$

---

[11] The ***direct product*** of two lattices $\{A_0, \vee_0, \wedge_0, 0_0, l_0\}$ and $\{A_1, \vee_1, \wedge_1, 0_1, l_1\}$ is the lattice $\{A_0 \times A_1, \vee_{0 \times 1}, \wedge_{0 \times 1}, 0_{0 \times 1}, l_{0 \times 1}\}$ where $(z_0, z_1) \vee_{0 \times 1} (y_0, y_1) = (z_0 \vee_{0 \times 1} y_0, z_1 \vee_{0 \times 1} y_1)$.

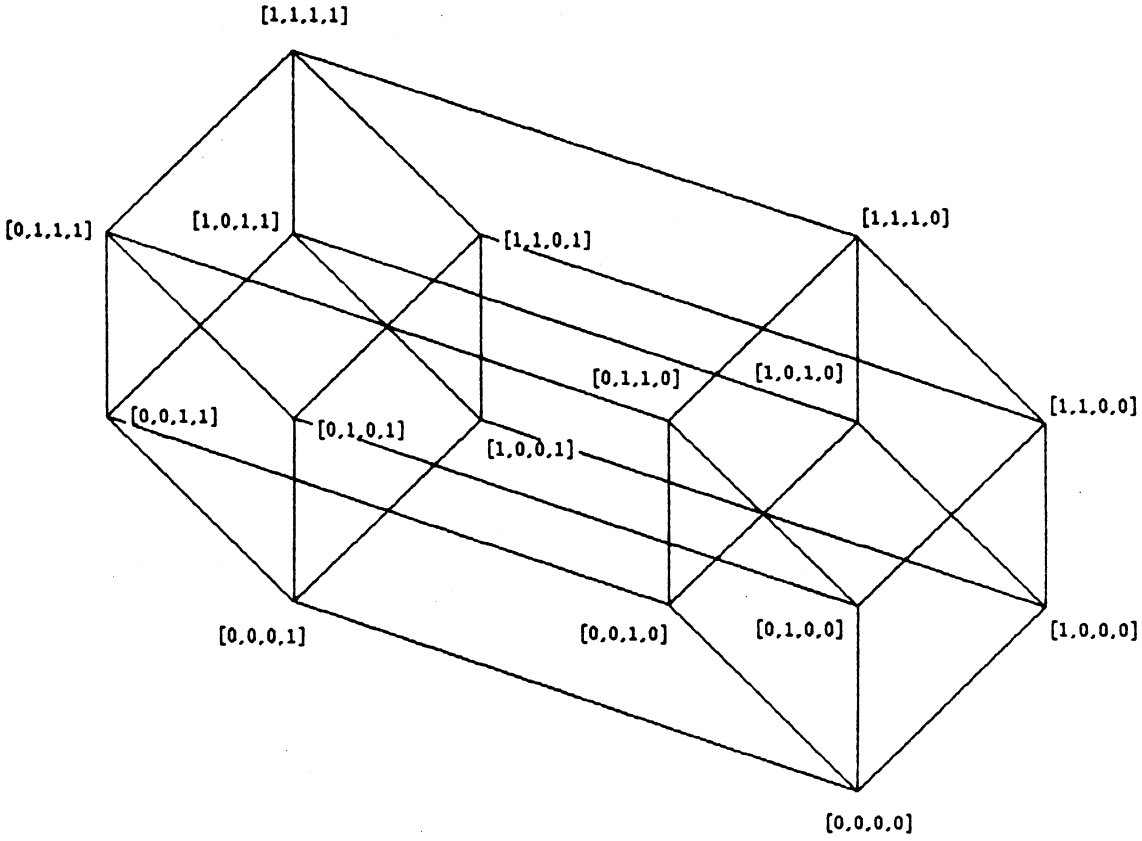


Fig. 4.1 Lattice of the functions $\{f:\{0,1\}^2 \to \{0,1\}\}$

elements) is shown in Fig 4.2.

4.2.3.2.3. Lattice Exponentiation

We now extend the exponentiation of lattice elements (e.g., see MacLane and Birkhoff [183], p. 503 or Grätzer [182], p. 82). Let $\mathbf{x} = [x_j]_r = [x_0, x_1, \ldots, x_r]$ be a vector of $r+1$ variables such that x_j is a variable taking values in U_j. Then if $(C_j) \subseteq U_j$, define the ***lattice exponentiation*** (Davio, Deschamps, and Thayse [165]) $x_j^{(C_j)}$ to be the function:

$$\cdot_j^{(C_j)} : U_j \rightarrow A \tag{4.89}$$

where

$$x_j^{(C_j)} = \begin{cases} 1 & \text{if } x_j \in (C_j) \\ 0 & \text{otherwise .} \end{cases} \tag{4.90}$$

Note that $x_j^{(U_j)} = 1$ and $x_j^{(\phi)} = 0$. Also, note how the following expression evaluates:

$$a \wedge x_j^{(C_j)} = \begin{cases} a & \text{if } x_j \in (C_j) \\ 0 & \text{otherwise .} \end{cases} \tag{4.91}$$

If $(C_j) = \{u_{j0}, u_{j1}, \ldots, u_{jg_j}\}$, write

$$x_j^{(C_j)} = x_j^{(u_{j0}, u_{j1}, \ldots, u_{jg_j})} . \tag{4.92}$$

An example will be given below. We shall use lattice exponentiation and constant functions as building blocks to represent more complicated functions (called "cube functions.") Cube functions will then be used to represent yet more complex (indeed all) discrete functions.

4.2.3.2.4. Lattice Expressions

Define a ***lattice expression*** to be a well-formed expression[12] made of the following

---

and $(x_0, x_1) / {}_{0 \times 1}(y_0, y_1) = (x_0 / {}_{0 \times 1} y_0, x_1 / {}_{0 \times 1} y_1)$.

[12] A well formed expression (wfe) is an inductively defined string of constants (0, 1, . . . , 1), variables

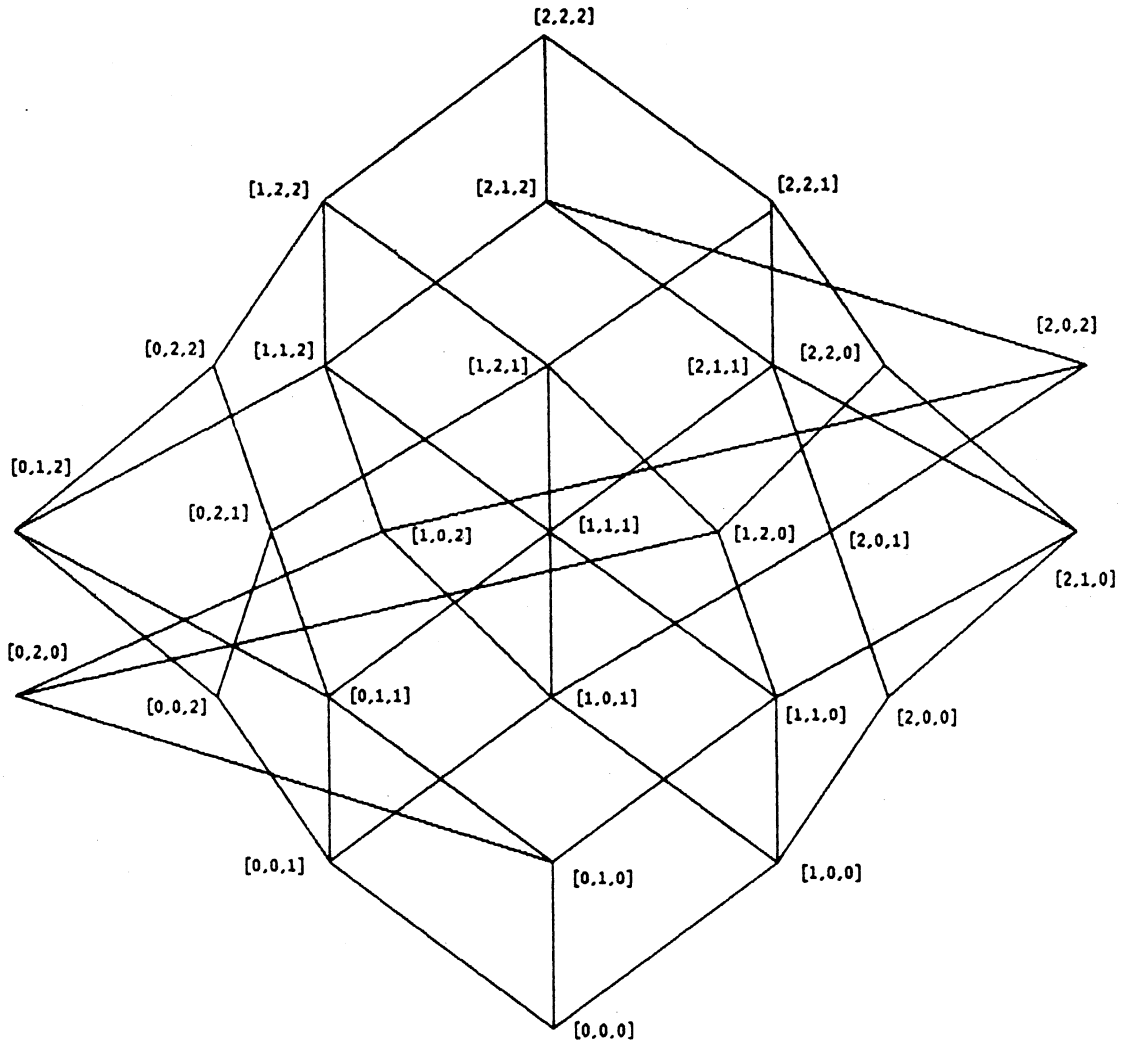


Fig. 4.2 Lattice of the functions {f:{0,1,2}→{0,1}}

symbols:

a) the constants are the lattice elements 0, 1, . . . , l

b) the variables are the $x_j^{(C_j)}$

c) the operators are the two binary lattice operators $\vee$ and $\wedge$.

(Note the lack of a complement operator; complements do not necessarily exist in the lattice A^N.) The operator $\wedge$ takes precedence over $\vee$. Sometimes in an expression, the operator $\wedge$ will be omitted, e.g., $a \wedge b$ will be written ab. Also, matching parenthesis "(" and ")" will sometimes be embedded within an expression to denote a local change in the order of operator precedence.

Every lattice expression describes a discrete function, since by choosing any value $\mathbf{u} \in \mathbf{U}$, substituting $\mathbf{u}$ into the expression and evaluating, a value in A can be obtained. Further, as shall be seen, every discrete function can be represented by at least one lattice expression.

As an example of representing a discrete function by lattice expressions, consider the discrete function

$$f: U_0 \times U_1 \rightarrow A \tag{4.93}$$

where $U_0 = \{0, 1, 2, 3, 4\}$, $U_1 = \{0, 1, 2\}$, and $A = \{0, 1, 2, 3\}$. A tabular

---

$(z_0, z_1, \ldots, z_N)$, and r-ary operators $(\theta_0, \theta_1, \ldots, \theta_k)$. The rules of construction are

1) Any constant is a wfe.

2) Any variable is a wfe.

3) If $E_0, E_1, \ldots, E_r$ are wfe's, and θ_j is an (r)-ary operator, then $\theta_j(E_0, E_1, \ldots, E_r)$ is a wfe. (If $r = 2$, the wfe is often written $E_1\, \theta_j\, E_2$.)

4) The only wfe's are those obtained by finite applications of rules 1), 2), and 3).

representation for f is presented below:

		x_0				
		0	1	2	3	4
	0	1	3	3	0	2
x_1	1	0	3	3	2	1
	2	1	0	3	2	3.

f can be represented by the following lattice expression:

$$\begin{aligned} F(x_0, x_1) = \quad & 1x_0^{(0)}x_1^{\{0,2\}} \wedge 1x_0^{(4)}x_1^{\{1\}} \\ & \wedge 2x_0^{(3)}x_1^{\{1,2\}} \wedge 2x_0^{(4)}x_1^{(0)} \\ & \wedge 3x_0^{\{1,2\}}x_1^{\{0,1\}} \wedge 3x_0^{\{2,4\}}x_1^{\{2\}} . \end{aligned} \tag{4.94}$$

Note that F is composed of the conjunction of six functions ("cube functions"), the first being $1x_0^{(0)}x_1^{\{0,2\}}$. In turn, each of those functions is composed of the conjunction of three functions: A constant function and two lattice exponentiations, e.g., $1x_0^{(0)}x_1^{\{0,2\}}$ is the conjunction of the lattice exponentiations $x_0^{(0)}$ and $x_1^{\{0,2\}}$.

Using trajectory set notation (Section 4.1 [Trajectory Sets: Basic Notation and Operations]), the same function would be represented by the preimages:

$$\begin{aligned} f^{-1}(0) &= [\,0\ 1\,] \cup [\,1\ 2\,] \cup [\,3\ 0\,] \\ f^{-1}(1) &= [\,0\ \{0,2\}\,] \cup [\,4\ 1\,] \\ f^{-1}(2) &= [\,3\ \{1,2\}\,] \cup [\,4\ 0\,] \\ f^{-1}(3) &= [\,\{1,2\}\ \{0,1\}\,] \cup [\,\{2,4\}\ 2\,] . \end{aligned} \tag{4.95}$$

Note the similarity between Eq. 4.94 and Eq. 4.95; the entries of the array products of Eq.

4.95 are simply the exponents of Eq. 4.94. With the entries of $f^{-1}(0)$ being all the entries not represented by $f^{-1}(1)$, $f^{-1}(2)$, and $f^{-1}(3)$. Indeed, it is clear that lattice expressions are a more general method of representation of discrete functions than are trajectory sets in the sense that every trajectory set can be written as a lattice expression similar to Eq. 4.94, but not every lattice expression can be written as a trajectory set. We shall find that every lattice expression denotes a discrete function (i.e., represents a point on A^N), but not every trajectory set represents a complete function. Yet, as shall be seen, every discrete function can be represented by an expression similar to Eq. 4.94, so using transformations of the kind used to obtain Eq. 4.95 from Eq. 4.94, trajectory sets are sufficiently powerful to represent every discrete function.

There is a countably infinite number of different lattice expressions that represent function f. For example, f can also be described by

$$\begin{aligned} G(x_0, x_1) = {} & (0 \vee x_0^{\{0,1,2,3\}} \vee x_1^{\{1,2\}})(0 \vee x_0^{\{1,2,3,4\}} \vee x_1^{\{0,2\}})(0 \vee x_0^{\{0,2,3,4\}} \vee x_1^{\{0,1\}}) \\ & (1 \vee x_0^{\{1,2,3,4\}} \vee x_1^{\{1\}})(1 \vee x_0^{\{0,1,2,3\}} \vee x_1^{\{0,2\}}) \\ & (2 \vee x_0^{\{0,1,2,4\}} \vee x_1^{\{0\}})(2 \vee x_0^{\{0,1,2,3\}} \vee x_1^{\{1,2\}}) . \end{aligned} \tag{4.96}$$

4.2.3.2.5. Classes and Properties of Lattice Expressions

Let us now formalize the above discussion. We begin by considering a more general function than simple lattice exponentiation. A ***cube function*** (Davio, Deschamps, and Thayse [165]) is a lattice expression $c(\mathbf{x})$ of the form

$$c(\mathbf{x}) = a \wedge \bigwedge_{j=0}^{r} x_j^{(C_j)}, \quad a \in A \ . \tag{4.97}$$

a is the ***weight*** of the cube. The elements of the set (C_j) are ***entries of weight a***. In particu-

lar,

$$c(\mathbf{x}) = \begin{cases} a & \text{if } x_j \in C_j \text{ for all j} \\ 0 & \text{otherwise .} \end{cases} \tag{4.98}$$

If f is a switching function, i.e., $f:\{0,1\}^r \rightarrow \{0,1\}$, then cube functions correspond to ***implicants*** in switching theorey. Since we shall not be dealing with any other class of implicants (see [165] for other classes), we shall freely use the term "implicant" for cube function.

If each (C_j) contains exactly one element, then $c(\mathbf{x})$ is a ***join-irreducible element*** of the lattice A^N. If $c(\mathbf{x})$ is such a join-irreducible element and $a = 1$, then $c(\mathbf{x})$ is an ***atom*** of A^N. An atom is hence a cube function

$$1 \wedge \bigwedge_{j=0}^{r} x_j^{(u_j)}, \quad u_j \in U_j . \tag{4.99}$$

An atom $c(\mathbf{x})$ is 1 if $x_j = u_j$ for all j, and is 0 otherwise. If at least one set $c_j = \phi$, then $c(\mathbf{x}) = 0$ for all x and so $c(\mathbf{x})$ is the minimum element (called the ***empty cube***) of A^N. This corresponds to a null array Φ of the trajectory set calculus. Let $c_0(\mathbf{x})$ and $c_1(\mathbf{x})$ be cubes

$$\begin{aligned} c_0(\mathbf{x}) &= a_0 \wedge \bigwedge_{j=0}^{r} x_j^{(C_{j0})} \\ c_1(\mathbf{x}) &= a_1 \wedge \bigwedge_{j=0}^{r} x_j^{(C_{j1})} \end{aligned} \tag{4.100}$$

where $a_0, a_1 \in A$. Then the conjunction of the two cubes is

$$c(\mathbf{x}_0) \wedge c(\mathbf{x}_1) = a_0 \wedge a_1 \wedge \bigwedge_{j=0}^{r} x_j^{(C_{j0} \cap C_{j1})} . \tag{4.101}$$

The duals for the above concepts follow. An ***anticube function*** is a lattice expression

$d(\mathbf{x})$ of the form

$$d(\mathbf{x}) = a \vee \bigvee_{j=0}^{r} x_j^{(D_j)}, \quad a \in A, \quad D_j \subseteq U_j \ . \tag{4.102}$$

a is the ***weight*** of the anticube. Now,

$$d(\mathbf{x}) = \begin{cases} l & \text{if } x_j \in D_j \quad \text{for some } j \\ a & \text{otherwise .} \end{cases} \tag{4.103}$$

In switching theory, anticube functions correspond to ***implicates***. If each $\bar{D}_j = U_j - D_j$ contains exactly one element, then $d(\mathbf{x})$ is a ***meet- irreducible element*** of the lattice A^N. A meet-irreducible element with $a = l$ is an ***antiatom*** of A^N:

$$l \vee \bigvee_{j=0}^{r} x_j^{(\bar{u}_j)}, \quad u_j \in U_j \ . \tag{4.104}$$

An antiatom $d(\mathbf{x})$ is l–1 if $x_j = u_j$ for all j, and is l otherwise. If at least one set $d_j = \phi$, then $d(\mathbf{x}) = l$ for all $\mathbf{x}$ and so $d(\mathbf{x})$ is the maximum element (called the ***empty anticube***) of A^N. This corresponds to a "full set" of the trajectory set calculus. Let $d_0(\mathbf{x})$ and $d_1(\mathbf{x})$ be cubes

$$\begin{aligned} d_0(\mathbf{x}) &= a_0 \vee \bigvee_{j=0}^{r} x_j^{(D_{j0})} \\ d_1(\mathbf{x}) &= a_1 \vee \bigvee_{j=0}^{r} x_j^{(D_{j1})} \end{aligned} \tag{4.105}$$

where $a_0, \ a_1 \in A$. Then the disjunction of the two cubes is

$$d(\mathbf{x}_0) \wedge d(\mathbf{x}_0) = a_0 \wedge a_1 \wedge \bigvee_{j=0}^{r} x_j^{(D_{j0} \cup D_{j1})} . \tag{4.106}$$

The ***negation*** (as opposed to "complement") $\bar{f}$ of a discrete function f is

$$[\bar{f}_u] = [1 - f_u] \ . \tag{4.107}$$

The minus sign is arithmetic subtraction. Negation relates the above concepts of cube and anticube functions. Clearly, $\overline{(\bar{f})} = f$, and De Morgan's Laws apply, i.e., for a set of functions $\{f_h\}$:

$$\begin{aligned} \overline{\bigvee_h f_h(x)} &= \bigwedge_h \bar{f}_h(x) \\ \overline{\bigwedge_h f_h(x)} &= \bigvee_h \bar{f}_h(x) \end{aligned} \tag{4.108}$$

The negation of a cube function is an anticube and conversely:

$$\begin{aligned} \overline{a \wedge \bigwedge_{j=0}^{r} x_j^{(C_j)}} &= \bar{a} \vee \bigvee_{j=0}^{r} x_j^{(\bar{C}_j)} \\ \overline{a \vee \bigvee_{j=0}^{r} x_j^{(C_j)}} &= \bar{a} \wedge \bigwedge_{j=0}^{r} x_j^{(\bar{C}_j)} \end{aligned} \tag{4.109}$$

Also, the following identities can be shown:

$$\overline{x_j^{(C_j)}} = x_j^{(\bar{C}_j)} \tag{4.110}$$

$$x_j^{(\bigcup_h C_{jh})} = \bigvee_h x_j^{(C_{jh})} \tag{4.111}$$

$$x_j^{(\bigcap_h C_{jh})} = \bigwedge_h x_j^{(C_{jh})} \tag{4.112}$$

$$1 \wedge x_j^{(C_j)} = x_j^{(C_j)} \tag{4.113}$$

$$1 \vee x_j^{(C_j)} = 1 \tag{4.114}$$

$$0 \wedge x_j^{(C_j)} = 0 \tag{4.115}$$

$$0 \vee x_j^{(C_j)} = x_j^{(C_j)} . \tag{4.116}$$

Distributivity of $\wedge$ over $\vee$ and $\vee$ over $\wedge$ holds:

$$\begin{aligned} x_j^{(C_j)} \vee \left[x_k^{(C_k)} \wedge x_h^{(C_h)} \right] &= \left[x_j^{(C_j)} \vee x_k^{(C_k)} \right] \wedge \left[x_j^{(C_j)} \vee x_h^{(C_h)} \right] \\ x_j^{(C_j)} \wedge \left[x_k^{(C_k)} \vee x_h^{(C_h)} \right] &= \left[x_j^{(C_j)} \wedge x_k^{(C_k)} \right] \vee \left[x_j^{(C_j)} \wedge x_h^{(C_h)} \right] \end{aligned} \tag{4.117}$$

4.2.3.2.6. Normal Forms of Lattice Expressions

Let F be a lattice expression representing the function f. If F is a disjunction of cube functions, i.e., if

$$F(\mathbf{x}) = \bigvee_h c_h(\mathbf{x}) , \tag{4.118}$$

then F is a ***disjunctive normal form*** of f. Similarly, if G is a lattice expression representing f and G is a conjunction of anticube functions

$$G(\mathbf{x}) = \bigwedge_h d_h(\mathbf{x}) , \tag{4.119}$$

then G is a ***conjunctive normal form*** of f. For example, Eq. 4.94 is in disjunctive normal form. The trajectory set notation is a variation of the disjunctive normal form. Nothing corresponds to the conjunctive normal form in the trajectory set calculus. Every lattice expression may be transformed into an equivalent disjunctive normal form and an equivalent

conjunctive norm form by repetitive application of the distributive laws (Eq. 4.117).

A notation that is close to trajectory sets is the "cubical representation." Represent a cube function

$$a \wedge \bigwedge_{j=0}^{r} x_j^{(C_j)} \tag{4.120}$$

by the vector of one scalar and $r+1$ sets: $[a\ C_0\ C_1\ \cdots\ C_r]$. The disjunctive normal form of f,

$$F(\mathbf{x}) = \bigvee_{k=0}^{p} \left[w_k \wedge \bigwedge_{h=0}^{r} x_h^{(C_{kh})} \right], \tag{4.121}$$

(where $w_k \in A$) being the disjunction of $p+1$ cube functions, is then represented by the array of $s+1$ scalars and $(p+1)(r+1)$ sets

$$\mathbf{F} = [a_k\ C_{kh}]_{kh:p \times r} = \begin{bmatrix} a_0 & C_{00} & C_{01} & \cdots & C_{0r} \\ a_1 & C_{10} & C_{11} & \cdots & C_{1r} \\ \vdots & \vdots & \vdots & & \vdots \\ \cdot & \cdot & \cdot & & \cdot \\ a_p & C_{p0} & C_{p1} & \cdots & C_{pr} \end{bmatrix}. \tag{4.122}$$

Note that the above matrix could be decomposed into a matrix $\mathbf{C}$ of the sets C_{kh} and a vector $\mathbf{W}$ of the w_k:

$$\mathbf{C} = [C_{kh}]_{p \times r} = \begin{bmatrix} C_{00} & C_{01} & \cdots & C_{0r} \\ C_{10} & C_{11} & \cdots & C_{1r} \\ \vdots & \vdots & & \vdots \\ \cdot & \cdot & & \cdot \\ C_{p0} & C_{p1} & \cdots & C_{pr} \end{bmatrix}, \tag{4.123}$$

$$\mathbf{W} = [w_k]_{k:p} = [w_0\ w_1\ \cdots w_p]\ . \tag{4.124}$$

It is convenient to represent the subset C_{jk} of U_j as a binary vector

$$(C_{jk}) = [b_{kh}^l]_{g:N_j} = [b_{kh}^0, b_{kh}^1, \ldots, b_{kh}^{N_j}] \tag{4.125}$$

where N_j is the number of elements in U_j and

$$b_{kh}^{g} = \begin{cases} 1 & \text{if } g \in (C_{kh}) \\ 0 & \text{otherwise} \end{cases} . \tag{4.126}$$

Then the representation **F** of f given in Eq. 4.122 can be written

$$\mathbf{F} = [w_k \ [b_{kh}^{g}]_{g:N_k}]_{kh:p \times r} \tag{4.127}$$

$$= \begin{bmatrix} w_0 & [b_{00}^{g}]_{g:N_0} & [b_{01}^{g}]_{g:N_0} & \cdots & [b_{0r}^{g}]_{g:N_0} \\ w_1 & [b_{10}^{g}]_{g:N_1} & [b_{11}^{g}]_{g:N_1} & \cdots & [b_{1r}^{g}]_{g:N_1} \\ \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & \cdot & & \cdot \\ w_p & [b_{p0}^{g}]_{g:N_p} & [b_{p1}^{g}]_{g:N_p} & \cdots & [b_{pr}^{g}]_{g:N_p} \end{bmatrix} . \tag{4.128}$$

For instance, the expression of Eq. 4.94

$$\begin{aligned} F(x_0, x_1) = \quad & 1x_0^{(0)}x_1^{(0,2)} \wedge 1x_0^{(4)}x_1^{(1)} \\ & \wedge 2x_0^{(3)}x_1^{(1)} \wedge 2x_0^{(4)}x_1^{(0)} \\ & \wedge 3x_0^{(1,2)}x_1^{(0,1)} \wedge 3x_0^{(2,4)}x_1^{(2)} \end{aligned} \tag{4.94}$$

would have the following cubical representation:

$$\mathbf{F} = \begin{bmatrix} 1 & [0\ 1\ 0\ 1] & [0\ 0\ 0\ 0\ 1] \\ 1 & [0\ 0\ 1\ 0] & [1\ 0\ 0\ 0\ 0] \\ 2 & [0\ 1\ 1\ 0] & [0\ 1\ 0\ 0\ 0] \\ 2 & [0\ 0\ 0\ 1] & [1\ 0\ 0\ 0\ 0] \\ 3 & [0\ 0\ 1\ 1] & [0\ 0\ 1\ 1\ 0] \\ 3 & [0\ 1\ 0\ 0] & [1\ 0\ 1\ 0\ 0] \end{bmatrix} \tag{4.129}$$

(or

$$\mathbf{C} = \begin{bmatrix} [0\ 1\ 0\ 1] & [0\ 0\ 0\ 0\ 1] \\ [0\ 0\ 1\ 0] & [1\ 0\ 0\ 0\ 0] \\ [0\ 1\ 1\ 0] & [0\ 1\ 0\ 0\ 0] \\ [0\ 0\ 0\ 1] & [1\ 0\ 0\ 0\ 0] \\ [0\ 0\ 1\ 1] & [0\ 0\ 1\ 1\ 0] \\ [0\ 1\ 0\ 0] & [1\ 0\ 1\ 0\ 0] \end{bmatrix} \tag{4.130}$$

and

$$\mathbf{W} = \left[\, 1\,1\,2\,2\,3\,3 \,\right] \;). \tag{4.131}$$

4.2.3.2.7. Canonical Forms of Lattice Expressions

As mentioned above, a discrete function can be represented in many different ways. Even when restricted to disjunctive normal forms (Eq. 4.118), an infinite number of expressions representing a given function can be written. With the machinery of the preceding section, the possibilities of specifying canonical or "standard" forms of discrete functions can now be discussed. Also, that every discrete function can be represented by at least one lattice expression will be shown.

First, define a ***minterm*** to be a join-irreducible element of the lattice with weight greater than 0, i.e., a minterm is a cube function of the form

$$a \vee \bigwedge_{j=0}^{n} x_j^{(u_j)} \tag{4.132}$$

where $\mathbf{u} = (u_0, u_1, \ldots, u_n)$, $0 \leq u_j \leq N_j$ is a point in the lattice. Note that there is a minterm for each non-zero weight a at each point in the lattice, and hence, there are $N \cdot (l-1)$ distinct minterms (where $N = \prod_{j=0}^{n} N_j$). We have the following important

Theorem 4.1: (Discrete function representation) Shannon [184]

Every discrete function f can be represented by a lattice expression in disjunctive normal form, each of whose cube functions is a minterm. Further, this lattice expression is unique up to a permutation of the minterms.

Proof:

Construct the expression

$$F(\mathbf{x}) = \bigvee_{\mathbf{u}} \left[f(\mathbf{u}) \wedge \bigwedge_{j=0}^{n} x_j^{(u_j)} \right] , \text{ where} \tag{4.133}$$

$$\mathbf{u} = (u_0, u_1, \ldots, u_n) \qquad 0 \leq u_j \leq N_j \ .$$

and note that each cube function inside the brackets is a minterm. For a given $\mathbf{u} = (u_0, u_1, \ldots, u_n)$, at most one cube function is non-zero, specifically

$$f(\mathbf{u}) \wedge x_0^{(e_0)} \wedge x_1^{(e_1)} \wedge \cdots \wedge x_n^{(e_n)} , \tag{4.134}$$

which has weight $f(\mathbf{u})$. Thus, the expression $F(\mathbf{x})$ assumes the value $f(\mathbf{u})$ at the point $\mathbf{u}$. Every function f can be so represented since the construction of F does not depend on the nature of f.

To show uniqueness, assume there is a second qualifying expression $G(\mathbf{x})$. Suppose $G(\mathbf{x})$ contains a minterm $c(\mathbf{x})$ (where $c(\mathbf{u}) \neq 0$) not in $F(\mathbf{x})$. Then clearly $G(\mathbf{u}) \neq F(\mathbf{u}) = 0$ and either $F(\mathbf{x})$ or $G(\mathbf{x})$ does not represent f. Next, suppose $F(\mathbf{x})$ contains all the minterms $G(\mathbf{x})$ and, in addition, the minterm $c(\mathbf{x})$ [where $c(\mathbf{u}) \neq 0$]. Then again $G(\mathbf{u}) = 0 \neq F(\mathbf{u})$, and once more either $G(\mathbf{x})$ or $F(\mathbf{x})$ fails. Hence $F(\mathbf{x})$ is unique.

||

The expression $F(\mathbf{x})$ of the proof is the ***canonical disjunctive form*** of function f. Two lattice expressions are ***equivalent*** if and only if they represent the same discrete function. Suppose a lattice expression $G(\mathbf{x})$ representing function f is equivalent to the canonical disjunctive form $F(\mathbf{x})$ of f. Then $F(\mathbf{x})$ is said to be the canonical disjunctive form of the expression $G(\mathbf{x})$. Any two expression are ***equivalent*** if they have the same disjunctive normal form.

A form of Shannon's expansion theorem provides one method of obtaining the canonical disjunctive form representing a discrete function from any expression $G(\mathbf{x})$ representing that function:

Theorem 4.2: (Shannon"s first expansion theorem)

A discrete function f represented by an expression $G(\mathbf{x})$ can be expressed

$$F(\mathbf{x}) = \bigvee_{u_0=0}^{N_0} \{x_0^{(0)} \vee G(u_0, x_1, x_2, \ldots, x_n)\} . \tag{4.135}$$

Applying Eq. 4.135 repeatedly with respect to every other variable $x_1, x_2, \ldots, x_n$ results in the canonical disjunctive form. Algorithm 2.1 of Davio, Deschamps, and Thayse [165] describes a simpler and faster procedure for obtaining the canonical disjunctive form.

The dual concepts for canonical conjunctive normal forms can also be described. Our interest in these forms vis-a-vis applications such as METAPHOR is somewhat academic since these forms are not practical in use. This impracticality results from the size of these normal forms. For example, in the worst case, a function f that is identically l everywhere (i.e., $f(\mathbf{u}) = l$ for all $\mathbf{u}$), would require $N = \prod_{j=0}^{n} N_j$ cube functions to be represented in canonical disjunctive form. However, the canonical disjunctive form does demonstrate that every function can be represented by a lattice expression, and hence, every discrete capability function γ and interlevel translation κ can be denoted by a set of trajectory sets. Further, any two lattice expressions can be compared for equivalence. Thus, our machinery is "complete" in the sense that everything we could want to represent can be represented.

All that remains, then, is to describe and implement a set of algorithms which allow us to manipulate trajectory sets (lattice expressions) efficiently so that we can obtain our goal, viz., $\gamma^{-1}(a)$ (Eq. 4.78). Such a set of algorithms is presented in Section 4.3.3 [Algorithms] and a discussion of a computer implementation is given in Section 4.4 [METAPHOR–A Performability Modeling and Evaluation Tool].

4.2.3.2.8. Exponentiation and Composition of Lattice Expressions

Exponentiation of lattice expressions and composition of lattice expressions are closely related and follow naturally from the definition of lattice exponentiation (Eq. 4.90). Consider the discrete functions

$$\begin{aligned} f&:\mathbf{U}_0 \rightarrow U \\ g&:U \rightarrow A \end{aligned} \tag{4.136}$$

where $\mathbf{U}_0 = U_0 \times U_1 \times \cdots \times U_r$, $U = \{0, 1, \ldots, L\}$, and $A = \{0, 1, \ldots, l\}$. Let $\mathbf{x} = [x_j]_r = [x_0, x_1, \ldots, x_r]$ be a vector of $r+1$ variables such that x_j is a variable taking values in U_j, and let x be a variable taking values in U. Then if $(C_j) \subseteq U_j$ and $(C) \subseteq U$, there are functions

$$\bullet_j^{(C_j)} : U_j \rightarrow U \tag{4.137}$$

$$\bullet^{(C)} : U \rightarrow A \tag{4.138}$$

where

$$x_j^{(C_j)} = \begin{cases} L & \text{if } x_j \in (C_j) \\ 0 & \text{otherwise} \end{cases} \tag{4.139}$$

$$x^{(C)} = \begin{cases} 1 & \text{if } x \in (C) \\ 0 & \text{otherwise .} \end{cases} \tag{4.140}$$

Then the composition of $x_j^{(C_j)}$ and $x_j^{(C)}$ yields the ***exponentiation of a lattice exponentiation***:

$$\left(\cdot_j^{(C_j)} \right)^{(C)} : U_j \to A \tag{4.141}$$

where

$$\left(x_j^{(C_j)}\right)^{(C)} = \begin{cases} 1 & \text{if } \{x_j \in (C_j) \text{ and } L \in (C)\} \text{ or } \{x_j \notin (C_j) \text{ and } 0 \in (C)\} \\ 0 & \text{otherwise .} \end{cases} \tag{4.142}$$

Note that $(x_j^{(U_j)})^{(U)} = 1$ and $(x_j^{(\phi)})^{(C)} = (x_j^{(C_j)})^{(\phi)} = 0$. ***Exponentiation of a cube function***, ***exponentiation of an anti-cube function***, and ***exponentiation of a lattice expression*** are similar extensions. Thus, for $u \in U$ and $a \in A$,

$$a \wedge \left(u \wedge x_j^{(C_j)}\right)^{(C)} = \begin{cases} a & \text{if } \{x_j \in (C_j) \text{ and } u \in (C)\} \text{ or } \{x_j \notin (C_j) \text{ and } 0 \in (C)\} \\ & \quad \text{or } q = 0 \in (C) \\ 0 & \text{otherwise .} \end{cases} \tag{4.143}$$

The ***composition*** of two lattice functions $F(\mathbf{x})$ and $G(x)$ representing f and g (Eq. 4.136) is then $G(F(\mathbf{x}))$, i.e., G with all occurences of x replaced by $F(\mathbf{x})$.

4.3. Calculation of Trajectory Sets

We now wish to state a set of algorithms which will enable us to compute the performability of discrete performance variable models. These algorithms have all been implemented in the METAPHOR package (see Sections 3.4.4 [METAPHOR–A Performability Modeling and Evaluation Tool] and 4.4 [METAPHOR–A Performability Modeling and Evaluation Tool]). In the development below, the names of the METAPHOR functions implementing specific algorithms and steps of algorithms are noted.

We begin (Section 4.3.1 [Representation of Discrete Functions Within METAPHOR]) by examining the internal representation that METAPHOR uses to represent discrete functions; this representation is a hybrid of the trajectory set notation (Section 4.1 [Trajectory Sets: Basic Notation and Operations]) and lattice expressions (Section 4.2.3.2 [Representations Using Lattice Expressions]). Then, in Section 4.3.2 [Notation], some specific notation for representing the discrete functions of METAPHOR is introduced. Finally (Section 4.3.3 [Algorithms] discusses the algorithms.

4.3.1. Representation of Discrete Functions Within METAPHOR

METAPHOR's internal representation of discrete functions is based on the disjunctive normal form of lattice expressions (Eq. 4.118); the actual representation is exemplified by the array of binary vectors of Eq. 4.127. However, the form of METAPHOR's representations differs slightly from Eq. 4.129, and METAPHOR places some restrictions on the freedom of the representations. The differences are described in this section.

First, the variables in METAPHOR have two indexes (attribute and phase), so a cube function (Eq. 4.97) is written

$$a \wedge \bigwedge_{j=0}^{r} \bigwedge_{k=0}^{s} x_{jk}^{(C_{jk})} . \tag{4.144}$$

The exponents of the cube function, (C_{jk}), can be denoted by a matrix

$$[C_{jk}]_{r \times s} = \begin{bmatrix} C_{00} & C_{01} & \cdots & C_{0s} \\ C_{10} & C_{11} & \cdots & C_{1s} \\ \vdots & \vdots & & \vdots \\ \cdot & \cdot & & \cdot \\ C_{r0} & C_{r1} & \cdots & C_{rs} \end{bmatrix} \tag{4.145}$$

where $(C_{jk}) \subseteq U_{jk}$ and the weight of the cube function denoted by $[C_{jk}]_{r \times s}$ is understood to be a. Thus, we see the relation between array products (Section 4.1 [Trajectory Sets: Basic

Notation and Operations]) and cube functions of the form Eq. 4.144, viz.,

$$[C_{jk}]_{r \times s} = [(C_{jk})]_{r \times s}, \tag{4.146}$$

where the weight a is implicit. In the remainder of this chapter, we use the terms "array products" and "exponents of a cube function" interchangeably.

With this correspondence in mind, define the "complement" of a cube function to be analogous to the complement of an array product (see Eq. 4.49). Let $c(\mathbf{x})$ be a cube function described by $[(C_{jk})]_{r \times s}$. The ***complement*** of $c(\mathbf{x})$ is the lattice expression $c(\mathbf{x})^c$ constructed from the complement of $[(C_{jk})]_{r \times s}$, i.e., $[(C_{jk})]^c_{r \times s}$, Note that

$$[(C_{jk})]^c_{r \times s} = \bigcup_{a=0}^{r} \bigcup_{b=0}^{s} [(\hat{C}_{jk})]_{r \times s} \tag{4.147}$$

where

$$\hat{C}_{jk} = \begin{cases} C^c_{jk} & \text{if } a = j,\ b = k, \\ * & \text{otherwise} \end{cases} \tag{4.148}$$

and

$$C^c_{jk} = Q_{jk} - C_{jk}\,. \tag{4.149}$$

Of course, we would like to have the various arrays in $c(\mathbf{x})$ to be disjoint; Eq. 4.147 can also be written

$$[(C_{jk})]^c_{r \times s} = \bigcup_{a=0}^{r} \bigcup_{b=0}^{s} [(C_{jk})]_{r \times s} \tag{4.150}$$

where

$$C_{jk} = \begin{cases} C^c_{jk} & \text{if } a = j,\ b = k, \\ C_{jk} & \text{if } a < j, \text{ or if } a = j,\ b < k, \\ * & \text{otherwise} \end{cases} \tag{4.151}$$

and the $[(C_{jk})]_{r \times s}$ are disjoint. Using the properties of Section 4.2.3.2.5 [Classes and Properties of Lattice Expressions], the complement of a cube function can be written

$$c(\mathbf{x})^c = a \wedge \left[\bigvee_{j_1=0}^{r} \bigvee_{k_1=0}^{s} x_{j_1 k_1}^{(C^c_{j_1 k_1})} \wedge \left[\bigvee_{j=0}^{r} \bigvee_{k=0}^{s} x_{jk}^{(C_{jk})} \right] \right] . \tag{4.152}$$

Note that the complement of a cube function is <u>not</u> necessarily a cube function. The terminology "complement" is somewhat abusive, especially since the complement of a complemented cube function is not defined. However, the definition follows naturally from array products and has a natural interpretation, namely the complement of a cube function $c(\mathbf{x})$ with weight a is the lattice expression yielding value a only when $c(\mathbf{x})$ does not, and conversely.

A disjunctive normal form representation of a function f is (see Eq. 4.118)

$$F = \bigvee_{h=0}^{p} \left[w_h \wedge \bigwedge_{j=0}^{r} \bigwedge_{k=0}^{s} x_{jk}^{(C_{(jk)h})} \right] \tag{4.153}$$

where there are $p+1$ terms in the representation, $w_h \in A$ is the weight of the h^{th} cube function, and $(C_{(jk)h})$ is the exponent for the jk^{th} term of the h^{th} cube function (implicant). The array products corresponding to the exponents $(C_{(jk)h})$ will be written $[C_{jkh}]_{r \times s}$. The inverse of f can now be characterized in terms of sets of (C_{jk}) (or of $[C_{jkh}]_{r \times s}$) as follows:

$$\begin{gathered} f^{-1}(a) \text{ is denoted by } \{(C_{(jk)h}) \mid w_h = a\} \\ \text{or } \{[C_{jkh}]_{r \times s} \mid w_h = a\} . \end{gathered} \tag{4.154}$$

We shall generally use the first form, i.e., $\{(C_{(jk)h}) \mid w_h = a\}$.

In representing discrete functions, the second difference between METAPHOR and the form of Eq. 4.127 is that in METAPHOR the cube function of weight 0 must be explicitly included in the characterization of a function. Thus, while every lattice expression denotes a discrete function, not every set of array products denotes a function. Requiring specification

of cube function with weight 0 would be unnecessary if the same convention employed by lattice expressions were used, namely, any $\mathbf{u} \in \mathbf{U}$ that does not appear in the representation would have value 0. With such a convention, the cube function with weight 0 could be determined by the conjunction of the complements (see Eq. 4.152) of each non-zero weight cube function (a form of DeMorgan's Law):

$$\bigvee_{h=1}^{p} c_p(\mathbf{x})^c \,. \tag{4.155}$$

However, computationally, it is easier to store these trajectories rather than to recompute them every time it is necessary to refer to them. Further, by forcing the user of METAPHOR to enter these trajectories, a form of error checking can be implemented. Thus, if the user forgets to include a point in the cube functions associated with a given weight a, that point will not be automatically assigned weight 0; instead, an error message will be generated, and the missing points can be computed using Eq. 4.155.

The third difference between METAPHOR's internal representation of discrete functions and that of Eq. 4.127 is that, in METAPHOR, a point $\mathbf{u} \in \mathbf{U}$ cannot appear in more than one array product (exponent of a cube function). Such a restriction is not inherent in lattice expressions, since, if u appears in the exponents of two different cube functions, the resulting disjunction would still be a function. Notice that this restriction allows another form of error checking by METAPHOR. METAPHOR opperates on the premise that if the user includes the same point u in two different cube functions, than that point has not been properly considered. Upon detection of such an overlap, METAPHOR generates an error message and can determine which point has been considered twice.

Finally, in METAPHOR the lattice expression in disjunctive normal form is factored such that each weight $a \in A$ is written only once, i.e., if p_n is the number of cube functions having weight a_n in the lattice expression (hence $p = \sum_{n=0}^{l} p_n$), then the representation of Eq.

4.153 can be written

$$\bigvee_{h=0}^{p}\left[w_h \wedge \bigwedge_{j=0}^{r}\bigwedge_{k=0}^{s} x_{jk}^{(C_{(jk)h})}\right] = \bigvee_{n=0}^{l} a_n \wedge \left[\bigvee_{h=1}^{p_n}\bigwedge_{j=0}^{r}\bigwedge_{k=0}^{s} x_{jk}^{(C_{(jk)nh})}\right] \tag{4.156}$$

where $(C_{(jk)nh})$ is the jk^{th} exponent of the h^{th} cube function having weight a_n. The representation used by both the APL and C versions of METAPHOR use the form of Eq. 4.156.

From Eq. 4.156, the function f can be denoted by a four dimensional (ragged[13]) array (compare the following discussion with Eqs. 4.123-4.128)

$$\mathbf{C} = [(C_{jknh})]_{jknh:r \times s \times l \times p_i} \tag{4.157}$$

along with a vector

$$\mathbf{p} = [p_0\ p_1\ \cdots\ p_l]\ . \tag{4.158}$$

Actually, each (C_{jknh}) is also a vector

$$(C_{jknh}) = [b^g_{jknh}]_{g:N_{jk}} = [b^0_{jknh}, b^1_{jknh}, \ldots, b^{N_{jk}}_{jknh}] \tag{4.159}$$

where N_{jk} is the number of elements in U_{jk} and

$$b^g_{jknh} = \begin{cases} 1 & \text{if } g \in (C_{jknh}) \\ 0 & \text{otherwise} \end{cases} \tag{4.160}$$

[see Eq. 4.125], and so the computer representation of f is a five dimensional array

$$\mathbf{C} = [b^g_{jknh}]_{jknhg:r \times s \times l \times p_i \times N_{jk}}\ . \tag{4.161}$$

---

[13]We define a ***ragged array*** to be a structure $[a_{ij}]_{i \in I, j \in J}$ of elements a_{ij} indexed by i and j taking values in the sets I and J, respectively. In the simplest case, $I = \{0, 1, \ldots, m\}$, $J = \{0, 1, \ldots, n\}$, and the resulting structure is the array $[a_{ij}]_{m \times n}$. The indexes i and j can be functions of one another, though we shall use only the case where the maximum value of one index is dependent on a second, relatively independent index, i.e., $[a_{ij}]_{i \in I, j \in J_i}$, where $I = \{0, 1, \ldots, m\}$, and $J_i = \{0, 1, \ldots, n_i\}$. Such an array will be denoted by $[a_{ij}]_{m \times n_i}$ or $[a_{ij}]_{i,j:m \times n_i}$. The concept generalizes in the way to higher dimensional structures.

The representation **F** of f given in Eq. 4.122 can be then written

$$\mathbf{F} = [w_i \ [b^q_{jknh}]_{q:N_{jk}}]_{jknh:r \times s \times l \times p_i} . \tag{4.162}$$

In APL, the length of the vectors C_{jk} is the size of the largest set U_{jk}. In the programming language C, the arrays are implemented more efficiently as structures of pointers and data.

As an illustration, consider the trajectory set of Eq. 4.95, i.e.,

$$\begin{aligned} f^{-1}(0) &= [\,0\ 1\,] \cup [\,1\ 2\,] \cup [\,3\ 0\,] \\ f^{-1}(1) &= [\,0\ \{0,2\}\,] \cup [\,4\ 1\,] \\ f^{-1}(2) &= [\,3\ \{1,2\}\,] \cup [\,4\ 0\,] \\ f^{-1}(3) &= [\,\{1,2\}\ \{0,1\}\,] \cup [\,\{2,4\}\ 2\,] . \end{aligned} \tag{4.95}$$

The vector **p** is

$$\mathbf{p} = [3\ 2\ 2\ 2] \tag{4.163}$$

since there are three cube functions denoting $f^{-1}(0)$ (i.e., have weight 0), and two cube functions representing each of $f^{-1}(1)$, $f^{-1}(2)$, and $f^{-1}(3)$ (i.e., with weights 1, 2, and 3). "Stacking" the arrays of Eq. 4.95,

$$\left[\begin{array}{c} [\,0\ 1\,] \\ [\,1\ 2\,] \\ [\,3\ 0\,] \\ \hline [\,0\ \{0,2\}\,] \\ [\,4\ 1\,] \\ \hline [\,3\ \{1,2\}\,] \\ [\,4\ 0\,] \\ [\,\{1,2\}\ \{0,1\}\,] \\ [\,\{2,4\}\ 2\,] \end{array}\right] \tag{4.164}$$

the array **C** is constructed as below:

$$\mathbf{C} = \begin{bmatrix} [\,[0\,0\,0\,1]\;\;[0\,0\,0\,1\,0]\,]\;] \\ [\,[0\,0\,1\,0]\;\;[0\,0\,1\,0\,0]\,]\;] \\ [\,[1\,0\,0\,0]\;\;[0\,0\,0\,0\,1]\,]\;] \\ [\,[0\,1\,0\,1]\;\;[0\,0\,0\,0\,1]\,]\;] \\ [\,[0\,0\,1\,0]\;\;[1\,0\,0\,0\,0]\,]\;] \\ [\,[0\,0\,1\,0]\;\;[0\,1\,0\,0\,0]\,]\;] \\ [\,[0\,0\,0\,1]\;\;[1\,0\,0\,0\,0]\,]\;] \\ [\,[0\,0\,1\,1]\;\;[0\,0\,1\,1\,0]\,]\;] \\ [\,[0\,1\,0\,0]\;\;[1\,0\,1\,0\,0]\,]\;] \end{bmatrix} \tag{4.165}$$

Note that the computer memory required to store Eq. 4.165 is the same as that of Eq. 4.129 (if Eq. 4.129 also includes the cubes with weight 0), but that the indexing of Eq. 4.165 is more versatile (in the sense of quick access to the h^{th} term with value a_n; see Eq. 4.156) when the domain has two coordinates.

As a somewhat more complex (in terms of dimensioning) example, consider the model hierarchy of Furchtgott [89], Meyer, Ballance, Furchtgott, and Wu [30], Meyer [38], and Meyer [29]. Specifically, we take function γ_1^{-1} of Table 2 in Furchtgott [89], which also appears as Table 3 of Meyer, Ballance, Furchtgott, and Wu [30]:

$$\begin{aligned}
\gamma_1^{-1}(a_0) &= \begin{bmatrix} 0 & 0 \\ * & * \end{bmatrix} \cup \begin{bmatrix} 0 & 1 \\ 0 & * \end{bmatrix} \cup \begin{bmatrix} 1 & \{0,1\} \\ 0 & * \end{bmatrix} \\
\gamma_1^{-1}(a_1) &= \begin{bmatrix} 2 & \{0,1\} \\ 0 & * \end{bmatrix} \cup \begin{bmatrix} 0 & 2 \\ * & * \end{bmatrix} \cup \begin{bmatrix} 0 & 3 \\ 0 & * \end{bmatrix} \cup \begin{bmatrix} 1 & \{2,3\} \\ 0 & * \end{bmatrix} \\
\gamma_1^{-1}(a_2) &= \begin{bmatrix} 1 & \{0,1\} \\ 1 & * \end{bmatrix} \\
\gamma_1^{-1}(a_3) &= \begin{bmatrix} 2 & \{0,1\} \\ 1 & * \end{bmatrix} \cup \begin{bmatrix} 1 & \{2,3\} \\ 1 & * \end{bmatrix} \\
\gamma_1^{-1}(a_4) &= \begin{bmatrix} 3 & * \\ * & * \end{bmatrix} \cup \begin{bmatrix} 2 & \{2,3\} \\ * & * \end{bmatrix} \cup \begin{bmatrix} 0 & 1 \\ \{2,3\} & * \end{bmatrix}
\end{aligned} \tag{4.166}$$

The vector **p** is

$$\mathbf{p} = [3\ 4\ 1\ 2\ 3]\ . \tag{4.167}$$

Again "stacking" the arrays of Eq. 4.166,

$$\left|\begin{array}{c}
\begin{bmatrix} 0 & 0 \\ * & * \end{bmatrix} \\
\begin{bmatrix} 0 & 1 \\ 0 & * \end{bmatrix} \\
\begin{bmatrix} 1 & \{0,1\} \\ 0 & * \end{bmatrix} \\
\hline
\begin{bmatrix} 2 & \{0,1\} \\ 0 & * \end{bmatrix} \\
\begin{bmatrix} 0 & 2 \\ * & * \end{bmatrix} \\
\begin{bmatrix} 0 & 3 \\ 0 & * \end{bmatrix} \\
\begin{bmatrix} 1 & \{2,3\} \\ 0 & * \end{bmatrix} \\
\hline
\begin{bmatrix} 1 & \{0,1\} \\ 1 & * \end{bmatrix} \\
\hline
\begin{bmatrix} 2 & \{0,1\} \\ 1 & * \end{bmatrix} \\
\begin{bmatrix} 1 & \{2,3\} \\ 1 & * \end{bmatrix} \\
\hline
\begin{bmatrix} 3 & * \\ * & * \end{bmatrix} \\
\begin{bmatrix} 2 & \{2,3\} \\ * & * \end{bmatrix} \\
\begin{bmatrix} 0 & 1 \\ \{2,3\} & * \end{bmatrix}
\end{array}\right| \tag{4.168}$$

the array **C** is constructed as below:

$$
\begin{bmatrix} [0\,0\,0\,1] & [0\,0\,0\,1] \\ [1\,1\,1\,1] & [1\,1\,1\,1] \end{bmatrix}
\begin{bmatrix} [0\,0\,0\,1] & [0\,0\,1\,0] \\ [0\,0\,0\,1] & [1\,1\,1\,1] \end{bmatrix}
\begin{bmatrix} [0\,0\,1\,0] & [0\,0\,1\,1] \\ [0\,0\,0\,1] & [1\,1\,1\,1] \end{bmatrix}
\begin{bmatrix} [0\,1\,0\,0] & [0\,0\,1\,1] \\ [0\,0\,0\,1] & [1\,1\,1\,1] \end{bmatrix}
\begin{bmatrix} [0\,0\,0\,1] & [0\,1\,0\,0] \\ [1\,1\,1\,1] & [1\,1\,1\,1] \end{bmatrix}
\begin{bmatrix} [0\,0\,0\,1] & [1\,0\,0\,0] \\ [0\,0\,0\,1] & [1\,1\,1\,1] \end{bmatrix}
\begin{bmatrix} [0\,0\,1\,0] & [1\,1\,0\,0] \\ [0\,0\,0\,1] & [1\,1\,1\,1] \end{bmatrix}
\begin{bmatrix} [0\,0\,1\,0] & [0\,0\,1\,1] \\ [0\,0\,1\,0] & [1\,1\,1\,1] \end{bmatrix}
\begin{bmatrix} [0\,1\,0\,0] & [0\,0\,1\,1] \\ [0\,0\,1\,0] & [1\,1\,1\,1] \end{bmatrix}
\begin{bmatrix} [0\,0\,1\,0] & [1\,1\,0\,0] \\ [0\,0\,1\,0] & [1\,1\,1\,1] \end{bmatrix}
\begin{bmatrix} [1\,0\,0\,0] & [1\,1\,1\,1] \\ [1\,1\,1\,1] & [1\,1\,1\,1] \end{bmatrix}
\begin{bmatrix} [0\,1\,0\,0] & [1\,1\,0\,0] \\ [1\,1\,1\,1] & [1\,1\,1\,1] \end{bmatrix}
\begin{bmatrix} [0\,0\,0\,1] & [0\,0\,1\,0] \\ [1\,1\,0\,0] & [1\,1\,1\,1] \end{bmatrix}
\tag{4.169}
$$

In conclusion, note that with the combined lattice expression/trajectory set notation discussed in this section, $f^{-1}(a)$ can be compactly represented in general discussions using sets of $\mathbf{C}_n$ (as in Eq. 4.153), and, for applications, can be efficiently represented in computer memory using arrays of form Eq. 4.165. As shall be seen in Section 4.3 [Calculation of Trajectory Sets] (and as might be inferred from such results as Eqs. 3.4 and 4.78), the inverses of discrete functions are important quantities (both conceptually and computationally) in the determination of γ^{-1}.

4.3.2. Notation

There are two broad classes of discrete functions with which we shall deal in the remainder of this chapter, namely γ_i (the level-i based capability function; see Sections 3.3.4 [The Capability Function γ], 3.3.8 [The Model Hierarchy], and 4.2.2 [Discrete Functions and Capability Functions]) and κ_i (the level-i interlevel translation; see Section 3.3.8 [The Model Hierarchy]). Below, we describe how these functions are represented by the hybrid lattice expression/trajectory set notation discussed in Section 4.3.1 [Representation of Discrete Functions Within METAPHOR]. Generally, the inverse f^{-1} of function f will be denoted by sets of array products (see Eq. 4.154): $f^{-1}(a)$ is denoted by

$$f^{-1}(a) = \{(C_{(jk)h}) \mid a = w_h\} \tag{4.170}$$

where (in the "extended" disjunctive normal form of Eq. 4.156)

$$F(\mathbf{x}) = \bigvee_{n=0}^{l} a_n \wedge \left[\bigvee_{h=1}^{p_n} \bigwedge_{j=0}^{r} \bigwedge_{k=0}^{s} x_{jk}^{(C_{(jk)nh})} \right] \tag{4.171}$$

and there are $p = \sum_{n=0}^{l} p_n$ terms in the representation and $(C_{(jk)nh})$ is the h^{th} set of entries of weight a_n for the jk^{th} term.

Briefly reviewing Sections 3.3.8 [The Model Hierarchy], 4.1 [Trajectory Sets: Basic Notation and Operations], and 4.2.2 [Discrete Functions and Capability Functions], decompose the level-i trajectory space $\mathbf{U}^i$ into composite and basic trajectory spaces:

$$\mathbf{U}^i = \mathbf{U}_c^i \times \mathbf{U}_b^i . \tag{4.172}$$

Let $\overline{\mathbf{U}}^i$ denote the level-i trajectory space along with all the basic trajectory spaces of higher

level models:

$$\overline{\mathbf{U}}^i = \mathbf{U}^0 \times \mathbf{U}_b^1 \times \cdots \times \mathbf{U}_b^i \tag{4.173}$$

where $\overline{\mathbf{U}}^0 = \mathbf{U}^0$ and $\overline{\mathbf{U}}^m = \mathbf{U}$. $\mathbf{U}_c^{i-1}$ $(i = 1, 2, \ldots, m)$ is related to $\mathbf{U}^i$ by the level-i interlevel translation

$$\kappa_i: \mathbf{U}^i \rightarrow \mathbf{U}_c^{i-1} \quad (i = 1, 2, \ldots, m) \tag{4.174}$$

$$\kappa_0: \mathbf{U}^0 \rightarrow A \ . \tag{4.175}$$

Now

$$\mathbf{U}_c^i = [U_{c_j, t_k}^i]_{jk: r_i \times s_i} \ , \tag{4.176}$$

(see Eq. 4.14), i.e., the level-i composite trajectory space has r_i component coordinates and s_i time coordinates (observations). Also, (see Eq. 4.5)

$$\mathbf{U}^i = [U_{jk}^i]_{jk: (r_i + \rho_i + 1) \times s_i} \ . \tag{4.177}$$

Using the projection function ξ_{jk} (see Section 4.1.1 [Notation and Terminology])

$$\xi_{jk} \mathbf{U}_c^{i-1} = U_{c_j, t_k}^{i-1} \quad (i = 1, 2, \ldots, m) \ , \tag{4.178}$$

κ_i can be decomposed (see also Eq. 4.16)

$$\xi_{jk} \kappa_i: \mathbf{U}^i \rightarrow U_{c_j, t_k}^{i-1} \ . \tag{4.179}$$

We represent κ_0 and the other κ_i as follows:

$$\kappa_0(\mathbf{u}) = \bigvee_{n=0}^{l} a_n \wedge \left[\bigvee_{h=1}^{\kappa_0} \bigwedge_{j=0}^{r_0 + \rho_0 + 1} \bigwedge_{k=0}^{s_0} u_{jk}^{(K_{(jk)nh}^0)} \right] \tag{4.180}$$

$$\xi_{j_0k_0\kappa_i}(\mathbf{u}) = \bigvee_{n=0}^{N_{j_0k_0}^{i}-1} w_n \wedge \left[\bigvee_{h=1}^{p_n^{\xi_{j_0k_0\kappa_i}}} \bigwedge_{j_1=0}^{r_i+\rho_i+1} \bigwedge_{k_1=0}^{s_i} u_{j_1k_1}^{(K_{(j_1k_1)nh}^{i,j_0k_0})} \right] \tag{4.181}$$

$$(i = 1, 2, \ldots, m) .$$

$(K_{(jk)nh}^{0})$ denotes the h^{th} array product (exponents of the cube functions) associated with accomplishment level a_n of κ_0, while $(K_{(j_1k_1)nh}^{i,j_0k_0})$ represents the h^{th} array product (exponents of the cube functions) associated with value w_n of the $j_0k_0^{\text{th}}$ projection of κ_i. Eqs. 4.180 and 4.181 can be written as the four dimensional ragged arrays (see Eq. 4.157)

$$\kappa_0(\mathbf{u}) = \mathbf{K}_0 = \left[K_{(jk)nh}^{0}\right]_{jknh:(r_i+\rho_0+1)\times s_0\times l\times p_n^{\kappa_0}} \tag{4.182}$$

$$\xi_{j_0k_0\kappa_i}(\mathbf{u}) = \mathbf{K}_i^{j_0k_0} = \left[K_{(j_1k_1)nh}^{i,j_0k_0}\right]_{j_1k_1nh:(r_i+\rho_i+1)\times s_i\times N_{j_0k_0}^{i}-1\times p_n^{\xi_{j_0k_0\kappa_i}}} \tag{4.183}$$

$$(i = 1, 2, \ldots, m) ,$$

along with vectors

$$\mathbf{p}^{\kappa_0} = [p_n^{\kappa_0}]_{n:l} = [p_0^{\kappa_0}\ p_1^{\kappa_0}\ \cdots\ p_l^{\kappa_0}] \tag{4.184}$$

$$\mathbf{p}^{\xi_{j_0k_0\kappa_1}} = [p_n^{\xi_{j_0k_0\kappa_i}}]_{n:N_{j_0k_0}^{i}-1} = [p_i^{\xi_{j_0k_0\kappa_i}}\ p_1^{\xi_{j_0k_0\kappa_i}}\ \cdots\ p_{N_{j_0k_0}^{i}-1}^{\xi_{j_0k_0\kappa_i}}] . \tag{4.185}$$

As in Eq. 4.154, the inverses $\kappa_0^{-1}(a)$ and $(\xi_{j_0k_0\kappa_i})^{-1}$ can be written

$$\kappa_0^{-1}(a) = \left\{ \left[K_{(j_1k_1)n}^{i,j_0k_0}\right]_{j_1k_1n:(r_i+\rho_i+1)\times s_i\times N_{j_0k_0}^{i}-1} \;\middle|\; a_n = a \right\} \tag{4.186}$$

$$(\xi_{j_0k_0\kappa_i})^{-1} = \left\{ \left[K_{(j_1k_1)n}^{i,j_0k_0}\right]_{j_1k_1n:(r_i+\rho_i+1)\times s_i\times N_{j_0k_0}^{i}-1} \;\middle|\; w_n = u \right\} \tag{4.187}$$

The ***level-i-based capability function*** is

$$\gamma_i : \overline{\mathbf{U}}^i \rightarrow A \tag{4.188}$$

defined inductively as follows: If $j = 0$ and $\mathbf{u} \in \overline{\mathbf{U}}^0$, then

$$\gamma_0(\mathbf{u}) = \kappa_0(\mathbf{u}) , \tag{4.189}$$

and, for $i > 0$, if $(\mathbf{u}, \mathbf{u}') \in \overline{\mathbf{U}}^i$ where $\mathbf{u} \in \mathbf{U}^i$, $\mathbf{u}' \in \mathbf{U}_b^0 \times \mathbf{U}_b^1 \times \cdots \times \mathbf{U}_b^{i-1}$, then

$$\gamma_i(\mathbf{u}, \mathbf{u}') = \gamma_{i-1}(\kappa_i(\mathbf{u}), \mathbf{u}') . \tag{4.190}$$

If $i = m$, then $\gamma_m = \gamma$.

We shall represent the level-0 capability function γ_0 (Eq. 4.189) as follows:

$$\gamma_0(\mathbf{u}) = \bigvee_{n=0}^{l} a_n \wedge \left[\bigvee_{h=1}^{p_n^{\gamma_0}} \bigwedge_{j=0}^{r_0+\rho_0+1} \bigwedge_{k=0}^{s_0} u_{jk}^{(\Gamma^0_{(jk)nh})} \right] \tag{4.191}$$

where $(\Gamma^0_{(jk)nh})$ denotes the h^{th} array product (exponents of the cube functions) of accomplishment level a_n. γ_0 can then be written as the four dimensional (ragged) array (see Eq. 4.157)

$$\Gamma^0 = [(\Gamma^0_{(jk)nh})]_{jknh:(r_0+\rho_0+1) \times s_0 \times l \times p_n^{\gamma_0}} \tag{4.192}$$

along with a vector

$$\mathbf{p}^{\gamma_0} = [p_n^{\gamma_0}]_{n:l} = [p_0^{\gamma_0}\ p_1^{\gamma_0}\ \cdots\ p_l^{\gamma_0}] . \tag{4.193}$$

sub $\{\{\mathbf{n}\} : \hat{}\ \mathbf{l}\}$ The inverse γ_0^{-1} can be represented

$$\gamma_0^{-1}(a) = \left\{ [(\Gamma^0_{(jk)nh})]_{jkn:(r_0+\rho_0+1) \times s_0 \times l} \;\middle|\; a_n = a \right\} \tag{4.194}$$

It will be convenient to write γ_0 (Eq. 4.191) so that its basic and composite projections

are clear (see Eq. 4.176):

$$\gamma_0(\mathbf{u}) = \bigvee_{n=0}^{l} a_n \wedge \left[\bigvee_{h=1}^{p_n^{\gamma_0}} \left[\bigwedge_{j=0}^{r_0} \bigwedge_{k=0}^{s_0} u_{c_j, t_k}^{0\,(\Gamma^0_{(jk)nh})} \right] \right. \tag{4.195}$$

$$\left. \wedge \left[\bigwedge_{j=0}^{\rho_0} \bigwedge_{k=0}^{s_0} u_{b_j, t_k}^{0\,(\Gamma^0_{(jk)nh})} \right] \right] .$$

In a manner similar to Eq. 4.192, γ_0 can then be denoted by the array

$$\Gamma^0 = [\Gamma_c^0 \;\; \Gamma_b^0] = \left[[(\Gamma^0_{c_{(jk)nh}})]_{jknh:r_0 \times n \times l \times p_n^{\gamma_0}} \quad [(\Gamma^0_{b_{(jk)nh}})]_{jknh:\rho_0 \times n \times l \times p_n^{\gamma_0}} \right] . \tag{4.196}$$

The vector $\mathbf{p}^{\gamma_0}$ is still as in Eq. 4.193.

Then, if $(\mathbf{u}, \mathbf{u}') \in \overline{\mathbf{U}}^1$ (Eq. 4.173), from Eqs. 4.179 and 4.190:

$$\gamma_1(\mathbf{u}, \mathbf{u}') = \bigvee_{n=0}^{l} a_n \wedge \left[\bigvee_{h=1}^{p_n^{\gamma_0}} \left[\bigwedge_{j=0}^{r_0} \bigwedge_{k=0}^{s_0} \right. \right.$$

$$\left. \left[\bigvee_{n_1=0}^{N_{jk}^0} w_{n_1} \wedge \left[\bigvee_{h_1=1}^{p_{n_1}^{\kappa_1}} \bigwedge_{j_1=0}^{r_1+\rho_1+1} \bigwedge_{k_1=0}^{s_1} u_{j_1 k_1}^{(K^{1,jk}_{(j_1 k_1)n_1 h_1})} \right] \right]^{(\Gamma^0_{(jk)nh})} \right. \tag{4.197}$$

$$\left. \left. \wedge \left[\bigwedge_{j=0}^{\rho_0} \bigwedge_{k=0}^{s_0} u_{b_j, t_k}^{0\,(\Gamma^0_{(jk)nh})} \right] \right] \right] .$$

Eq. 4.197 appears quite complex; concisely, the equation is specifying a collection of unions and intersections of level-1 array products (exponents of cube functions). In Section 4.3.3.2 [Intersection Trees], Eq. 4.197 will be discussed in detail. The particular sets which are operated upon are dictated by γ_0 and κ_1. When those operations are complete, Eq. 4.197 can be written in reduced form:

$$\gamma_1(\mathbf{u}, \mathbf{u}') = \bigvee_{n=0}^{l} a_n \wedge \left[\bigvee_{h=1}^{p_n^{\gamma_1}} \left[\bigwedge_{j=0}^{r_1} \bigwedge_{k=0}^{s_1} u_{jk}^{1\,(\Gamma^{11}_{(jk)nh})} \right] \right. \tag{4.198}$$

$$\left. \wedge \left[\bigwedge_{j=0}^{\rho_0} \bigwedge_{k=0}^{s_0} u_{b_j, t_k}^{0\,(\Gamma^{10}_{(jk)nh})} \right] \right]$$

where $(\Gamma^{11}_{(jk)nh})$ denotes the h^{th} array product (exponents of the cube functions) of accomplishment level a_n, which are associated with level-1 trajectories, and $(\Gamma^{10}_{(jk)nh})$ denotes the h^{th} array product associated with level-0 basic trajectories. γ_1 can then be written as the (ragged) array

$$\Gamma^1 = [\,\Gamma^{11} \quad \Gamma^{10}\,]$$

$$= \left[[(\Gamma^{11}_{(jk)nh})]_{jknh:(r_1+\rho_1+1)\times s_1 \times l \times p_n^{\gamma_1}} \quad [(\Gamma^{10}_{(jk)nh})]_{jknh:(r_0+\rho_0+1)\times s_0 \times l \times p_n^{\gamma_0}} \right] \tag{4.199}$$

along with a vector

$$\mathbf{p}^{\gamma_1} = [p_n^{\gamma_1}]_{n:l} = [p_0^{\gamma_1}\ p_1^{\gamma_1}\ \cdots\ p_l^{\gamma_1}]\,. \tag{4.200}$$

The inverse γ_1^{-1} can be written (see Eq. 4.154)

$$\gamma_1^{-1}(a) = \left\{ [(\Gamma^{11}_{(jk)n})]_{jkn:(r_1+\rho_1+1)\times s_1 \times l} \quad [(\Gamma^{10}_{(jk)n})]_{jkn:(r_1+\rho_1+1)\times s_1 \times l} \,\middle|\, a_n = a \right\}. \tag{4.201}$$

Continuing iteratively (see Eq. 4.190) and letting $(\mathbf{u}, \mathbf{u}') \in \overline{\mathbf{U}}^i$, $(i = 1, 2, \ldots, m)$, we obtain for level-i:

$$\gamma_i(\mathbf{u},\mathbf{u}') = \bigvee_{n=0}^{l} a_n \wedge \left[\bigvee_{h=1}^{p_n^{\gamma_{i-1}}} \left[\bigwedge_{j=0}^{r_{i-1}} \bigwedge_{k=0}^{s_{i-1}} \left[\bigvee_{n_1=0}^{N_{jk}^{i-1}} w_{n_1} \wedge \left[\bigvee_{h_1=1}^{p_{n_i}^{\kappa_i}} \bigwedge_{j_1=0}^{r_i+\rho_i+1} \bigwedge_{k_1=0}^{s_i} u_{j_1 k_1}^{(K_{(jk)n_1 h_1}^{i,jk})} \right] \right]^{(\Gamma_{(jk)nh}^{i-1})} \right.\right.$$

$$\left.\left. \wedge \left[\bigvee_{e=0}^{i-1} \bigwedge_{j=0}^{\rho_e} \bigwedge_{k=0}^{s_e} u_{b_j,t_k}^{e\,(\Gamma_{(jk)nh}^{ie})} \right] \right] \right] \tag{4.202}$$

$$\gamma_i(\mathbf{u},\mathbf{u}') = \bigvee_{n=0}^{l} a_n \wedge \left[\bigvee_{h=1}^{p_n^{\gamma_i}} \left[\bigwedge_{j=0}^{r_i} \bigwedge_{k=0}^{s_i} u_{jk}^{i\,(\Gamma_{(jk)nh}^{ii})} \right] \right.$$

$$\left. \wedge \left[\bigvee_{e=0}^{i-1} \bigwedge_{j=0}^{\rho_e} \bigwedge_{k=0}^{s_e} u_{b_j,t_k}^{e\,(\Gamma_{(jk)nh}^{ie})} \right] \right] \tag{4.203}$$

where $(\Gamma_{(jk)nh}^{ii})$ denotes the h^{th} array product (exponents of the cube functions) of accomplishment level a_n, which are associated with level-i trajectories, and $(\Gamma_{(jk)nh}^{ie})$ denotes the h^{th} array product associated with level-e basic trajectories. γ_i can then be written as the (ragged) array

$$\begin{aligned} \Gamma^i &= [\,\Gamma^{ii}\ \Gamma^{i(i-1)}\ \cdots\ \Gamma^{i0}\,] = [\,\Gamma^{ie}\,]_{e:i} \\ &= \left[[(\Gamma_{(jk)nh}^{ii})]_{jkh:(r_i+\rho_i+1)\times s_i \times p_n^{\gamma_i}} \quad [(\Gamma_{(jk)nh}^{ie})]_{jkh:\rho_e \times s_e \times p_n^{\gamma_e}} \right]_{ne:l \times i-1} \end{aligned} \tag{4.204}$$

along with a vector

$$\mathbf{p}^{\gamma_i} = [p_n^{\gamma_i}]_{n:l} = [p_1^{\gamma_i}\ p_1^{\gamma_i}\ \cdots\ p_l^{\gamma_i}]\,. \tag{4.205}$$

Finally, the inverse γ_i^{-1} can be written (see Eq. 4.154)

$$\gamma_i^{-1}(a) = \left\{ \left[(\Gamma^{ie}_{(jk)n}) \right]_{jkne:(r_1+\rho_1+1)\times s_1 \times l \times i} \;\middle|\; a_n = a \right\} . \tag{4.206}$$

Detailed algorithms for computing Γ^i are discussed in Section 4.3.3 [Algorithms].

4.3.3. Algorithms

4.3.3.1. High Level Algorithms

Recall (Section 3.3.6 [Solving for the Performability]) that to obtain the probabilistic

description of the accomplishment set A, the following two-step procedure can be employed:

Algorithm 3.2b: (Algorithm for obtaining the performability of a system)

For each (measurable[14]) set $B \subseteq A$ of accomplishment levels,

METAPHOR Function meta_discrete		
	Step	METAPHOR Function
1)	Determine $\gamma^{-1}(B) = \{\mathbf{u} \mid \gamma(\mathbf{u}) \in B\}$ (3.4) i.e., the set of all trajectories **u** that result in an accomplishment level in the set B.	commandbuild
2)	Determine the probability of the set of trajectories $\gamma^{-1}(B)$.	commandeval

For the present, let us concern ourselves with Step 1), viz., finding $\gamma^{-1}(B)$.

Since A is finite, then to specify Prob(B), it suffices to specify Prob(a) for each $a \in A$; then

$$\text{Prob}(B) = \sum_{a \in B} \text{Prob}(a) \,. \tag{4.207}$$

In terms of trajectory sets, Algorithm 3.2b can be written

[14] Since A is finite, then every subset of A is finite, and so every $B \subseteq A$ is measurable.

Algorithm 3.2c: (Algorithm for obtaining the performability of a system)

METAPHOR Function meta_discrete		
	Step	METAPHOR Function
1)	For each $a \in A$, determine $\gamma^{-1}(a)$, i.e., the $[\Gamma^{ie}_{(jk)n}]$ of Eq. 4.206.	commandbuild
2)	For each $a \in A$, determine the probability of the set of trajectory sets $\gamma^{-1}(a)$.	commandeval

Note that the iteration "for each $a \in A$" has been distributed so that the entire capability function γ^{-1} is determined [step 1)] before any probability calculations are made [step 2)].

Finding $\gamma^{-1}(a)$ [step 1) of Algorithm 3.2c] will be done in a top-down manner. (See Section 3.3.8 [The Model Hierarchy] for details on the model hierarchy employed.) Beginning with the level-0-based capability function, we have (see also Eqs. 3.22 and 3.4)

$$\gamma_0^{-1}(a) = \kappa_0^{-1}(a)\,, \tag{4.208}$$

$$\gamma_i^{-1}(a) = \bigcup_{(\mathbf{u},\mathbf{u}')\,\in\,\gamma_{i-1}^{-1}(a)} (\kappa_i^{-1}(\mathbf{u}),\mathbf{u}')\,,\ m \geq i > 0 \tag{4.209}$$

where $(\kappa_i^{-1}(\mathbf{u}),\mathbf{u}') = \{(\mathbf{v},\mathbf{u}') \mid \kappa_i(\mathbf{v}) = \mathbf{u}\}$.

The algorithm can thus be stated iteratively:

Algorithm 4.2: (Algorithm for constructing γ^{-1})

METAPHOR Function commandbuild		
	Step	**METAPHOR Function**
1)	Build the top levels of the hierarchy:	commandbuild
	a) Specify the accomplishment set A.	getacclev
	b) Specify level-0, i.e., $\mathbf{U}^0$ (see Section 3.4 [Models Having Finite Performance Variables])	getlevel0
	c) Specify level-1, i.e., $\mathbf{U}^1$.	getlevel1
	d) In terms of trajectory sets, specify γ_0^{-1} (κ_0^{-1}), i.e., the inverse of the level-0 capability function. This is done by stating the $[\Gamma_{(jk)nh}]$ of Eq. 4.192 and the $p_n^{\gamma_0}$ of Eq. 4.193.	getlcarray
	e) Specify κ_1^{-1}, i.e., the inverse of the level-1 to level-0 interlevel translation. This is done by stating the $[K^0_{(jk)nh}]$ of Eq. 4.182 and the $p_n^{\kappa_0}$ of Eq. 4.184.	getkarray
	f) Using γ_0^{-1} and κ_1^{-1}, calculate γ_1^{-1}, i.e., the inverse of the level-1 capability function (Eq. 4.208, and more specifically, Eq. 4.206).	intarraysets
2)	Build each of the remaining levels of the hierarchy, i.e., for $i = 2$ to m:	commandnext
	a) Specify level-i, i.e., $\mathbf{U}^i$.	getlevel1
	b) Specify κ_i^{-1}, i.e., the inverse of the level-i to level-$(i-1)$ interlevel translation. This is done by stating the $[K^{i,j_0^k0}_{(jk)nh}]$ of Eq. 4.183 and the $p_n^{\xi_{j_0^k0}^{\kappa_i}}$ of Eq. 4.185.	getkarray
	c) Using γ_i^{-1} and κ_i^{-1}, calculate γ_{i+1}^{-1}, i.e., the inverse of the level-$(i+1)$ capability function (Eq. 4.209).	intarraysets

Most of Algorithm 4.2 consists of the relatively straightforward actions of inputing data and checking the consistency of that data. The computational heart of the algorithm is Steps

1)f) and 2)c). Each of the functions in Algorithm 4.2 will be discussed in detail below.

4.3.3.2. Intersection Trees

Algorithm 4.3: (Basic algorithm for constructing γ_i^{-1})

METAPHOR Function intarraysets		
	Step	METAPHOR Function
For each $a \in A$: (for $n = 0$ to l:)		
1)	Consider each level-$(i-1)$ cube function associated with $\gamma_{i-1}(a)$, recursively calculating $\gamma_i^{-1}(a)$, i.e., the set of level-i array products that map into a. (for $h = 1$ to $p_n^{\gamma_{i-1}}$: find $[(\Gamma^{i}_{(jk)n})]_{jkne:(r_1+\rho_1+1)\times s_1 \times l \times i}$ such that $a_n = a$ (Eq. 4.206).)	buildtree
2)	Reduce (if possible) the results of step 1) to obtain a smaller set for Eq. 4.206.	reducelcarray

Step 1) works by constructing a tree of partial array product intersections. Recall Eq. 4.202:

$$\gamma_i(\mathbf{u},\mathbf{u}') = \bigvee_{n=0}^{l} a_n \wedge \left[\bigvee_{h=1}^{p_n^{\gamma_{i-1}}} \left[\bigwedge_{j=0}^{r_{i-1}} \bigwedge_{k=0}^{s_{i-1}} \right.\right.$$

$$\left[\bigvee_{n_1=0}^{N_{jk}^{i-1}} w_{n_1} \wedge \left[\bigvee_{h_1=1}^{p_{n_1}^{n_i}} \bigwedge_{j_1=0}^{r_i+\rho_i+1} \bigwedge_{k_1=0}^{s_i} u_{j_1 k_1}^{(K^{i}_{(j_1 k_1)n_1 h_1})} \right] \right]^{(\Gamma^{i-1}_{(jk)nh})} \tag{4.202}$$

$$\left.\left. \wedge \left[\bigvee_{e=0}^{i-1} \bigwedge_{j=0}^{\rho_e} \bigwedge_{k=0}^{s_e} u_{b_j, t_k}^{e\,(\Gamma^{e}_{(jk)nh})} \right] \right] \right]$$

As can be seen from Eq. 4.202, each level-$(i-1)$ array product [denoted by the $(\Gamma^{i-1}_{(jkk)nh})$ in the

exponent of the middle line] must be replaced, [level-$(i-1)$] componentwise, with the corresponding cube functions in κ_i [denoted by the terms within the outer set of braces in the middle line, i.e.,

$$\bigvee_{n_1=0}^{N_{jk}^{i-1}} w_{n_1} \wedge \left[\bigvee_{h_1=1}^{p_{n_1}^{\kappa_i}} \bigwedge_{j_1=0}^{r_i+\rho_i+1} \bigwedge_{k_1=0}^{s_i} u_{j_1 k_1}^{(K_{(j_1 k_1)n_1 h_1}^{i,jk})} \right] \quad (4.210)$$

These componentwise replacements generate level-i array products, which in turn, can be (level-i) componentwise intersected.

Rather than doing the intersections from scratch for each new level-i array product which we calculate, we shall "telescope" the intersections: first, we shall take one of the cube functions involving level-$(i-1)$ projection $(0, 0)$); then, we shall take one of the cube functions involving level-$(i-1)$ projection $(0, 1)$), intersect it with the previous set, and save it (if the intersection is non-null). We continue taking one cube function from each level-$(i-1)$ projection, intersecting it with the previously derived cube function, and saving the result until we have reached level-$(i-1)$ projection (r_{i-1}, s_{i-1}). This (combined with the basic part from higher levels, i.e., the last line of Eq. 4.202) is a level-i array product. Then we take the next cube function from projection (r_{i-1}, s_{i-1}) and intersect it with the saved result from projection (r_{i-1}, s_{i-2}). We continuing taking cube functions from projection (r_{i-1}, s_{i-1}) until they are exhausted. Then we back down to projection (r_{i-1}, s_{i-2}), take the next cube function from there, and repeat the above procedure. This entire procedure of traversing back and forth among level-$(i-1)$ projections and taking intersections is continued until we return to $(0, 0)$ and there are no more cube function at that level.

Of course, what we have just long-windedly described is the recursive traversal of a tree. The tree has the following interpretation: depth[15] d of the tree denotes a specific level-$(i-1)$ projection (j, k); the root corresponds to projection $(0, 0)$, depth 1 corresponds to

[15]The *depth* d of a node of a tree is the distance (number of arcs) to the root. For example, the root has depth 0, any nodes connected by a single arc to the root have depth 1, etc.

projection (0, 1), and so forth. Generally, depth d represents projection

$$(j, k) = (d \text{ div } (s_{i-1}+1), d \text{ mod } (s_{i-1}+1)) , \tag{4.211}$$

where "div" is integer division. Each node of the tree has associated a level-i "partial" cube function (partial in the sense that the cube function is not necessarily part of γ_i). For the h^{th} node at depth d

$$\bigwedge_{j=0}^{r_i+\rho_i+1} \bigwedge_{k=0}^{s_i} u_{jk}^{(\Delta_{(jk)}^{dh})} \tag{4.212}$$

which we shall sometimes write as

$$\Delta^{dh} = [(\Delta_{(jk)}^{dh})]_{jk:(r_i+\rho_i+1)\times s_i} . \tag{4.213}$$

The root contains the "full cube function"

$$(\Delta_{(jk)}^{00}) = U_{jk} . \tag{4.214}$$

The node Δ^{dhn} at the end of the n^{th} branch (i.e., the n^{th} child) of the h^{th} node $\Delta^{(d-1)h}$ at depth $d-1$ is obtained by the componentwise intersection of the n^{th} cube function of κ_i with the cube function associated with the node $\Delta^{(d-1)h}$. Thus, we have $(j, k) = (d \text{ div } (s_{i-1}+1), d \text{ mod } (s_{i-1}+1))$, and for a specific $\Gamma_{(jk)nh}^{i-1}$,

$$\bigwedge_{j_1=0}^{r_i+\rho_i+1} \bigwedge_{k_1=0}^{s_i} u_{j_1k_1}^{(\Delta_{U_{j_1k_1}}^{dh})} = \left[\bigwedge_{j_1=0}^{r_i+\rho_i+1} \bigwedge_{k_1=0}^{s_i} w_n \wedge u_{j_1k_1}^{(K_{U_{j_1k_1}}^{i,jk})} \right]^{(\Gamma_{(jk)nh}^{i-1})} , \tag{4.215}$$

where w_n is the weight of the n^{th} cube function of κ_i. (Compare Eq. 4.215 with the middle line of Eq. 4.202.)

Thus, we are constructing a tree of level-i cube functions. Call this tree the ***intersection tree*** of the level-$(i-1)$ array products $[\Gamma_{(jk)nh}^{i-1}]_{jk:r_i\times s_i}$. The leaves at depth

$(r_i + \rho_i + 2)^*(s_i + 1)$ are the cube functions specified by the following term of Eq. 4.202:

$$\left(\bigwedge_{j=0}^{r_i-1} \bigwedge_{k=0}^{s_i-1} \left(\bigvee_{n_1=0}^{N_{jk}^{i-1}} w_{n_1} \wedge \left(\bigvee_{h_1=1}^{p_{n_1}^{\kappa_i}} \bigwedge_{j_1=0}^{r_i+\rho_i+1} \bigwedge_{k_1=0}^{s_i} u_{j_1 k_1}^{((K_{j_1 k_1}^{i,jk})_{n_1 h_1})} \right) \right) \right)^{(\Gamma_{jk}^{i-1})_{nh}} \tag{4.216}$$

By ***evaluating*** an intersection tree, we mean determining the lowest leaves, i.e., the values of Eq. 4.216.

Finally, once any given leaf of the tree has been determined, the corresponding basic components of the underlying level-(i–1) cube function (i.e., the last line of Eq. 4.202) can be adjoined (the $\wedge$ operation).

Note that unlike, say, a parse tree, we are simultaneously constructing and evaluting the intersection tree. We can therefore toss away a "partial" cube function (Eqs. 4.212-4.213) once we have constructed and evaluated all nodes below it. Indeed, we never have to store more than a single chain of partial cube functions (as we go down the tree), and that chain's length is at most the depth of the tree. (Of course, coming back up the tree, we must store any newly computed level-i cube functions.)

We are now prepared to state in more detail how an intersection tree is constructed and

evaluated. The following algorithm is called from intarrays as "buildtree(Δ^{00}, 0)."

Algorithm 4.4: (Recursive algorithm for constructing and evaluating an intersection tree)

METAPHOR Function buildtree(Δ^d, d)	
Comment	Step
a) Initialize this level's result to null.	a) result $\leftarrow \Phi$;
b) Consider each level-i cube function associated with the (nonbasic) jk^{th} projection of κ_i.	b) **for** $n_1 = 0$ **to** N_{jk}^{i-1} **do**
c) For each cube function (4.217) $[K_{(j_1k_1)n_1h_1}^{i,jk}]_{j_1k_1:(r_i+\rho_i+1)\times s_i}$ with weight w_{n_1} in $\Gamma_{(jk)nh}^{i-1}$:	c) **if** $w_{n_1} \in \Gamma_{(jk)nh}^{i-1}$ **then** **for** $h_1 = 0$ **to** $p_{n_i}^{\kappa_i}$ **do begin**
d) Do componentwise intersections of Δ^d and Eq. 4.217 to form the child $\Delta^{(d+1)h_1n_1}$.	d) $\Delta^{(d+1)h_1n_1} \leftarrow \Delta^d \cap [K_{(j_1k_1)n_1h_1}^{i,jk}]_{j_1k_1:(r_i+\rho_i+1)\times s_i}$;
e) If this is not the bottom level, do the next level with the child of step c). Save the result above along with all the higher level basic components of the level-(i–1) cube function of Eq. 4.217, i.e., the bottom line of Eq. 4.202.	e) **if** $d \neq (r_i + \rho_i + 2)^*(s_i + 1)$ **then** result $\leftarrow$ [[result] [buildtree($\Delta^{(d+1)h_1n_1}$, d+1) $[(\Gamma_{(jk)n}^{ie})]_{jke:\rho_e \times s_e \times i-1}$]] **else** result $\leftarrow$ [[result] $\Delta^{(d+1)h_1n_1}$] **end**;
f) Return the result.	f) **return** result **end**;

Of course, many refinements can be made to Algorithm 4.4. In particular, we need not continue constructing the tree below any given node if that node is null, i.e., the result of the intersection of step c) is a null array. Also, as discussed above, at a given level, we can compute all the children of that node "before" proceeding down the tree. Then, we can attempt to simplify those children to derive a smaller set of children to traverse. Another technique that can possibly lower the computation time is to reduce the cube functions in

$[K^{i,jk}_{(j_1k_1)n_1}]_{j_1k_1:(r_i + \rho_i + 1) \times s_i}$ before doing intersections with Δ^d [step c)]. The point of this latter step is to reduce the number of cube functions that will appear in the collection of children. Thus, we have the following algorithm:

Algorithm 4.4b: (Algorithm for constructing and evaluating an intersection tree)

METAPHOR Function buildtree(Δ^d, d)

Comment	Step
a) Initialize this level's result to null.	a) result $\leftarrow \Phi$; $\leftarrow \Phi$;
b) Consider each level-i cube function associated with the (nonbasic) jk^{th} projection of κ_i.	b) **for** $n_1 = 0$ **to** N_{jk}^{i-1} **do**
c) For each cube function (4.218) $[K^{i;jk}_{(j_1k_1)n_1h_1}]_{j_1k_1:(r_i+\rho_i+1)\times s_i}$ with weight w_{n_1} in $\Gamma^{i-1}_{(jk)nh}$:	c) **if** $w_{n_1} \in \Gamma^{i-1}_{(jk)nh}$ **then** **for** $h_1 = 0$ **to** $p^{\kappa_i}_{n_i}$ **do**
d) Remember the cube function satisfying step c).	d) K $\leftarrow$ [[K] $[K^{i;jk}_{(j_1k_1)n_1h_1}]_{j_1k_1:(r_i+\rho_i+1)\times s_i}$];
e) Reduce the cube functions stored in K.	e) K $\leftarrow$ reducearray(K);
f) Note how many reduced cube functions there are.	f) N $\leftarrow$ <number of cube functions in reduced K>;
g) For each cube function in K:	g) **for** $n_1 = 1$ **to** N **do begin**
h) Do componentwise intersections of Δ^d and the reduced versions of Eq. 4.218 to form the child $\Delta^{(d+1)n_1}$.	h) $\Delta^{(d+1)n_1} \leftarrow \Delta^d \cap K_{n_1}$;
i) If this is not the bottom level, do the next level with the child of step c). Save the result above along with all the higher level basic components of the level-$(i-1)$ cube function of Eq. 4.218, i.e., the bottom line of Eq. 4.202.	i) **if** $d \neq (r_i + \rho_i + 2)*(s_i + 1)$ **then** result $\leftarrow$ [[result] [buildtree($\Delta^{(d+1)n_1}$, $d+1$) $[(\Gamma^{i\epsilon}_{(jk)n})]_{jk\epsilon:\rho_\epsilon \times s_\epsilon \times i-1}$]] **else** result $\leftarrow$ [[result] $\Delta^{(d+1)n_1}$] **end**;
j) Return the result.	j) **return** result **end**;

4.3.3.3. Reduction

Several algorithms described above (Algorithms 4.3 and 4.4) specify that a set of cube functions be "reduced" or made smaller. In this section, we describe what the operation of "reduction" on cube functions means and how it is performed.

To motivate this section, recall that in switching theory, two commonly performed operations are 1) finding the prime implicants of a switching function, and 2) from the resulting set of prime implicants, finding an optimal (under some cost criterion) cover of the function. The Quine-McCluskey algorithm [185], [186], and less commonly, consensus algorithms (Quine [185]; see also Tison [187]), are typically employed to solve the first problem. Various covering algorithms are used to attack the second.

There is a similar problem when representing discrete functions. We desire to find an "optimal" representation of a discrete function. In Section 4.2.3.2.5 [Classes and Properties of Lattice Expressions], the correspondence between cube functions and implicants in switching theory is noted. Therefore, one might conjecture that the representation problem can be solved by appropriate extensions of the prime implicant extraction algorithms and associated covering algorithms. This is indeed the case. Some previous research addresses such extensions. For example, work by Tison [188] examines generalized consensus for discrete functions, while Davio, Deschamps, and Thayse [165] discuss a generalized Quine-McCluskey method and consensus.

As discussed in Section 3.4.3 [Algorithms for Calculating Trajectory Sets], much of the above theory [165], [188] concerns the optimization of the space required to represent discrete functions. That is because in many applications, such as the design of switching circuits, discrete functions do not need to be calculated repeatedly. Hence, the cost of the time required to compute a spatially optimal representation is slight, compared to the cost of the space itself. However in the application considered here, we are computing many representations, any one of which is relatively ephemeral. If we are not careful, the expense of

searching for a spatially optimal function may outweigh the advantage of possessing such a optimization. Our approach shall be to find a sub-optimal representation as quickly as possible.

Unfortunately, the problem of finding a sub-optimal (or even an optimal) disjoint representation cannot be solved by straightforward extensions of, say, the Quine-McCluskey algorithm, since there is a constraint that prevents us from using "prime implicants" or "prime cube functions." The constraint is that the cube functions must be disjoint so that when probabilities are computed (using the cube functions as events), no event is measured twice. Instead of using prime implicants, we shall define a "maximal disjoint cube function" and specify an algorithm similar to the Quine-McCluskey algorithm. We need not worry about a covering algorithm because the set of maximal disjoint cube functions which are derived will all be necessary to cover the discrete function, and therefore, there will be no redundant cube functions.

We begin by specifying the cost for a representation of a discrete function: The ***cost*** of a normal disjunctive form of a discrete function is the number of cube functions (minterms) in the representation. For example,

$$c(\mathbf{x}) = \bigvee_{h=1}^{p_n} \bigwedge_{j=0}^{r} \bigwedge_{k=0}^{s} x_{jk}^{(C_{(jk)nh})} \tag{4.219}$$

has cost p_n because $c(\mathbf{x})$ has p_n cube functions.

The set of prime implicants of a function is unique and so the cost of the disjunction of all of its prime implicants is fixed. On the other hand, we shall find that a discrete function may have many sets of maximal disjoint cube functions, and that the cost of such sets may vary. Therefore, to find the optimal maximal disjoint cube function representation, we may have to determine all such representations and choose the least costly one. If, however, it is somehow relatively "expensive" to determine all such representations, and if the variance of the cost of the representations is small, i.e., there is not a significant difference in cost among representations, we may be satisfied with a sub-optimal representation, i.e., one with a higher

(hopefully not much higher) cost than the optimal. This is the case for our application. Our criterion for choosing a set of maximal disjoint cube functions is straightforward: We choose the first such set we can find. We shall not formally analyze the above problem; the intuitive argument should, however, explain the reasoning.

Consider two cube functions

$$c_0(\mathbf{x}) = \bigwedge_{j=0}^{r} \bigwedge_{k=0}^{s} x_{jk}^{(C^0_{(jk\,nh)})}$$
$$c_1(\mathbf{x}) = \bigwedge_{j=0}^{r} \bigwedge_{k=0}^{s} x_{jk}^{(C^1_{(jk)})} . \tag{4.220}$$

If $(C^0_{(jk)}) = (C^1_{(jk\,nh)}) = (C_{(jk\,nh)})$ for all j, k except one pair $j'\ k'$, then

$$c_0(\mathbf{x}) \vee c_1(\mathbf{x}) = \wedge \left[x_{jk}^{(C^0_{(j'\,k')})} \vee x_{jk}^{(C^1_{(j'\,k')})} \right] \left[\bigwedge_{\substack{j=0 \\ j \neq j'}}^{r} \bigwedge_{\substack{k=0 \\ k \neq k'}}^{s} x_{jk}^{(C^1_{(jk)})} \right] . \tag{4.221}$$

Recalling the identity (Eq. 4.111)

$$x_j^{(\bigcup_h C_{(jh)})} = \bigvee_h x_j^{(C_{(jh)})} , \tag{4.112}$$

we have

$$c_0(\mathbf{x}) \vee c_1(\mathbf{x}) = \left[x_{jk}^{(C^0_{(j'\,k')}) \cup (C^1_{(j'\,k')})} \right] \wedge \left[\bigwedge_{\substack{j=0 \\ j \neq j'}}^{r} \bigwedge_{\substack{k=0 \\ k \neq k'}}^{s} x_{jk}^{(C_{(jk)})} \right] \tag{4.222}$$

and setting

$$(C_{(j'\,k')}) = (C^0_{(j'\,k')}) \cup (C^1_{(j'\,k')}) \tag{4.223}$$

we find that the two cube functions of Eq. 4.220 can be "squeezed" into a single cube func-

tion

$$c_0(\mathbf{x}) \vee c_1(\mathbf{x}) = \bigwedge_{j=0}^{r} \bigwedge_{k=0}^{s} x_{jk}^{(C_{(jk)})} \tag{4.224}$$

$$= c(\mathbf{x}) \ .$$

In words, the disjunction of two cube functions whose exponents are identical componentwise, except for possibly one component, can be written as a single cube function. Eq. 4.224 is the basis for a ***reduction*** operation *:

$$c_0(\mathbf{x}) * c_1(\mathbf{x}) = \begin{cases} c_0(\mathbf{x}) & \text{if every } (C^0_{(jk)}) = (C^1_{(jk)}) \\ c(\mathbf{x}) & \text{if every } (C^0_{(jk)}) = (C^1_{(jk)}) \text{ except one} \\ 0 & \text{otherwise.} \end{cases} \tag{4.225}$$

Let

$$F(\mathbf{x}) = \bigvee_{h=1}^{p} \bigwedge_{j=0}^{r} \bigwedge_{k=0}^{s} x_{jk}^{(C_{(jk)_h})} \ . \tag{4.226}$$

Then each cube function $x_{jk}^{(C_{(jk)_a})}$ is a ***maximal disjoint cube function*** if for every pair of cube functions $x_{jk}^{(C_{(jk)_a})}$ and $x_{jk}^{(C_{(jk)_b})}$,

$$x_{jk}^{(C_{(jk)_a})} * x_{jk}^{(C_{(jk)_b})} = 0 \tag{4.227}$$

and

$$x_{jk}^{(C_{(jk)_a})} \wedge x_{jk}^{(C_{(jk)_b})} = 0 \ . \tag{4.228}$$

That is, the cube functions cannot be reduced by Eq. 4.224 and are pairwise disjoint.

In an earlier paragraph, the claim was made that there may be more than one representation of a discrete function whose cube functions are maximal disjoint. Consider the simple switching function whose Karnaugh map is shown in Figure 4.3. The cube functions (min-

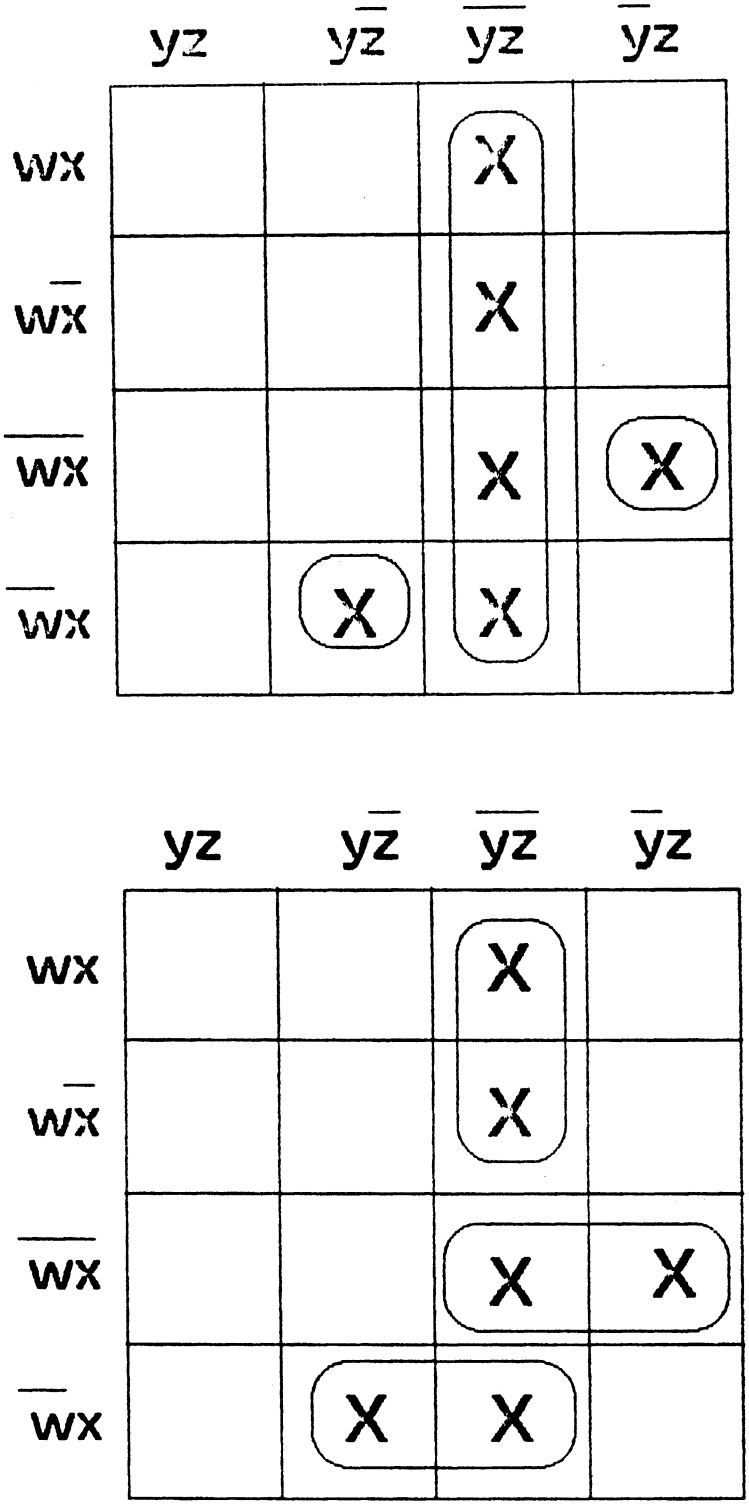


Fig. 4.3 Example of a switching function with multiple maximal disjoint cube function representations

terms) are noted by the rounded boxes. Clearly, the representations in each diagram are disjoint and maximal and represent the same function, yet are not the same representations.

We are now in a position to describe how to take a disjunctive normal form of a function (e.g., Eq. 4.228) whose cube functions are mutually disjoint and extract from it a set of maximally disjoint cube functions. Note the assumption that we are initially presented with mutually disjoint cube functions. This is a valid assumption for our algorithms since all representations are always kept disjoint. Basically, the algorithm selects the first cube function and compares it with each other cube function. If the two cube functions can be reduced (Eq. 4.224), they are reduced; the first cube function is replaced by the reduction and the other cube function is deleted. When all the other cube functions have been compared, the algorithm selects the second cube function and compares it with all the other cube functions except the first. Again, reductions are made if possible. The algorithm continues selecting one cube function and comparing it with the remaining cube functions until all cube functions (except the last) have been selected as the "first" cube function.

Algorithm 4.5: (Algorithm for determining a set of maximal disjoint cube functions)

METAPHOR Function reducetraj	
Comment	Step
a) Pretend changes have been made. b) While it is still possible to have reductions, keep repeating the algorithm: c) Reset the counter noting changes have been made. d) Consider each cube function (except the last) as the "first." e) Consider each of the "remaining" cube functions. f) If the second cube function exists, and if the two cube functions can be compressed (Eq. 4.224): g) then compress them together. h) Note that we lost a cube function.	a) changeflag ← 1; b) **while** changeflag = 1 **do begin** c) changeflag ← 0; d) **for** $h_1 = 0$ **to** $p-1$ **do** e) **for** $h_2 = h_1 + 1$ **to** p **do** f) **while** $h_2 < p$ **and** compresstest(K_{h_1}, K_{h_2}) **do begin** g) $K \leftarrow$ squeeze($K, K_{h_1}, K_{h_2}, h_1, h_2$); h) $p \leftarrow p - 1$ **end** **end**;

When the above process completes, if any two cube functions were reduced, the process is repeated from the beginning until no reductions are made during a pass.

The two algorithms used in reducetraj (compresstest and squeeze) are straightforward:

Algorithm 4.6: (Algorithm for detecting whether two cube functions can be compressed)

METAPHOR Function compresstest(K_0, K_1)	
Comment	**Step**
a) Note that no matching components have been found.	a) found ← 0;
b) consider each of the first dimension coordinates.	b) **for** j = 0 **to** r **do**
c) consider each of the second dimension coordinates.	c) **for** k = 0 **to** s **do**
d) Check if the exponents are identical, componentwise.	d) **if** $(C_{jk}^0) \neq (C_{jk}^1)$ **then**
e) If not, see if we have any other nonequal exponents. If we have, then these two cube functions cannot be compressed, so return false immediately.	e) **if** found = 1 **then return false**
f) Otherwise, these cube functions may potentially be compressible. Note that have found dissimliar exponents:	f) **else begin**
g) and store their coordinate so that if we do squeeze the two cube functions, we do not have to recalculate the coordinate.	found ← 1;
h) If we got this far without returning, then the cube functions can be compressed, either a single coordinate is different or the two cube functions are identical. Either way, return "true."	g) <*squirrel away j and k*>
	end
	else;
	h) **return true;**
	end;

Algorithm 4.7: (Algorithm for compressing two cube functions)

METAPHOR Function squeeze($K, K_{h_1}, K_{h_2}, h_1, h_2$)	
Comment	Step
a) Take the union of the two differing exponents and use it to replace the exponent of the first cube function. The values of j and k were stored when compresstest (Algorithm 4.6) was executed. b) Delete the second cube function.	a) $(C^{h_1}_{(jk)}) \leftarrow (C^{h_1}_{(jk)}) \cup (C^{h_2}_{(jk)})$ b) <*purge K of* K_{h_2}> **end;**

Algorithm 4.3 refers to a METAPHOR function called reducelctraj and Algorithm algogetktraj (to be defined in Section 4.3.3.4 [Consistency Checking Algorithms]) refers to a METAPHOR function called reducektraj. At level-i, reducelctraj considers for each accomplishment level a, the set $\gamma_i^{-1}(a)$, and tries to reduce that set. Similarly, reducektraj attempts to reduce $(\xi_{j_k}\kappa_i)^{-1}(\mathbf{u})$ (see Eq. 4.16). These two functions are specific (and slightly more effecient) instances of the more global function reducetraj (Algorithm 4.5); the major differences are in the looping.

4.3.3.4. Consistency Checking Algorithms

The high level Algorithm 4.2 describes the basic information which must be input to METAPHOR, mainly descriptions of the model hierarchy and the interlevel translations κ_i. It is important that the input information be correct and consistent. As with any algorithm or program, the correct answer can be produced only if the data is correct. There are several algorithms that METAPHOR can employ to insure that the input is consistent; the user has responsibility for the correctness of the input.

Consider the high-level input algorithms (see Algorithm 4.2) for specifying the model hierarchy. The first three (getacclev, getlevel0, and getlevel1) are straightforward:

Algorithm 4.8: (Algorithm for obtaining the accomplishment level)

METAPHOR Function getacclev		
	Step	METAPHOR Function
1)	Fetch the number of accomplishment levels	
2)	Fetch the name of each accomplishment level	getname

Algorithm 4.9: (Algorithm for obtaining the specification for level-0)

METAPHOR Function getlevel0		
	Step	METAPHOR Function
1)	Get the attributes for leve-0	getattr0val
2)	Get the phases for level-0	getphase0val
3)	Get the basic variables for level-0	getbasicval

Algorithm 4.10: (Algorithm for obtaining the specification for level-1)

METAPHOR Function getlevel1		
	Step	METAPHOR Function
1)	Get the attributes for level-1	getattr1val
2)	Get the phases for level-1	getphase1val

Each of the functions getattr0val, getattr0val, getbasicvar, getattr1val, and getattr1val are similar to getacclev (Algorithm 4.8).

The function that inputs $\gamma_0 = \kappa_0$ is getlctraj:

Algorithm 4.11: (Algorithm for obtaining $\gamma_0 = \kappa_0$)

METAPHOR Function getlctraj		
	Step	METAPHOR Function
1)	For each accomplishment level a, get the number of cube functions in $\gamma_0^{-1}(a)$.	getlccount
2)	For each accomplishment level a, get the cube functions in $\gamma_0^{-1}(a)$. (A scanner and parser.)	getlctrajsets
3)	Check that the cube functions input in step 2) describe (according to the conventions of Section 4.3.1 [Representation of Discrete Functions Within METAPHOR]) a (possibly non-total) function, i.e., that there is no point $\mathbf{u} \in \mathbf{U}^0$ that maps into more than one accomplishment level. This involves checking that no point appears in more than one cube function.	checklctraj
4)	Check that the cube functions input in step 2) describe (according to the conventions of Section 4.3.1 [Representation of Discrete Functions Within METAPHOR]) a total function, i.e., that every point $\mathbf{u} \in U^0$ is defined. This involves checking that every point appears in at least one cube function.	checktotal
5)	If the user desires, reduce γ_0.	reducelctraj

The function getktrajset is similar to getlctraj and fetches the level-i interlevel translations

Algorithm 4.12: (Algorithm for obtaining κ_1)

METAPHOR Function getktraj		
	Step	METAPHOR Function
1)	For each possible value u of each attribute of each phase of the level-0 trajectory space (i.e., each value of each projection of $\mathbf{U}^0$), get the number of cube functions in $(\xi_{jk}\kappa_1)^{-1}(u)$.	getkcount
2)	For each possible value u of each attribute of each phase of the level-0 trajectory space (i.e., each value of each projection of $\mathbf{U}^0$), get the of cube functions in $(\xi_{jk}\kappa_1)^{-1}(u)$. (A scanner and parser.)	getktrajsets
3)	Check that the cube functions that were input in step 2) describe (according to the conventions of Section 4.3.1 [Representation of Discrete Functions Within METAPHOR]) a (possibly non-total) function, i.e., that there is no point $\mathbf{u} \in \mathbf{U}^1$ that maps into more than u at level-0. This involves checking that no point appears in more than one cube function.	checkktraj
4)	Check that the cube functions input in step 2) describe (according to the conventions of Section 4.3.1 [Representation of Discrete Functions Within METAPHOR]) a total function, i.e., that every point $\mathbf{u} \in U^0$ is defined. This involves checking that every point appears in at least one cube function.	checktotal
5)	If the user desires, reduce κ_1.	reducektraj

Note that only a function getlevel1 (but no getlevel2 is defined (see Algorithm 4.2). Also note that we compute γ_1 before we examine κ_{i+1} [step 2) of Algorithm 4.2]. Once the function γ_1 has been determined, we can "reorganize" the hierarchy as follows: discard all references to level-0 and γ_0, rename the "old" level-1 to be the "new" level-0, rename the "old" γ_1 to be the "new" γ_0, and continue as though the next level of the model is level-1. Thus, internally to METAPHOR, we never progress beyond level-1. Hence, we do not require any functions that deal with more than two levels (level-0 and level-1).

Consider the function checklctraj (see Algorithm 4.11). checklctraj verifies that the given cube functions describe a "legal" γ_0 in the sense that no trajectory $\mathbf{u} \in \mathbf{U}^0$ is mapped into more than one accomplishment level. This is checked by making sure the various cube functions are mutually exclusive. The technique employed is direct: Take the pairwise meet of each cube function with every other cube function and make sure that the result is null. If the result is non-null, then the cube functions do not describe a function. METAPHOR will print the offending two cube functions. If the user is requesting that reduction of the input cube functions is to be made, then the cube functions associated with a given accomplishment level do not have to be mutually exclusive (since the reduction function will produce mutually exclusive cube functions) and so no checking is performed between those cube functions. Recall Eq. 4.191:

$$\gamma_0(\mathbf{u}) = \bigvee_{n=0}^{l} a_n \wedge \left[\bigvee_{h=1}^{p_n^{\gamma_0}} \bigwedge_{j=0}^{r_0+\rho_0+1} \bigwedge_{k=0}^{s_0} u_{jk}^{(\Gamma^0_{(k)nh})} \right] \tag{4.191}$$

Then

Algorithm 4.13: (Algorithm for checking that $\gamma_0(\mathbf{u})$ is unique)

METAPHOR Function checklctraj	
Comment	Step
a) Consider each accomplishment level. b) Consider each cube function associated with that accomplishment level. c) Consider each accomplishment level up to the present one. d) Consider each cube function associated with the accomplishment level of step c). e) Assume the two cube functions of steps b) and d) are not mutually exclusive. f) Consider each component (attribute): g) We can quit if the meet of the cube functions are null h) Consider each phase: i) We can quit if the meet of the cube functions are null j) See if the intersection of the compenents is null. If so, then the meet is empty. k) If the cube functions are not mutually exclusive, then process the error. l) If the cube functions are to be reduced (See Algorithm 4.5), then we need not worry about the mutual exclusiveness of cube functions within the same accomplishment level. Otherwise repeat steps b) through k) for the current accomplishment level.	a) **for** $n_1 = 0$ **to** l **do** b) **for** $h_1 = 1$ **to** $p_{n_1}^{\gamma_0}$ **do begin** c) **for** $n_2 = 0$ **to** n_1–1 **do** d) **for** $h_2 = 1$ **to** $p_{n_2}^{\gamma_0}$ **do begin** e) nonnull ← **true** f) **for** $j = 0$ **to** $(r_0 + \rho_0 + 1)$ **do** f) **if** nonnull **then** h) **for** $k = 0$ **to** s_0 **do** i) **if** nonnull **then** j) nonnull ← < $\Gamma^0_{(k)n_1h_1} \cap \Gamma^0_{(k)n_2h_2}$ *is non-empty?*>; k) **if** nonnull **then** error() **end**; l) **if not** reducelc **begin** m) **for** $h_2 = 1$ **to** $h_1 - 1$ **do begin** n) nonnull ← **true** o) **for** $j = 0$ **to** $(r_0 + \rho_0 + 1)$ **do** p) **if** nonnull **then** q) **for** $k = 0$ **to** s_0 **do** r) **if** nonnull **then** s) nonnull ← < $\Gamma^0_{(k)n_1h_1} \cap \Gamma^0_{(k)n_1h_2}$ *is non-empty?*>; t) **if** nonnull **then** error() **end**; **end**;

The function checkktraj is essentially the same as checklctraj. Recall Eq. 4.181:

$$\xi_{j_0k_0\kappa_i}(\mathbf{u}) = \bigvee_{n=0}^{N_{j_0k_0}^{i-1}} w_n \wedge \left[\bigvee_{h=1}^{p_n^{\xi_{j_0k_0}^{\kappa_i}}} \bigwedge_{j_1=0}^{r_i+\rho_i+1} \bigwedge_{k_1=0}^{s_i} u_{j_1k_1}^{(K_{(j_1k_1)nh}^{i,j_0k_0})} \right] \qquad 4.181$$

$$(i = 1, 2, \ldots, m) .$$

The major difference between function checkktraj and checklctraj are that checkktraj is executed for every projection $\xi_{jk\kappa_i}$ rather than once:

Algorithm 4.14: (Algorithm for checking that $\kappa_1(\mathbf{u})$ is unique)

METAPHOR Function checkktraj	
Comment	Step
a) Consider each level-0 component (attribute). b) Consider each level-0 phase. c) Check the projection of κ_1.	a) **for** $j_0 = 0$ **to** r_0 **do** b) **for** $k_0 = 0$ **to** s_0 **do** c) checkkprojtraj(j_0, k_0) **end**;

where

Algorithm 4.15: (Algorithm for checking that $\xi_{j_0k_0\kappa}(u)$ is unique)

METAPHOR Function checkkprojtraj(j_0, k_0)	
Comment	Step
a) Consider each value that $\xi_{j_0k_0}$ can assume b) Consider each cube function associated with that value. c) Consider each value up to the present one. d) Consider each cube function associated with the value of step c). e) Assume the two cube functions of steps b) and d) are not mutually exclusive. f) Consider each component (attribute): g) We can quit if the meet of the cube functions are null h) Consider each phase: i) We can quit if the meet of the cube functions are null j) See if the intersection of the compenents is null. If so, then the meet is empty. k) If the cube functions are not mutually exclusive, then process the error. l) If the cube functions are to be reduced (See Algorithm 4.5), then we need not worry about the mutual exclusiveness of cube functions within the same value. Otherwise repeat steps b) through k) for the current value.	a) **for** $n_1 = 0$ **to** $N^0_{j_0k_0}$ **do** b) **for** $h_1 = 1$ **to** $p_{n_1}^{\xi_{j_0k_0\kappa_1}}$ **do begin** c) **for** $n_2 = 0$ **to** n_1-1 **do** d) **for** $h_2 = 1$ **to** $p_{n_2}^{\xi_{j_0k_0\kappa_1}}$ **do begin** e) nonnull ← **true** f) **for** $j_1 = 0$ **to** $(r_1 + \rho_1 + 1)$ **do** f) **if** nonnull **then** h) **for** $k_1 = 0$ **to** s_1 **do** i) **if** nonnull **then** j) nonnull ← $< K^{1,j_0k_0}_{(j_0k_0)n_1h_1} \cap K^{1,j_0k_0}_{(j_0k_0)n_2h_2}$ *is non-empty?*>; k) **if** nonnull **then** error() **end**; l) **if not** reducek **begin** m) **for** $h_2 = 1$ **to** $h_1 - 1$ **do begin** n) nonnull ← **true** o) **for** $j = 0$ **to** $(r_0 + \rho_0 + 1)$ **do** p) **if** nonnull **then** q) **for** $k = 0$ **to** s_0 **do** r) **if** nonnull **then** s) nonnull ← $< K^{1,j_0k_0}_{(j_0k_0)n_1h_1} \cap K^{1,j_0k_0}_{(j_0k_0)n_1h_2}$ *is non-empty?*>; t) **if** nonnull **then** error() **end**; **end**;

The other function in Algorithms 4.11 and 4.12 which checks the consistency of the input is checktotal. This function verifies that the function (either γ_0 or κ_1) is total, i.e., that every point in the domain is defined. This is done by compressing all of the cube functions

(regardless of weight) associated with the function until they will no longer compress. The function is total if the result of the compression is the single "full" cube function:

$$\bigwedge_{j=0}^{r} \bigwedge_{k=0}^{s} x_{jk}^{(Q_{jk})} . \tag{4.229}$$

If the result of the comparison is not the full cube function, the function may still be total; any missing values can be found by taking the difference between the full cube function and the result of the reduction operations. The algorithm to find any missing trajectories is implemented by the function complementraj.

Algorithm 4.16: (Algorithm for checking that γ_0 or κ_1 is total)

METAPHOR Function checktotal	
Comment	Step
a) Reduce the cube functions. b) If we do not get the full cube function then: c) Complement the cube functions we do have.	a) C ← reducetraj b) **if** C ≠ <*full cube function*> **then** c) complementtraj(C) **end**;

Function complementtraj works by using Eq. 4.155:

$$\bigvee_{h=1}^{p} c_p(\mathbf{x})^c . \tag{4.230}$$

See Eq. 4.152 for the definition of the complement of a cube function, $c_p(\mathbf{x})^c$. complementtraj complements each of its input cube functions and then intersects them, using an intersection tree (see Section 4.3.3.2 [Intersection Trees]) algorithm similar to Algorithm 4.4.

Algorithm 4.17: (Algorithm for complementing a set of cube functions)

METAPHOR Function complementtraj(C)	
Comment	*Step*
a) Determine how many cube functions are given. b) Initialize the set which will hold the complemented cube functions. c) For each given cube function: d) complement the cube funtion. e) Intersect all the complemented cube functions.	a) N ← <*number of cube functions in* C> b) D ← *NULL*; c) **for** n = 1 **to** N **do** d) D ← [D complement(C_n)]; *e)* comptree(D); **end**;

The function 4.18 implements Eq. 4.230.

Algorithm 4.18: (Algorithm for complementing a cube function)

METAPHOR Function complement(C)	
Comment	Step
a) Initialize the structure which will hold the complement of the cube function C.	a) COMP ← Φ;
b) Consider each row.	b) **for** $a = 0$ **to** r **do**
c) Consider each column.	c) **for** $b = 0$ **to** s **do begin**
d) If the exponent of concern is not full, continue. Otherwise, we do not have to consider this element since the resulting array would be null.	d) **if** $C_{jk} \neq Q_{jk}$ **then begin**
e) Initialize the next cube function in the complement.	e) D ← Φ;
f) Go through each row up till just before the present:	f) **for** $j = 0$ **to** a–1 **do**
g) Go through each column in that row:	g) **for** $k = 0$ **to** s **do**
h) And copy the exponent of that element.	h) $D_{jk} \leftarrow C_{jk}$;
i) Now, do the present row. Consider each column up till just before the present.	i) **for** $k = 0$ **to** b–1 **do**
j) And copy the exponent of that element.	j) $D_{ak} \leftarrow C_{ak}$;
k) Now complement the exponent under consideration.	k) $D_{ab} \leftarrow C_{ab}$;

METAPHOR Function complement(C) (continued)	
Comment	Step
l) To finish up, go through the remaining columns on this row. m) And make those exponents full. n) Go trough the remaining rows. o) Go through each column in that row: p) And make those exponents full. q) Store the newly constructed cube function before constructing the remaining ones. r) Finally, return the result.	l) **for** k = b+1 **to** s **do** m) $D_{ak} \leftarrow Q_{ak}$; n) **for** j = a+1 **to** r **do** o) **for** k = 0 **to** s **do** p) $D_{jk} \leftarrow C_{jk}$; q) COMP ← [COMP D] **end** **end**; r) **return** COMP **end**;

Finally, the function comptree (see Algorithm 4.17) is a version of function buildtree (see 4.4).

4.4. METAPHOR--A Performability Modeling and Evaluation Tool

As discussed in Section 3.4.4 [METAPHOR–A Performability Modeling and Evaluation Tool], the algorithms in Section 4.3.3 [Algorithms] (as well as other supporting and ancillary algorithms) have been implemented in METAPHOR. The current version is numbered 3 and is implemented in C [176]. The portion dealing with discrete performance variables is called "meta_discrete", contains approximiately 8,000 lines of source, and has approximately 256 Kbytes of executable code, although the runtime size varies considerably due to dynamic

allocation and deallocation of memory. Appendix A contains the Unix manual entry for METAPHOR, while Appendix B contains the manual entry for meta_discrete. Appendix D describes the calling structure of meta_discrete.

Among the supporting algorithms in METAPHOR are routines for calculating the probability of the derived trajectory sets. These algorithms are described in Ballance, Furchtgott, Meyer, and Wu [30], and Wu [59], while their implementation within METAPHOR is discussed in Furchtgott [41], [46].

4.5. Examples

We now present some examples of the use of METAPHOR as a tool for generating γ^{-1} and calculating performability. As discussed in Section 3.3.8.2 [Examples of a Model Hierarchy], examples of model hierarchy construction by Furchtgott [89] is available in [29], [30], [38] and a second example of model hierarchy construction appears in [31], [32], [39], [40], [44]. These examples will not be redescribed here, but METAPHOR sessions for those examples will be described. In addition, two examples by Hitt, Bridgman, and Robinson [156] will be discussed. The latter two examples were also solved using fault-tree evaluation and a tablular method based on the program TASRA [117].

All of the examples were also solved completely by hand, though the labor was sometimes extensive. Except for the fourth example, the hand generated answers agreed with METAPHOR's results. The fourth example was so large that the hand generated answer had to make simplifying assumptions which led to errorneous results. The fault-tree and TASRA results were also errorneous because of the way in which they handled statistical dependencies.

4.5.1. Simple Reliability Network Example

This example is by Hitt, Bridgman, and Robinson [156], who call it the "series-parallel problem." Consider the simple reliability network in Figure 4.4. The mission length is 10 hours, and the subsystems fail with the following constant rates:

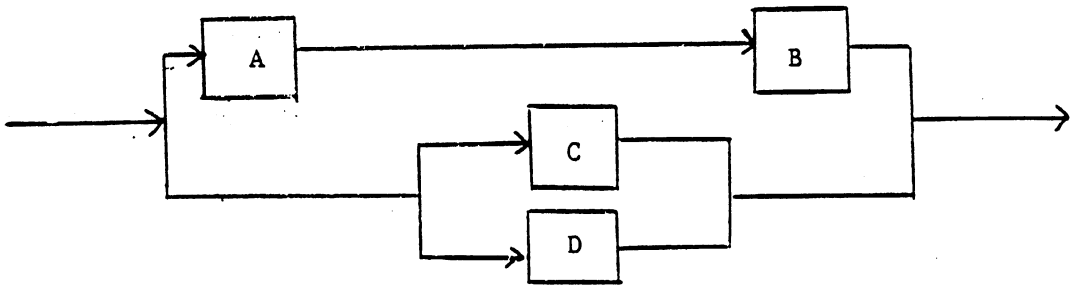


Fig. 4.4 Simple reliability network

Subsystem	λ
A	5×10^{-4}
B	4×10^{-4}
C	1×10^{-3}
D	1×10^{-3}

The system is successful if, at the end of the mission, there is a path from A to B. The accomplishment levels are {success, fail} and the performability can be identified with the reliability. Appendix F shows the METAPHOR session deriving the result.

The answer agrees with hand calculation, and presumably with the results of [156], though those results were not reported.

4.5.2. Simple Air Transport Mission Example

The second example we discuss is a simple air transport mission [29], [30], [38], [89]. As discussed in Section 4.5 [Examples], we will not repeat the scenario here. Consider the system P_{S_2} of [29]; the METAPHOR session for this example is in Appendix G.

Accomplishment Level a	$p(a)$
a_0 = (ALL GOOD)	0.9998
a_1 = (BAD FUEL EFFECIENCY)	$337. \times 10^{-7}$
a_2 = (DIVERSION)	0.0
a_3 = (BAD FUEL EFFECIENCY AND DIVERSION)	$1450. \times 10^{-7}$
a_4 = (CRASH)	$5. \times 10^{-7}$

4.5.3. SIFT Computer Example

The third example we present is the SIFT computer [67]. Again, as disussed in Section 4.5 [Examples], we will not repeat the scenario here. The entire METAPHOR session is comparitively lengthy; it follows closely the sessions in Appendices F and G in structure. The input data for this example is in Appendix H. The specific instance of [32] calculated is the London to New York flight having initial state (6,6) with probability one.

Accomplishment Level a	$p(a)$
a_0 = (ALL GOOD)	0.9962
a_1 = (BAD FUEL EFFECIENCY)	3.80×10^{-3}
a_2 = (DIVERSION)	3.77×10^{-12}
a_3 = (BAD FUEL EFFECIENCY AND DIVERSION)	6.02×10^{-6}
a_4 = (CRASH)	8.34×10^{-13}

4.5.4. Dual-Dual Example

The fourth example is by Hitt, Bridgman, and Robinson [156]. The scenario is given in Appendix I. As with the SIFT example (Section 4.5.3 [SIFT Computer Example]), the entire METAPHOR session is comparitively lengthy. The input data is in Appendix J. Table 4.1 contains the results for the solution techniques using METAPHOR, hand calculation, fault trees, and TASRA. The hand calculation as described in [156] was quite involved and hence prone to error. However, there is a significant discrepency between METAPHOR and the fault tree and TASRA results. The reason for this lies in the technique used in the fault tree and TASRA evaluations to combine two separate outcomes (diversion and crash) which are statistically dependent. For the evaluation, separate fault tree and TASRA evaluations were performed for diversion (ignoring any possibility of crash) and for crash (ignoring diversion). Thus, the derived probability of diversion does not take into account any crashes which might occur. Similarly, there are trajectories in which attempted CAT-II landings crash and hence lower the probability of a successful CAT-II landing.

It may be possible to evaluate such a mission correctly with fault-tree tools. However, with a program such as METAPHOR, such evaluation becomes significantly easier. Also, it is interesting to note that the example does not contain any time dependencies (mission outcomes which depend on sequential combinations of events) such as those of the examples of Sections 4.5.2 [Simple Air Transport Mission Example] and 4.5.3 [SIFT Computer Example]. Time dependencies are much harder to model using the "snap-shot" techniques of fault trees

Mission Outcome Probabilities	METAPHOR	Hand Performability	Fault Trees	TASRA
Safe Flight and Landing at Primary Destination	0.983740	0.974212	0.974245	0.974236
Safe Flight and Landing at Alternate Destination	0.016193	0.025763	0.025701.	0.025740
Loss of Aircraft	66.59×10^{-6}	25.98×10^{-6}	25.98×10^{-6}	23.69×10^{-6}

Table 4.1 Dual-dual computer system results for four solution techniques.

CHAPTER 5

CONTINUOUS PEFORMANCE VARIABLE (CPV) METHODOLOGY

5.1. Introduction

Much of the recent work in analytical evaluation of degradable fault-tolerant computing systems has concerned the evaluation of the total "benefit" derived from the computing system during some specified interval of time [15], [17]-[22], [24], [26]-[28], [33], [56], [189]-[194]. In these studies, a variety of concepts and terminology have been used to formalize benefit, including "work"[15], "capacity"[17], "reward"[22], and "performance"[18], Other studies have concerned the evaluation of "cost" in conjunction with the evaluation of some form of benefit [26], [189]-[193]. Much of this work [19]-[22], [27], [91], [189]-[194] has considered systems in steady-state which are utilized over an unbounded period of time: repair of components is allowed (indeed, required), and the evaluated quantity is the expected rate at which benefit is derived under steady-state conditions. However, many important applications of degradable systems have bounded utilization periods. In such cases, a transient solution of the system benefit is usually required. Moreover, in many such applications, the user is interested in how benefit is distributed probabilistically (i.e., the probability distribution function of the benefit variable). This further complicates the evaluation process as compared, say, with the evaluation of expected benefit.

As discussed in Section 3.3 [An Informal Introduction to Performability], we view system "performance" as a relatively general concept which includes various types of benefit as possible specializations. Briefly reviewing some of the terminology of Section 3.3 [An Informal Introduction to Performability], we define the ***performance*** of a system S over a specified time period T to be a random variable Y taking values in a set A. Elements of A are ***accomplishment levels*** representing how well S performs during T. Relative to a designated performance variable Y, ***performability*** is the probability measure p induced by Y where, for any measurable set B of accomplishment levels ($B \subseteq A$), $p(B)$ is the probability that S performs at a level B. Evaluation of performability thus requires solution of the probability distribution function of Y. This solution is based on an underlying stochastic process X, called the ***base model*** (of S), which represents the dynamics of the system's structure, internal state, and environment during utilization.

Specific interpretations of performability thus depend on how values of Y (the accomplishment levels A) are interpreted. In particular, if elements of A represent levels of benefit then performability describes the system's ability to benefit its user. In the discussion of this chapter, we consider this type of performability where benefit is equated with ***reward*** (see Howard [155] for example) derived from using the system during a specified period T. We are interested in the case where the time period T is bounded, although unbounded periods are considered during an intermediate step of the evaluation process. Also, by the definition of performability (see above), we seek to determine (either analytically or numerically) the probability distribution function of the reward variable Y.

Prior work relating to this problem has dealt primarily with evaluations of expected reward, i.e., the expected value of the reward variable Y. This background is nevertheless relevant and two approaches are of particular interest. The first is based on the concept of the "potential" of a semi-Markov process; see Cinlar [195] for an introductory discussion of potentials. Gay [91] (see also Gay and Ketelsen [19]) has applied potentials of Markov processes to modeling the performance of degradable systems. Most of this work deals with steady-state solutions of systems having repairable components. The second approach

considers transient solutions of the expected reward and is based on semi-Markov reward models such as those discussed by Howard [155]. The formulation is in terms of a set of integrations and variables which must be solved in order to determine the expected reward. Because of the many inherent convolutions, the equations can be conveniently written in terms of Laplace transforms. However, the equations are generally difficult to solve (and if done in transform space, to invert), and so practical applications are limited. De Souza [22] has applied Howard's work to fault-tolerant computing systems via a unified reliability/cost model; however, these methods presume that the system is in steady state. Though some of the underlying characteristics (especially the nature of transition graphs) of these potential and reward models are allowed to be more general than those considered in this thesis, these models are restricted to be semi-Markov and, more importantly, the evaluation results are expected values rather than probability distribution functions.

Relatively little work has been done on obtaining transient solutions of the probability distribution function (PDF) of the reward variable. Notably, Cherkesov [92] presents an elegant solution for the probability of accumulating a minimum reward during a finite interval when the underlying process is semi-Markov. The length of the interval can itself be random. However, the solution is in terms of a system of equations of two-dimensional Laplace-Carson transforms (see p. 39 of [196] for a description of this transform). Solving and inverting these equations is intractable except in the simplest of cases. Instead, Cherkesov employs the transforms to obtain expected values. This work is mainly of theoretical interest.

In the presentation that follows, we obtain a method for determining the PDF of the reward variable, subject to certain conditions imposed on the base model and on the corresponding reward model. Specifically, we assume that the base model is a finite-state process and is "acyclic" in the sense that states are not revisited by the process. However, the process need not be Markov or even semi-Markov. We assume further that the corresponding reward model is a nonrecoverable process in the sense that a future state (reward rate) of the model cannot be greater than the present state. These conditions reflect properties that are typically exhibited by degradable, nonrepairable computing sytems. For

this model class, we are able to obtain the probability distribution function of the performance (reward) variable and, hence, the performability of the corresponding system. Moreover, this is done for bounded utilization periods, an assumption which demands a relatively complex solution.

The approach is based on the strategy used in [33], [56] to solve performability, once the performance (reward) rates were obtained via a queueing model analysis. (There, in reward terms, the reward rate of a given structure state is taken to be the normalized throughput rate; performance is viewed as average reward, i.e., total reward divided by the duration of utilization.) This earlier work, however, does not suggest specific techniques for implementing the prescribed steps. The computational example presented in [33], [56] (namely a 3-state process) is solved using graphical arguments to determine appropriate regions of integration. Such an approach becomes more difficult when the number of states is four and becomes intractable when the number of states is five or more. This chapter describes a solution technique wherein regions of integration are determined in a tractable, algorithmic fashion. Section 5.2 [Reward Models] discusses the model; Section 5.3 [Reward Model Solution] presents the solution and describes a program implementing the solution procedure. Examples of closed-form solutions are presented in Section 5.3.9 [Closed-Form Solutions] and applications are illustrated in Section 5.3.11 [Numerical Solutions].

5.2. Reward Models

5.2.1. Definition of Reward Models

We consider reward models of the type discussed in [155], although the stochastic process need not be semi-Markovian. We restrict our attention,however, to the case where reward is determined solely by reward rates (referred to in [55], [59], [62] as "operational rates") associated with states of the model. We assume further that these rates are constant, i.e., time-invariant. More general reward structures (time-varying rates and "bonuses" associated with state transitions) are also accommodated by our approach, and they are considered

in Section 5.4 [History Based Reward Models (HBRMs)].

A state, in this context, typically represents a certain "operational status" of the system, including configurations wherein the system is not operating (system failure). In a given state, the associated reward rate reflects the pace at which the system rewards its user. Such rates can thus be identified with aspects of system performance such as productivity, responsiveness, and utilization, or, at a higher level, with broader measures such as economic benefit. In Section cpvorates, we discuss reward rates in greater detail—what such rates can represent and how to obtain them.

More formally, let X_τ be the random variable denoting the state of the system at time τ. Accordingly, the base model of the system is the stochastic process

$$X = \{X_\tau \mid \tau \in [0, \infty)\} \ . \tag{5.1}$$

For reasons given later in the discussion, we restrict X to be finite-state, i.e., $X_\tau \in Q = \{q_N, q_{N-1}, \ldots, q_0\}$. Suppose further that each $q_i \in Q$ has an associated reward rate r_i (a nonnegative real number). Then there is a natural function

$$r: Q \rightarrow \mathbb{R}^{0,+} \tag{5.2}$$

where $r(q_i) = r_i$ is the rate associated with q_i. r is referred to as a ***reward structure*** of X. Let $\overline{X}_\tau$, $\tau \in [0, \infty)$, be a random variable [taking values in $r(Q)$] representing the reward rate of S at time τ, that is

$$\overline{X}_\tau = r(X_\tau) \ . \tag{5.3}$$

Then the stochastic process

$$\overline{X} = \{\overline{X}_\tau \mid \tau \in [0, \infty)\} \ . \tag{5.4}$$

is a ***reward model*** of the system. Such reward models are a special class of the type of "operational models" investigated by Wu [62]. (Also, compare with the "capacity models" of

Gay [91], Gay and Ketelsen [19].)

The performability models we consider consist of a reward model $\overline{X}$ along with a performance (reward) variable Y representing the total reward accrued during utilization of the system. More precisely, given a reward model $\overline{X}$ and a utilization period $T = [0, t]$, we take Y to be the random variable

$$Y = \int_0^t \overline{X}_\tau \, d\tau \ . \tag{5.5}$$

Equivalently, in terms of the base model X and the reward structure r,

$$Y = \int_0^t r(X_\tau) \, d\tau \ . \tag{5.6}$$

5.2.2. An Example of a Reward Based Performability Model

Consider a computing system S consisting of N processors, each processor having a computational capacity of δ jobs/hour. In addition, there is an air conditioner which does not affect the computational capacity of the system (though it may the affect the failure characteristics of the processors). The base model X is a $(2\cdot(N+1))$-state (not necessarily Markov) stochastic process whose state-transition diagram[1] appears in Fig. 5.1. In state (i,j), $i \in \{0, 1, \ldots, N\}$ denotes the number of operational processors and j is 1 if the air condi-

---

[1]Informally, a ***state-transition diagram*** for a stochastic process X is a diagram representing a directed graph whose nodes (points) denote the states of X and which has a directed arc (line) from node q_i to q_j if and only if, with nonzero probability, X can visit state q_i and then state q_j without visiting any intermediate states That is, there exist times $\tau, \nu \in [0, \infty)$ where $\tau < \nu$, such that

$$\text{Prob}[X_\tau = q_i \text{ and } X_\nu = q_j \text{ and } X_\psi \in \{q_i, q_j\} \text{ for all } \psi \text{ such that } \tau < \psi < \nu] > 0 \ .$$

(To be able to discuss the behavior of X during an interval, we assume X is separable.) If X is Markov and if the transition rate for the transition from q_i to q_j is associated with the directed arc, then the diagram of the graph is a ***state-transition-rate diagram***.

State (node) q_i is ***reachable*** from state (node) q_j if there exists a nontrivial sequence of distinct states (nodes) $q_i, q_1, q_2, \ldots, q_n, q_j$ beginning at q_i and ending at q_j such that each neighboring pair in the sequence [i.e., (q_i, q_1), (q_1, q_2), $\ldots$, (q_n, q_j)] is connected by a directed arc. The state-transition diagram is ***acyclic*** if the graph is acyclic, i.e., no point is reachable from itself.

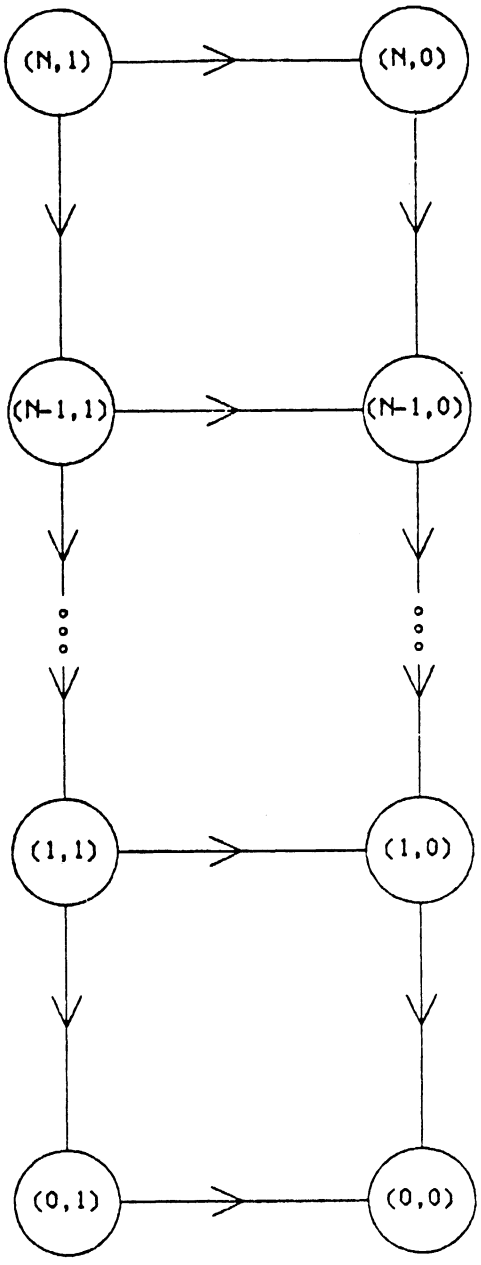


Fig. 5.1 Transition diagram for the Markov model of the example of Section 5.2.2

tioner is operational and 0 otherwise. Assuming system capacity is proportional to the number of operational processors, the reward structure is the function $r(i,j) = i \cdot \delta$. Accordingly, the reward model $\overline{X}$ has $N+1$ states corresponding to the $N+1$ different reward rates. Relative to a specified utilization period T, the performance (reward) variable Y is the number of jobs processed during T.

5.2.3. Determination of Reward Rates

An important consideration when applying the methodology described in this chapter is the interpretation and derivation of the reward rates. In this section, we delineate several possible interpretations and quantifications of reward. All determinations can be made using one of at least four methods: A) measurement, B) simulation, C) analytic techniques, and D) subjective judgement. The accuracy of the resulting reward model depends on which of the above techniques is employed. Generally, A) is more accurate than B), which is more accurate than C), which, in turn, is more accurate than D); conversely, the most accurate methods are usually the most expensive to employ.

The following paragraphs discuss several possible interpretations of reward. This survey is by no means complete; there are certainly many other applications not considered here.

1) Capacity and Workload: One basic measure of a computing system's reward rate is the speed at which the system is able to perform computations. This is commonly referred to as ***capacity***. Work of Beaudry [17], Gay [91], Gay and Ketelsen [19], Castillo and Siewiorek [21], Oda, Tohma, and Furuya [192], and Munarin [194] model capacity. A related concept is that of ***workload***, which is the demand for computation placed on the system by the environment. Studies by Gay [91] and Gay and Ketelsen [19] examine workload models.

2) Queues and Networks of Queues: A detailed view of the system's behavior often can be obtained by studying queueing models of the system. Examples of such models abound in the performance evaluation literature; see Kobayashi [3], Ferrari [2], Chandy and Sauer [4], and Trivedi [73]. For instance, S could be a multiprocessor (k servers) and a buffer (queue)

for storing arriving tasks; S is then a $G/G/i$ queueing system. If arrival and service rates are exponential and the buffer length is L, then S is the $M/M/k/L+k$ system considered by Meyer [33].

The reward rate and total reward associated with a queueing system depends on what is of concern to the analyst. One could associate reward with throughput rate, system (or service) time of customers, server utilization, number of customers in the system, etc. Hence, for a given structural configuration, standard queueing analysis (e.g., see Gross and Harris [197] or Kleinrock [198]) or simulation could provide any of the above rates. In [33], for instance, the reward is based on a "normalized" throughput rate. A study by Huslende [24] considers response times of $M/M/n$ queues, with suggested generalizations to $M/G/1$ and $G/M/n$ queues.

We generally wish to avoid the difficult problem of transient solutions of queueing models, so we make some reasonable assumptions. Usually, therefore, one assumes that the average time a customer is in the system is very small compared to the utilization period $[0, t]$ and the average time between changes in the structural state of the system. The differences in these times may be many orders of magnitude. For instance, in the $M/M/k/L+k$ system of [33], the customers are computational tasks arriving and being serviced in terms of seconds or milliseconds, while structural changes occur when processors and buffers fail with MTBF's of hundred's of hours.

3) Profit and Expense: An important measure of the reward derived from a system is the profit (say, in terms of dollars) obtained from the system, or, in a negative context, the expense of the system. Each system state has an associated rate of profit or expense. Such values may be obtained, for instance, by economic analysis or by utility techniques, e.g., Raiffa [199]. Koren and Berg [189] and Huslende [193] have based analyses on economic factors.

4) Control Theoretic: When the system is a real-time control processor, the reward rate associated with a given state could be specified as functions of such control theoretic con-

cepts as response time. For instance, Krishna and Shin [26] have examined such descriptions, while Gai and Adams [28] have discussed tradeoffs between "optimal" and "robust" response times.

5.2.4. Nonrecoverable Processes

5.2.4.1. Definition of a Nonrecoverable Process

Consider the "desired" behavior of a degradable computing system used over a bounded period of time. Given a set of available resources, a well-designed degradable computing system configures itself to *maximize* the reward rate. Thus, when a component fails, the system does not reconfigure itself in a manner that causes the reward rate to be *greater* than before the failure. We will assume there is no repair (i.e., component replacement via an external source), so an increase in the reward rate due to the acquisition of additional components cannot occur. Transient faults that could lower the reward rate temporarily, thereby raising the rate again when the fault is corrected, are not be considered. Under these conditions, the reward rate of the system is non-increasing in time, which has the interpretation that the system does not become a "better" system after a change in state (e.g., a component failure). These assumptions are formalized by requiring that the reward model $\overline{X}$ be a ***nonrecoverable process***[62] relative to the usual ordering of real numbers. In other words, for all states $q_1, q_2 \in Q$ and all times $\tau, \nu \in [0, \infty)$ such that $\tau \leq \nu$, we require that

$$\text{Prob}[\overline{X}_\tau = r(q_1) \text{ and } \overline{X}_\nu = r(q_2)] > 0 \Rightarrow \; r(q_2) \leq r(q_1) \,. \tag{5.7}$$

Consider the process X of the multiprocessor/air conditioner example discussed above. Clearly, by the state-transition diagram of Fig. 1 and the definition of the reward structure, the reward rate of the process cannot increase (with positive probability). Hence, the associated reward model $\overline{X}$ is an example of a nonrecoverable process.

5.2.4.2. Some Properties of Nonrecoverable Processes

The following are derivable immediately from the definition of a nonrecoverable process:

Theorem 5.1: (Necessary and sufficient conditions for X to be a nonrecoverable process) *Let X be a finite-state stochastic process with state space Q and reward structure r. For all states $q_1, q_2 \in Q$ and all times $\tau, \nu \in [0, \infty)$ such that $\tau \leq \nu$, the following are equivalent statements:*

i) $\overline{X} = \{r(X_\tau) \mid \tau \in [0, \infty)\}$ is a nonrecoverable process. (5.8)

ii) $\text{Prob}[\overline{X}_\tau = r(q_1) \text{ and } \overline{X}_\nu = r(q_2)] > 0 \Rightarrow r(q_2) \leq r(q_1)$. (5.9)

iii) $\text{Prob}[X_\tau = q_1 \text{ and } X_\nu = q_2] > 0 \Rightarrow r(q_2) \leq r(q_1)$. (5.10)

iv) $\text{Prob}[\overline{X}_\nu = r(q_2) \mid \overline{X}_\tau = r(q_1)] > 0 \Rightarrow r(q_2) \leq r(q_1)$. (5.11)

v) $\text{Prob}[X_\nu = q_2 \mid X_\tau = q_1] > 0 \Rightarrow r(q_2) \leq r(q_1)$. (5.12)

vi) $\text{Prob}[\overline{X}_\nu \leq r(q_1) \mid \overline{X}_\tau = r(q_1)] = 1$. (5.13)

vii) $\text{Prob}[\overline{X}_\nu \leq \overline{X}_\tau] = 1$. (5.14)

Proof:

i) $\Leftrightarrow$ *ii)*: by definition of nonrecoverable process. (5.15)

ii) $\Leftrightarrow$ *iii)*: by definition of $\bar{X}_\tau$. (5.16)

ii) $\Leftrightarrow$ *iv)*: $\{ \text{Prob}[\bar{X}_\tau = r(q_1) \text{ and } \bar{X}_\nu = r(q_2)] > 0 \Rightarrow r(q_2) \leq r(q_1) \}$

(5.17)

$$\Leftrightarrow \left\{ \frac{\text{Prob}[\bar{X}_\tau = r(q_1) \text{ and } \bar{X}_\nu = r(q_2)]}{\text{Prob}[\bar{X}_\tau = r(q_1)]} > 0 \Rightarrow r(q_2) \leq r(q_1) \right\}$$

$$\Leftrightarrow \{ \text{Prob}[\bar{X}_\nu = r(q_2) \mid X_\tau = r(q_1)] > 0 \Rightarrow r(q_2) \leq r(q_1) \}$$

iv) $\Leftrightarrow$ *v)*: by definition of $\bar{X}_\tau$. (5.18)

iv) $\Leftrightarrow$ *vi)*: Since $\sum_{q \in Q} \text{Prob}[\bar{X}_\nu = r(q) \mid X_\tau = r(q_1)] = 1$, then

$$\{ \text{Prob}[\bar{X}_\nu = r(q_2) \mid \bar{X}_\tau = r(q_1)] > 0 \Rightarrow r(q_2) \leq r(q_1) \}$$

$$\Leftrightarrow \sum_{\substack{q \in Q \\ r(q) \leq r(q_1)}} \text{Prob}[\bar{X}_\nu = r(q) \mid \bar{X}_\tau = r(q_1)] = 1 \tag{5.19}$$

$$\Leftrightarrow \text{Prob}[\bar{X}_\nu \leq r(q_1) \mid \bar{X}_\tau = r(q_1)] = 1$$

vi) $\Leftrightarrow$ *vii)*: $\{ \text{Prob}[\bar{X}_\nu \leq r(q_1) \mid \bar{X}_\tau = r(q_1)] = 1 \}$

$$\Leftrightarrow \{ \text{Prob}[\bar{X}_\nu \leq r(q_1) \text{ and } \bar{X}_\tau = r(q_1)] = \text{Prob}[\bar{X}_\tau = r(q_1)] \}$$

(5.20)

$$\Leftrightarrow \left\{ \sum_{q \in Q} \text{Prob}[\bar{X}_\nu \leq r(q) \text{ and } \bar{X}_\tau = r(q)] = 1 \right\}$$

$$\Leftrightarrow \{ \text{Prob}[\bar{X}_\nu \leq \bar{X}_\tau] = 1 \} \ .$$

| |

The following result (which we shall not prove here) is slightly more difficult and requires a separability condition:[2]

Theorem 5.2: (Necessary and sufficient conditions for X to be a nonrecoverable process) *Let X be a separable process with reward structure r and state space Q. Then $\overline{X}$ is a nonrecoverable process if and only if for all states $q \in Q$ and for all times $\nu \in [0, \infty)$:*

$$\text{Prob}[\overline{X}_\tau \geq r(q) \text{ for all } \tau \leq \nu \text{ and } X_\nu = r(q)] = \text{Prob}[X_\nu = r(q)] \,. \tag{5.21}$$

5.2.5. Acyclic Processes

5.2.5.1. Definition of an Acyclic Process

An essential step of our model solution procedure (Section 5.3 [Reward Model Solution]) is to first evaluate conditional performabilities, conditioned on the sequence of states entered by the process during the unbounded period $[0, \infty)$. For this reason, we restrict our attention to finite-state base models. To insure a finite number of state sequences and hence conditions, we also require that the base model be "acyclic" in the sense that states are not revisited by the process. More precisely, let M_i be the random variable denoting the total number of visits of the process X to state $q_i \in Q$ during the interval $[0, \infty)$. X is an ***acyclic process*** if for all $q_i \in Q$, $\text{Prob}[M_i > 1] = 0$.

5.2.5.2. Properties of Acyclic Processes

A well-known necessary and sufficient condition for an acyclic process follow:

---

[2]We implicitly assume all underlying stochastic processes discussed in this thesis are separable.

Theorem 5.3: (Sufficient and necessary condition for a process to be acyclic)

Let X be a stochastic process with state space Q. Then X is acyclic if and only if its state-transition diagram[3] *is acyclic.*

Proof: Clear from the definitions of acyclic process and acyclic state-transition diagram.

||

An absorbing state is one which the process can never leave, i.e., $q_i \in Q$ is an ***absorbing state*** if for all $\tau, \nu \in [0, \infty)$ such that $\tau < \nu$, $\text{Prob}[X_\nu = q_i \mid X_\tau = q_i] = 1$.

Theorem 5.4: (Every finite-state, acyclic process must have at least one absorbing state)

Let X be an acyclic process with finite state space Q. Then at least one state $q_i \in Q$ is absorbing.

Proof: Assume otherwise, i.e., no state is absorbing. Let $|Q| = N$. Suppose for some $q_1 \in Q$ and some $\tau_1 \in [0, \infty)$ that $\text{Prob}[X_{\tau_1} = q_1] > 0$. Then there exists a $q_2 \in Q$ and a $\tau_2 \in [0, \infty)$ such that $q_1 \neq q_2$, $\tau_1 < \tau_2$, and $\text{Prob}[X_{\tau_2} = q_2 \mid X_{\tau_1} = q_1] > 0$. Continuing iteratively, we find that there exist $q_2, q_3, \ldots, q_{N+1}$ and $\tau_2, \tau_3, \ldots, \tau_{N+1}$ such that $q_2 \neq q_3$, $q_3 \neq q_4, \ldots, q_N \neq q_{N+1}$, $\tau_2 < \tau_3 \leq \cdots \leq \tau_{N+1}$, and so

$$\begin{aligned} &\text{Prob}[X_{\tau_{N+1}} = q_{N+1} \mid X_{\tau_N} = q_N] \text{Prob}[X_{\tau_N} = q_N \mid X_{\tau_{N-1}} = q_{N-1}] \\ &\qquad \cdots \text{Prob}[X_{\tau_2} = q_2 \mid X_{\tau_1} = q_1] \text{Prob}[X_{\tau_1} = q_1] \\ &\qquad > 0 \end{aligned} \tag{5.22}$$

$$\Rightarrow \quad \text{Prob}[X_{\tau_{N+1}} = q_{N+1}, X_{\tau_N} = q_N \cdots X_{\tau_1} = q_1] > 0 .$$

---

[3] See the footnote in Section 5.2.2 [An Example of a Reward Based Performability Model].

However, $|Q| = N$ and so at least one $q_j = q_k, j \neq k$. Thus,

$$0 < \text{Prob}[X_{\tau_k} = q_j, X_{\tau_j} = q_j] \leq \text{Prob}[M_{q_j} = 2] \tag{5.23}$$

which contradicts X being acyclic.

||

5.2.5.3. Acyclic Properties of Nonrecoverable Models

This section examines the relationship between acyclic processes and nonrecoverable models. We shall find that nonrecoverable models are acyclic and that the underlying base model must either be acyclic or else have an acyclic nature in the sense that all states in a cycle must have the same reward rate.

Theorem 5.5: *Nonrecoverable models are acyclic*

Proof: Assume otherwise, i.e., there exists a cyclic nonrecoverable model $\overline{X}$. Then there must be a state (reward) r_i reachable from itself, i.e., there must be a second state (reward) r_j such that $\overline{X}$ can, with probability greater than zero, visit r_i then r_j and r_i again. But since $\overline{X}$ is nonrecoverable, $r_i \geq r_j \geq r_i \Rightarrow r_i = r_j$, that is, r_i and r_j are the same state (reward). Therefore, $\overline{X}$ cannot be cyclic.

||

Theorem 5.6: (Acyclic nature of the base models of nonrecoverable models)
Let $\overline{X}$ be a nonrecoverable model. X can be either cyclic or acyclic. If X is cyclic, then if $q_1, q_2 \in Q$ are in a common cycle (i.e., q_1 and q_2 are reachable from each other), then $r(q_1) = r(q_2)$.

Proof: Let X can be acyclic. If for every $q_1, q_2 \in Q$ such that q_2 is reachable from q_1, we have $r(q_2) \leq r(q_1)$, then $\overline{X}$ is nonrecoverable and so X could be acyclic. Suppose now that

X is cyclic and $q_1, q_2 \in Q$ are in a common cycle. Then $r(q_2) \leq r(q_1)$ since X can reach q_2 after q_1. Similarly, $r(q_1) \leq r(q_2)$. Hence $r(q_1) = r(q_2)$.

||

5.2.5.4. Approximating Cyclic Processes

When evaluating fault-tolerant systems where repair is not allowed, the acyclic condition is not restrictive since once the system leaves a state due to a fault, the system can never return to that state. Often, however, one is interested in analyzing systems with transient faults or temporary failures. Models of such systems are generally cyclic because if the system recovers from the fault, the system returns to some previously entered state. Such recoveries can occur an unbounded number of times during $[0, t]$, resulting in an infinite number of state sequences.

One approach for approximating cyclic base models is to "unravel" the process into an acyclic base model. This is done by simply choosing a finite set of state sequences to consider. For example, the set chosen can be those sequences which have at least some threshold probability of occurring during the utilization period $[0, t]$. In other words, those sequences of sufficiently low probability are ignored. Alternatively, the set could include all sequences less than or equal to some specified length.

A second approach is to "lump" all the states in each cycle. Because each state in a cycle must have the same reward rate (see Theorem 5.6) the reward rate of each lumped state is the rate of each of the component states. The difficulty with this approach is determining the equivalent stochastic behavior of the lumped model.

5.3. Reward Model Solution

5.3.1. A Partition of the Trajectory Space

As noted in [56], one approach to determining performability is to partition the trajectory space and solve for each of the resulting classes of trajectories. During the unbounded period $[0, \infty)$, the base model process X will, with probability 1, pass through some finite sequence of distinct states, say $\mathbf{u} = (u_n, u_{n-1}, \ldots, u_0)$, where state u_n is the initial state and state u_0 is an absorbing state. $\overline{X}$ is nonrecoverable (see above) so there is a sequence of reward rates $(r(u_n), r(u_{n-1}), \ldots, r(u_0))$, corresponding to $\mathbf{u}$, such that

$$r(u_n) \geq r(u_{n-1}) \geq \cdots \geq r(u_0) \ . \tag{5.24}$$

Let $\mathbf{U}$ be a random variable denoting the sequence of states visited by the base model during $[0, \infty)$. Since the base model is acyclic, and since there are $N < \infty$ base model states, there are only a finite number of possible state sequences, i.e., sequences $\mathbf{u}$ such that $\text{Prob}[\mathbf{U} = \mathbf{u}] > 0$. Let Y be the performance (reward) variable defined in (5.5) and let F_Y be the PDF of Y. Further, if $F_{Y|\mathbf{U}}$ is the conditional PDF of Y given $\mathbf{U}$, then F_Y can be expressed as the following summation over a *finite* index set:

$$F_Y(y) = \sum_{\mathbf{u}} F_{Y|\mathbf{U}}(y|\mathbf{u}) \text{Prob}[\mathbf{U} = \mathbf{u}] \ . \tag{5.25}$$

5.3.2. Notation

At this point, it is convenient to introduce a body of notation for dealing with the time the process spends in various states during both $T = [0, t]$ and $[0, \infty)$. Unless otherwise noted, the remarks that follow assume $\mathbf{U}$ to be some arbitrary sequence $\mathbf{u} = (u_n, u_{n-1}, \ldots, u_0)$ such that $\text{Prob}[\mathbf{U} = \mathbf{u}] > 0$.

For each state sequence $\mathbf{u} = (u_n, u_{n-1}, \ldots, u_0)$ there is a vector-valued random variable $\mathbf{V_u} = (V_n, V_{n-1}, \ldots, V_0)$ taking values in $(\mathbb{R}^{0,+})^n$, which describes the time the pro-

cess resides in each state in $\mathbf{u}$. For $0 \leq i \leq n$, V_i is a random variable representing the time of the base model process resides in state q_i during the interval $[0, \infty)$. State u_0 is an absorbing state, and so v_0 is ∞. We will be interested in the PDF for $\mathbf{V_u}$ conditioned on $\mathbf{U} = \mathbf{u}$, i.e.,

$$\begin{aligned} F_{\mathbf{V_u}|\mathbf{U}}(\mathbf{v_u} \mid \mathbf{u}) &= \text{Prob}[\mathbf{V_u} \leq \mathbf{v_u} \mid \mathbf{U} = \mathbf{u}] \\ &= \text{Prob}[V_n \leq v_n \wedge V_{n-1} \leq v_{n-1} \wedge \cdots \wedge V_0 \leq v_0 \mid \mathbf{U} = \mathbf{u}] \ . \end{aligned} \tag{5.26}$$

If the conditional probability density function for $\mathbf{V_u}$ exists, it will be written $f_{\mathbf{V_u}}$. We suppose that $f_{\mathbf{V_u}}$ exists and we find it convenient to expand $f_{\mathbf{V_u}}$ as follows:

$$\begin{aligned} f_{\mathbf{V_u}|\mathbf{U}}(\mathbf{v_u} \mid \mathbf{u}) = f_{V_n|\mathbf{U}}(v_n \mid \mathbf{u}) f_{V_{n-1}|V_n\mathbf{U}}(v_{n-1} \mid v_n, \mathbf{u}) \cdots \\ f_{V_0|V_n, V_{n-1}, \ldots, V_1, \mathbf{U}}(v_0 \mid v_n, v_{n-1}, \ldots, v_1, \mathbf{u}) \ . \end{aligned} \tag{5.27}$$

For a given $\mathbf{u}$, define W_t^i to be a random variable denoting the amount of time the process is in state u_i during the utilization period T. The basic relationship between W_t^i and V_i can be expressed as

$$W_t^i = \begin{cases} \min(t, V_n), \text{ if } i = n \\ \max(\min(t - \sum_{j=i+1}^{n} V_j, V_i), 0) \ , \quad \text{otherwise,} \end{cases} \tag{5.28}$$

Note that if $\mathbf{U} = \mathbf{u}$ and if $W_t^i = 0$ and $V_i \geq 0$, then $W_t^{i-1}, W_t^{i-2}, \ldots, W_t^0 = 0$. Also note that the value of W_t^i can always be expressed by one of the following alternatives: t, V_i, $t - \sum_{j=i+1}^{n} V_j$, or 0.

5.3.3. The Approach

We seek to determine the PDF F_Y of the reward varible Y (Eq. 5.25). The algorithm delineated in [56] will be employed:

Algorithm 5.2: (Determining the probability distribution function, F_Y)

1) Determine all state sequences $\mathbf{u}$, i.e., the range of the random variable $\mathbf{U}$.

2) For each of these $\mathbf{u}$, determine Prob$[\mathbf{U} = \mathbf{u}]$.

3) For each $\mathbf{u}$ such that Prob$[\mathbf{U} = \mathbf{u}] > 0$, determine $F_{Y|\mathbf{U}}(y)$.

4) Apply Eq. 5.25 to arrive at F_Y.

For the present, we assume that steps 1) and 2) have been completed. Consider step 3), the calculation of $F_{Y|\mathbf{U}}$.

Using the notation introduced in the previous section, the reward variable Y can be directly expressed as a linear combination of the random variables W_t'. If $\mathbf{U} = \mathbf{u}$, then

$$Y = \sum_{j=0}^{n} r(u_j) W_t' \quad . \tag{5.29}$$

If the W_t' were independent random variables, we could obtain the PDF F_Y by mathematical convolution. However, as discussed in [33], [56], the W_t' are statistically dependent and a probabilistic characterization is generally difficult to obtain. Such complications are avoided by formulating Y as a function $\gamma_{\mathbf{u}}$ of $\mathbf{V_u}$ (if it is known that $\mathbf{U} = \mathbf{u}$). Although the basic concept is simple to state, the details are somewhat complex.

Using the relationship between W_t' and V_i of Eq. 5.28, (see Eq. 5.6)

$$Y = \gamma(\mathbf{V_u}) = \begin{cases} r(u_j)t + \sum_{k=j+1}^{n} (r(u_k) - r(u_j)) V_k & \\ \qquad \text{if } \sum_{k=j+1}^{n} V_k < t, \ \sum_{k=j}^{n} V_k \geq t \ , \text{ for } n > j & \\ r(u_n)t \quad \text{if } V_n \geq t \ . & \end{cases} \tag{5.30}$$

See Fig. 5.2 and compare with Eq. 24 of [33].

Let B_y be the set of accomplishment levels (reward outcomes) not greater than y, i.e., $B_y = \{b \geq 0 \mid b \leq y\}$ and let $\mathbf{C}_y = \gamma^{-1}(B_y)$ be the set of all base model trajectories which

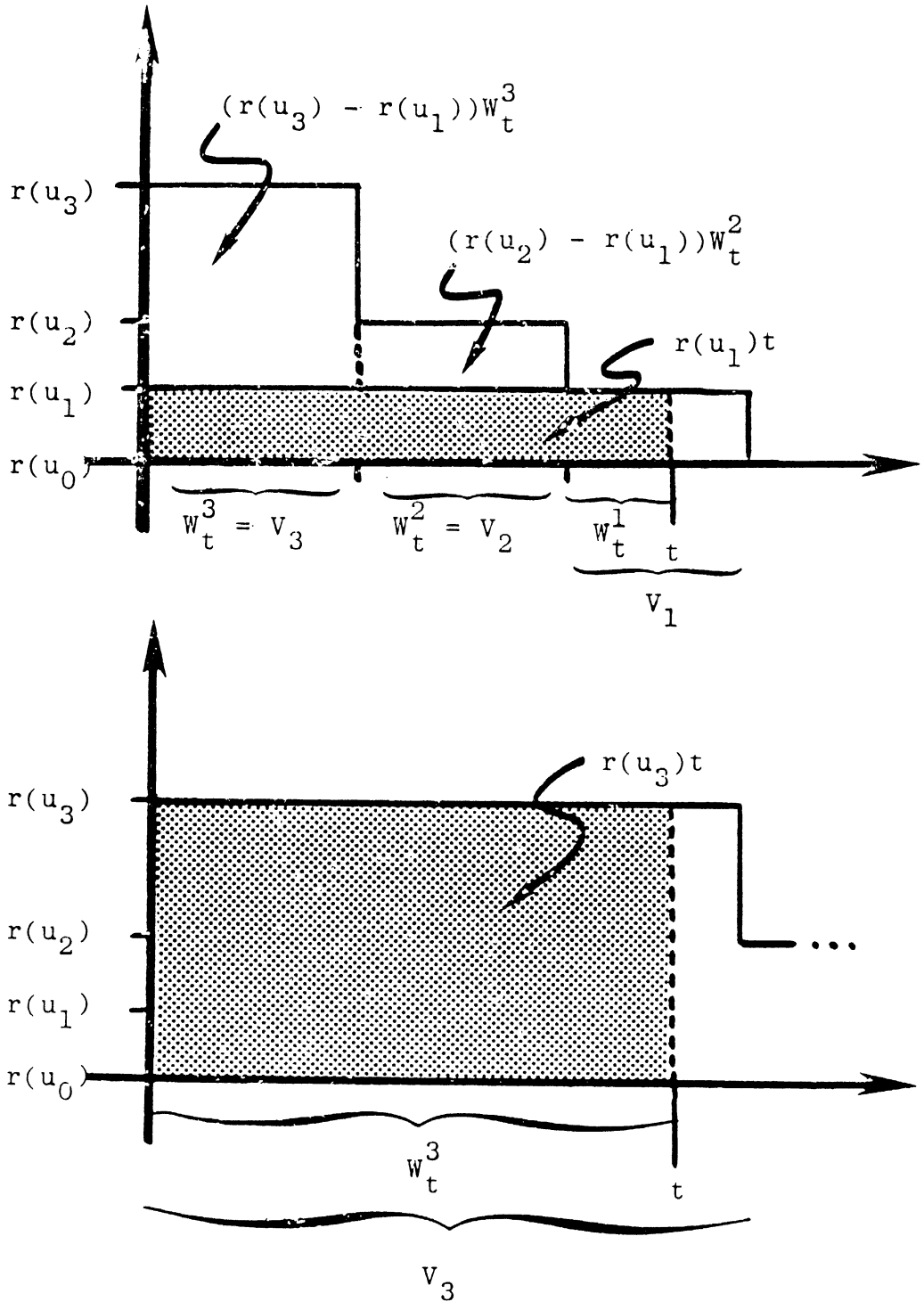


Fig. 5.2 Decomposition of $\overline{\gamma}_u(V_u)$

traverse the state sequence $\mathbf{u}$ and provide reward no greater than y. Then

$$F_{Y|U}(y|\mathbf{u}) = \text{Prob}[\mathbf{V_u} \in \mathbf{C}_y \mid \mathbf{U} = \mathbf{u}] = \int_{\mathbf{C}_y} f_{\mathbf{V_u}|\mathbf{U}}(\mathbf{v_u} \mid \mathbf{u})\, d\mathbf{v_u} \,. \tag{5.31}$$

Since the probability density function $f_{\mathbf{V_u}|\mathbf{U}}(\,\cdot\,)$ is assumed to exist and be known, to formulate $F_{Y|U}$ we must determine $\mathbf{C}_y$.

5.3.4. Formulation of $F_{Y|U}$

In the next section (Section 5.3.5 [i-Resolvability]), we show that, depending on the values of $\mathbf{u}$ and y, $\mathbf{C}_y$ can be easily partitioned, i.e., expressed as a union of disjoint regions $\mathbf{C}_y^i$. Such a partition provides the ability to break up $F_{Y|U}$ into a sum of smaller, less complex integrations. In particular, if the length of the sequence $\mathbf{u}$ is $n+1$, $\mathbf{C}_y$ will be decomposed into $n+2$ disjoint regions

$$\{\mathbf{C}_y^{n+1}, \mathbf{C}_y^n, \ldots, \mathbf{C}_y^0\} \,. \tag{5.32}$$

(The disjoint sets $\mathbf{C}_y^i$ will correspond to those trajectories which are "i-resolvable". See Section 5.3.5 [i-Resolvability]. For the discussion of this section, it is sufficient to consider the $\mathbf{C}_y^i$ to be arbitrary disjoint sets.) Let $\mathbf{v_u}$ be a point in $\mathbf{C}_y^i$ and let

$$C_{y,n}^i = \Big\{ v_n \mid \text{there exist } v_{n-1}, v_{n-2}, \ldots, v_0 \text{ such that } (v_n, v_{n-1}, \ldots, v_0) \in \mathbf{C}_y^i \Big\} \,, \tag{5.33}$$

and for $n > j \geq 0$, let

$$C_{y,j}^i(v_n, v_{n-1}, \ldots, v_{j+1}) = \Big\{ v_j \mid \text{there exist } v_{j-1}, v_{j-2}, \ldots, v_0 \text{ such that } (v_n, v_{n-1}, \ldots, v_0) \in \mathbf{C}_y^i \Big\} \,. \tag{5.34}$$

In other words, $C_{y,j}^i(v_n, v_{n-1}, \ldots, v_{j+1})$ is the set of all values v_j that are "contained" in

some $\mathbf{v}_u \in \mathbf{C}_y^i$. For conciseness, we shall write $C_{y,j}^i$ in place of $C_{y,j}^i(v_n, v_{n-1}, \ldots, v_{j+1})$ when the $(v_n, v_{n-1}, \ldots, v_{j+1})$ is implicit.

Consider $\bar{\gamma}_u$ (Eq. 5.30). The regions $\mathbf{C}_y^i$ are *not* generally Cartesian, that is, $\mathbf{C}_y^i$ is usually not expressible as the cross-product of the $C_{y,j}^i$. Rather, a different relationship based on "i-resolvability" (see Section 5.3.5 [i-Resolvability]) will be employed to define the $\mathbf{C}_y^i$.

By the definitions of the $\mathbf{C}_y^i$ and $C_{y,j}^i$, Eq. 5.31 can now be expressed as

$$\begin{aligned} F_{Y|U}(y|\mathbf{u}) &= \sum_{i=0}^{n+1} \text{Prob}[\mathbf{V}_u \in \mathbf{C}_y^i \,|\, \mathbf{U} = \mathbf{u}] \;=\; \sum_{i=0}^{n+1} \int_{\mathbf{C}_y^i} f_{\mathbf{V}_u|\mathbf{U}}(\mathbf{v}_u \,|\, \mathbf{u})\, d\mathbf{v}_u \\ &= \sum_{i=0}^{n+1} \int_{C_{y,n}^i} \int_{C_{y,n-1}^i} \cdots \int_{C_{y,0}^i} f_{\mathbf{V}_u|\mathbf{U}}(\mathbf{v}_u \,|\, \mathbf{U})\, d\mathbf{v}_u \;. \end{aligned} \tag{5.35}$$

The expansion of $f_{\mathbf{V}_u|\mathbf{U}}(\mathbf{v}_u \,|\, \mathbf{u})$ in Eq. 5.27 is especially appropriate in this representation. Finally, combining Eqs. 5.25, 5.27, and 5.35 with the characterization of the $C_{y,j}^i$ developed below, we arrive at an integral solution for the PDF F_Y.

5.3.5. i-Resolvability

5.3.5.1. Definition of i-Resolvability

In this section, we describe a method of partitioning the set $\mathbf{C}_y = \gamma_u^{-1}(B_y)$ into the $\mathbf{C}_y^i$ (Eq. 5.32) and characterizing $C_{y,j}^i$ (Eqs. 5.33 and 5.34). The immediate difficulty with characterizing the $\mathbf{C}_y^i$ arises from the nature of γ_u (Eq. 5.30), viz., γ_u is a sum of a random number of random variables $V_n, V_{n-1}, \ldots, V_{j+1}$. We introduce the notion of "i-resolvability," which allows the decomposition of the single large problem of describing the sum of a varying number of random variables into a set of smaller problems, each consisting of describing the sum of a fixed number of random variables. The number of random variables to be considered will be determined by i-resolvability, a notion which takes full advantage of both the monotonicity of γ_u and the finite utilization period. The concept can be loosely stated: "For a given subtrajectory, based on the past and regardless of the future, what can we say about the entire trajectory?"

We are interested in determining the probability that the total accumulated reward $\gamma_u(\mathbf{V}_u) \leq y$. Recall that the time the process resides in each state of the sequence $\mathbf{u}$ is given by the random variable $\mathbf{V}_u = (v_n, v_{n-1}, \ldots, v_0)$. Suppose that the sojourn times for the first $n-i+1$ of these values are known, i.e., $(V_n, V_{n-1}, \ldots, V_i) = (v_n, v_{n-1}, \ldots, v_i)$. Knowledge of these times conveys significant information regarding the set of possible sojourn times $(V_{i-1}, V_{i-2}, \ldots, V_0)$ that can be associated with the remaining states and still satisfy $\gamma_u(\mathbf{V}_u) \leq y$. As an informal introduction, imagine that someone chooses at random some vector $\mathbf{v}_u$ of "possible" sequence times and starts dealing, in order and one at a time, the values $v_n, v_{n-1}, \ldots, v_0$. We look at the values as they are dealt. If ***after*** the value v_i has been dealt–but not before–we can determine with certainty that $\gamma_u(\mathbf{v}_u) \leq y$, then we shall say that $\mathbf{v}_u$ is such that $\gamma(\mathbf{v}_u) \leq y$ is *i-resolvable based on* $\mathbf{v}_u$, or, more succinctly, $\mathbf{v}_u$ is *i-resolvable* (y and γ_u are implicit). If at any time we can determine with certainty that $\gamma_u(\mathbf{v}_u) > y$, then $\mathbf{v}_u \notin \mathbf{C}_y$; we are not interested in that value $\mathbf{v}_u$ (since it does not contribute to Prob$[Y \leq y]$) and so we reject it.

We now formalize this property. Let $\gamma_u : \mathbb{R}^n \rightarrow \mathbb{R}$ and $\mathbf{v}_u = (v_n, v_{n-1}, \ldots, v_0) \in \mathbb{R}^n$ be such that $\gamma_u \leq y$. In addition, let $\gamma_u(v_n, v_{n-1}, \ldots, v_i, \bullet, \bullet, \ldots, \bullet)$ be the $(n-i+1)$-variable function derived from $\gamma_u(\bullet, \bullet, \ldots, \bullet)$ by setting $V_n = v_n$, $V_{n-1} = v_{n-1}$, $\ldots$, $V_i = v_i$. When y is (with probability one) an upper bound[4] of the function $\gamma_u(v_n, v_{n-1}, \ldots, v_i, \bullet, \bullet, \ldots, \bullet)$ but not of $\gamma_u(v_n, v_{n-1}, \ldots, v_{i+1}, \bullet, \bullet, \ldots, \bullet)$, we shall say that $\mathbf{v}_u$ is i-resolvable. [As a special case, when y bounds $\gamma_u(\bullet, \bullet, \ldots, \bullet)$ (with probability one), then $\mathbf{v}_u$ is $(n+1)$-resolvable.] In other words, when a determination as to whether $\gamma_u(\mathbf{v}_u) \leq y$ can <u>not</u> be made based solely on the values $(v_n, v_{n-1}, \ldots, v_{i+1})$, but such a determination <u>can</u> be made with the additional knowledge of the value of v_i, we shall say that $\mathbf{v}_u$ is i-resolvable. [When $\mathbf{v}_u$ is $(n+1)$-resolvable, then the determination can be made with no knowledge of $\mathbf{v}_u$.] (Except where necessary, we shall drop the phrase "with probability one" when referring to y bounding $\gamma_u(v_n, v_{n-1}, \ldots, v_i, \bullet, \bullet, \ldots, \bullet)$.)

---

[4] A constant y is an ***upper bound*** of the function $f : D \rightarrow R$ if for all $x \in D$, $f(x) \leq y$.

To motivate our interest in i-resolvability, consider the following example. Suppose U is such that $(r(u_2), r(u_1), r(u_0)) = (2, 1, 0)$ and that $y = 5$ and $t = 4$. If $V_2 = 1$, then for $\gamma_u(\mathbf{V}_u) \leq y$ to be true, we must have $W_t^1 \leq t - V_2 = 4 - 1 = 3$ (see Eq. 5.28), and

$$\sum_{i=0}^{2} r(u_i) W_t^i = 1 \cdot W_t^1 + 1 \cdot 2 = W_t^1 + 2 \leq 3 + 2 = 5 = y \ . \tag{5.36}$$

Now $W_t^1 \leq 3$ irrespective of V_1, so V_1 can therefore take on *any* value in the interval $[0, \infty)$ and still satisfy $\gamma_u(\mathbf{V}_u) \leq y$. In other words, since $V_2 = 1$ then, without further examination of V_1 or V_0, we know $\gamma_u(\mathbf{V}_u) \leq y$. Thus, every $(1, V_1, V_0)$ is 2-resolvable.

$\mathbf{v}_u$ is 3-resolvable if $y \geq v_2 t = 2 \cdot 4 = 8$ since we know that $\gamma_u(\mathbf{v}_u) \leq y$ without examining any of the v_i. $\mathbf{v}_u$ is 1-resolvable if

$$\begin{aligned} &i)\ y < 8 \ , \text{ and} \\ &ii)\ v_2 \in \left[0, \frac{y - 1t}{2 - 1}\right] = [0, y - 4] \ , \end{aligned} \tag{5.37}$$

(e.g., if $v_2 = 0$ when $y = 4$), since if v_2 is in the above range, then $\gamma_u(\mathbf{v}_u) \leq y$ regardless of the values of v_1 and v_0. Also, $\mathbf{v}_u$ is 2-resolvable if

$$\begin{aligned} &i)\ y < 8 \ , \text{ and} \\ &ii)\ v_2 \in (y - 4, \infty) \ . \end{aligned} \tag{5.38}$$

5.3.5.2. Properties of i-Resolvability

5.3.5.2.1. General Properties

From the definition of i-resolvability, we have the following important

Theorem 5.7: (i-resolvability and bounds of γ_u)

$\mathbf{v}_u$ *is* i*-resolvable if and only if*

A. for $i = n+1$ *then*

$\gamma_u(\bullet, \bullet, \ldots, \bullet)$, $\gamma_u(v_n, \bullet, \ldots, \bullet), \ldots,$

$\gamma_u(v_n, v_{n-1}, \ldots, v_0)$ *are* <u>*all*</u> *bounded by* y, *and*

B. for $n \geq i \geq 0$ *then*

i) $\gamma_u(v_n, v_{n-1}, \ldots, v_i, \bullet, \bullet, \ldots, \bullet)$,

$\gamma_u(v_n, v_{n-1}, \ldots, v_{i-1}, \bullet, \bullet, \ldots, \bullet)$,

$\ldots, \gamma_u(v_n, v_{n-1}, \ldots, v_0)$ *are* <u>*all*</u> *bounded by* y, *and*

ii) <u>*None*</u> *of* $\gamma_u(\bullet, \bullet, \ldots, \bullet)$, $\gamma_u(v_n, \bullet, \ldots, \bullet), \ldots,$

$\gamma_u(v_n, v_{n-1}, \ldots, v_{i+1}, \bullet, \bullet, \ldots, \bullet)$ *are bounded by* y.

<u>Proof:</u> A: This is simply the definition of (n+1)-resolvability.

B: $\Rightarrow$: Suppose $\mathbf{v}$ is i-resolvable. Then by definition $\gamma_u(v_n, v_{n-1}, \ldots, v_i, \bullet, \bullet, \ldots, \bullet) \leq y$. Let $\hat{v}_{i-1}, \hat{v}_{i-2}, \ldots, \hat{v}_0$ be the values of $v_{i-1}, v_{i-2}, \ldots, v_0$ that maximizes $\gamma_u(v_n, v_{n-1}, \ldots, v_i, \bullet, \bullet, \ldots, \bullet)$. But then

$$y \geq \gamma_u(v_n, v_{n-1}, \ldots, v_i, \hat{v}_{i-1}, \bullet, \ldots, \bullet) \geq \geq \gamma_u(v_n, v_{n-1}, \ldots, v_i, v_{i-1}, \bullet, \ldots, \bullet) \tag{5.39}$$

$$y \geq \gamma_u(v_n, v_{n-1}, \ldots, v_i, \hat{v}_{i-1}, \hat{v}_{i-2}, \bullet, \ldots, \bullet) \geq \geq \gamma_u(v_n, v_{n-1}, \ldots, v_i, v_{i-1}, v_{i-2}, \bullet, \ldots, \bullet) \tag{5.40}$$

.
.
.

$$y \geq \gamma_u(v_n, v_{n-1}, \ldots, v_i, \hat{v}_{i-1}, \hat{v}_{i-2}, \ldots, \hat{v}_0) \geq$$

$$\geq \gamma_u(v_n, v_{n-1}, \ldots, v_i, v_{i-1}, v_{i-2}, \ldots, v_0) \tag{5.41}$$

and so y is the upper bound of $\gamma_u(v_n, v_{n-1}, \ldots, v_i, v_{i-1}, \bullet, \ldots, \bullet)$,

$\gamma_u(v_n, v_{n-1}, \ldots, v_i, v_{i-1}, v_{i-2}, \bullet, \ldots, \bullet), \ldots, \gamma_u(v_n, v_{n-1}, \ldots, v_i, v_{i-1}, v_{i-2}, \ldots, v_0)$.

Now again by the definition of i-resolvability, $\gamma_u(v_n, v_{n-1}, \ldots, v_{i+1}, \bullet, \bullet, \ldots, \bullet)$ is not bounded by y and so there must exist values $\hat{v}_i, \hat{v}_{i-1}, \ldots, \hat{v}_0$ such that $\gamma_u(v_n, v_{n-1}, \ldots, v_{i+1}, \hat{v}_i, \ldots, \hat{v}_0) > y$. But then $\gamma_u(v_n, v_{n-1}, \ldots, v_{i+2}, \bullet, \bullet, \ldots, \bullet)$ is not bounded by y for we could choose $v_{i+1}, \hat{v}_i, \ldots, \hat{v}_0$ as arguments. Similarly for $\gamma_u(v_n, v_{n-1}, \ldots, v_{i+3}, \bullet, \bullet, \ldots, \bullet), \ldots, \gamma_u(\bullet, \bullet, \ldots, \bullet)$. Thus, $\gamma_u(v_n, v_{n-1}, \ldots, v_{i+2}, \bullet, \bullet, \ldots, \bullet), \ldots, \gamma_u(\bullet, \bullet, \ldots, \bullet)$ are not bounded by y.

$\Leftarrow$: The converse is trivial, since if y bounds $\gamma_u(v_n, v_{n-1}, \ldots, v_i, \bullet, \bullet, \ldots, \bullet)$ but not $\gamma_u(v_n, v_{n-1}, \ldots, v_{i+1}, \bullet, \bullet, \ldots, \bullet)$ then by definition $\mathbf{v}$ is i-resolvable.

||

One can define i-resolvability recursively, as follows:

Theorem 5.8: (Recursive definition of i-resolvability)

For a given $\mathbf{v}_u = (v_n, v_{n-1}, \ldots, v_0)$:

i) $\mathbf{v}$ is $(n+1)$-resolvable if and only if $\gamma_u(\mathbf{v}_u) \leq y$ regardless of the value $\mathbf{v}_u$.

ii) $\mathbf{v}$ is i-resolvable if $\mathbf{v}$ is not j-resolvable for all $j > i$ but $\gamma_u(v_n, v_{n-1}, \ldots, v_i, \bullet, \bullet, \ldots, \bullet) \leq y$ regardless of the values of $(v_{i-1}, v_{i-2}, \ldots, v_0)$.

Proof: The initialization step i) is obviously equivalent to the definition of $n+1$-resolvability. (Indeed, it is the definition.) Consider now step ii):

$\Rightarrow$: Let $\mathbf{v}_u$ be i-resolvable. Then by Theorem 5.7, y does not bound $\gamma_u(v_n, v_{n-1}, \ldots, v_{i+1}, \bullet, \bullet, \ldots, \bullet)$,

$\gamma_u(v_n, v_{n-1}, \ldots, v_{i+2}, \bullet, \bullet, \ldots, \bullet), \ldots, \gamma_u(\bullet, \bullet, \ldots, \bullet)$. Therefore, $\mathbf{v}_u$ cannot be j-

resolvable for any $j > i$. Further, by the definition of i-resolvability, $\gamma_u(v_n, v_{n-1}, \ldots, v_i, \bullet, \bullet, \ldots, \bullet) \leq y$.

$\Leftarrow$: Suppose the premise is true. Then $\mathbf{v}_u$ is not $i+1$-resolvable and so $\gamma_u(v_n, v_{n-1}, \ldots, v_{i+1}, \bullet, \bullet, \ldots, \bullet)$ is not bounded by y. But $\gamma_u(v_n, v_{n-1}, \ldots, v_i, \bullet, \bullet, \ldots, \bullet)$ is bounded, and so by definition, $\mathbf{v}$ is i-resolvable.

||

Theorem 5.9: (Conditions for $v_j = w_t^j$)

Let $n > i > 0$. *If* $\mathbf{v}_u$ *is* i*-resolvable, then* $v_j = w_t^j$ *for all* $n \geq j \geq i + 1$.

Proof: Assume otherwise, i.e., $v_j \neq w_t^j$. But then $w_t^{j-1} = w_t^{j-2} = \cdots = w_t^0 = 0$ (from Eq. 5.28). Thus,

$$\begin{aligned} \gamma_u(v_n, v_{n-1}, \ldots, v_j, \bullet, \bullet, \ldots, \bullet) &= \gamma_u(v_n, v_{n-1}, \ldots, v_{j-1}, \bullet, \bullet, \ldots, \bullet) \\ &= , \ldots , = \gamma_u(v_n, v_{n-1}, \ldots, v_i, \bullet, \bullet, \ldots, \bullet) \end{aligned} \tag{5.42}$$

since regardless of the values of $v_{j-1}, v_{j-2}, \ldots, v_0$, their contribution to γ_u will be zero. (See Eq. 5.30.) This contradicts the premise that $\mathbf{v}_u$ is i-resolvable.

||

5.3.5.2.2. Existence and Uniqueness Properties

Theorem 5.10: *If* $\mathbf{v}_u$ *is* i*-resolvable then* $\mathbf{v}_u$ *is not* j*-resolvable* for $j \neq i$

Proof: Clear from the definition of i-resolvability.

||

Theorem 5.11: $\mathbf{v_u}$ is $(n+1)$-resolvable if and only if $y \geq r(u_n)t$.

Proof: Definition of (n+1)-resolvability.

||

Theorem 5.12: (maximizing $\gamma_u(v_n, v_{n-1}, \ldots, v_{i+1})$)

Let $n \geq i > 0$. The minimum value v_i that maximizes the function $\gamma_u(v_n, v_{n-1}, \ldots, v_{i+1})$ is

$$\max\left\{ t - \sum_{j=i+1}^{n} v_j, 0 \right\} . \tag{5.43}$$

Further, this value is independent of the values $v_{i-1}, v_{i-2}, \ldots, v_0$.

Proof: Let $\gamma_u(v_n, v_{n-1}, \ldots, v_{i+1}, 0, 0, \ldots, 0) = \hat{y}$. Then

$$\gamma_u(v_n, v_{n-1}, \ldots, v_{i+1}, v_i, v_{i-1}, \ldots, v_0) = \hat{y} + \sum_{j=0}^{i} w_t^j r(u_j) . \tag{5.44}$$

Since every $w_t^j \geq 0$ and $r(u_j) \geq 0$, then $\sum_{j=0}^{i} w_t^j r(u_i) \geq 0$. So, $\gamma_u(v_n, v_{n-1}, \ldots, v_{i+1})$ is maximum when $\sum_{j=0}^{i} w_t^j r(u_i) \geq 0$ is maximum. Furthermore, $r(u_i) \geq r(u_{i-1}) \geq \cdots \geq r(u_0)$, so $\sum_{j=0}^{i} w_t^j r(u_i) \geq 0$ is maximum when w_t^i is maximum, and that value of w_t^i is $\max\left\{ t - \sum_{j=i+1}^{n} v_j, 0 \right\}$, i.e., w_t^i is all the remaining time in the utization interval $[0, t]$ after accounting for the intervals $(v_n, v_{n-1}, \ldots, v_{i+1})$. Also, note that in the derivation, the values $v_{i-1}, v_{i-2}, \ldots, v_0$ do not affect the value of v_i which maximizes $\gamma_u(v_n, v_{n-1}, \ldots, v_{i+1})$.

||

Theorem 5.13: *Every* $\mathbf{v}_u$ *such that* $\gamma_u(\mathbf{v}_u) \leq y$ *is* i*-resolvable for some* i*, where* $n+1 \geq i \geq 0$.

Proof: If $y \geq r(u_n)t$, then by Theorem 5.11, $\mathbf{v}_u$ is $(n+1)$-resolvable. Suppose $y < r(u_n)t$. Then $\gamma_u(\bullet, \bullet, \ldots, \bullet)$ is not bounded by y, but $\gamma_u(v_n, v_{n-1}, \ldots, v_0)$ is. Further, γ_u is monotonically non-decreasing, so for some i, it must be the case that $\gamma_u(v_n, v_{n-1}, \ldots, v_i)$ is bounded by y, but $\gamma_u(v_n, v_{n-1}, \ldots, v_{i+1})$ is not.

||

Theorem 5.14: (i for which there exist i-resolvable $\mathbf{v}_u$)

Let $y \in [r(u_{l-1})t, r(u_l)t)$, $n \geq l > 0$. *Then for all* i *such that* $l \geq i > 0$ *there exist vectors* $\mathbf{v}_u$ *such that* $\mathbf{v}_u$ *is* i*-resolvable. Further, for all other* i*, there do not exist any vectors* $\mathbf{v}_u$ *such that* $\mathbf{v}_u$ *is* i*-resolvable.*

Proof: (Here we will show only the existence of a single $\mathbf{v}_u$ that is i-resolvable. In Section 5.3.8.1 [Characterization of $C^i_{y,j}$], we will characterize all of the i-resolvable $\mathbf{v}_u$.) Let $y \in [r(u_{l-1})t, r(u_l)t)$, $n \geq l > 0$, and let i be such that $l \geq i > 0$. Let

$$v_n = \frac{y - r(u_{i-1})t}{r(u_n) - r(u_{i-1})}\ ,\ v_{n-1} = 0, \ldots, v_0 = 0\ . \tag{5.45}$$

Note that $v_i \leq t$ since

$$\begin{aligned} r(u_n) \geq r(u_{l-1}) \Rightarrow\ v_n &= \frac{y - r(u_{i-1})t}{r(u_n) - r(u_{u-1})} \\ &\leq \frac{r(u_{l-1}) - r(u_{i-1})t}{r(u_n) - r(u_{u-1})} \\ &\leq t\left[\frac{r(u_n) - r(u_{i-1})t}{r(u_n) - r(u_{u-1})}\right] \\ &= t\ . \end{aligned} \tag{5.46}$$

Then the function

$$\gamma_u(v_n, v_{n-1}, v_{n-2}, \ldots, v_{i+1}, v_i, \cdot, \cdot, \ldots, \cdot)$$

$$= \gamma_u\left(\frac{y - r(u_{i-1})t}{r(u_n) - r(u_{n-1})}, 0, 0, \ldots, 0, 0 \cdot, \cdot, \ldots, \cdot\right)$$

$$\leq r(u_n)v_n + r_{u-1})[t - v_n] \qquad \text{(by Theorem 5.12)} \tag{5.47}$$

$$= r(u_n)\left[\frac{y - r(u_{i-1})t}{r(u_n) - r(u_{i-1})}\right] + r(u_{i-1})\left[t - \frac{y - r(u_{i-1})t}{r(u_n) - r(u_{i-1})}\right]$$

$$= y$$

is bounded by y. Further, consider the function

$$\gamma_u(v_n, v_{n-1}, v_{n-2}, \ldots, v_{i+1}, \cdot, \cdot, \ldots, \cdot)$$

$$= \gamma_u\left(\frac{y - r(u_{i-1})t}{r(u_n) - r(u_{i-1})}, 0, 0, \ldots, 0, \cdot, \cdot, \ldots, \cdot\right). \tag{5.48}$$

If $v_i = t - v_n = t - \dfrac{y - r(u_{i-1})t}{r(u_n) - r(u_{i-1})}$, then the function of Eq. 5.48 has value

$$r(u_n)\left[\frac{y - r(u_{i-1})t}{r(u_n) - r(u_{i-1})}\right] + r(u_i)\left[\frac{y - r(u_{i-1})t}{r(u_n) - r(u_{i-1})}\right]$$

$$> r(u_n)\left[\frac{y - r(u_{i-1})t}{r(u_n) - r(u_{i-1})}\right] + r(u_{i-1})\left[\frac{y - r(u_{i-1})t}{r(u_n) - r(u_{i-1})}\right] \tag{5.49}$$

$$= y$$

The proper inequality is due to the condition $r(u_i) \neq r(u_{i-1})$. Therefore, $\gamma_u(v_n, v_{n-1}, \ldots, v_i)$ is bounded by y, but $\gamma_u(v_n, v_{n-1}, \ldots, v_i)+1$ is not, and so $\mathbf{v}_u$ is i-resolvable.

To prove the second part of the theorem, we must consider two cases: $> l$ and $ieq0$. First, suppose there exists a vector $\mathbf{v}_u$ which is i-resolvable for $i > l$. Then $\gamma_u(v_n, v_{n-1}, \ldots, v_i)$ is bounded by y. By Theorem 5.9, $v_n = w_i^n$,

$v_{n-1} = w_t^{n-1}, \ldots, v_{i+1} = w_t^{i+1}$, and by Theorem 5.12, $\gamma_u(v_n, v_{n-1}, \ldots, v_i)$ has maximum value when

$$\begin{aligned} v_{i-1} &= \max\left\{t - \sum_{j=i+1}^{n} v_j, 0\right\} \\ &= t - \sum_{j=i}^{n} v_j . \end{aligned} \tag{5.50}$$

That value is

$$\begin{aligned} \sum_{j=i+1}^{n} r(u_j)v_j + r(u_i)\left[t - \sum_{j=i+1}^{n} v_j\right] &\geq \sum_{j=i+1}^{n} r(u_i)v_j + r(u_i)\left[t - \sum_{j=i+1}^{n} v_j\right] \\ &\geq r(u_i)t \\ &> y \end{aligned} \tag{5.51}$$

and so $\gamma_u(v_n, v_{n-1}, \ldots, v_i)$ is not bounded by y, which contradicts $\mathbf{v}_u$ being i-resolvable.

Finally, if $i = 0$, then by definition, $\gamma_u(v_n, v_{n-1}, \ldots, v_0) \leq y$ but $\gamma_u(v_n, v_{n-1}, \ldots, v_1, \cdot)$ is not, i.e., there exists some $\hat{v}_0$ such that $\gamma_u(v_n, v_{n-1}, \ldots, v_1, \hat{v}_0) > y$. However, $v_0 = \infty$ with probability one and so $\gamma_u(v_n, v_{n-1}, \ldots, v_1, \cdot) = \gamma_u(v_n, v_{n-1}, \ldots, v_0) > y$, and there can be no 0-resolvable $\mathbf{v}_u$.

||

Theorem 5.15: (Some conditions for which there does not exist any $\mathbf{v}_u$ such that $\mathbf{v}_u$ is 0-resolvable)

There does not exist any $\mathbf{v}_u$ *such that* $\mathbf{v}_u$ is i-resolvable *if any of the following are true:*

A. $i = n + 1$ *and* $y < r(u_n)t$;

B. $i \leq n$ *and* $r(u_i) = 0$;

C. $0 \leq i \leq n$ *and* $r(u_i) = r(u_{i-1})$;

D. $0 < i \leq n$, $r(u_i) \neq r(u_n)$, *and there does not exist any* $\mathbf{v}_u$ *that is* j*-resolvable, where* j *is the largest* $k \neq 0$ *such that* $k < i$ *and for which* $r(u_k) \neq r(u_i)$;

E. $i \leq n$ *and* $y \geq r(u_i)t$; *or*

F. $i = 0$.

Proof:

A. Theorem 5.11.

B. $r(u_i) = 0 \Rightarrow \gamma_u(v_n, v_{n-1}, \ldots, v_{i+1}) = \gamma_u(v_n, v_{n-1}, \ldots, v_i)$ for all v_i .

$\Rightarrow$ y bounds $\gamma_u(v_n, v_{n-1}, \ldots, v_{i+1})$ if and only if y bounds $\gamma_u(v_n, v_{n-1}, \ldots, v_i)$.

$\Rightarrow$ There is no $\mathbf{v}_u$ such that y is an upper bound of $\gamma_u(v_n, v_{n-1}, \ldots, v_i)$ but not of $\gamma_u(v_n, v_{n-1}, \ldots, v_{i+1})$.

$\Rightarrow$ There is no $\mathbf{v}_u$ that is i-resolvable .

C. $r(u_i) = r(u_{i-1}) \Rightarrow \gamma_u(v_n, v_{n-1}, \ldots, v_{i+1}, v_i, v_{i-1}, \cdot, \cdot, \ldots, \cdot)$

$= \gamma_u(v_n, v_{n-1}, \ldots, v_{i+1}, 0, v_{i-1} + v_i, \cdot, \cdot, \ldots, \cdot)$,

i.e., v_i is replaced by 0 and v_{i-1} is replaced by $v_{i-1} + v_i$. Now, for $\mathbf{v}_u$ to be i-resolvable, $\mathbf{v}_u$ cannot be $(n+1)$-resolvable, n-resolvable, . . . , $(i+1)$-resolvable; see Theorem 5.10.

Consider first the case $i = n$ and suppose $\mathbf{v}_u$ is not $(n+1)$-resolvable. Then

$$y < \gamma_u(t, \cdot, \cdot, \ldots, \cdot) \quad \text{(Theorem 5.11)}$$

$$\leq \gamma_u(0, v_{n-1} + t, \cdot, \cdot, \ldots, \cdot) \tag{5.52}$$

$$\leq \gamma_u(v_n, v_{n-1} + t, \cdot, \cdot, \ldots, \cdot) \quad \text{for any } v_n \geq 0 .$$

$\Rightarrow$ y does not bound $\gamma_u(\mathbf{v}_u, \cdot, \cdot, \ldots, \cdot)$.

$\Rightarrow$ There does not exist a $\mathbf{v}_u$ such that $\mathbf{v}_u$ is n-resolvable.

Consider now the case $i < n$ and suppose $\mathbf{v}_u$ is not $(n+1)$-resolvable, n-resolvable, . . . , $(i+1)$-resolvable. Then in particular, y does not bound $\gamma_u(v_n, v_{n-1}, \ldots, v_{i+1})$ and so

$$y \leq \gamma_u(v_n, v_{n-1}, \ldots, v_{i+1}, t - \sum_{i+1}^{n} v_j, \cdot, \cdot, \ldots, \cdot) \tag{5.53}$$

(see Theorem 5.12), i.e., v_i is replaced by the value that maximizes the function $\gamma_u(v_n, v_{n-1}, \ldots, v_{i+1})$.

$$\Rightarrow y \leq \gamma_u(v_n, v_{n-1}, \ldots, v_{i+1}, 0, v_{i-1} + t - \sum_{i+1}^{n} v_j, \cdot, \cdot, \ldots, \cdot) ,$$

that is, the maximizing value of v_i is replaced by 0 and v_{i-1} is replaced by $v_{i-1} + t - \sum_{i+1}^{n} v_j$.

$$\Rightarrow y \leq \gamma_u(v_n, v_{n-1}, \ldots, v_{i+1}, v_i, v_{i-1} + t - \sum_{i+1}^{n} v_j, \cdot, \cdot, \ldots, \cdot) , \text{ for any } v_i \geq 0 .$$

$\Rightarrow$ y does not bound $\gamma_u(v_n, v_{n-1}, \ldots, v_i)$ for any $\mathbf{v}_u$.

$\Rightarrow$ there does not exist a $\mathbf{v}_u$ such that $\mathbf{v}_u$ is i-resolvable.

D. If $y \geq r(u_n)t$, then by Theorem 5.11, no $\mathbf{v}_u$ are i-resolvable for $n \geq i \geq 0$ and the claim is obvious. Suppose $y < r(u_n)t$. Then y must be in some region $[r(u_{l-1})t, r(u_l)t)$ for $n \geq l > 0$. Then by Theorem 5.14, there exist i-resolvable $\mathbf{v}_u$ for all $l \geq i > 0$. If, as given, there is a $j < i$ such that $r(u_j) \neq r(u_i)$ and no $\mathbf{v}_u$ is j-resolvable, then $j > l$ and again by Theorem 5.14, there is no k-resolvable $\mathbf{v}_u$; specifically, there is no i-resolvable $\mathbf{v}_u$.

E. Theorem 5.14.

F. Theorem 5.14.

||

5.3.6. An Algorithm for Determining $\mathbf{C}_y$

Recalling that $\mathbf{C}_y = \gamma^{-1}(B_y)$, let

$$\mathbf{C}_y^i = \{\mathbf{v}_u \in \gamma^{-1}(B_y) \text{and} \mathbf{v}_u \text{ is } i\text{-resolvable}\}. \tag{5.54}$$

Lemma 5.16: The $\mathbf{C}_y^i$ *are mutually disjoint. That is,*

$$\mathbf{C}_y^i \bigcap \mathbf{C}_y^j = \phi \quad \text{for } i \neq j. \tag{5.55}$$

Proof: There can be no $\mathbf{v}_u$ such that simultaneously $\mathbf{v}_u$ is i-resolvable and $\mathbf{v}_u$ is j-resolvable, $i \neq j$ (Theorem 5.10). Therefore, no $\mathbf{v}_u$ can be in both $\mathbf{C}_y^i$ and $\mathbf{C}_y^j$, for $i \neq j$.

||

Further, we also have

Lemma 5.17: *The* $\mathbf{C}_y^i$ *cover* $\mathbf{C}_y$. That is,

$$\mathbf{C}_y = \mathbf{C}_y^n \bigcup \mathbf{C}_y^{n-1} \bigcup \cdots \bigcup \mathbf{C}_y^0 \tag{5.56}$$

Proof: Since ever $\mathbf{v}_u$ is i-resolvable for some i (see Theorem 5.13), every $\mathbf{v}_u$ must belong to some $\mathbf{C}_y^i$.

||

Combining Lemmas 5.16 and 5.17, we find that

Theorem 5.18: *The* $\mathbf{C}_y^i$ *partition* $\mathbf{C}_y$.

The iterative definition of "i-resolvability" (Theorem 5.8) yields the following algorithm for determining the $\mathbf{C}_y^i$:

Algorithm 5.3: (Determining the $\mathbf{C}_y^i$)

For a given B_y,

i) Determine $\mathbf{C}_y^{n+1}$, i.e., all $\mathbf{v}_u \in \gamma^{-1}(B_y)$ such that $\mathbf{v}_u \in \gamma^{-1}(B_y)$ regardless of the value of $\mathbf{v}_u$. These are the $\mathbf{v}_u$ which are $(n+1)$-resolvable. ($\mathbf{C}_y^{n+1}$ will either be empty or all $\mathbf{C}_y$, depending on the value of y.)

ii) For each $i = n, n-1, \ldots, 0$, determine all $\mathbf{v}_u \in \gamma^{-1}(B_y)$ such that $\mathbf{v}_u \notin \mathbf{C}_y^n \bigcup \mathbf{C}_y^{n-1} \bigcup \cdots \bigcup \mathbf{C}_y^{i+1}$ and $\mathbf{v}_u \in \gamma^{-1}(B_y)$ regardless of the values of $(v_{i-1}, v_{i-2}, \ldots, v_0)$.

5.3.7. Conditions for $\mathbf{v}_u$ Being i-Resolvable

To show that a given vector $\mathbf{v}_u$ is i-resolvable, the definition of i-resolvability may be used directly. We need to consider three cases of i:

1) $i = n + 1$: $\mathbf{v}_u$ is $(n+1)$-resolvable if and only if $y \geq r(u_n)t$. (See Theorem 5.11.)

2) $i = 0$: No $\mathbf{v}_u$ is 0-resolvable. (See Theorem 5.15.)

3) $n \geq i > 0$: Whether $\mathbf{v}_u$ is i-resolvable depends on the values of the components v_j in $\mathbf{v}_u$.

The first two are the trivial boundary cases $i = n+1$ and $i = 0$; the third, more difficult, case consists of all the values in between. The remainder of this section considers only case 3). **In this section, i is restricted to $n, n-1, \ldots, 1$.** By the definition of i-resolvable, to show $\mathbf{v}_u$ is i-resolvable, one need only show:

a) y bounds $\gamma(v_n, v_{n-1}, \ldots, v_i, \bullet, \bullet, \ldots, \bullet)$, i.e., there does not exist any $\tilde{v}_{i-1}$, $\tilde{v}_{i-2}, \ldots, \tilde{v}_0$ such that

$$\gamma(v_n, \ldots, v_i, \tilde{v}_{i-1}, \tilde{v}_{i-2}, \ldots, \tilde{v}_0) > y, \tag{5.57}$$

and,

b) if $i \neq n$, then y does not bound $\gamma(v_n, v_{n-1}, \ldots, v_{i+1}, \bullet, \bullet, \ldots, \bullet)$ [if $i = n$ then y does not bound $\gamma(\bullet, \bullet, \ldots, \bullet)$], i.e., there exist $\hat{v}_i, \hat{v}_{i-1}, \ldots, \hat{v}_0$ such that

$$\gamma(v_n, \ldots, v_{i+1}, \hat{v}_i, \hat{v}_{i-1}, \ldots, \hat{v}_0) > y \tag{5.58}$$

Condition a) can be shown by examining the set of $\tilde{v}_{i-1}, \tilde{v}_{i-2}, \ldots, \tilde{v}_0$ that maximizes $\gamma(v_n, v_{n-1}, \ldots, v_i, \bullet, \bullet, \ldots, \bullet)$. From Theorem 5.12, those values are

$$\tilde{v}_{i-1} = \max(t - \sum_{j=1}^{n} v_j, 0), \quad i \neq 1 \tag{5.59}$$

$$\begin{aligned} &\tilde{v}_j \text{ is arbitrary, } i-1 > j > 0, \\ &\tilde{v}_0 = \infty \text{ (since } v_0 \text{ is absorbing).} \end{aligned} \tag{5.60}$$

That is, $\tilde{v}_{i-1}$ is all the remaining time in $[0, t]$ after accounting for the intervals

$(v_n, v_{n-1}, \ldots, v_i)$. Let[5]

$$\tilde{\mathbf{v}}_u = (v_n, \ldots, v_i, \tilde{v}_{i-1}, \tilde{v}_{i-2}, \ldots, \tilde{v}_0) \ . \tag{5.61}$$

Then $\gamma(\tilde{\mathbf{v}}_u)$ is the maximum value of $\gamma(v_n, v_{n-1}, \ldots, v_i, \bullet, \bullet, \ldots, \bullet)$. To examine the behavior of the w_t^i, let

$$\tilde{\mathbf{w}}^u = (w_t^n, \ldots, w_t^i, \tilde{w}_{i-1}, \tilde{w}_{i-2}, \ldots, \tilde{w}_0) \tag{5.62}$$

be the vector of $\tilde{w}_i$ corresponding to $\tilde{\mathbf{v}}_u$. Now

$$\tilde{v}_{i-1} \geq w_t^{i-1} \tag{5.63}$$

and (if $i > 1$)

$$(\tilde{w}_{i-2}, \tilde{w}_{i-3}, \ldots, \tilde{w}_0) = (0, 0, \ldots, 0) \tag{5.64}$$

and so $\gamma(\mathbf{v}_u) \leq \gamma(\tilde{\mathbf{v}}_u)$ (see Eq. 5.30 and Theorem 5.12). Thus if $\gamma(\tilde{\mathbf{v}}_u) \leq y$, then $\gamma(v_n, v_{n-1}, \ldots, v_i, \bullet, \bullet, \ldots, \bullet) \leq y$.

To show condition b), define $\hat{\mathbf{v}}$ similarly to $\tilde{\mathbf{v}}_u$ above, i.e.,[6]

$$\hat{\mathbf{v}}_u = (v_n, \ldots, v_{i+1}, \hat{v}_i, \hat{v}_{i-1}, \ldots, \hat{v}_0) \ , \tag{5.65}$$

where again from Theorem 5.12

$$\hat{v}_i = \max(t - \sum_{j=i+1}^{n} v_j, 0) \ , \tag{5.66}$$

$$\hat{v}_j \text{ is arbitrary, } i > j > 0 \text{ , and} \qquad \hat{v}_0 = \infty \ . \tag{5.67}$$

---

[5] $\tilde{\mathbf{v}}_u$ should be modified by the index i, e.g., $\tilde{\mathbf{v}}_{u,i}$, since there is a different vector $\tilde{\mathbf{v}}_u$ for each i. However, throughout this discussion, the only index to be used is i; so to simplify notation, $\mathbf{v}_u$ will be employed.

[6] Again, $\hat{\mathbf{v}}_u$ should be modified by the index i, e.g., $\hat{\mathbf{v}}_{u,i}$, since there is a different vector $\hat{\mathbf{v}}_u$ for each i. However, as with $\mathbf{v}_u$, we shall simplify the notation by employing only $\mathbf{v}_u$.

That is, $\hat{v}_i$ is all the remaining time in $[0, t]$ after accounting for the intervals $(v_n, v_{n-1}, \ldots, v_{i+1})$. To study the w_t^i, let

$$\hat{\mathbf{w}}^{\mathbf{u}} = (w_t^n, \ldots, w_t^{i+1}, \hat{w}_t^i, \hat{w}_t^{i-1}, \ldots, \hat{w}_t^0) \tag{5.68}$$

be the vector of $\hat{w}_t^i$ corresponding to $\hat{\mathbf{v}}_{\mathbf{u}}$. When $\gamma(\hat{\mathbf{v}}) > y$, then it is <u>not</u> necessarily the case that $\gamma(\mathbf{v}_{\mathbf{u}}) > y$.

From conditions a) and b) (Eqs. 5.57 and 5.58), we have the following

Theorem 5.19: (Sufficient and necessary conditions for $\mathbf{v}_{\mathbf{u}}$ to be i-resolvable)

$\mathbf{v}_{\mathbf{u}}$ *is* i*-resolvable* $(n \geq i > 0)$ *if and only if*

$$\begin{aligned} &a) \quad \gamma(\tilde{\mathbf{v}}_{\mathbf{u}}) \leq y, \text{ and} \\ &b) \quad \gamma(\hat{\mathbf{v}}_{\mathbf{u}}) > y \ . \end{aligned} \tag{5.69}$$

5.3.8. Solution of Finite State Acyclic Reward Models

5.3.8.1. Characterization of $C_{y,j}^{i}$

5.3.8.1.1. Introduction

This section characterizes $C_{y,j}^i$. (See Eq. 5.33 for the definition of $C_{y,j}^i$.) Let $\mathbf{v}_{\mathbf{u}} = (v_n, v_{n-1}, \ldots, v_0)$ and suppose $\mathbf{v}_{\mathbf{u}}$ is i-resolvable. We can now state some restrictions on the possible values of the w_t^i and v_i; more restrictions will soon become apparent, but the following are obvious. (See Theorem 5.9.) For $\tilde{\mathbf{v}}$:

$$\left.\begin{aligned} 0 \leq \sum_{k=j}^{n} v_k \leq t \\ 0 \leq w_t^j = v_j \end{aligned}\right\} \quad \text{for each } j \text{ such that } n \geq j \geq i \tag{5.70}$$

$$\begin{aligned} &\sum_{k=i-1}^{n} v_k = t \\ &0 \le w_t^{i-1} \le \tilde{w}_{i-1} = \tilde{v}_{i-1} \end{aligned} \tag{5.71}$$

$$\left.\begin{aligned} &0 \le \sum_{k=i}^{n} w_t^k + \sum_{k=j}^{i-1} \tilde{w}_k = t \\ &\tilde{w}_j = 0 \end{aligned}\right\} \quad \text{for each } j \text{ such that } i-1 > j \ge 0 \ , \tag{5.72}$$

and for $\hat{\mathbf{v}}$:

If $n > i \ge 1$:

$$\left.\begin{aligned} &0 \le \sum_{k=j}^{n} v_k \le t \\ &0 \le w_t^j = v_j \end{aligned}\right\} \quad \text{for each } j \text{ such that } n \ge j \ge i+1 \tag{5.73}$$

$$\begin{aligned} &\sum_{k=i}^{n} v_k = t \\ &0 \le w_t^i \le \hat{w}_t^i = \hat{v}_i \end{aligned} \tag{5.74}$$

$$\left.\begin{aligned} &0 \le \sum_{k=i+1}^{n} w_t^k + \sum_{k=j}^{i} \hat{v}_k = t \\ &\hat{w}_t^j = 0 \end{aligned}\right\} \quad \text{for each } j \text{ such that } i > j \ge 0 \ . \tag{5.75}$$

Recalling the capability function (Eq. 5.30),

$$Y = \gamma(\mathbf{V}_u) = \begin{cases} r(u_j)t + \sum_{k=j+1}^{n} (r(u_k) - r(u_j))V_k \\ \qquad \text{if } \sum_{k=j+1}^{n} V_k < t, \ \sum_{k=j}^{n} V_k \ge t \ , \text{ for } n > j \\ r(u_n)t \quad \text{if } V_n \ge t \ , \end{cases} \tag{5.30}$$

along with the information concerning which of the $\tilde{w}_j$ and $\hat{w}_t^j$ are 0 (Eqs 5.72 and 5.75), the two conditions a) and b) of Eq. 5.69 can be rewritten as in the following (see also Eq. 5.29)

Corollary 5.20: (Sufficient and necessary conditions for $\mathbf{v}_u$ to be i-resolvable)

$\mathbf{v}_u$ *is* i*-resolvable* ($n \geq i > 0$) *if and only if*

$$a) \qquad y \geq \tilde{\gamma}(\mathbf{v}) = r(u_{i-1})t + \sum_{j=i}^{n} (r(u_j) - r(u_{i-1}))w_t^j \tag{5.76}$$

and

$$b) \qquad y < \begin{cases} \hat{\gamma}(\mathbf{v}) = r(u_i)t + \sum_{j=i+1}^{n} (r(u_j) - r(u_i))w_t^j \\ \qquad\qquad \text{if } n > i > 0 \\ \hat{\gamma}(\mathbf{v}) = r(u_n)t \qquad \text{if } i = n \end{cases} \tag{5.77}$$

As mentioned in Section 5.3.3 [The Approach], the w_t^j are too statistically dependent for convenient use, and so conditions a) and b) (Eqs. 5.76 and 5.77) must be rewritten in terms of the v_j (see Eq. 5.28). We use the conditions of Eqs. 5.76 and 5.77 along with the observations of Eq. 5.70-5.75.

The following sections develop in detail the regions $C_{y,j}^i$. Section 5.3.8.1.2 [Restatement of Condition a) in Terms of the v_j] characterizes Eq. 5.76 in terms of the v_j, while Section 5.3.8.1.3 [Restatement of Condition b) in Terms of the v_j] does the same for Eq. 5.77. Section 5.3.8.1.4 [The Regions $C_{y,j}^i$] then combines the results of Sections 5.3.8.1.2 [Restatement of Condition a) in Terms of the v_j] and 5.3.8.1.3 [Restatement of Condition b) in Terms of the v_j] and states the regions $C_{y,j}^i$.

5.3.8.1.2. Restatement of Condition a) in Terms of the v_j

Consider first condition a) (Eq. 5.76). In the derivation of this section will appear division by the term $r(u_j) - r(u_{i-1})$, $n > j \geq i$. It is possible that $r(u_j) = r(u_{i-1})$, in which case we would be dividing by zero. However, in such a case, since $r(u_j) \geq r(u_{j-1}) \geq \cdots \geq r(u_i) \geq r(u_{i-1})$, then $r(u_i) = r(u_{i-1})$. Thus, by Theorem 5.15 C), since $r(u_i) = r(u_{i-1})$, then there are no i-resolvable $\mathbf{v}_u$, in which case $\mathbf{C}_y^i = \phi$ and there are no

$C^{i}_{y,j}$ to characterize. Hence, because we are interested only in i-resolvable $\mathbf{v_u}$, we assume $r(u_i) \neq r(u_{i-1})$ and division by zero will never be performed.

Since every component of Eq. 5.76 is positive, and since all $v_i \geq 0$, we have a collection of conditions which must satisfied:

$$y \geq r(u_{i-1})t + (r(u_n) - r(u_{i-1}))w_t^n \geq 0 \tag{5.78}$$

(5.79)

$$y \geq r(u_{i-1})t + (r(u_n) - r(u_{i-1}))w_t^n + (r(u_{n-1}) - r(u_{i-1}))w_t^{n-1} \geq 0$$

$$\vdots$$

$$y \geq r(u_{i-1})t + \sum_{j=i}^{n} (r(u_j) - r(u_{i-1}))w_t^j \geq 0 \ . \tag{5.80}$$

Since *each* of the Eqs. 5.78-5.80 must be satisfied, we will have a set of $n - i + 1$ conditions, one each for w_t^n, w_t^{n-1}, . . . , w_t^i. The restrictions on the range of w_t^j will depend on the values of w_t^n, w_t^{n-1}, . . . , w_t^{j+1}. In addition, there are constraints on the w_t^j and v_j, noted in Eqs. 5.70-5.72. Incorporating the information of Eqs. 5.70-5.72 into Eqs. 5.78-5.80, we will obtain qualifications on the ranges of v_n, v_{n-1}, . . . , v_0. To be valid, the resulting ranges of the v_j present restrictions on the particular ranges of y which are admissible. We emphasize that the allowed ranges of the v_j are determined first–the corresponding ranges of y are derived from the v_j. Those restrictions on y are also noted below.

We do the easy ones first. The definition of $\tilde{v}_j$ (Eq. 5.60) yields for each j such that $i > j \geq 0$:

$$\begin{aligned} &0 \leq v_j < \infty \quad i > j > 0, \text{ and} \\ &v_0 = \infty \end{aligned} \tag{5.81}$$

Now suppose $j = n$. Then from Eq. 5.78 and Eq. 5.70:

$$0 \leq v_n = w_t^n \leq \frac{y - r(u_{i-1})t}{r(u_n) - r(u_{i-1})} \leq t, \tag{5.82}$$

and constraint ✳ must hold, where constraint ✳ is based on the two boundary constraints for v_n:

$$1) \quad 0 \leq v_n \Rightarrow \frac{y - r(u_{i-1})t}{r(u_n) - r(u_{i-1})} \geq 0 \tag{5.83}$$

$$\Rightarrow \quad y - r(u_{i-1})t \geq 0 \ \text{(since } r(u_n) > r(u_{i-1})\text{)} \tag{5.84}$$

$$\Rightarrow \quad y \geq r(u_{i-1})t \ , \tag{5.85}$$

$$2) \quad v_n \leq t \Rightarrow \frac{y - r(u_{i-1})t}{r(u_n) - r(u_{i-1})} < t \tag{5.86}$$

$$\Rightarrow \quad y < r(u_n)t \ . \tag{5.87}$$

Combining Eqs. 5.85 and 5.87, constraint ✳ reduces to

$$r(u_n)t > y \geq r(u_{i-1})t \ . \tag{5.88}$$

Now we consider the other v_j. From Eqs. 5.79-5.80 and Eq. 5.70, for each j such that $n > j \geq i$:

$$0 \leq v_j = w_t^j \leq \frac{y - r(u_{i-1})t - \sum_{k=j+1}^{n} (r(u_k) - r(u_{i-1}))v_k}{r(u_j) - r(u_{i-1})} \tag{5.89}$$

$$\leq t - \sum_{k=j+1}^{n} v_k$$

and constraint ✳✳ must hold, where constraint ✳✳ is based on the two boundary con-

straints for v_j:

$$1)\quad 0 \le v_j \Rightarrow \frac{y - r(u_{i-1})t - \sum_{k=j+1}^{n} (r(u_k) - r(u_{i-1}))v_k}{r(u_j) - r(u_{i-1})} \ge 0 \tag{5.90}$$

$$\Rightarrow \quad y - r(u_{i-1})t - \sum_{k=j+1}^{n} (r(u_k) - r(u_{i-1}))v_k \ge 0 \tag{5.91}$$

$$(\text{since } r(u_j) > r(u_{i-1}))$$

$$\Rightarrow \quad y \ge r(u_{i-1})t + \sum_{k=j+1}^{n} (r(u_k) - r(u_{i-1}))v_k \ , \tag{5.92}$$

$$2)\quad v_j \le t - \sum_{k=j+1}^{n} v_k \tag{5.93}$$

$$\Rightarrow \quad \frac{y - r(u_{i-1})t - \sum_{k=j+1}^{n} (r(u_k) - r(u_{i-1}))v_k}{r(u_j) - r(u_{i-1})} \le t - \sum_{k=j+1}^{n} v_k \tag{5.94}$$

$$\Rightarrow \quad y < r(u_j)t + \sum_{k=j+1}^{n} r(u_k)v_k \ . \tag{5.95}$$

Eq. 5.92 and 5.95 combine, allowing constraint ** to be written

$$r(u_j)t + \sum_{k=j+1}^{n} r(u_k)v_k \ > \ y \ \ge \ r(u_{i-1})t - \sum_{k=j+1}^{n} (r(u_k) - r(u_{i-1}))v_k \ . \tag{5.96}$$

At first glance, Eq. 5.96 appears more restrictive than Eq. 5.88 concerning the range of y which allows the existence of a vector $\tilde{\mathbf{v}}$ satisfying $\gamma(\tilde{\mathbf{v}}) \le y$. However, Eq. 5.96 follows from Eq. 5.88, which can be seen by combining Eqs. 5.88 and 5.96:

$$r(u_n)t \ \ge \ r(u_j)t + \sum_{k=j+1}^{n} r(u_k)v_k > \ y \ \ge \ r(u_{i-1})t - \sum_{k=j+1}^{n} (r(u_k) - r(u_{i-1}))v_k \tag{5.97}$$

$$\ge \ r(u_{i-1})t \ .$$

As long as y satisfies Eq. 5.88, then clearly there are $\mathbf{v}_u = (v_n, v_{n-1}, \ldots, v_j)$ for which $\gamma(\tilde{\mathbf{v}}) \leq y$.

The results of this section are summarized by the following:

Lemma 5.21: (Sufficient and necessary conditions on the v_i for $\gamma(\tilde{\mathbf{v}}) \leq y$)

Let $n \geq i > 0$. Then $\gamma(\tilde{\mathbf{v}}) \leq y$ if and only if

i) $r(u_i) \neq r(u_{i-1})$; (5.98)

ii) $r(u_n)t > y \geq r(u_{i-1})t$; (5.88

iii) *for each j such that $i > j \geq 0$:*

$0 \leq v_j < \infty \quad i > j > 0$, *and* (5.89)

$v_0 = \infty$;

iv) *for $j = n$:*

(5.82)

$$0 \leq v_n \leq \frac{y - r(u_{i-1})t}{r(u_n) - r(u_{i-1})} \text{ ; and}$$

v) *for each j such that $n > j \geq i$:*

$$0 \leq v_j \leq \frac{y - r(u_{i-1})t - \sum_{k=j+1}^{n} (r(u_k) - r(u_{i-1}))v_k}{r(u_j) - r(u_{i-1})} \tag{5.89}$$

$$\leq t - \sum_{k=j+1}^{n} v_k \ .$$

Note that in Eq 5.89, if $r(u_j) = r(u_{i-1})$ then $r(u_i) = r(u_{i-1})$ and so condition i) fails. Thus, division by zero does not occur.

5.3.8.1.3. Restatement of Condition b) in Terms of the v_j

In order that condition b) (Eq. 5.77) of Corollary 5.20 be satisfied, we have somewhat different constraints on the v_i than we had with condition a) (see the previous section). Division by the term $r(u_j) - r(u_i)$, $j > i$, occurs in the derivation of this section. It is possible that there exist $\mathbf{v}_u$ which are i-resolvable even when $r(u_j) = r(u_{j-1})$, $n \geq j > i$. To avoid interrupting the discussion with special cases, we temporarily assume $r(u_j) > r(u_i)$ for all $j > i$, and postpone considerationnn of the case $r(u_j) = r(u_{j-1})$ tilll the end of this section.

Lower bound restrictions on <u>every</u> v_j, $j \geq i$ are not required; only in certain cases does a given v_j have any lower bound constraints at all. The intuition here is as follows: If $\gamma(\hat{\mathbf{v}}) > y$, then only $v_n, v_{n-1}, \ldots, v_{i+1}, \hat{v}_i$ make any positive contribution to $\gamma(\hat{\mathbf{v}})$. $\hat{v}_i$ is defined to be $t - \sum_{j=i+1} v_j$ (see Eq. 5.66), so we are concerned just with specifying the ranges of $v_n, v_{n-1}, \ldots, v_{i+1}$. Once $v_n, v_{n-1}, \ldots, v_{j+1}$ are set, then either v_j must be at least as large as the value which causes $\gamma(\hat{\mathbf{v}}) > y$, or else, if v_{j-1} can be large enough to cause $\gamma(\hat{\mathbf{v}}) > y$, then there is no lower bound on v_j at all. As with condition a) (Eq. 5.76), there is a certain range of y for which the bounds on v_j hold, and this restriction will be noted momentarily.

Again we start with the easy case. For each j such that $i \geq j \geq 0$, the definition of $\hat{v}_j$ (Eq. 5.67) yields

$$0 \leq \hat{v}_j < \infty, \quad i \geq j > 0, \text{ and}$$
$$\hat{v}_0 = \infty . \tag{5.99}$$

When $i = n$, the bounds on the v_j can be quickly characterized:

If $i = n$, then for each j such that $n \geq j \geq 0$:

$$\begin{aligned} &0 \leq \hat{w}_t^n = \hat{v}_n = t \\ &0 \leq \hat{v}_j \leq \infty, \quad n > j > 0 \\ &\hat{v}_0 = \infty \\ &\hat{w}_t^j = 0, \quad n > j \geq 0 . \end{aligned} \tag{5.100}$$

and tautologically, the conditioning on y is $y < r(u_n)t$.

Now suppose $n > i$. From Eq. 5.77, we have only the following condition:

$$y < r(u_i)t + \sum_{j=i+1}^{n} (r(u_j) - r(u_i))w_t^j \ . \tag{5.101}$$

Replacing w_t^j with v_j (Eq. 5.73), Eq. 5.101 is written

$$y < r(u_i)t + \sum_{j=i+1}^{n} (r(u_j) - r(u_i))v_j \ . \tag{5.102}$$

We now find lower bounds on the v_j:

for $j = i+1$ (if $i < n-1$; a similar argument holds if $i = n - 1$):

$$v_{i+1} > \frac{y - r(u_i)t - \sum_{k=i+2}^{n} (r(u_k) - r(u_i))v_k}{r(u_{i+1}) - r(u_i)} \ , \qquad \text{(From Eq. 5.102)} \tag{5.103}$$

$$0 \leq v_{i+1} \leq t - \sum_{k=i+2}^{n} v_k \qquad \text{(from Eq. 5.73)} . \tag{5.104}$$

Then combining Eqs. 5.103 and 5.104,

$$t - \sum_{k=i+2}^{n} v_k \geq v_{i+1} > \frac{y - r(u_i)t - \sum_{k=i+2}^{n} (r(u_k) - r(u_i))v_k}{r(u_{i+1}) - r(u_i)} \tag{5.105}$$

The notation $a \underset{0}{>} b$ denotes

$$a \underset{0}{>} b \Rightarrow \begin{cases} a > b & \text{if } b > 0 \\ a \geq 0 & \text{if } b \leq 0 \end{cases} . \tag{5.106}$$

Intuitively, $a \underset{0}{>} b$ means "a can be any non-negative (or zero) number exceeding b." Now suppose the right hand term in in Eq. 5.105 causes the inequality to be false, i.e., suppose v_n, $v_{n-1}, \ldots, v_{i+2}$ are such that

$$t - \sum_{k=i+2}^{n} v_k \leq v_{i+1} < \frac{y - r(u_i)t - \sum_{k=i+2}^{n} (r(u_k) - r(u_i))v_k}{r(u_{i+1}) - r(u_i)} . \tag{5.107}$$

To prevent this inconsistency from occurring, we must insure that v_n, $v_{n-1}, \ldots, v_{i+2}$ are sufficiently large. Suppose $i < n-2$ (a similar argument applies if $i = n-2$) and consider v_{i+2}. The following condition prevents the situation of Eq. 5.107:

$$v_{i+2} > \frac{y - r(u_{i+1})t - \sum_{k=i+3}^{n} (r(u_k) - r(u_{i+1}))v_k}{r(u_{i+2}) - r(u_{i+1})} , \tag{5.108}$$

since this would imply

$$(r(u_{i+2}) - r(u_{i+1}))v_{i+2} > y - r(u_{i+1})t - \sum_{k=i+3}^{n} (r(u_k) - r(u_{i+1}))v_k \tag{5.109}$$

$$\Rightarrow \quad (r(u_{i+2}) - r(u_{i+1}))v_{i+2} - r(u_i)t + \sum_{k=i+2}^{n} r(u_i)v_k$$

$$> y - r(u_{i+1})t - \sum_{k=i+3}^{n} (r(u_k) - r(u_{i+1}))v_k \tag{5.110}$$

$$- r(u_i)t + \sum_{k=i+2}^{n} r(u_i)v_k$$

$$\Rightarrow \quad r(u_{i+1})t - r(u_i)t + \sum_{k=i+2}^{n} r(u_i)v_k$$

$$- \sum_{k=i+3}^{n} r(u_{i+1})v_k - r(u_{i+1})v_{i+2} \tag{5.111}$$

$$> \; y - r(u_i)t - \sum_{k=i+3}^{n} r(u_k)v_k - r(u_{i+2})v_{i+2} + \sum_{k=i+2}^{n} r(u_i)v_k$$

$$\Rightarrow \quad (r(u_{i+1}) - r(u_i))\left(t - \sum_{k=i+2}^{n} v_k\right)$$

$$> \; y - r(u_i)t - \sum_{k=i+2}^{n} (r(u_k) - r(u_i))v_k \tag{5.112}$$

$$\Rightarrow \quad t - \sum_{k=i+2}^{n} v_k > \frac{y - r(u_i)t - \sum_{k=i+2}^{n} (r(u_k) - r(u_i))v_k}{r(u_{i+1}) - r(u_i)} \tag{5.113}$$

which precludes Eq. 5.107. So, to insure Eq. 5.105, we need to insure Eq. 5.108.

Assume then that v_{i+2} does satisfy Eq. 5.108 (and that $i < n-3$; again, if $i = n-3$ then a similar argument holds). Then v_{i+2} must also satisfy

$$t - \sum_{k=i+3}^{n} v_k \geq v_{i+2} > 0 \frac{y - r(u_{i+1})t - \sum_{k=i+3}^{n} (r(u_k) - r(u_{i+1}))v_k}{r(u_{i+2}) - r(u_{i-1})} \quad , \tag{5.114}$$

which is similar to the condition of Eq. 5.105. Indeed, this is exactly Eq. 5.105 with i replaced by $i+1$. Proceeding recursively, we find that we must place conditions on v_n, $v_{n-1}, \ldots, v_{i+3}$ such that Eq. 5.114 is satisfied. There is thus a chain of conditions of the form

$$v_n > 0 \frac{y - r(u_{n-1})t}{r(u_n) - r(u_{n-1})} \quad , \tag{5.115}$$

and for $n > j > i$:

$$v_j \underset{0}{>} \frac{y - r(u_{j-1})t - \sum_{k=j+1}^{n} (r(u_k) - r(u_{j-1}))v_k}{r(u_j) - r(u_{j-1})} \quad . \tag{5.116}$$

Based on the above discussion, we make the following

Claim 5.22: (Conditions on $v_n, v_{n-1}, \ldots, v_{i+1}$ for v_{i+1} to satisfy Eq. 5.105)

Suppose $n > i \geq 0$. If $v_n, v_{n-1}, \ldots, v_{i+1}$ are all non-negative and satisfy Eqs. 5.115 and 5.116, then v_{i+1} satisfies Eq. 5.105. Indeed,

$$t \geq v_n \underset{0}{>} \frac{y - r(u_{n-1})t}{v_n - v_{n-1}} \quad , \tag{5.117}$$

and for any j such that $n > j > i$,

$$t - \sum_{k=j+1}^{n} v_k \geq v_j \underset{0}{>} \frac{y - r(u_{j-1})t - \sum_{k=j+1}^{n} (r(u_k) - r(u_{j-1}))v_k}{r(u_j) - r(u_{j-1})} \quad . \tag{5.118}$$

For condition b) (Eq. 5.77), constraints on y are straightforward. [For the constraints on condition a), (Eq. 5.76), see ✱ (Eqs. 5.83-5.88) and ✱✱ (Eqs. 5.90-5.97).] Because of the manner in which the above ranges of the v_j are constructed, the v_j automatically fall in the constrained range $0 \leq v_j \leq t - \sum_{k=j+1}^{n} v_k$ (or $0 \leq v_n \leq t$, if $j = n$). Therefore, the only restriction on y is

$$y < r(u_n)t \quad . \tag{5.119}$$

Now, as discussed at the start of this section, if $r(u_j) = r(u_{j-1})$, $n \geq j > i$, there are apparently cases in Eqs. 5.117 and 5.118 when there is division by zero. However, if we re-examine the derivation of Eqs 5.117 and condej, we find that the term $r(u_j) - r(u_{j-1})$ is used

as a divisor in Eqs. 5.103 and 5.113. Referring to Eq. 5.101, if $r(u_j) = r(u_{j-1})$, then

$$y < r(u_i)t + \sum_{j=i+1}^{n} (r(u_j) + r(u_i))v_j = r(u_i)t + \sum_{l=k}^{n} (r(u_j) - r(u_i))v_j \tag{5.120}$$

where k is the smallest $m > j$ such that $u_m > u_j$. Then we see that v_j does not affect the value of γ_u and so v_j can be any value ≥ 0. Therefore, we find that Eqs. 5.117 and 5.118 have the exceptions:

$$\begin{aligned} &t \geq v_j \geq 0 \text{ if } u_n = u_{n-1}, \text{ and} \\ &t - \sum_{k=j+1}^{n} v_k \geq v_j \geq 0 \text{ if } u_j = u_{j-1} . \end{aligned} \tag{5.121}$$

The results of this section are summarized by the following:

Lemma 5.23: (Sufficient and necessary conditions on the v_i for $\gamma_u(\hat{\mathbf{v}}) > y$)

Let $n \geq i > 0$. Then $\gamma(\hat{\mathbf{v}}) > y$ if and only if

i) $$y < r(u_n)t \; ; \tag{5.122}$$

ii) *for each j such that $i \geq j \geq 0$:*

$$0 \leq v_j < \infty, \; i \geq j > 0 \tag{5.123}$$

$$v_0 = \infty \; ;$$

iii) *if $i \neq n$, then:*

a) for $j = n$:

$$t \geq v_n \geq_0 \frac{y - r(u_{n-1})t}{r(u_n) - r(u_{n-1})} \quad \text{if } r(u_n) > r(u_{n-1}) \tag{5.124}$$

$$t \geq v_n \geq 0 \ \text{if } r(u_n) = r(u_{n-1}) ;$$

b) for each j *such that* $n > j > i$:

$$t - \sum_{k=j+1}^{n} v_k \geq v_j \geq_0 \frac{y - r(u_{j-1})t - \sum_{k=j+1}^{n} (r(u_k) - r(u_{j-1}))v_k}{r(u_j) - r(u_{j-1})} \tag{5.125}$$

$$\text{if } r(u_j) > r(u_{j-1})$$

$$t - \sum_{k=j+1}^{n} v_k \geq v_j \geq 0 \text{if } r(u_j) = r(u_{j-1}) .$$

5.3.8.1.4. The Regions $C^i_{y,j}$

Using the observations of the previous sections, we can now make further statements about the ranges of the v_j which make a vector $\mathbf{v_u}$ i-resolvable. By Theorem 5.19, $\mathbf{v_u}$ is i-resolvable if it satisfies both Lemma 5.21 and 5.23. Intersecting the conditions of Lemma 5.21 and 5.23, we arrive at the following key result:

Theorem 5.24: (Sufficient and necessary conditions for $\mathbf{v_u}$ to be i-**resolvable**)

$\mathbf{v_u}$ *is* i*-resolvable* ($n \geq i > 0$) *if and only if*

$$i) \ r(u_i) \neq r(u_{i-1}) \quad (Eq.\ 5.126) ; \tag{5.126}$$

ii) $r(u_n)t > y \geq r(u_{i-1})t$ *(Eqs. 5.88 and 5.97 and 5.122)* ; (5.127)

iii) For each j *such that* $i > j \geq 0$:

$$0 \leq v_j < \infty, \quad i > j > 0, \text{ and}$$
$$v_0 = \infty \qquad \textit{(Eqs. 5.89 and 5.123)} \; ; \tag{5.128}$$

iv) If $i = n$:

a) for $j = n$:

$$0 \leq v_n \leq \frac{y - r(u_{n-1})t}{r(u_n) - r(u_{n-1})} \quad \textit{(Eq. 5.82)} \tag{5.129}$$

b) for each j *such that* $n > j \geq 0$:

$$0 \leq v_i \leq \frac{y - r(u_{n-1}) - (r(u_n) - r(u_{n-1}))v_n}{r(u_n) - r(u_{n-1})} \tag{5.130}$$

(Eq. 5.89) ;

v) If $i \neq n$ *then*

a) for $j = n$:

$$\frac{y - r(u_{i-1})t}{r(u_n) - r(u_{i-1})} \geq v_n > \frac{y - r(u_{n-1})t}{r(u_n) - r(u_{n-1})} \quad \text{if } r(u_n) > r(u_{n-1})$$

$$\frac{y - r(u_{i-1})t}{r(u_n) - r(u_{i-1})} \geq v_n \geq 0 \quad \text{if } r(u_n) > r(u_{n-1})$$

(Eqs. 5.82 and 5.124) ;

b) for each j such that n > j > i :

$$\frac{y - r(u_{i-1})t - \sum_{k=j+1}^{n} (r(u_k) - r(u_{i-1}))v_k}{r(u_j) - r(u_{i-1})} \geq v_j$$

$$> \frac{y - r(u_{j-1})t - \sum_{k=j+1}^{n} (r(u_k) - r(u_{j-1}))v_k}{r(u_j) - r(u_{j-1})} \geq 0$$

$$\text{if } r(u_j) > r(u_{j-1}) \tag{5.131}$$

$$\frac{y - r(u_{i-1})t - \sum_{k=j+1}^{n} (r(u_k) - r(u_{i-1}))v_k}{r(u_j) - r(u_{i-1})} \geq v_j \geq 0$$

$$\text{if } r(u_j) = r(u_{j-1})$$

(Eqs. 5.89 and 5.118) ;

c) for j = i:

$$\frac{y - r(u_{i-1})t - \sum_{k=j+1}^{n} (r(u_k) - r(u_{i-1}))v_k}{r(u_i) - r(u_{i-1})} \geq v_i \geq 0 \tag{5.132}$$

(Eqs. 5.89 and 5.123) .

In Eq. 5.131, if $r(u_j) = r(u_{i-1})$, then $r(u_i) = r(u_{i-1})$, condition *i)* then fails, and so there can be no division by zero.

Now, by definition of $C^i_{y,j}$ (see Eqs. 5.54 and 5.33), the ranges of the v_j described in Theorem 5.24 are the $C^i_{y,j}$. Hence, we have the main result of this section:

Theorem 5.25: (Characterization of $C^i_{y,j}$)

i) If $r(u_i) = r(u_{i-1})$, $n \geq i > 0$, *then no* $\mathbf{v_u}$ *is i-resolvable.* $\mathbf{C}^i_y = \phi$ *and for each j such that* $n \geq k \geq 0$*:*

$$C^i_{y,j} = \phi \,. \tag{5.133}$$

ii) Let $y \in [r(u_n)t, \infty)$. *Then every* $\mathbf{v_u}$ *is* $n+1$*-resolvable.* $\mathbf{C}^{n+1}_y = \mathbf{C}_y$, $\mathbf{C}^i_y = \phi$ *for* $n \geq i \geq 0$, *and for each j such that* $n \geq j \geq 0$ *and for each k such that* $n \geq k \geq 0$*:*

$$C^{n+1}_{y,j} = [0, \infty) \tag{5.134}$$

$$C^k_{y,j} = \phi \,. \tag{5.135}$$

iii) Let $y \in [r(u_{l-1})t, r(u_l)t)$, $n \geq l > 0$. *Then for every i such that* $l \geq i > 0$ *and* $r(u_i) \neq r(u_{i-1})$, *there exist vectors* $\mathbf{v_u}$ *such that* $\mathbf{v_u}$ *is i-resolvable.*

a) If $i = n$, *then*

$$\text{i)} \quad C^n_{y,n} = \left[0, \frac{y - r(u_{n-1})t}{r(u_n) - r(u_{n-1})}\right] \tag{5.136}$$

and for each j such that $n > j \geq 0$

$$\text{ii)} \quad C^n_{y,j} = [0, \infty), \quad n > j > 0 \text{ and} \tag{5.137}$$

$$\text{iii)} \quad C^n_{y,0} = \infty \,. \tag{5.138}$$

b) If $i \neq n$, *then*

$$\text{i)}\quad C^{i}_{y,n} = \left(\frac{y - r(u_{n-1})t}{r(u_n) - r(u_{n-1})}\,,\ \frac{y - r(u_{i-1})t}{r(u_n) - r(u_{i-1})} \right] \quad \text{if } r(u_n) > r(u_{n-1})$$

$$C^{i}_{y,n} = \left[0,\ \frac{y - r(u_{i-1})t}{r(u_n) - r(u_{i-1})} \right] \quad \text{if } r(u_n) = r(u_{n-1}) \tag{5.139}$$

and for each j *such that* $n > j > i$

$$\text{ii)}\quad C^{i}_{y,j} = \left(\frac{y - r(u_{j-1})t - \sum_{k=j+1}^{n} (r(u_k) - r(u_{j-1}))v_k}{r(u_j) - r(u_{j-1})}\,, \right.$$

$$\left. \frac{y - r(u_{i-1})t - \sum_{k=j+1}^{n} (r(u_k) - r(u_{i-1}))v_k}{r(u_j) - r(u_{i-1})} \right]$$

$$\text{if } r(u_j) > r(u_{j-1}) \tag{5.140}$$

$$C^{i}_{y,j} = \left(0,\ \frac{y - r(u_{i-1})t - \sum_{k=j+1}^{n} (r(u_k) - r(u_{i-1}))v_k}{r(u_j) - r(u_{i-1})} \right]$$

$$\text{if } r(u_j) = r(u_{j-1})$$

and for $j = i$

$$\text{iii)}\quad C^{i}_{y,i} = \left[0,\ \frac{y - r(u_{i-1})t - \sum_{k=j+1}^{n} (r(u_k) - r(u_{i-1}))v_k}{r(u_i) - r(u_{i-1})} \right] \tag{5.141}$$

and for each j *such that* $i > j \geq 0$

$$\text{iv)}\quad C^{i}_{y,j} = [0, \infty),\ i > j > 0, \text{ and}$$

$$C^{i}_{y,0} = \infty\,. \tag{5.142}$$

c) For each k such that $n + 1 \geq k > l$ (or $k = 0$) and for each j such that $n \geq j \geq 0$

$$C^k_{y,j} = \phi \,. \tag{5.143}$$

where the notation $\underset{\geq 0}{(\ a, b]}$ means

$$\underset{\geq 0}{(\ a, b]} = \begin{cases} [0, b] & \text{if } a \leq 0 \\ (a, b] & \text{if } a > 0 \end{cases} \tag{5.144}$$

In Eq. 5.140, if $r(u_j) - r(u_{i-1})$ for $n > j > i$, then $r(u_i) = r(u_{i-1})$ and Eq. 5.140 is replaced by ϕ [see case i)].

An integral solution for the PDF F_Y (and hence the system's performability) is thus obtained by employing Eqs. 5.25, 5.27, 5.35, and 5.134-5.143.

5.3.8.2. Simplified Notation and Results

As discussed in Section 5.3.2 [Notation] the notation used in Sections 5.3.3 [The Approach]-5.3.8.1.4 [The Regions $C^i_{y,j}$] makes explicit qualifications for the boundary cases, e.g., for situations dealing with n. Such fastidious detail is necessary for observing the exceptions occurring at the boundaries. However, such meticulousness is not necessary in order to present the results; the notation and results presented in this section are aimed at removing the special boundary cases and are arguably more elegant than those of the previous sections.

We begin by defining countably many "states" "above" state n: For all $k > n$, let u_k have reward rate 0, i.e., $r(u_k) = 0$, and let $w^k_t = v_k = 0$. $r(u_k)$, w^k_t, and v_k will allow a kind of non-contributory "overflow" above state n. In addition, define "state" $u_{(-1)}$ "below" u_0, let $u_{(-1)}$ have reward rate 0 (i.e., $r(u_{(-1)}) = 0$), and let $w^{(-1)}_t = v_{(-1)} = 0$. *We emphasize that these definitions are notational contrivances only. There is no physical interpretation of these additional states.* This extended concept of the set of "states" will allow us to write unified expressions with no boundary conditions.

Also define the operator

$$a \dot{-} b = \begin{cases} a - b & \text{if } a \geq b \\ 0 & \text{otherwise} . \end{cases} \tag{5.145}$$

We adopt the convention that, if $b \geq a$,

$$\begin{aligned} \sum_{k=a}^{b} X_k &= X_a + X_{a+1} + \cdots + X_b \\ &= X_b + X_{b-1} + \cdots + X_a \\ &= \sum_{k=b}^{a} X_k . \end{aligned} \tag{5.146}$$

Let $i \geq 0$. Then for a given $\mathbf{v}_u$:

$$w_t^i = \max(\min(t - \sum_{j=i+1}^{n} v_j, v_i), 0) \quad \text{(See Eq. 5.28)} \tag{5.147}$$

$$Y = \gamma(\mathbf{v}_u) = r(u_j)t + \sum_{k=j+1}^{n} (r(u_k) \dot{-} r(u_j))v_k \quad \text{when} \sum_{k=j+1}^{n} v_k < t, \sum_{k=j}^{n} v_k \geq t \tag{5.148}$$

(See Eq. 5.30) .

Next we adopt the convention that $\pm\frac{1}{0} = \pm\infty$. The notation $[a, b\ ]_{\substack{\geq 0 \\ <\infty}}$ means

$$\{a, b\ ]_{\substack{\geq 0 \\ <\infty}} = \begin{cases} [a, b] & \text{if } 0 \leq b < \infty \\ [a, \infty) & \text{if } b = \infty \\ [a, 0) & \text{if } b < 0 \end{cases} \tag{5.149}$$

where the left bracket { denotes any interval demarking symbol, e.g., {, (, or $(_{\geq 0}$. The notation $(_{\geq 0}$ is extended as follows:

$$\begin{matrix}(\ f(i,j),\ b] \\ \scriptstyle \geq 0 \\ \scriptstyle i \leq j \\ \scriptstyle \neq \infty\end{matrix} = \begin{cases} (\ f(i,j),\ b] & \text{if } i \leq j \\ \scriptstyle \geq 0 & \\ [0,\ b) & \text{if } i > j \text{ or } f(i,j) = \infty\ . \end{cases} \tag{5.150}$$

If $b \leq a$, then the region (a, b) is empty, i.e., $(a, b) = \phi$.

Theorem 5.25 can now be written:

Theorem 5.26: (Compact characterization of $C^i_{y,j}$)

i) If $r(u_i) = r(u_{i-1})$, $n \geq i > 0$, *then no* $\mathbf{v}_u$ *is i-resolvable.* $\mathbf{C}^i_y = \phi$ *and for each j such that* $n \geq k \geq 0$*:*

$$C^i_{y,j} = \phi\ . \tag{5.151}$$

ii) Let $y \in [r(u_n)t, \infty)$. *Then every* $\mathbf{v}_u$ *is* $n+1$*-resolvable.* $\mathbf{C}^{n+1}_y = \mathbf{C}_y$, $\mathbf{C}^i_y = \phi$ *for* $n \geq i \geq 0$, *and for each j such that* $n \geq j \geq 0$ *and for each k such that* $n \geq k \geq 0$*:*

$$C^{n+1}_{y,j} = [0, \infty) \tag{5.152}$$

$$C^k_{y,j} = \phi\ . \tag{5.153}$$

iii) Let $y \in [r(u_{l-1})t, r(u_l)t)$, $n > l \geq 0$. *Then for every i such that* $n+1 \geq i > 0$ *and* $r(u_i) \neq r(u_{i-1})$ *and for each j such that* $n \geq j > 0$*:*

$$C'_{y,j} = \left(\frac{y - r(u_{j-1})t - \sum_{k=j+1}^{n} (r(u_k) - r(u_{j-1}))v_k}{r(u_j) - r(u_{j-1})} , \right.$$

$$\left. \frac{y - r(u_{i-1})t - \sum_{k=j+1}^{n} (r(u_k) \dot{-} r(u_{i-1}))v_k}{r(u_j) \dot{-} r(u_{i-1})} \right]_{\substack{i \geq 0 \\ i \leq j \\ \neq \infty}} \tag{5.154}$$

$$C'_{y,0} = \infty \ .$$

Note that Eq. 5.154 of Theorem 5.26 captures all the information of Eqs. 5.136 - 5.143 of Theorem 5.25.

The derivation of the $C'_{y,j}$ presented in Section 5.3.5 [i-Resolvability] can be restated using the more compact notation above.

We can simplify the integrations of Eq. 5.35 by observing:

i) in certain cases $C'_{y,j} = \phi$ (Eqs. 5.143 and 5.153) and the associated integrations of Eq. 5.35 will be 0, and

ii) in certain cases, $C'_{y,j} = [0, \infty)$ (Eqs. 5.137 and 5.142) and the associated integrations of Eq. 5.35 will be 1 since the integration is over a probability density.

Hence, we have:

i) Let $y \in [r(u_n)t, \infty)$. Then

$$F_{Y|U}(y|\mathbf{u}) = \sum_{i=0}^{n+1} \int_{C'_{y,n}} \int_{C'_{y,n-1}} \cdots \int_{C'_{y,0}} f_{\mathbf{V_u}|\mathbf{U}}(\mathbf{v_u} \mid \mathbf{U}) \, d\mathbf{v_u} \tag{5.155}$$

$$= \int_0^\infty \int_0^\infty \cdots \int_0^\infty f_{\mathbf{V}_u}(\mathbf{v}_u)\, d\mathbf{v}_u$$

$$+ \sum_{i=0}^{n} \int_\phi \int_\phi \cdots \int_\phi f_{\mathbf{V}_u}(\mathbf{v}_u)\, d\mathbf{v}_u \tag{5.156}$$

$$= 1 \ .$$

ii) Let $y \in [r(u_{l-1})t, r(u_l)t)$, $n \geq l > 0$. Then

$$F_{Y|\mathbf{U}}(y|\mathbf{u}) = \sum_{i=0}^{n+1} \int_{C^i_{y,n}} \int_{C^i_{y,n-1}} \cdots \int_{C^i_{y,0}} f_{\mathbf{V}_u \,|\, \mathbf{U}}(\mathbf{v}_u \,|\, \mathbf{U})\, d\mathbf{v}_u \tag{5.157}$$

$$= \int_\phi \int_\phi \cdots \int_\phi f_{\mathbf{V}_u}(\mathbf{v}_u)\, d\mathbf{v}_u$$

$$+ \int_{C^1_{y,n}} \int_{C^1_{y,n-1}} \cdots \int_{C^1_{y,1}} \int_0^\infty f_{\mathbf{V}_u}(\mathbf{v}_u)\, d\mathbf{v}_u$$

$$+ \int_{C^2_{y,n}} \int_{C^2_{y,n-1}} \cdots \int_{C^2_{y,2}} \int_0^\infty \int_0^\infty f_{\mathbf{V}_u}(\mathbf{v}_u)\, d\mathbf{v}_u \tag{5.158}$$

$$+ \cdots + \int_{C^l_{y,n}} \int_{C^l_{y,n-1}} \cdots \int_{C^l_{y,l}} \int_0^\infty \cdots \int_0^\infty f_{\mathbf{V}_u}(\mathbf{v}_u)\, d\mathbf{v}_u$$

$$+ \sum_{i=l+1}^{n} \int_\phi \int_\phi \cdots \int_\phi f_{\mathbf{V}_u}(\mathbf{v}_u)\, d\mathbf{v}_u$$

(the last term applicable only if $n > l$)

$$= \begin{cases} \int_{C^n_{y,n}} f_n(v_n)dv_n + \sum_{i=1}^{n-1} \int_{C^i_{y,n}} \int_{C^i_{y,n-1}} \cdots \int_{C^i_{y,1}} f_n(v_n) \\ \quad \left(\prod_{j=n-1}^{1} f_{j|n,n-1,\ \ ,j+1}(v_j|v_n, v_{n-1}, \ldots, v_{j+1})\ dv_j \right) dv_n \\ \quad \text{if } n = l \\ \\ \sum_{i=1}^{l} \int_{C^i_{y,n}} \int_{C^i_{y,n-1}} \cdots \int_{C^i_{y,1}} f_n(v_n) \\ \quad \left(\prod_{j=n-1}^{1} f_{j|n,n-1,\ \ ,j+1}(v_j|v_n, v_{n-1}, \ldots, v_{j+1})\ dv_j \right) dv_n \\ \quad \text{if } n > l \end{cases} \tag{5.159}$$

We next present a compact version of Eq. 5.159. Adopt the convention that the product of any increasing series is 1, i.e., if $a < b$ then for any sequence $\{X_i\}$

$$\prod_{j=a}^{b} X_j = 1 . \tag{5.160}$$

If $y \in [r(u_{l-1})t, r(u_l)t)$ for some $n > l > 0$, then

$$F_{Y|U}(y|\mathbf{u}) = \sum_{i=0}^{l} \int_{C^i_{y,n}} \int_{C^i_{y,n-1}} \cdots \int_{C^i_{y,1}} f_n(v_n) \left(\prod_{j=n-1}^{1} f_{j|n,n-1,\ \ ,j+1}(v_j|v_n, v_{n-1}, \ldots, v_{j+1})\ dv_j \right) dv_n . \tag{5.161}$$

5.3.9. Closed-Form Solutions

In this section are discussed some examples of the performability solution of Eq. 5.35 and regions of Theorem 5.25. Numerical applications to engineering problems of the solution will be discussed in Section 5.3.11 [Numerical Solutions]. Some of these results will be

rederived in Section 5.3.9 [Closed-Form Solutions] using a recursive formulation of $F_{Y|U}$. To restate the problem, we wish to determine the distribution of reward for a finite-state acyclic nonrecoverable process. (See Section 5.2.4.1 [Definition of a Nonrecoverable Process] for the definition of a nonrecoverable process.) We consider two classes of processes: time-invariant Markovian and non-Markovian. For the Markovian process, the distribution of the sojurn time in state i is exponential with rate λ_i and is independent of the sojurn times of the process in any other state j and of the present time. No restrictions are placed on sojurn time distributions for the non-Markovian case.

5.3.9.1. Two State Markovian Acyclic Process

As the simplest non-trivial example, suppose $n = 1$. See Figure 5.3. The regions $C^1_{y,1}$, $C^1_{y,0}$, $C^0_{y,1}$, and $C^0_{y,0}$ are as follows; the applicable case of Theorem 5.25 is indicated in brackets.

For $y \in [\, 0, r(u_1)t\,)$, (i.e., $l = 1$)

$i = n$ (i.e., checking for n-resolvability):

$$j = n = 1: \quad C^1_{y,1} = \left[0, \frac{y - r(u_0)t}{r(u_1) - r(u_0)} \right) \qquad [\text{case } a\text{-}i)] \tag{5.162}$$

$$j = i = 0: \quad C^1_{y,0} = \infty \qquad [\text{case } a\text{-}ii)] \tag{5.163}$$

$$F_Y(y) = \int_{C^1_{y,1}} f_1(v_1)dv_1 = \int_0^{\frac{y - r(u_0)t}{r(u_1) - r(u_0)}} \lambda_1 e^{-\lambda_1 v_1} dv_1 = 1 - e^{-\lambda_1 \frac{y - r(u_0)t}{r(u_1) - r(u_0)}} . \tag{5.164}$$

For $y \in [r(u_1)t, \infty)$:

$$F_Y(y) = 1 \tag{5.165}$$

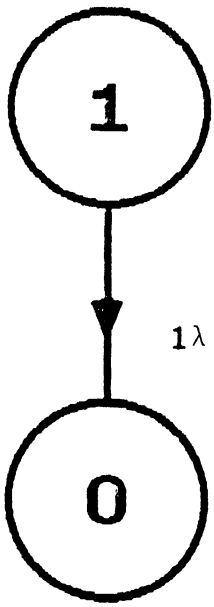


Fig. 5.3 Markov state-transition-rate diagram for the case n=1

The performability distribution is then

$$F_Y(y) = \begin{cases} 1 - e^{-\lambda_1 \frac{y - r(u_0)t}{r(u_1) - r(u_0)}} & y \in [0,\ r(u_1)t\ ) \\ 1 & y \in [r(u_1)t, \infty)\ . \end{cases} \tag{5.166}$$

which is the anticipated result. If $r(u_0) = 0$, then

$$F_Y(y) = \begin{cases} 1 - e^{-\frac{\lambda_1 y}{r(u_1)}} & y \in [0,\ r(u_1)t\ ) \\ 1 & y \in [r(u_1)t, \infty)\ . \end{cases} \tag{5.167}$$

Note that t does not appear in the expression for $F_Y(y)$, other than to delimit the "breakpoint."

5.3.9.2. Three State Markovian Acyclic Process

Consider now $n = 2$. See Fig. 5.4. This was the case considered in [33]. The derivation is relatively long and is presented in Appendix K. The results are:

$$F_Y(y) = \begin{cases} \int\limits_{C^1_{y,2}} \int\limits_{C^1_{y,1}} f_2(v_2) f_1(v_1) dv_1 dv_2 & \text{if } y \in [0, r(u_1)t) \\ \int\limits_{C^2_{y,2}} f_1(v_1) dv_1 + \int\limits_{C^1_{y,2}} \int\limits_{C^1_{y,1}} f_2(v_2) f_1(v_1) dv_1 dv_2 & \\ \qquad \text{if } y \in [r(u_1)t, r(u_2)t) & \\ 1 \qquad \text{if } y \in [r(u_2)t, \infty) & \end{cases} \tag{5.168}$$

If $y \in [0, r(u_1)t)$:

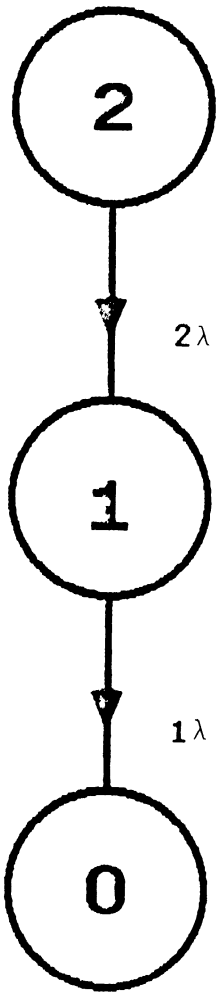


Fig. 5.4 Markov state-transition-rate diagram for the case n=2

$$F_Y(y) = 1 - e^{-\frac{\lambda_2(y - r(u_0)t)}{r(u_2) - r(u_0)}} - \frac{(r(u_1) - r(u_0))\lambda_2}{\lambda_1(r(u_2) - r(u_0)) + \lambda_2(r(u_1) - r(u_0))} e^{-\frac{\lambda_1(y - r(u_0)t)}{r(u_1) - r(u_0)}} \left[1 - e^{-\frac{\left[\lambda_1(r(u_2) - r(u_0)) + \lambda_2(r(u_1) - r(u_0))\right](y - r(u_0)t)}{r(u_1) - r(u_0)}} \right] \tag{5.169}$$

If $y \in [r(u_1), r(u_2)t)$:

$$F_Y(y) = 1 - e^{-\frac{\lambda_2(y - r(u_0)t)}{r(u_2) - r(u_0)}} + \frac{(r(u_1) - r(u_0))\lambda_2}{\lambda_2(r(u_1) - r(u_0)) - \lambda_1(r(u_2) - r(u_0))} e^{-\frac{\lambda_1(y - r(u_0)t)}{r(u_2) - r(u_0)}} \left[e^{-\frac{(y - r(u_0)t)\left[\lambda_2(r(u_1) - r(u_0)) - \lambda_1(r(u_2) - r(u_0))\right]}{(r(u_1) - r(u_0))(r(u_2) - r(u_0))}} - e^{-\frac{(y - r(u_1)t)\left[\lambda_2(r(u_1) - r(u_0)) - \lambda_1(r(u_2) - r(u_0))\right]}{(r(u_1) - r(u_0))(r(u_2) - r(u_0))}} \right] \tag{5.170}$$

If $y \in [r(u_1), r(u_2)t)$:

$$F_Y(y) = 1 \tag{5.171}$$

5.3.9.3. Four State Markovian Acyclic Process

Consider now $n = 3$. See Fig. 5.5. The solution is presented in Appendix L.

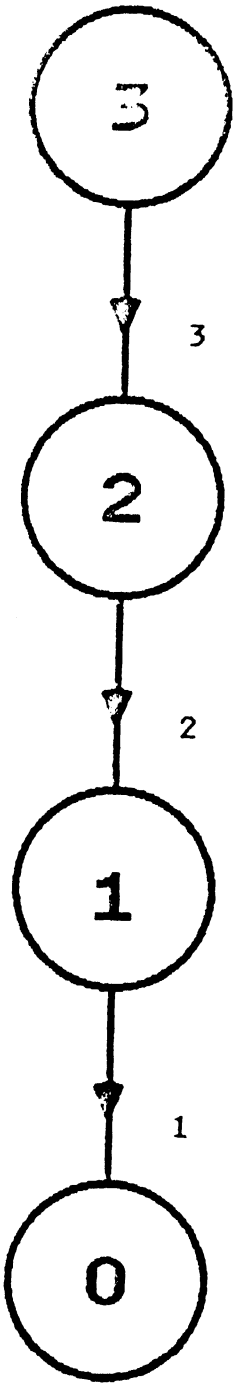


Fig. 5.5 -- Markov state-transition-rate diagram
for the case n=3

5.3.10. Recursive Formulation of $F_{Y|U}$

As an example to motivate this section, consider the following scenario. Suppose we have modeled a multiprocessor computing system having n processors as an $n+1$ state stochastic process, where state i denotes i operational processors. Suppose too that we wish to compute the performability of this system for the cases $n = 0, 1, \ldots, n_{\max}$. That is, we desire a family of performability distributions, $p_{S_0}, p_{S_1}, \ldots, p_{S_{n_{\max}}}$ where S_i is the i processor system. For any system S_i that contains state u_j, the reward rate $r(u_j)$ as well as the distribution of the sojourn time in u_j are the same. Rather than calculating each p_{S_i} independently from the ground up, we may find it possible to use information about $p_{S_0}, p_{S_1}, \ldots, p_{S_{i-1}}$ to help determine p_{S_i}. This section examines that possibility.

In more general terms, assume that we have solved $F_{Y|U}$ for a series of increasingly complex models, where the first model S_0 has only the single state sequence $\mathbf{u}_0 = (u_0)$, the second model S_1 has only the single sequence $\mathbf{u}_1 = (u_1, u_0)$, the third model S_2 has only $\mathbf{u}_2 = (u_2, u_1, u_0)$, and so forth, to S_{n-1} having only $\mathbf{u}_{n-1} = (u_{n-1}, u_{n-1}, \ldots, u_0)$. Assume, too, that these solutions are "parametric" (closed-form) in terms of the reward rates $r(u_i)$ and the sojourn time densities in states u_i.

Using the above information, we now wish to solve the next model in the series, i.e., S_n, the model that only contains the sequence $\mathbf{u}_n = (u_n, u_{n-1}, \ldots, u_0)$. Below we present a formulation of $F_{Y|U}(\bullet \mid \mathbf{u}_n)$ based on the knowledge of $F_{Y|U}(\bullet \mid \mathbf{u}_{n-1})$.

We need one reasonable assumption to insure some similarity between systems. Given two of the systems, the sojourn time distributions of "corresponding" states must be "parametrically equivalent" in the sense to be defined below.

Let each state u_i have a ***parameter*** λ_i which can be reflected in the sojourn time distribution of that state. For example, if the sojourn time in state u_i is exponential and independent of the process's history, then the density of the sojourn time in state i is $f_i(v) = e^{-\lambda_i v}$. The parameter λ_i is allowed to vary from system to system, and state u_i of system S_j will

have parameter λ_i^j. For system S_j, let $f^{\lambda^j}_{i\,|\,j,j-1,\ \ldots\ ,i+1}$ be the density of the sojourn time in state i, conditioned on the history of the process, and a function of parameter λ_i^j.

To recursively build $F_{Y|U}$ of S_n from the $F_{Y|U}$ of S_{n-1}, we require some notation of state correspondence between models. In particular, we need to know which state of system S_k corresponds to state u_i of system S_j. Since the history of the process can affect sojourn time distributions, we make the number of state transitions the basis of correspondence. Thus, for S_j and S_k, the pairings are u_j and u_k; u_{j-1} and u_{k-1}; $\cdots$; u_{j-i} and u_{k-i}. To understand intuitively the reasoning behind this definition, note that the densities $f^{\lambda^j}_{j-i\,|\,j,j-1,\ \ldots\ ,j-i+1}$ and $f^{\lambda^k}_{k-i\,|\,k,k-1,\ \ldots\ ,k-i+1}$ are similar in the sense that both are functions of the same number of variables.

Consider the systems S_j and S_k. Let $0 \leq i \leq j < k$. State u_{j-i} of S_j is ***parametrically equivalent*** to state u_{k-i} of S_k if, for any λ,

$$f^{\lambda}_{j-i\,|\,j,j-1,\ \ldots\ ,j-i+1} = f^{\lambda}_{k-i\,|\,k,k-1,\ \ldots\ ,k-i+1} \tag{5.172}$$

(and if $i = 0$,

$$f^{\lambda}_{j} = f^{\lambda}_{k}\ ). \tag{5.173}$$

In other words, $f^{\lambda}_{j-i\,|\,j,j-1,\ \ldots\ ,j-i+1}$ and $f^{\lambda}_{k-i\,|\,k,k-1,\ \ldots\ ,k-i+1}$ are identical functions of the parameter λ.

Let $j < k$. We shall say that system S_j is parametrically equivalent to system S_k if, for every i such that $0 \leq i \leq j$, state u_{j-i} of S_j is parametrically equivalent to state u_{k-i} of S_k.

For example, suppose the sojourn time density in state u_{j-i} of S_j is $e^{\lambda^j_{j-i}\,t}$ and the density in state u_{k-i} of S_k is $e^{\lambda^j_{k-i}\,t}$. Then clearly u_{j-i} and u_{k-i} are parametrically equivalent.

Let $\boldsymbol{\lambda}^k = (\lambda_k, \lambda_{k-1}, \ldots, \lambda_0)$ and let $0 \leq i < k$. Then define $F^{k;\boldsymbol{\lambda}^k}_{Y|U}(\bullet\,|\,\mathbf{u}_k)$ to be the distribution of Y for system S_k given the sequence $\mathbf{u}_k$ and the vector of parameters $\boldsymbol{\lambda}^k$. The

vector $\boldsymbol{\lambda}^k$ denotes the parameters of the sojourn time densities of states $i, i-1, \ldots, 0$ as follows: $f_{i}^{\lambda_k}$, $f_{i-1|i}^{\lambda_{k-1}}$, $\ldots$, $f_{1|i,i-1,\ldots,2}^{\lambda_{k-i+1}}$, $f_{0|i,i-1,\ldots,1}^{\lambda_{k-i}}$. Note that if $k \neq i$, then the vector $\boldsymbol{\lambda}^k$ will contain elements $\lambda_{k-i-1}, \lambda_{k-i-2}, \ldots, \lambda_0$ which do not affect the distribution.

In much the same manner as we generalized the notation for $F_{Y|U}$, we now generalize the representation of the set $C_{y,j}^{i}$ (see Eq. 5.154) to reflect explicitly the sequence of reward rates $(r(u_m), r(u_{m-1}), \ldots, r(u_0))$. Let $m \geq k \geq i > 0$, $i \geq j \geq 0$. and $r(\mathbf{u}_m) = (r(u_m), r(u_{m-1}), \ldots, r(u_0))$. Then let $C_{y,j}^{i,k;r(\mathbf{u}_m)}$ be the set $C_{y,j}^{i}$ for the system S_k whose reward rates are the initial subsequence of $r(\mathbf{u}_m)$, i.e., the rates are $(r(u_m), r(u_{m-1}), \ldots, r(u_{m-k}))$. The rates $r(u_{m-k-1})$, $r(u_{m-k-2})$, $\ldots$, $r(u_0)$ do not affect $C_{y,j}^{i,k;r(\mathbf{u}_m)}$. Eq. 5.154 then translates to

$$C_{y,j}^{i,k;r(\mathbf{u}_m)} = \left(\frac{y - r(u_{m-k+j-1})t - \sum_{l=m-k+j+1}^{m} (r(u_l) - r(u_{m-k+j-1}))v_l}{r(u_{m-k+j}) - r(u_{m-k+j-1})}, \right. \tag{5.174}$$

$$\left. \frac{y - r(u_{m-k+i-1})t - \sum_{l=m-k+j+1}^{m} (r(u_l) \dot{-} r(u_{m-k+i-1}))v_l}{r(u_{m-k+j}) \dot{-} r(u_{m-k+i-1})} \right]_{\geq 0}^{\wedge} .$$

We make the following key proposition:

Lemma 5.27: (Relationship between $C_{y,k-j}^{k-i,k;r(\mathbf{u}_m)}$ and $C_{y,k-j}^{k'-i,k';r(\mathbf{u}_m)}$)

Let $m \geq k \geq i > 0$, $m \geq k' \geq i > 0$, and $i \geq j \geq 0$. Then

$$C_{y,k-j}^{k-i,k;r(\mathbf{u}_m)} = C_{y,k'-j}^{k'-i,k';r(\mathbf{u}_m)} . \tag{5.175}$$

Proof:

$$C_{y,k-j}^{k-i,k;r(u_m)} = \left(\frac{y - r(u_{m-k+k-j-1})t - \sum_{l=m-k+k-j+1}^{m} (r(u_l) - r(u_{m-k+k-j-1}))v_l}{r(u_{m-k+k-j}) - r(u_{m-k+k-j-1})} , \right.$$

$$\left. \frac{y - r(u_{m-k+k-i-1})t - \sum_{l=m-k+k-j+1}^{m} (r(u_l) \dot{-} r(u_{m-k+k-i-1}))v_l}{r(u_{m-k+k-j}) \dot{-} r(u_{m-k+k-i-1})} \right]_{\geq 0}^{\infty} \qquad (5.176)$$

$$= \left(\frac{y - r(u_{m-k'+k'-j-1})t - \sum_{l=m-k'+k'-j+1}^{m} (r(u_l) - r(u_{m-k'+k'-j-1}))v_l}{r(u_{m-k'+k'-j}) - r(u_{m-k'+k'-j-1})} , \right.$$

$$\left. \frac{y - r(u_{m-k'+k'-i-1})t - \sum_{l=m-k'+k'-j+1}^{m} (r(u_l) \dot{-} r(u_{m-k'+k'-i-1}))v_l}{r(u_{m-k'+k'-j}) \dot{-} r(u_{m-k'+k'-i-1})} \right]_{\geq 0}^{\infty}$$

$$= C_{y,k'-j}^{k'-i,k';r(u_m)} . \qquad \| \|$$

Finally, we can write the recursive solution:

Theorem 5.28: (Recursive representation of $F_{Y|U}^{n;\lambda^n}$)

Let S_n and S_{n-1} be parametrically equivalent. Let $y \in [u_{l-1}t, u_l t)$, $n \geq l > 0$. Then

$$F_{Y|U}^{n;\lambda^n}(y \mid \mathbf{u}_n) = \int_{C_{y,n}^{1,n;r(u_n)}} \int_{C_{y,n-1}^{1,n;r(u_n)}} \cdots \int_{C_{y,1}^{1,n;r(u_n)}} f_n^{\lambda^n}(v_n)$$

$$\left(\prod_{j=n-1}^{1} f_{j|n,n-1,\ldots,j+1}^{\lambda^n}(v_j \mid v_n, v_{n-1}, \ldots, v_{j+1})\, dv_j \right) dv_n \qquad (5.177)$$

$$+ F_{Y|U}^{n-1;\lambda^n}(y \mid \mathbf{u}_n)$$

(or, if $l = 1$

$$F_{Y|U}^{n;\lambda^n}(y \mid \mathbf{u}_n) = \int_{C_{y,n}^{1,n;r(u_n)}} \int_{C_{y,n-1}^{1,n;r(u_n)}} \cdots \int_{C_{y,1}^{1,n;r(u_n)}} f_n^{\lambda^n}(v_n)$$

$$\left(\prod_{j=n-1}^{1} f_{j|n,n-1,\ldots,j+1}^{\lambda^n}(v_j \mid v_n, v_{n-1}, \ldots, v_{j+1})\, dv_j \right) dv_n \ ). \qquad (5.178)$$

Proof: To reduce the "noise," we let $*$ denote the first term of Eq. 5.177 and all of Eq. 5.178, i.e.,

$$* = \int_{C_{y,n}^{1,n;r(u_n)}} \int_{C_{y,n-1}^{1,n;r(u_n)}} \cdots \int_{C_{y,1}^{1,n;r(u_n)}} f_n^{\lambda^n}(v_n) \left(\prod_{j=n-1}^{1} f_{j|n,n-1,\ldots,j+1}^{\lambda^n}(v_j | v_n, v_{n-1}, \ldots, v_{j+1})\, dv_j \right) dv_n \ . \tag{5.179}$$

If $l = 1$, then Eq. 5.178 follows directly from Eq. 5.35. Assume then that $1 < n \leq l$. We prove Eq. 5.177 by starting with Eq. 5.161 and Eq. 5.174, converting the regions $C_{y,j}^{i,n;r(u_n)}$ to $C_{y,n-j}^{n-i,n;r(u_n)}$, using Lemma 5.27 to translate $C_{y,n-j}^{n-i,n;r(u_n)}$ to $C_{y,n-1-j}^{n-1-i,n-1;r(u_n)}$, and then converting the regions $C_{y,n-1-j}^{n-1-i,n-1;r(u_n)}$ to $C_{y,j}^{i,n-1;r(u_n)}$. Proper bookkeeping of the i's and j's are necessary, and translations of the densities $f_{j|n,n-1,\ldots,j+1}$ are also required. The steps are:

$$F_{Y|U}^{n;\lambda^n}(y \,|\, u_n) = \sum_{i=1}^{l} \int_{C_{y,n}^{i,n;r(u_n)}} \int_{C_{y,n-1}^{i,n;r(u_n)}} \cdots \int_{C_{y,1}^{i,n;r(u_n)}} f_n^{\lambda^n}(v_n) \left(\prod_{j=n-1}^{1} f_{j|n,n-1,\ldots,j+1}^{\lambda^n}(v_j | v_n, v_{n-1}, \ldots, v_{j+1})\, dv_j \right) dv_n \tag{5.180}$$

$$= * + \sum_{i=2}^{l} \int_{C_{y,n}^{i,n;r(u_n)}} \int_{C_{y,n-1}^{i,n;r(u_n)}} \cdots \int_{C_{y,1}^{i,n;r(u_n)}} f_n^{\lambda^n}(v_n) \left(\prod_{j=n-1}^{1} f_{j|n,n-1,\ldots,j+1}^{\lambda^n}(v_j | v_n, v_{n-1}, \ldots, v_{j+1})\, dv_j \right) dv_n \tag{5.181}$$

(and "inverting" the index i:)

$$= * + \sum_{i=n-2}^{n-l} \int_{C_{y,n}^{n-i,n;r(u_n)}} \int_{C_{y,n-1}^{n-i,n;r(u_n)}} \cdots \int_{C_{y,n-1}^{n-i,n;r(u_n)}} f_n^{\lambda^n}(v_n) \left(\prod_{j=n-1}^{n-i} f_{j|n,n-1,\ldots,j+1}^{\lambda^n}(v_j | v_n, v_{n-1}, \ldots, v_{j+1})\, dv_j \right) dv_n \tag{5.182}$$

(using Lemma 5.27:)

$$= * + \sum_{i=n-2}^{n-l} \int_{C_{y,n-1}^{n-i-1,n-1;r(u_n)}} \int_{C_{y,n-2}^{n-i-1,n-1;r(u_n)}} \cdots \int_{C_{y,n-i-1}^{n-i-1,n-1;r(u_n)}} f_n^{\lambda^n}(v_n)$$

$$\left(\prod_{j=n-1}^{n-i} f_{j|n,n-1,\ \ ,j+1}^{\lambda^n}(v_j | v_n, v_{n-1}, \ldots, v_{j+1})\ dv_j \right) dv_n \tag{5.183}$$

(and since S_n is parametrically equivalent to S_{n-1}:)

$$= * + \sum_{i=n-2}^{n-l} \int_{C_{y,n-1}^{n-i-1,n-1;r(u_n)}} \int_{C_{y,n-2}^{n-i-1,n-1;r(u_n)}} \cdots \int_{C_{y,n-i-1}^{n-i-1,n-1;r(u_n)}} f_{n-1}^{\lambda^n}(v_n)$$

$$\left(\prod_{j=n-2}^{n-i-1} f_{j|n-1,n-2,\ \ ,j+1}^{\lambda^n}(v_j | v_{n-1}, v_{n-2}, \ldots, v_{j+1})\ dv_j \right) dv_{n-1} \tag{5.184}$$

(shifting the index i by 1:)

$$= * + \sum_{i=n-1}^{n-l+1} \int_{C_{y,n-1}^{n-i,n-1;r(u_n)}} \int_{C_{y,n-2}^{n-i,n-1;r(u_n)}} \cdots \int_{C_{y,n-i}^{n-i,n-1;r(u_n)}} f_{n-1}^{\lambda^n}(v_n)$$

$$\left(\prod_{j=n-2}^{n-i} f_{j|n-1,n-2,\ \ ,j+1}^{\lambda^n}(v_j | v_{n-1}, v_{n-2}, \ldots, v_{j+1})\ dv_j \right) dv_{n-1} \tag{5.185}$$

(and "reinverting" the index i:)

$$= * + \sum_{i=1}^{l-1} \int_{C_{y,n-1}^{i,n-1;r(u_n)}} \int_{C_{y,n-2}^{i,n-1;r(u_n)}} \cdots \int_{C_{y,n-i}^{i,n-1;r(u_n)}} f_{n-1}^{\lambda^n}(v_n)$$

$$\left(\prod_{j=n-2}^{i} f_{j|n-1,n-2,\ \ ,j+1}^{\lambda^n}(v_j | v_{n-1}, v_{n-2}, \ldots, v_{j+1})\ dv_j \right) dv_{n-1} \tag{5.186}$$

$$= * + F_{Y|U}^{n-1;\lambda^n}(y | u_n)\ .$$

| |

Some examples of this substitution are presented in Section Appendices M and N.

5.3.10.1. Examples of Recursively Derived Solutions

In this section are discussed some examples of the recursive performability solution of Theorem 5.28. To restate the problem, we wish to determine the distribution of reward for a finite-state nonrecoverable process. (See Section 5.2.4.1 [Definition of a Nonrecoverable Process] for the definition.) As in Section cpvexample, the distribution of the sojourn time in state i is exponential with rate λ_i and is independent of the sojourn times of the process in any other state j and of the present time. We also assume that $\lambda_i^j = \lambda_i^k$ for all j and k. Thus, for $i \geq 0, j > i$:

$$f^{\lambda^j}_{\cdot i \mid j, j-1, \ldots, i+1}(v_i) = e^{-\lambda_i v_i} \tag{5.187}$$

5.3.10.1.1. Two State Markovian Acyclic Process

Consider the simple two-state Markovian acyclic process of Section 5.3.9.1 [Two State Markovian Acyclic Process]. (See Fig. 5.3.) Call the system S_1; so $k = 1$, $\boldsymbol{\lambda}^1 = (\lambda_1, \lambda_0)$, and the only state sequence is $\mathbf{u} = (1, 0)$.

For $y \in [\, 0, r(u_1)t\,)$, (i.e., $l = 1$):

$$j = n = 1: \quad C_{y,1}^{1,1;r(u_1)} = \left[0, \frac{y - r(u_0)t}{r(u_1) - r(u_0)} \right) \qquad [\text{case } a\text{-}i)] \tag{5.188}$$

$$j = i = 0: \quad C_{y,0}^{1,1;r(u_1)} = \infty \qquad [\text{case } a\text{-}ii)] \tag{5.189}$$

$$F^{1;\boldsymbol{\lambda}^1}_{Y\ \mathbf{U}}(y \mid (u_1, u_0) = \int_{C_{y,1}^{1,1;r(u_1)}} f_1^{\boldsymbol{\lambda}^1}(v_1) dv_1 \tag{5.190}$$

$$= \int_0^{\frac{y - r(u_0)t}{r(u_1) - r(u_0)}} \lambda_1 e^{-\lambda_1 v_1} dv_1 = 1 - e^{-\lambda_1 \frac{y - r(u_0)t}{r(u_1) - r(u_0)}} . \tag{5.191}$$

For $y \in [r(u_1)t, \infty)$:

$$F_Y(y) = 1 \tag{5.192}$$

The performability distribution is then

$$F_Y(y) = \begin{cases} 1 - e^{-\lambda_1 \frac{y - r(u_0)t}{r(u_1) - r(u_0)}} & y \in [0,\ r(u_1)t\ ) \\ 1 & y \in [r(u_1)t, \infty) \ . \end{cases} \tag{5.193}$$

which is the same result of Section 5.3.9.1 [Two State Markovian Acyclic Process].

5.3.10.1.2. Three State Markovian Acyclic Process

Consider now $n = 2$. See Fig. 5.4. Call the system S_2; so $k = 2$, $\boldsymbol{\lambda}^2 = (\lambda_2, \lambda_1, \lambda_0)$, and the only state sequence is $\mathbf{u} = (2, 1, 0)$.

The derivation is presented in Appendix L . Of course, the results are the same as those in Appendix J, but the derivation is significantly shorter.

5.3.10.1.3. Four State Markovian Acyclic Process

Next, consider $n = 3$. See Fig. 5.5. The system is S_2, $k = 2$, $\boldsymbol{\lambda}^2 = (\lambda_2, \lambda_1, \lambda_0)$, and the only state sequence is $\mathbf{u} = (2, 1, 0)$. The derivation is presented in Appendix M The results are the same as those in Appendix K.

5.3.11. Numerical Solutions

5.3.11.1. METAPHOR

The remaining difficulty is calculating the integrations. For certain classes of probability densities $f_{V_i \mid V_n, V_{n-1}, \ldots, V_{i+1}, U}$, it may be possible to perform symbolically the integrations to obtain a true closed-form solution. For example, a closed-form solution for the performability of a queueing system is derived in [33]. The symbolic integrations may be performed either (laboriously) by hand or by computer programs that manipulate symbolic quantities, e.g., MAXIMA [200] and REDUCE2 [201]. A recursive version of Eqs. 5.35 and 5.152-5.143 is derived in Section 5.3.10 [Recursive Formulation of $F_{Y|U}$] and may help in reducing the amount of work necessary to perform symbolically the required integrations.

However, when performability is evaluated for complex systems with large state spaces, one is primarily interested in numerical results as opposed to closed-form solutions. For this purpose, we have written a numerical program based on Eqs. 5.25, 5.27, 5.35, and 5.152-5.143. This program is called "meta_continuous" and is now part of a larger performability evaluation package, METAPHOR (see Sections 3.4.4 [METAPHOR–A Performability Modeling and Evaluation Tool] and 4.4 [METAPHOR–A Performability Modeling and Evaluation Tool] for descriptions of METAPHOR). To obtain a system's performability, METAPHOR is given information about each trajectory sequence **u**, including the sequence itself, the conditional densities $f_{V_i \mid V_n, V_{n-1}, \ldots, V_{i+1}, U}$, and the utilization period $T = [0, t]$. METAPHOR then computes the regions of integrations $C^{'}_{y,j}$ and using Gaussian quadrature, computes $F_{Y \mid U}$. (See Section 5.3.4 [Formulation of $F_{Y|U}$].)

meta_continuous contains approximately 4,000 lines of C [176], including menu and project management functions. Appendix C contains the Unix manual entry for meta_continuous, while Appendix E contains the calling structure for meta_continous.

5.3.11.2. Multiprocessor/Air Conditioner Example

To illustrate the application of the solution procedure for F_Y and to exhibit its ability to deal with non-semi-Markov base models, we consider the multiprocessor/air conditioner example discussed in Section 5.2.2 [An Example of a Reward Based Performability Model]. (See Fig. 5.1). Recall that the performance (reward) variable Y is the number of jobs processed during some utilization period $T = [0, t]$. State (i,j) reflects the number of processors operational and whether the air conditioner is operational; each processor is identical and has computation rate of δ jobs/hour. The reward structure is $r(i,j) = i \cdot \delta$. For simplicity, the example constrains the system to a single air conditioner. (A more complete example would include multiple air conditioners.)

Suppose that, relative to a specified threshold y ($y > 0$), the system user is interested in processing more than y jobs during the bounded utilization period. For instance, the system might be a computer in a business or university during a working day, or it might be a processor handling financial transactions overnight. Note that especially in the latter category of applications, availability over the entire utilization period is not required since most of the computation could be done early in the period, or, alternatively, spread out over the entire period. In performability terms, we are considering (for a given value of y) the set of accomplishment levels $B^y = \{b \mid b > y\}$. The performability $p(B^y)$, i.e., the probability that the number of jobs processed (reward) Y is greater than y, can then be obtained from F_Y since $p(B^y) = \text{Prob}[Y > y] = 1 - F_Y(y)$.

The stochastic properties of the processors are affected by the failure time of the air conditioner in such a way that the base model is not semi-Markov. At the beginning of the utilization period, the air conditioner and all of the processors are functioning and the temperature of the room containing the equipment is 20° C. Since the amount of heat which the air conditioner must dissipate places stress on the compressor, the air conditioner's failure rate is influenced by the number of processors it must cool. Assume the air conditioner fails with an constant failure rate $\lambda_{AC}(N) = N \cdot 0.05$ failures/hour, where N is the number of

processors in the room. If the air conditioner fails, the ambient temperature in the room R begins to increase to a higher steady-state temperature with an exponential rise time; the number of processors in the room affects the speed at which the temperature rises. More precisely,

$$R(\Delta\tau) = 55^\circ + 2N - (55^\circ + 2N - 20^\circ)e^{-N\mu\Delta\tau} . \tag{5.194}$$

where $(55^\circ + 2N)^\circ$ C is the new steady-state temperature, $\Delta\tau$ is the time (in hours) since the air conditioner failed, and $\mu = 10$ degrees/hour/processor is a constant reflecting the rate of temperature increase. If a processor fails, it does not shut off and so continues contributing heat to the room. The failure behavior of the processors is influenced by the ambient temperature; each processor fails at an constant rate which varies linearly with room temperature (over the range 20° C to 55° C) from 0.001 failures/hour to .1 failure/hour. If the room temperature is R° C,

$$\lambda_P(R) = 0.001 + \left[\frac{0.1-0.001}{55^\circ - 20^\circ}\right](R - 20^\circ) . \tag{5.195}$$

There are $N+1$ state sequences $\mathbf{u}$ such that $\text{Prob}[\mathbf{U} = \mathbf{u}] > 0$ (see Fig. 5.1):

$$\begin{aligned}\{\mathbf{u}\} = \{&((N,1),(N,0),(N-1,0),\ldots,(0,0)),\\ &((N,1),(N-1,1),(N-1,0),\ldots,(0,0)),\ldots,\\ &((N,1),(N-1,1),(N-2,1),\ldots,(0,1),(0,0))\}.\end{aligned} \tag{5.196}$$

Let $\mathbf{u} = ((N,1),(N-1,1),\ldots,(k,1),(k,0),\ldots,(0,0))$. The probability of the sequence $\mathbf{u}$ is

$$\text{Prob}[U = u] = \begin{cases} \dfrac{\lambda_{AC}}{k\lambda_P + \lambda_{AC}} \times \displaystyle\prod_{j=k+1}^{N} \dfrac{j\lambda_P}{j\lambda_P + \lambda_{AC}} & \text{if } k < N \\ \dfrac{\lambda_{AC}}{N\lambda_P + \lambda_{AC}} & \text{if } k = N . \end{cases} \tag{5.197}$$

Suppose $u_i = (j,1)$; the conditional probability density function for the time the process

spends in state u_i is

$$f_{V_i \mid V_n, V_{n-1}, \ldots, V_{i+1}, \mathbf{U}}(v_i \mid v_n, v_{n-1}, \ldots, v_{i+1}, \mathbf{u}) = (j\lambda_P(20^\circ) + \lambda_{AC})\, e^{-(j\lambda_P(20^\circ)+\lambda_{AC})v_i} . \tag{5.198}$$

Suppose $u_i = (j, 0)$ and that the first state in $\mathbf{u}$ representing a failed air conditioner is u_k, $k \geq i$; the conditional probability density function for the time the process spends in state u_i is

$$f_{V_i \mid V_n, V_{n-1}, \ldots, V_{i+1}, \mathbf{U}}(v_i \mid v_n, v_{n-1}, \ldots, v_{i+1}, \mathbf{u}) = \frac{1}{K}\, j\lambda_P(R(\Delta\tau))\, e^{-j\lambda_P(R(\Delta\tau)v_i)} \tag{5.199}$$

where $\Delta\tau = v_k + v_{k-1} + \cdots + v_i$, and K is a normalization constant equal to

$$K = \int_0^\infty j\lambda_P(R(\Delta\tau')\, e^{-j\lambda_P(R(\Delta\tau'))v}\, dv , \tag{5.200}$$

(where $\Delta\tau' = v_k + v_{k-1} + \cdots + v_{i+1} + v$). That the base model is not semi-Markov can be seen from the time-varying failure rates associated with states $(i, 0)$ [Eq. 5.199]; these rates are functions of the process' history and cannot be inferred from the present state, present time, and time of entry to the state.

It is clear how Eqs. 5.197, 5.198, and 5.199 can be applied to Eqs. 5.25 and 5.27. To see how Eqs. 5.35 and 5.152-5.143 are employed to calculate F_Y, consider $N = 3$, $\mathbf{u} = ((3,1),(3,0),(2,0),(1,0),(0,0))$ and $y = 1500 \in [r(u_1)t, r(u_2)t)$. From Eq. 5.35, we see that F_Y is the sum of six multiple integrals, three of which are 0 since they are integrated over empty $\mathbf{C}_y^i$ (these are $\mathbf{C}_y^5$, $\mathbf{C}_y^4$, and $\mathbf{C}_y^0$). The other 3 $\mathbf{C}_y^i$ correspond to the sets of base model trajectories which are 3, 2, and 1-resolvable. Consider $\mathbf{C}_y^2$. From Eq. 5.139:

$$C_{y,4}^2 = \left[0, \frac{(1500 - 1000)}{(300 - 100)}\right] = [0, 2.5) ; \tag{5.201}$$

from Eq. 5.140:

$$C^2_{y,3} = \left(\underset{\geq 0}{} \frac{1500 - 2000 - (300 - 200)v_4}{300 - 200}, \frac{1500 - 1000 - (300 - 100)v_4}{300 - 100} \right] \tag{5.202}$$

$$= \left[0, \frac{500 - 200v_4}{200} \right];$$

from Eq. 5.141:

$$C^2_{y,2} = \left[0, \frac{1500 - 1000 - (300 - 100)v_3 - (300 - 100)v_4}{200 - 100} \right] \tag{5.203}$$

$$= \left[0, \frac{500 - 200(v_4 + v_3)}{100} \right];$$

and from Eq. 5.142:

$$C^2_{y,1} = [0, \infty) \text{ and} \tag{5.204}$$

$$C^2_{y,0} = \infty .$$

Once regions $\mathbf{C}^3_y$ and $\mathbf{C}^1_y$ are similarly obtained, Eq. 5.35 can then be evaluated. This process must be repeated for each state sequence **u** and then Eq. 5.25 can be applied to obtain the performability $p(B^y)$.

Fig. 5.6 shows a plot of $p(B^y) = 1 - F_Y(y)$ for $N = 1, 2, 3,$ and 4 processors, $t = 10$ hours and $\delta = 100$ jobs/hour. Note that these evaluations provide a considerable amount of information regarding the system's ability to perform (provide reward) in the presence of faults. It is easy to show, for example, that the performability $p(B^y)$ is 0 when y is greater than or equal to $N \cdot \delta \cdot t$ (e.g., for $N = 2$, $p(B^{2000}) = 0$; see Fig. 5.6). Hence, to obtain a nonzero performability, the number of processors must be greater than $\frac{y}{\delta \cdot t}$. For instance, to have a nonzero probability of accomplishing more than 1500 jobs, one must have at least 2 processors. Generally, for the values of N shown on the plot, there is a significant

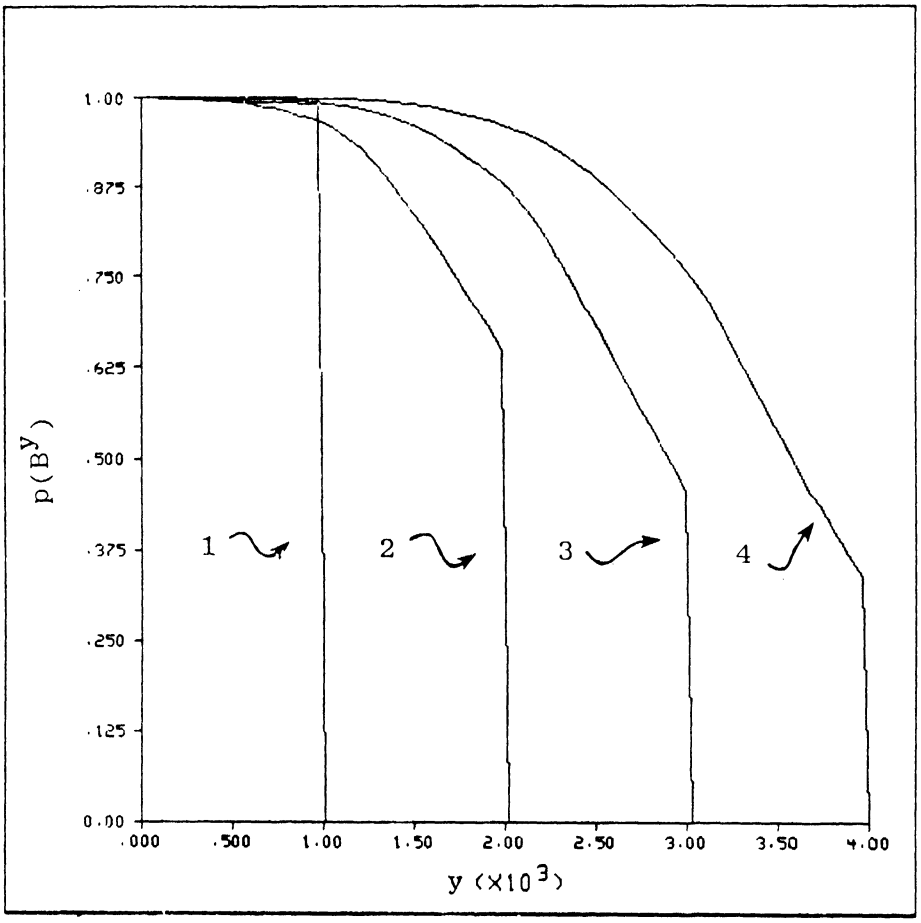


Fig. 5.6 Performability plot for the multiprocessor/air conditioner example of Section 5.3.11.2

gain in $p(B^y)$ for values of y above 1000 when additional processors are included in the system. For values of y below 1000, there is relatively little gain from having more than a single processor. Indeed, if the specified minimum reward is between about 500 and 1000 jobs, a single processor provides a greater probability of performing within B^y than do two processors.

In non-critical applications, a system designer may choose to settle for a lower performability to avoid the cost of additional processors. Information such as that provided by Fig. 5.6 can be quite useful in investigating such tradeoffs. For example, suppose the threshold is $y = 1500$. The difference between the performability with $N = 3$ and with $N = 4$ is about 0.03, while the difference between $N = 2$ and $N = 3$ is about 0.12. The probability of accomplishing more than 1500 jobs with 3 processors is about 0.96. If this probability is adequate for the application, and the additional .03 probability is not worth the cost of an extra processor, the designer may well choose a 3-processor system.

CHAPTER 6

CONCLUSIONS

6.1. Contributions

The need for combined reliability and performance measures for a broad class of systems is becoming increasingly recognized. Performability is a sufficiently general combined measure. Acceptance of performability as a measure requires simple, powerful, and automatable techniques for modeling, calculating, and using a system's performability. This thesis has addressed this issue. In particular,

a) we have generalized the analysis of systems having discrete performance variables, and

b) we extended the development of methods for analytically solving performability models using continuous performance variables.

With regard to a):

i) We have described a calculus for relating low-level structural behavior to high-level system performance.

ii) We have proposed algorithms and heuristics for efficiently performing the calculations associated with the calculus.

iii) We have implemented these algorithms in the computer package METAPHOR.

iv) Employing that computer package, we have analyzed example systems.

In the case of b):

i) We have developed the problem more generally in the context of reward models and nonrecoverable processes [55], [59].

ii) We have derived a general solution for the performability of a system modeled by a finite-state, acyclic, nonrecoverable process. This solution takes the form of an integral equation. The solution has been illustrated.

iii) We have derived a recursive form of the solution of ii). In addition, we have presented specific examples of the recursive solution.

iv) We have discussed a computer package we have written that implements the solution.

v) Using the above tools, we have analyzed a nontrivial example.

vi) We have begun consideration of still more general models.

These contributions constitute a significant extension of performability theory, and more broadly, of performance and reliability theory for degradable systems.

6.2. Further Research

The research described in this thesis has resulted in techniques for determining the performability of many useful computing systems. Building upon the work of this thesis, many questions can now be pursued, some "local" in scope, i.e., specific to the topics addressed in this dissertation, and others more "global," i.e., concerning the direction of general performability research.

In the area of the DPV (discrete performance variable, Chapter 4) methodology, the following are among the obvious refinements and extensions:

i) Still more efficient solution techniques must be developed. Aiding such derivations would be an analysis of the complexity of the present algorithms. Approximation methods would be useful for evaluating large systems. Also, it may be possible to describe special-purpose architectures for implementing many of the evaluation algorithms.

ii) The theory described in this dissertation is general, and so application of the work to areas other than those explicitly considered here should be investigated. In particular, systems such as nuclear reactors may be advantageously modeled by techniques having the time and spatial dependencies of the methodology of Chapter 4. However, more advanced solution techniques [as discussed in point i) above] would be required for large scale evaluations.

For the CPV (continuous performance variable, Chapter 5) methodology, the following remains to be done:

i) More efficient implementations (numerical integration programs) of the solution should be researched and the feasibility of using MACSYMA-class symbolic integration programs should be considered. Further, obtaining closed-form solutions with no integration terms for certain classes of systems (e.g., those with Markovian reward models) should be investigated. Additional study of approximation techniques, especially to remove the acyclic restriction (see Section 5.2.5.4 [Approximating Cyclic Processes]), may prove useful for studying other classes of systems, such as those with repair.

ii) A different method of specifying a system to METAPHOR should be developed. A "performability language" for describing the reward model could be written and a compiler constructed. For instance, the user could specify for each state: the next states with their probabilities, the sojourn time densities, and the reward.

iii) As with the DPV methodology, the CPV theory described in this dissertation is general, and so application of the work to areas other than computing systems should be

considered. For instance, applications to economic systems may be possible. The concept of reward can be interpreted many ways (see Section 5.2.3 [Determination of Reward Rates]); investigation of additional possible interpretations may lead to still more applications.

iv) Additional techniques for using the reward models considered in this thesis to study design tradeoffs should be developed.

v) The concept of reward model should be broadened. For example, the reward rate $r(i)$ of state i could be extended so that $r(i)$ is a function not just of i but of the entire history of the process to the present and of the sequence of states visited by the base model during $[0, \infty)$. We refer to such an extension as a "history based" reward model (HBRM). The interpretation of reward in history based reward models is identical to that in reward models; only the functional description of reward rate is different. The extension may be fairly straightforward, with the terms involving products $v_i r_i$ replaced with integrations. Such models would include the concepts of discounting (an exponential decay with time of the reward rate) and bonuses (instantaneous "impulse" reward upon entering a state).

vi) The restriction of a nonrecoverable model should be weakened. For example, it may be possible to allow an increase in the reward rate within a relative small range, e.g., the rate in a given state can never become greater than the smallest rate of the previous state.

APPENDICES

APPENDIX A

Unix manual entry for metaphor

NAME

metaphor – modeling and evaluation tool for performabilty

SYNOPSIS

metaphor

DESCRIPTION

The modeling and evaluation aid for performability, *metaphor* has two main components: a tool for discrete performance varible models (see meta_discrete(1)) and a tool for continuous performance variable models (see meta_continuous(1)). *metaphor* uses menu(1) to present the user with a simple menu-based method of choosing between the two classes of performability analysis. For details, refer to menmet(1), meta_continuous(1) and meta_discrete(1).

SEE ALSO

menu(1), menmet(1), meta_continuous(1) and meta_discrete(1).

AUTHOR

D. G. Furchtgott

APPENDIX B

Unix manual entry for meta_discrete

NAME

meta_discrete - calculate performability distributions

SYNOPSIS

meta_discrete

DESCRIPTION

meta_discrete computes performability distributions using the theory developed in the author's thesis. *meta_discrete* is part of the performabilty modeling and evaluation package *metaphor(1)*.

meta_discrete takes as input a logic description of a performability model hierarchy and a description of the probabilistic nature of the base model. The performability is then computed. The consistency of the model hierarchy can be checked if the user desires.

The following commands are recognized:

help Gives a short list of all available commands.

exit Leaves meta_discrete.

data This command has not been implemented in the C version of *meta_discrete*. It does work on the MTS APL version. data allows one to view the input data.

alter This command has not been implemented in the C version of *meta_discrete*. It does work on the MTS APL version. alter allows one to change the input data.

calc This command has not been implemented in the C version of *meta_discrete*. It does work on the MTS APL version. calc causes METAPHOR to enter the APL caluator mode.

com Allows the user to enter comments in midst of a *meta_discrete* session. All following input is ignored until the single word 'exit' appears by itself on a line.

eval Tells *meta_discrete* that the model construction is complete and the user is now ready to calculate the probability of the accomplishment levels. *meta_discrete* assumes that each bottom level attribute is statistically independent and so, for each array product, *meta_discrete* does a separate set of probability calculations for each attribute. Multiplication of the resulting probabilities is done to combine the results. *meta_discrete* will prompt the user for the relationships between the base model and the stochastic description that will be input. For each bottom level attribute, the user must specify:

the number of states in the stochastic model,

the number of attribute values which each stochastic model state represents,

for each stochastic model state, the specific attribute values represented,

for each base model phase, the transition **(P)** matrix (see below),

for each phase transition, the interphase **(H)** (or if the last phase, the **(F)** matrix–see below), and finally

for each bottom level attribute, the initial probability **(I** vector).

meta_discrete recognizes several interphase **(P)** matrices (see the METAPHOR User's Guide for more information about identity, given, nfail, and dedfail).

identity *meta_discrete* generates an identiy transition matrix, i.e., with probability one, th system stays in the same state.

given *meta_discrete* prompts for each entry and checks that a genuine stochastic matrix is input.

exponential *meta_discrete* constructs a transition matrix that assumes a system having N identical, statistically independent components with constant failure

rates. The user is prompted for the failure rates of the components and the length of the phase.

nfail *meta_discrete* constructs a transition matrix that assumes a system having N groups of identical, statistically independent components with constant failure rates. Group i has ni working components in each group, i.e., (n1, n2,..., nN). The user is prompted for the failure rate of the components, the length of the phase, the number of groups, and the number of components in each group. State i of the model corresponds to the following encoding (much like the decode operator of APL): Take the binary representation of (2^N)-1-i. Then the i-th digit of the binary representation (read left to right) represents the state of the corresponding component in the system, 0 if failed, 1 if not failed.

dedfail *meta_discrete* constructs a transition matrix that assumes a system having N components, each identical, statistically independent, and having a constant failure rate. The state of the model reflects specifically which components are working, i.e., dedfail is nfail with N groups of 1 component each. The user is prompted for the failure rate of the components, the length of the phase, and the number of components. The number of states must be a power of two. State i of the model corresponds to the following encoding (much like the encode operator of APL): Assign each component a unique integer between 1 and N. Take the binary representation of (2^N)-1-i. The the i-th digit of the binary representation (read from left to right) represents the state of the corresponding component in the system, 0 if failed, 1 if not failed.

sift A special transition matrix is generated which is suitable for the SIFT example. (See Meyer, Furchtgott, and Wu, *IEEE Trans. Computer*, June 1980.) The user is prompted for the number of processors, the number of busses, the failure rates of the components, and the length of the phase.

dualdual A special transition matrix is generated which is suitable for the dual-dual example of Hitt, Bridgman, and Robinson. The user is prompted for the failure rates of the components and the length of the phase.

meta_discrete also recognizes several intraphase **(G)** matrices (see the METAPHOR User's Guide for more information about identity, given, nfail, and dedfail).

identity *meta_discrete* generates an identiy transition matrix, i.e., with probability one, the system stays in the same state.

given *meta_discrete* prompts for each entry and checks that a genuine stochastic matrix is input.

build Initiates construction of a model hierarchy. Information is collected regarding the accomplishment levels, the first two model levels, the level-0 capability function, and the level-1 interlevel translation. When **meta_discrete** has gathered this data, the level-1 function is computed. The user is asked for:

the number of accomplishment levels,

their names,

the names of the top two hierarchy levels,

the number of level-0 attributes

their names, and

the number of values they can assume,

the number of level-0 phases,

their names, and

the number of level-1 attributes

their names, and

the number of values they can assume,

the number of level-1 phases,

their names, and

for each accomplishment level, the number of
array products associated with that accomplishment level,

whether the user wants consistency checked,
totalness checked, or reduction,

each level-0 to accomplishment level
array product, and

for each level-0 phase/attribute/value entry,
the number of array products associated with that entry,

whether the user wants consistency checked,
totalness checked, or reduction,

each level-1 to level-0
array product, and finally

whether the user wants a printout of the
derived level-1 capabilty function.

checkpoint
Makes a copy of the current program image (via vfork(1)) and begins execution of the copy. The user can return to the current state by using the *meta_discrete* exit command. Any number of checkpoints can be stacked (of course, up to limits of the machine and the number of user pids allowed).

echo Toggles whether input is echoed. By default, echo is off. This command is useful with the source command.

brief Toggles whether most *meta_discrete* output is printed or suppressed. By default, brief is off. Major error messages and performability results cannot be suppressed. This command is useful with the source command.

next Begin construction of the next level of the hierarchy. Most information regarding level-0 is discarded, level-1 is renamed to level-0, the level-1 capability function is renamed to the level-0 capability function, and the next level of the hierarchy is input. When **meta_discrete** has gathered this data, the level-1 function is computed. The user is asked for:

the number of level-1 attributes

their names, and

the number of values they can assume,

the number of level-1 phases,

their names, and

for each level-0 phase/attribute/value entry,
the number of array products associated with that entry,

whether the user wants consistency checked,
totalness checked, or reduction,

each level-1 to level-0
array product, and finally

whether the user wants a printout of the
derived level-1 capabilty function.

source *meta_discrete* prompts for the name of a file and takes its input from that file. By default, *meta_discrete* takes its input from the standard input, usually the user's terminal.

sink *meta_discrete* prompts for the name of a file and puts its output in that file. The special filename '-' indicates that input should be printed on the standard output, usually the user's terminal. By default, *meta_discrete* prints its output from the standard output.

printh Print the interphase **(H)** matrices.

printk Print the (current) level-1 interlevel translation (k or kappa function).

printlc Print the (current) level-1 capability function.

printm Print the interphase transition **(H)** matrices, the initial vector **(I)** and final vector **(F)**.

printp Print the state transition **(P)** matrices.

reducelc Reduce the (current) level-1 capability function.

stats Print statistics on current memory usage. The format is two rows of numbers. The top row shows the number of blocks of increasing sizes which the memory allocator has allocated and has since freed; this memory can be reused. The (n+2)-th row denotes blocks of size (2^n)-4 bytes. The bottom row shows the number of blocks which are being used.

Commands should be typed in lower case. When inputing a single line of an array product, the following syntax must be employed:
'|' <number>|<set>|<group> ... '|' <comment>
where: <number> is an attribute value. <set> is a collection of attributes written with set brackets, e.g., {0,3}. A set means that that any trajectory in that array product must contain one of the values in **every** set. <group> is a collection of attributes written with parenthesis, e.g., (0,3). A group means that that any trajectory in that array product must contain one of the values in at **least one** group.

SEE ALSO

menmet(1), metaphor(1), meta_continuous(1)

AUTHOR

D. G. Furchtgott

APPENDIX C

Unix manual entry for meta_continuous

NAME

meta_continuous - calculate performability distributions

SYNOPSIS

meta_continuous [option] ...

DESCRIPTION

meta_continuous computes performability distributions using the theory developed in the author's thesis.

It prints the values on standard output and optionally produces a file suitable for the plotting programs *gplp(1)* and *bp(1)*. *menmet(1)* is a menu-driven interface to meta_continuous. There are lots of options: the following are recognized, each as a separate argument followed by at least one numerical or string argument.

-A Solve the air conditioner example. The following options are then recognized:

- **-c** The processor temperature contribution at steady state with no air conditioning. Defaults to 0.
- **-C** The air conditioner failure rate. Defaults to 0.
- **-d** The computation rate for each processor. Defaults to 0.
- **-p** The final processor failure rate. Defaults to 0.
- **-P** The inital processor failure rate. Defaults to 0.
- **-R** The initial ambient room temperature. Defaults to 0.
- **-S** The first state in which the air conditioner does not work. Defaults to 0.
- **-T** The final ambient room temperature. Defaults to 0.
- **-u** The rate at which the temperature rises in the room when the air conditioner fails. Defaults to 0.

-f The name of the file in which the graphic image is to be placed. Defaults to graph.

-F The name of the file in which the graphic image of the input parameters is to be placed. Defaults to graph.par.

-g If followed by a 'y', a graphic image is produced. By default, no graph is produced.

-i The number of iterations to be performed. Should be used with the -I, -z, and -Z options. Defaults to 1. *meta_continuous* allows the user to specify a parameter to be varied.

-I The parameter to be varied with the -i, -z, and -Z options. This option should be followed by a single character specifying which parameter is to be varied. Legal characters are any of the options involving user specified data that involve a single data point (as opposed to a vector, such as with reward rates -r) e.g., n to vary the number of states and a to vary the arrival rates. If a graph is to be produced, the number of data points (-v) should not be varied.

-l The failure rates from state i to state i-1. The rates are listed in order from lambda[0] to lambda[n] after the -l. The first number should always be 0. The -n option must be specified before -l. These default to 0.

-M Solve the M/M/k+i/L example discussed in J. F. Meyer, 'Closed-form solutions of performability,' IEEE Trans. Computers, Vol C-32, July 1982, pp. 648-657. The following options are then recognized:

- **-a** The arrival rate for the queue. Defaults to 0.
- **-b** The buffer element failure rate. Defaults to 0.
- **-L** The length of the buffer. Defaults to 0.

-m The service rate of each processor. Defaults to 0.

-P The processor failure rate. Defaults to 0.

-n The number of non-trivial states (i.e., not counting state 0). Defaults to 1.

-N If followed by the character 'y', noramlize the performance variable y so that it lies in the range 0 to 1.

-q Number of quadrature points for the integration. Should be one of 2, 4, 8, or 16. Defaults to 4.

-r The reward rates of each state. The rates are listed in order from rate[0] to rate[n] after the -r. The -n option must be specified before -r. Defaults to 0.

-s The number of intervals which the density will be divided for the purposes of the quadrature. By default, a single interval is used.

-t The length of the utilization interval. Defaults to 0.

-v The number of data points to be printed. Defaults to 10.

-y The minimum value of t which will be considered. All caculations begin at this time.

-Y The maximum value of t which will be considered. All caculations end at this time.

-z The values to be iterated upon, if they are integers. The -i option must be specified before -z.

-Z The values to be iterated upon, if they are reals. The -i option must be specified before -Z.

SEE ALSO

bp(1), gplp(1), menmet(1), metaphor(1), meta_discrete(1)

AUTHOR

D. G. Furchtgott

NAME

menmet – menu based metaphor (continuous performance variable)

SYNOPSIS

menmet [project] [option] ...

DESCRIPTION

A modeling and evaluation aid for performability, METAPHOR(1), has two main components: a tool for discrete performance varible models (see meta_discrete(1)) and a tool for continuous performance variable models (see meta_continuous(1)). meta_continuous is a program with a myriad of options which can be specified on the run line. Indeed, meta_continuous is not designed to be run directly by a casual user. *menmet* is a preprocessor for generating all the commands and options necessary to run meta_continuous. *menmet* is based on Abe Schenker's *menu(1)* system.

The menus are grouped logically according to function.

A group of settings can be named and saved as a 'project.' Thus, if you are working with more than one document, you can preserve the settings associated with each document. To recall a particular project, include the project name on the eroff command line, e.g., menmet air_conditioner will start up menmet with all the settings saved under the project 'june.report.'

A handy convention to know in *menmet* is that commands which bring up another menu are bracketed. The best way to become familiar with all the commands available in *menmet* is to run it and explore each of the menus.

SEE ALSO

menu(1), metaphor(1), meta_discrete(1), meta_continuous(1)

AUTHOR

D. G. Furchtgott

APPENDIX D

Calling structure of meta_discrete

This appendix contains the calling structure for meta_discrete, i.e., the portion of METAPHOR that deals with discrete performance variables. The calling structure provides a good overview of the program's flow structure. Most of the major functions that concern construction of the hierarchy and error checking are discussed in Section 4.3.3 [Algorithms]. Many other functions dealing with the calculation of probailities are discussed in Furchtgott furchtgott sel128 .[furchtgott sel136 In the analysis below, external functions (e.g., system functions) and macros are denoted by "[ext]." Recursive functions are delimited by "^." Functions which appear earlier in the analysis are referred by line number.

```
1  main
2     fprintf
3     input
4        printquad [ext]
5        fscanf [ext]
6        commandexit
7           fprintf
8           exit [ext]
9        strlen [ext]
10       strcmp [ext]
11       strcpy [ext]
12       print [ext]
13       getc [ext]
14       command
15          commandhelp
16             fprintf
17          commandexit ... [see line 6]
18          commanddata
19             fprintf
20          commandalter
21             fprintf
22          commandcalc
23             fprintf
24          commandecho
25          commandcom
26             printquad [ext]
27             getc [ext]
28             commandexit ... [see line 6]
29             fprintf
30          commandbrief
31          commandeval
32             getstates
33                sprintf [ext]
34                print [ext]
35                strcpy [ext]
36                strcmp [ext]
```

```
37                ^ input ^
38              atoi [ext]
39              getnumstates
40                 print [ext]
41                 sprintf [ext]
42                 strcpy [ext]
43                 strcmp [ext]
44                   ^ input ^
45                 atoi [ext]
46                 fprintf
47              alloce
48                 calloc
49                 sizeof [ext]
50                 fprintf
51                 exit [ext]
52              gete
53                 calloc
54                 sizeof [ext]
55                 fprintf
56                 exit [ext]
57                 sprintf [ext]
58                 print [ext]
59                 strcpy [ext]
60                 strcmp [ext]
61                   ^ input ^
62                 EVALUE [ext]
63                 atoi [ext]
64                 free
65                    ASSERT [ext]
66           getpmatrices
67              allocp
68                 calloc
69                 sizeof [ext]
70                 fprintf
71                 exit [ext]
72              print [ext]
73              sprintf [ext]
74              generatepmatrix
75                 strcpy [ext]
76                 strcmp [ext]
77                 print [ext]
78                   ^ input ^
79                 pgiven
80                    print [ext]
81                    strcpy [ext]
82                    strcmp [ext]
83                    sprintf [ext]
84                      ^ input ^
85                    checkprob
86                       atof
87                       fprintf
88                    checkone
89                       atof
90                       fprintf
91                    PVALUE [ext]
92                    atof
93                 dedfail
94                    log [ext]
95                    fprintf
96                    print [ext]
97                    strcpy [ext]
98                    strcmp [ext]
99                      ^ input ^
100                   atof
101                   valueinfo
102                   PVALUE [ext]
103                   exp [ext]
104                   pow [ext]
105                nfail
106                   print [ext]
107                   strcpy [ext]
108                   strcmp [ext]
109                     ^ input ^
110                   atof
111                   atoi [ext]
```

```
112                 fprintf
113                 choose
114                 PVALUE [ext]
115                 exp [ext]
116                 pow [ext]
117             sift
118                 print [ext]
119                 strcpy [ext]
120                 strcmp [ext]
121                 ^ input ^
122                 atof
123                 atoi [ext]
124                 fprintf
125                 PVALUE [ext]
126                 choose
127                 exp [ext]
128                 pow [ext]
129             pidentity
130                 PVALUE [ext]
131             exponential
132                 print [ext]
133                 strcpy [ext]
134                 strcmp [ext]
135                 ^ input ^
136                 atof
137                 PVALUE [ext]
138                 exp [ext]
139             dualdual
140                 print [ext]
141                 strcpy [ext]
142                 strcmp [ext]
143                 ^ input ^
144                 atof
145                 exp [ext]
146                 PVALUE [ext]
147                 fprintf
148         gethmatrices
149             alloch
150                 calloc
151                 sizeof [ext]
152                 fprintf
153                 exit [ext]
154             print [ext]
155             sprintf [ext]
156             generatehmatrix
157                 strcpy [ext]
158                 strcmp [ext]
159                 print [ext]
160                 ^ input ^
161                 hgiven
162                     print [ext]
163                     strcpy [ext]
164                     strcmp [ext]
165                     sprintf [ext]
166                     ^ input ^
167                     checkprob ... [see line 85]
168                     checkone ... [see line 88]
169                     HVALUE [ext]
170                     atof
171                 hidentity
172                     HVALUE [ext]
173         getivector
174             strcpy [ext]
175             strcmp [ext]
176             print [ext]
177             sprintf [ext]
178             ^ input ^
179             checkprob ... [see line 85]
180             checkone ... [see line 88]
181             atof
182         getbasicvarprob
183             allocb
184                 calloc
185                 sizeof [ext]
186                 fprintf
```

```
187                 exit [ext]
188              strcpy [ext]
189              strcmp [ext]
190              print [ext]
191              sprintf [ext]
192                 ^ input ^
193              checkprob ... [see line 85]
194              checkone ... [see line 88]
195              BVALUE [ext]
196              atof
197           getperformability
198              allocg
199                 calloc
200                 sizeof [ext]
201                 fprintf
202                 exit [ext]
203              getacclevprob
204                 calcarrayprob
205                    PVALUE [ext]
206                    GVALUE [ext]
207                    HVALUE [ext]
208                    BVALUE [ext]
209                    bVALUE [ext]
210                 LCBOUNDARY
211                 expandg
212                    GVALUE [ext]
213                    valueinfo
214                    LCARRAYVALUES [ext]
215                    EVALUE [ext]
216                 expandf
217                    valueinfo
218                    LCARRAYVALUES [ext]
219                    EVALUE [ext]
220                 expandb
221                    bVALUE [ext]
222                    valueinfo
223                    BASICARRAYVALUES [ext]
224           printperformability
225              print [ext]
226              sprintf [ext]
227              fprintf
228           free ... [see line 64]
229        commandbuild
230           getacclev
231              print [ext]
232              strcpy [ext]
233              strcmp [ext]
234                 ^ input ^
235              atoi [ext]
236              sprintf [ext]
237              getname
238                 sprintf [ext]
239                 print [ext]
240                 getc [ext]
241                 commandexit ... [see line 6]
242                 fprintf
243              strlen [ext]
244           print [ext]
245           getname ... [see line 237]
246           getlevel0
247              getattr0val
248                 print [ext]
249                 sprintf [ext]
250                 strcpy [ext]
251                 strcmp [ext]
252                    ^ input ^
253                 atoi [ext]
254                 getname ... [see line 237]
255                 strlen [ext]
256                 getattr0
257                    print [ext]
258                    sprintf [ext]
259                    strcpy [ext]
260                    strcmp [ext]
261                       ^ input ^
```

```
262                 atoi [ext]
263             getphase0val
264                 print [ext]
265                 sprintf [ext]
266                 strcpy [ext]
267                 strcmp [ext]
268                   ^ input ^
269                 atoi [ext]
270                 getname ... [see line 237]
271                 strlen [ext]
272             getbasicval
273                 sprintf [ext]
274                 print [ext]
275                 getname ... [see line 237]
276                 strlen [ext]
277                 getbasicvar
278                     print [ext]
279                     sprintf [ext]
280                     strcpy [ext]
281                     strcmp [ext]
282                       ^ input ^
283                     atoi [ext]
284         getlevel1
285             getattr1val
286                 print [ext]
287                 sprintf [ext]
288                 strcpy [ext]
289                 strcmp [ext]
290                   ^ input ^
291                 atoi [ext]
292                 getname ... [see line 237]
293                 strlen [ext]
294                 getattr1
295                     print [ext]
296                     sprintf [ext]
297                     strcpy [ext]
298                     strcmp [ext]
299                       ^ input ^
300                     atoi [ext]
301             getphase1val
302                 print [ext]
303                 sprintf [ext]
304                 strcpy [ext]
305                 strcmp [ext]
306                   ^ input ^
307                 atoi [ext]
308                 getname ... [see line 237]
309                 strlen [ext]
310         getlcarray
311             getlccount
312                 calloc
313                 sizeof [ext]
314                 fprintf
315                 exit [ext]
316                 print [ext]
317                 sprintf [ext]
318                 strcpy [ext]
319                 strcmp [ext]
320                   ^ input ^
321                 LCCOUNT [ext]
322                 atoi [ext]
323             getlcarrays
324                 calloc
325                 sizeof [ext]
326                 fprintf
327                 exit [ext]
328                 print [ext]
329                 inyes
330                     printquad [ext]
331                     fscanf [ext]
332                     commandexit ... [see line 6]
333                     print [ext]
334                     getc [ext]
335                 LCBOUNDARY
336                 LCCOUNT [ext]
```

```
337                 getonelcarrayset
338                     alloclc
339                         fprintf
340                         exit [ext]
341                         calloc
342                         sizeof [ext]
343                     print [ext]
344                     sprintf [ext]
345                     LCARRAY
346                     strcpy [ext]
347                     strcmp [ext]
348                     charinput
349                         getc [ext]
350                         commandexit ... [see line 6]
351                         print [ext]
352                         strcmp [ext]
353                         ^ command ^
354                         strcpy [ext]
355                     LCARRAYATTR
356                     fprintf
357                     exit [ext]
358                     parse0
359                         parserr0
360                             fprintf
361                             sprintf [ext]
362                             print [ext]
363                         GENARRAYVALUES [ext]
364                         isdigit [ext]
365                         atoi [ext]
366                     parsebasic
367                         parserrbasic
368                             fprintf
369                             sprintf [ext]
370                             print [ext]
371                         GENBASICARRAYVALUES [ext]
372                         isdigit [ext]
373                         atoi [ext]
374                     BASICARRAYATTR
375                     expandarray
376                         allocarray
377                             fprintf
378                             exit [ext]
379                             calloc
380                             sizeof [ext]
381                         GENARRAYVALUES [ext]
382                         GENBASICARRAYVALUES [ext]
383                     free ... [see line 64]
384                 checklcarray
385                     LCBOUNDARY
386                     LCARRAYVALUES [ext]
387                     BASICARRAYVALUES [ext]
388                     fprintf
389                 free ... [see line 64]
390                 checktotal
391                     copyarray
392                         allocarray ... [see line 376]
393                         GENARRAYVALUES [ext]
394                     compresstest
395                     squeeze
396                         free ... [see line 64]
397                     fprintf
398                     ALLONES [ext]
399                     GENARRAYVALUES [ext]
400                     print [ext]
401                     freearray
402                         free ... [see line 64]
403                     sprintf [ext]
404                     fflush
405                     inyes ... [see line 329]
406                     printf [ext]
407                     printreducedarray
408                         print [ext]
409                         printquad [ext]
410                         sprintf [ext]
411                         print0array
```

```
412                         valueinfo
413                         sprintf [ext]
414                         fprintf
415                         exit [ext]
416                         print [ext]
417                      print1array
418                         valueinfo
419                         sprintf [ext]
420                         fprintf
421                         exit [ext]
422                         print [ext]
423                   complementarray
424                      allocarray ... [see line 376]
425                      calloc
426                      sizeof [ext]
427                      fprintf
428                      exit [ext]
429                      complement
430                         allocarray ... [see line 376]
431                         ALLONES [ext]
432                         GENARRAYVALUES [ext]
433                         GENBASICARRAYVALUES [ext]
434                      print [ext]
435                      GENKNOWBOUNDARY [ext]
436                      GENARRAYVALUES [ext]
437                      comptree
438                         allocarray ... [see line 376]
439                         GENKNOWBOUNDARY [ext]
440                         GENARRAYVALUES [ext]
441                         ^ comptree ^
442                         free ... [see line 64]
443                      reducearray
444                         compresstest
445                         squeeze ... [see line 395]
446                      printreducedarray ... [see line 407]
447                      freearray ... [see line 401]
448                      free ... [see line 64]
449                reducelcarray
450                   print [ext]
451                   LCBOUNDARY
452                   compresstest
453                   squeeze ... [see line 395]
454                   inyes ... [see line 329]
455                   printlcarray
456                      print [ext]
457                      printquad [ext]
458                      sprintf [ext]
459                      LCBOUNDARY
460                      print0array ... [see line 411]
461                      LCARRAY
462                      printbasicarray
463                         valueinfo
464                         sprintf [ext]
465                         fprintf
466                         exit [ext]
467                         print [ext]
468          getkarray
469             getkcount
470                calloc
471                sizeof [ext]
472                fprintf
473                exit [ext]
474                print [ext]
475                sprintf [ext]
476                strcpy [ext]
477                strcmp [ext]
478                ^ input ^
479                KCOUNT [ext]
480                atoi [ext]
481             getkarrays
482                calloc
483                sizeof [ext]
484                fprintf
485                exit [ext]
486                print [ext]
```

```
487                inyes ... [see line 329]
488                sprintf [ext]
489                KCOUNT [ext]
490                KBOUNDARY
491                getonekarrayset
492                   allock
493                      fprintf
494                      exit [ext]
495                      calloc
496                      sizeof [ext]
497                   print [ext]
498                   sprintf [ext]
499                   strcpy [ext]
500                   strcmp [ext]
501                   charinput ... [see line 348]
502                   parse1
503                      parserr1
504                         fprintf
505                         sprintf [ext]
506                         print [ext]
507                      GENARRAYVALUES [ext]
508                      isdigit [ext]
509                      atoi [ext]
510                   expandarray ... [see line 375]
511                   free ... [see line 64]
512                checkkarray
513                   KBOUNDARY
514                   KARRAYVALUES [ext]
515                   fprintf
516                free ... [see line 64]
517                checktotal ... [see line 390]
518                setbasicvar
519                   sprintf [ext]
520                   strlen [ext]
521                   fprintf
522                reducekarray
523                   print [ext]
524                   KBOUNDARY
525                   compresstest
526                   squeeze ... [see line 395]
527                   inyes ... [see line 329]
528                   printkarray
529                      print [ext]
530                      printquad [ext]
531                      sprintf [ext]
532                      KBOUNDARY
533                      print1array ... [see line 417]
534                      KARRAY
535             intarrays
536                allocarray ... [see line 376]
537                calloc
538                sizeof [ext]
539                fprintf
540                exit [ext]
541                GENARRAYVALUES [ext]
542                print [ext]
543                GENLCBOUNDARY
544                LCBOUNDARY
545                GENBASICARRAYVALUES [ext]
546                BASICARRAYVALUES [ext]
547                buildtree
548                   allocarray ... [see line 376]
549                   KBOUNDARY
550                   fprintf
551                   sprintf [ext]
552                   mstats
553                   valueinfo
554                   LCARRAYVALUES [ext]
555                   GENARRAYVALUES [ext]
556                   GENBASICARRAYVALUES [ext]
557                   copyarray ... [see line 391]
558                   commandexit ... [see line 6]
559                   reducearray ... [see line 443]
560                   quickreduce
561                      compresstest
```

```
562                     squeeze ... [see line 395]
563                  ^ buildtree ^
564                     free ... [see line 64]
565                  free ... [see line 64]
566                  setnext
567                     freearray ... [see line 401]
568                     free ... [see line 64]
569                     strcpy [ext]
570                     strncpy [ext]
571                  reducelcarray ... [see line 449]
572                  inyes ... [see line 329]
573                  printlcarray ... [see line 455]
574            commandnext
575               sprintf [ext]
576               print [ext]
577               getname ... [see line 237]
578               getlevel1 ... [see line 284]
579               getkarray ... [see line 468]
580               intarrays ... [see line 535]
581            commandsource
582               print [ext]
583               charinput ... [see line 348]
584               fclose [ext]
585               fopen [ext]
586               fprintf
587            commandsink
588               print [ext]
589               charinput ... [see line 348]
590               fclose [ext]
591               fopen [ext]
592               fprintf
593            commandprinth
594               fprintf
595               HVALUE [ext]
596            printkarray ... [see line 528]
597            printlcarray ... [see line 455]
598            commandprintm
599               fprintf
600               GVALUE [ext]
601            commandprintp
602               fprintf
603               PVALUE [ext]
604            reducelcarray ... [see line 449]
605            mstats
606            commandcheckpoint
607               fork [ext]
608               fprintf
609               sigignore [ext]
610               wait [ext]
611               fclose [ext]
612               print [ext]
613               sigset [ext]
614            commandbotch
615            fprintf

616   realloc
617     malloc
618        morecore
619           sbrk [ext]
620           vlimit [ext]
621           write [ext]
622     copymem
623        asm [ext]
624     free ... [see line 64]
```

APPENDIX E

Calling structure of meta_continuous

This appendix contains the calling structure for meta_continuous, i.e., the portion of METAPHOR that deals with continuous performance variables. The calling structure provides a good overview of the program's flow structure. In the analysis below, external functions (e.g., system functions) and macros are denoted by "[ext]." Recursive functions are delimited by "^." Functions which appear earlier in the analysis are referred by line number.

```
1  main
2      init
3          sprintf
4      input
5          fetch [ext]
6          fprintf [ext]
7          exit [ext]
8      iterate
9          printf [ext]
10         checkit
11             floor [ext]
12             printf [ext]
13             exit [ext]
14             queuesetup
15                 ipow
16                     fprintf [ext]
17                     exit [ext]
18                 fact
19                     fprintf [ext]
20                     exit [ext]
21             acsetup
22                 intnorm
23                     Loww
24                     Uppp
25                     Funevv
26                         exp [ext]
27                     printf [ext]
28                     pow [ext]
29             point
30             printsetup
31                 printf [ext]
32         calc
33             perf
34                 {C}
35                     multint
36                         Low
37                             adj [ext]
38                             printf [ext]
39                             sum [ext]
40                         Upp
41                             adj [ext]
42                             printf [ext]
43                             sum [ext]
44                             log [ext]
```

```
45                  Funev
46                      intnorm ... [see line 22]
47                      adj [ext]
48                      exp [ext]
49                      fprintf [ext]
50                  printf [ext]
51                  pow [ext]
52          printit
53              printf [ext]
54              ffflush
55                  fflush [ext]
56                  printf [ext]
57                  exit [ext]
58      info
59          printf [ext]
60          printit ... [see line 52]
61      plotit
62          plotinit
63              plots [ext]
64              fname [ext]
65              plot [ext]
66          scale [ext]
67          plot [ext]
68          axisv [ext]
69          axis [ext]
70          dline [ext]
71      plotinput
72          plotinit ... [see line 62]
73          plot [ext]
74          doprint
75              symbol
76                  sprintf
77                      symbol
78              dline [ext]
79              plot [ext]
```

APPENDIX F

METAPHOR session for the simple reliability network example

This appendix contains a METAPHOR session for evaluating the series-parallel reliability network example of Section 4.5.1 [Simple Reliability Network Example].

```
Michigan EvaluaTion Aid for PerpHORmability
      version 3.0

type help for assistance
#: build


number of accomplishment levels?
#: 2
getting the name of each accomplishment level
What is the name of accomplishment level 0?
SUCCESS
What is the name of accomplishment level 1?
FAIL
getting the names of the top two levels:
What is the name of model hierarchy level-0 (the highest level)?
THE MISSION LEVEL
What is the name of model hierarchy level-1 (the second highest level)?
THE COMPONENT LEVEL

number of level-0 (THE MISSION LEVEL) attributes?
#: 1

getting the name of each level-0 (THE MISSION LEVEL) attribute:
What is the name of attribute 0?
RELIABITY GRAPH CONNECTIVITY

number of values per level-0 (THE MISSION LEVEL) attribute?
#: 2

number of level-0 (THE MISSION LEVEL) phases?
#: 1
getting the name of each level-0 (THE MISSION LEVEL) phase:
What is the name of phase 0?
LEVEL 0 MISSION

number of level 1 (THE COMPONENT LEVEL) attributes?
#: 4

getting the name of each level 1 (THE COMPONENT LEVEL) attribute:
What is the name of attribute 0?
COMPONENT A
What is the name of attribute 1?
COMPONENT B
What is the name of attribute 2?
COMPONENT C
What is the name of attribute 3?
COMPONENT D
```

```
number of values per level 1 (THE COMPONENT LEVEL) attribute?
#: 2 2 2 2

number of level 1 (THE COMPONENT LEVEL) phases?
#: 1
getting the name of each level 1 (THE COMPONENT LEVEL) phase:
What is the name of phase 0?
LEVEL 1 MISSION

for each accomplishment level
enter the number of array products associated
with that accomplishment level.

accomplishment level 0 (SUCCESS)
#: 1
accomplishment level 1 (FAIL)
#: 1


partial capability function array product specification

will you want consistency checked?
#: yes

will you want totalness checked?
#: yes

will you want reduction?
#: yes

enter each array product corresponding to each accomplishment level


accomplishment level 0 (SUCCESS):  array product 0

 the level-0 (THE MISSION LEVEL) attribute 0 (RELIABITY GRAPH CONNECTIVITY)
#THIS IS THE TRAJECTORY SET FOR ACC LEV 0
 the level-0 (THE MISSION LEVEL) attribute 0 (RELIABITY GRAPH CONNECTIVITY)
| 0 |

accomplishment level 1 (FAIL):  array product 0

 the level-0 (THE MISSION LEVEL) attribute 0 (RELIABITY GRAPH CONNECTIVITY)
#THIS IS THE TRAJECTORY SET FOR ACC LEV 1
 the level-0 (THE MISSION LEVEL) attribute 0 (RELIABITY GRAPH CONNECTIVITY)
| 1 |
the inverse capability function is already reduced

for each value that each level-0 (THE MISSION LEVEL) attribute and phase can
assume, enter the number of array products to be
input for that value, phase, and attribute.
if an attribute and phase is to be basic, enter BASIC.

level-0 (THE MISSION LEVEL) attribute 0 (RELIABITY GRAPH CONNECTIVITY)
     phase 0 (LEVEL 0 MISSION)
   value = 0:
#: 5
   value = 1:
#: 2


interlevel translation array product specification

will you want consistency checked?
#: yes

will you want totalness checked?
#: yes

will you want reduction?
#: yes

enter each array product corresponding to each combination of
  level-0 (THE MISSION LEVEL) attribute, phase, and value
```

```
 level-0 (THE MISSION LEVEL) attribute 0 (RELIABITY GRAPH CONNECTIVITY)
  phase 0 (LEVEL 0 MISSION)
  value = 0
    array product 0

the level-1 (THE COMPONENT LEVEL) attribute 0 (COMPONENT A)
#CONNECTIVITY = 0  set 1
the level-1 (THE COMPONENT LEVEL) attribute 0 (COMPONENT A)
| 0 |
the level-1 (THE COMPONENT LEVEL) attribute 1 (COMPONENT B)
| 0 |
the level-1 (THE COMPONENT LEVEL) attribute 2 (COMPONENT C)
| • |
the level-1 (THE COMPONENT LEVEL) attribute 3 (COMPONENT D)
| • |

 level-0 (THE MISSION LEVEL) attribute 0 (RELIABITY GRAPH CONNECTIVITY)
  phase 0 (LEVEL 0 MISSION)
  value = 0
    array product 1

the level-1 (THE COMPONENT LEVEL) attribute 0 (COMPONENT A)
#CONNECTIVITY = 0  set 2
the level-1 (THE COMPONENT LEVEL) attribute 0 (COMPONENT A)
| 1 |
the level-1 (THE COMPONENT LEVEL) attribute 1 (COMPONENT B)
| • |
the level-1 (THE COMPONENT LEVEL) attribute 2 (COMPONENT C)
| 0 |
the level-1 (THE COMPONENT LEVEL) attribute 3 (COMPONENT D)
| • |

 level-0 (THE MISSION LEVEL) attribute 0 (RELIABITY GRAPH CONNECTIVITY)
  phase 0 (LEVEL 0 MISSION)
  value = 0
    array product 2

the level-1 (THE COMPONENT LEVEL) attribute 0 (COMPONENT A)
#CONNECTIVITY = 0  set 3
the level-1 (THE COMPONENT LEVEL) attribute 0 (COMPONENT A)
| 1 |
the level-1 (THE COMPONENT LEVEL) attribute 1 (COMPONENT B)
| • |
the level-1 (THE COMPONENT LEVEL) attribute 2 (COMPONENT C)
| 1 |
the level-1 (THE COMPONENT LEVEL) attribute 3 (COMPONENT D)
| 0 |

 level-0 (THE MISSION LEVEL) attribute 0 (RELIABITY GRAPH CONNECTIVITY)
  phase 0 (LEVEL 0 MISSION)
  value = 0
    array product 3

the level-1 (THE COMPONENT LEVEL) attribute 0 (COMPONENT A)
#CONNECTIVITY = 0  set 4
the level-1 (THE COMPONENT LEVEL) attribute 0 (COMPONENT A)
| 0 |
the level-1 (THE COMPONENT LEVEL) attribute 1 (COMPONENT B)
| 1 |
the level-1 (THE COMPONENT LEVEL) attribute 2 (COMPONENT C)
| 0 |
the level-1 (THE COMPONENT LEVEL) attribute 3 (COMPONENT D)
| • |

 level-0 (THE MISSION LEVEL) attribute 0 (RELIABITY GRAPH CONNECTIVITY)
  phase 0 (LEVEL 0 MISSION)
  value = 0
    array product 4

the level-1 (THE COMPONENT LEVEL) attribute 0 (COMPONENT A)
#CONNECTIVITY = 0  set 5
the level-1 (THE COMPONENT LEVEL) attribute 0 (COMPONENT A)
| 0 |
the level-1 (THE COMPONENT LEVEL) attribute 1 (COMPONENT B)
| 1 |
```

```
the level-1 (THE COMPONENT LEVEL) attribute 2 (COMPONENT C)
| 1 |
the level-1 (THE COMPONENT LEVEL) attribute 3 (COMPONENT D)
| 0 |

 level-0 (THE MISSION LEVEL) attribute 0 (RELIABITY GRAPH CONNECTIVITY)
  phase 0 (LEVEL 0 MISSION)
  value = 1
     array product 0

the level-1 (THE COMPONENT LEVEL) attribute 0 (COMPONENT A)
#CONNECTIVITY = 1   set 1
the level-1 (THE COMPONENT LEVEL) attribute 0 (COMPONENT A)
| 1 |
the level-1 (THE COMPONENT LEVEL) attribute 1 (COMPONENT B)
| * |
the level-1 (THE COMPONENT LEVEL) attribute 2 (COMPONENT C)
| 1 |
the level-1 (THE COMPONENT LEVEL) attribute 3 (COMPONENT D)
| 1 |

 level-0 (THE MISSION LEVEL) attribute 0 (RELIABITY GRAPH CONNECTIVITY)
  phase 0 (LEVEL 0 MISSION)
  value = 1
     array product 1

the level-1 (THE COMPONENT LEVEL) attribute 0 (COMPONENT A)
#CONNECTIVITY = 1   set 2
the level-1 (THE COMPONENT LEVEL) attribute 0 (COMPONENT A)
| 0 |
the level-1 (THE COMPONENT LEVEL) attribute 1 (COMPONENT B)
| 1 |
the level-1 (THE COMPONENT LEVEL) attribute 2 (COMPONENT C)
| 1 |
the level-1 (THE COMPONENT LEVEL) attribute 3 (COMPONENT D)
| 1 |
the inverse interlevel translation is already reduced

do you want a list of the array products for the capability function?
#: no
recommended commands are checkpoint, next, or
#: eval
getting the stochastic model information for attribute 0 (COMPONENT A)?

number of stochastic model states for attribute 0 (COMPONENT A)?
#:
number of stochastic model states for attribute 0 (COMPONENT A)?
#: 2
getting the number of stochastic model states corresponding to
value of bottom-level attribute 0 (COMPONENT A):
value 0
#: 1
value 1
#: 1
for bottom attribute 0 (COMPONENT A),
getting the states corresponding to each value:
value 0 of attribute 0 (COMPONENT A)
#: 0
value 1 of attribute 0 (COMPONENT A)
#: 1


for bottom level attribute 0 (COMPONENT A),
specify the p matrices for each phase, 1 phase at a time

phase 0 (LEVEL 1 MISSION):

what type of p matrix?
#: exponential

enter phase length
#: 10
enter component failure rate
#: .0005
```

```
for bottom level attribute 0 (COMPONENT A),
specify the h matrices for each phase, 1 phase at a time

for the stochastic model corresponding to
bottom level attribute 0 (COMPONENT A)
enter the inital probabilities of each state,
beginning with the lowest numbered state:
#: 0.0 1.0
getting the stochastic model information for attribute 1 (COMPONENT B)?

number of stochastic model states for attribute 1 (COMPONENT B)?
#:
number of stochastic model states for attribute 1 (COMPONENT B)?
#: 2
getting the number of stochastic model states corresponding to
value of bottom-level attribute 1 (COMPONENT B):
value 0
#: 1
value 1
#: 1
for bottom attribute 1 (COMPONENT B),
getting the states corresponding to each value:
value 0 of attribute 1 (COMPONENT B)
#: 0
value 1 of attribute 1 (COMPONENT B)
#: 1


for bottom level attribute 1 (COMPONENT B),
specify the p matrices for each phase, 1 phase at a time

phase 0 (LEVEL 1 MISSION):

what type of p matrix?
#: exponential

enter phase length
#: 10
enter component failure rate
#: .0004


for bottom level attribute 1 (COMPONENT B),
specify the h matrices for each phase, 1 phase at a time

for the stochastic model corresponding to
bottom level attribute 1 (COMPONENT B)
enter the inital probabilities of each state,
beginning with the lowest numbered state:
#: 0.0 1.0
getting the stochastic model information for attribute 2 (COMPONENT C)?

number of stochastic model states for attribute 2 (COMPONENT C)?
#:
number of stochastic model states for attribute 2 (COMPONENT C)?
#: 2
getting the number of stochastic model states corresponding to
value of bottom-level attribute 2 (COMPONENT C):
value 0
#: 1
value 1
#: 1
for bottom attribute 2 (COMPONENT C),
getting the states corresponding to each value:
value 0 of attribute 2 (COMPONENT C)
#: 0
value 1 of attribute 2 (COMPONENT C)
#: 1


for bottom level attribute 2 (COMPONENT C),
specify the p matrices for each phase, 1 phase at a time

phase 0 (LEVEL 1 MISSION):
```

```
what type of p matrix?
#: exponential

enter phase length
#: 10
enter component failure rate
#: .001


for bottom level attribute 2 (COMPONENT C),
specify the h matrices for each phase, 1 phase at a time

for the stochastic model corresponding to
bottom level attribute 2 (COMPONENT C)
enter the inital probabilities of each state,
beginning with the lowest numbered state:
#: 0.0 1.0
getting the stochastic model information for attribute 3 (COMPONENT D)?

number of stochastic model states for attribute 3 (COMPONENT D)?
#:
number of stochastic model states for attribute 3 (COMPONENT D)?
#: 2
getting the number of stochastic model states corresponding to
value of bottom-level attribute 3 (COMPONENT D):
value 0
#: 1
value 1
#: 1
for bottom attribute 3 (COMPONENT D),
getting the states corresponding to each value:
value 0 of attribute 3 (COMPONENT D)
#: 0
value 1 of attribute 3 (COMPONENT D)
#: 1


for bottom level attribute 3 (COMPONENT D),
specify the p matrices for each phase, 1 phase at a time

phase 0 (LEVEL 1 MISSION):

what type of p matrix?
#: exponential

enter phase length
#: 10
enter component failure rate
#: .001


for bottom level attribute 3 (COMPONENT D),
specify the h matrices for each phase, 1 phase at a time

for the stochastic model corresponding to
bottom level attribute 3 (COMPONENT D)
enter the inital probabilities of each state,
beginning with the lowest numbered state:
#: 0.0 1.0



performability for this mission:
 accomplishment level 0 (SUCCESS): 9.999991e-01

 accomplishment level 1 (FAIL): 8.870545e-07

#: exit

bye.
```

APPENDIX G

METAPHOR session for the simple air transport example

This appendix contains a METAPHOR session for evaluating the simple air transport mission example of Section 4.5.2 [Simple Air Transport Mission Example].

```
Michigan EvaluaTion Aid for PerpHORmability
      version 3.0

type help for assistance
#: build

number of accomplishment levels?
#: 5
getting the name of each accomplishment level
What is the name of accomplishment level 0?
ALL GOOD (000)
What is the name of accomplishment level 1?
BAD FUEL EFFICIENCY (001)
What is the name of accomplishment level 2?
DIVERSION (010)
What is the name of accomplishment level 3?
BAD FUEL EFFICIENCY AND DIVERSION (011)
What is the name of accomplishment level 4?
1 • • (CRASH)
getting the names of the top two levels:
What is the name of model hierarchy level-0 (the highest level)?
THE MISSION LEVEL
What is the name of model hierarchy level-1 (the second highest level)?
THE AIRCRAFT FUNCTIONAL TASK LEVEL

number of level-0 (THE MISSION LEVEL) attributes?
#: 3

getting the name of each level-0 (THE MISSION LEVEL) attribute:
What is the name of attribute 0?
FUEL CONSUMPTION
What is the name of attribute 1?
DIVERSION
What is the name of attribute 2?
SAFETY

number of values per level-0 (THE MISSION LEVEL) attribute?
#: 2 2 2

number of level-0 (THE MISSION LEVEL) phases?
#: 1
getting the name of each level-0 (THE MISSION LEVEL) phase:
What is the name of phase 0?
MISSION

number of level 1 (THE AIRCRAFT FUNCTIONAL TASK LEVEL) attributes?
#: 2

getting the name of each level 1 (THE AIRCRAFT FUNCTIONAL TASK LEVEL) attribute:
```

```
What is the name of attribute 0?
ACTIVE CONTROL
What is the name of attribute 1?
WEATHER

number of values per level 1 (THE AIRCRAFT FUNCTIONAL TASK LEVEL) attribute?
#: 4 2

number of level 1 (THE AIRCRAFT FUNCTIONAL TASK LEVEL) phases?
#: 2
getting the name of each level 1 (THE AIRCRAFT FUNCTIONAL TASK LEVEL) phase:
What is the name of phase 0?
TAKEOFF AND CRUISE
What is the name of phase 1?
LANDING

for each accomplishment level
enter the number of array products associated
with that accomplishment level.

accomplishment level 0 (ALL GOOD (000))
#: 1
accomplishment level 1 (BAD FUEL EFFICIENCY (001))
#: 1
accomplishment level 2 (DIVERSION (010))
#: 1
accomplishment level 3 (BAD FUEL EFFICIENCY AND DIVERSION (011))
#: 1
accomplishment level 4 (1 * * (CRASH))
#: 1


partial capability function array product specification

will you want consistency checked?
#: YES

will you want totalness checked?
#: YES

will you want reduction?
#: YES

enter each array product corresponding to each accomplishment level


accomplishment level 0 (ALL GOOD (000)): array product 0

 the level-0 (THE MISSION LEVEL) attribute 0 (FUEL CONSUMPTION)
#THIS IS THE TRAJECTORY SET FOR ACC LEV 0
 the level-0 (THE MISSION LEVEL) attribute 0 (FUEL CONSUMPTION)
 | 0 |
 the level-0 (THE MISSION LEVEL) attribute 1 (DIVERSION)
 | 0 |
 the level-0 (THE MISSION LEVEL) attribute 2 (SAFETY)
 | 0 |

accomplishment level 1 (BAD FUEL EFFICIENCY (001)): array product 0

 the level-0 (THE MISSION LEVEL) attribute 0 (FUEL CONSUMPTION)
#ACC LEV 1
 the level-0 (THE MISSION LEVEL) attribute 0 (FUEL CONSUMPTION)
 | 1 |
 the level-0 (THE MISSION LEVEL) attribute 1 (DIVERSION)
 | 0 |
 the level-0 (THE MISSION LEVEL) attribute 2 (SAFETY)
 | 0 |

accomplishment level 2 (DIVERSION (010)): array product 0

 the level-0 (THE MISSION LEVEL) attribute 0 (FUEL CONSUMPTION)
#ACC LEV 2
 the level-0 (THE MISSION LEVEL) attribute 0 (FUEL CONSUMPTION)
 | 0 |
 the level-0 (THE MISSION LEVEL) attribute 1 (DIVERSION)
```

```
| 1 |
the level-0 (THE MISSION LEVEL) attribute 2 (SAFETY)
| 0 |

accomplishment level 3 (BAD FUEL EFFICIENCY AND DIVERSION (011)): array product 0

 the level-0 (THE MISSION LEVEL) attribute 0 (FUEL CONSUMPTION)
#ACC LEV 3
 the level-0 (THE MISSION LEVEL) attribute 0 (FUEL CONSUMPTION)
 | 1 |
 the level-0 (THE MISSION LEVEL) attribute 1 (DIVERSION)
 | 1 |
 the level-0 (THE MISSION LEVEL) attribute 2 (SAFETY)
 | 0 |

accomplishment level 4 (1 * * (CRASH)): array product 0

 the level-0 (THE MISSION LEVEL) attribute 0 (FUEL CONSUMPTION)
#ACC LEV 4
 the level-0 (THE MISSION LEVEL) attribute 0 (FUEL CONSUMPTION)
 | * |
 the level-0 (THE MISSION LEVEL) attribute 1 (DIVERSION)
 | * |
 the level-0 (THE MISSION LEVEL) attribute 2 (SAFETY)
 | 1 |
the inverse capability function is already reduced

for each value that each level-0 (THE MISSION LEVEL) attribute and phase can
assume, enter the number of array products to be
input for that value, phase, and attribute.
if an attribute and phase is to be basic, enter BASIC.

level-0 (THE MISSION LEVEL) attribute 0 (FUEL CONSUMPTION)
     phase 0 (MISSION)
   value = 0:
#: 1
   value = 1:
#: 2
level-0 (THE MISSION LEVEL) attribute 1 (DIVERSION)
     phase 0 (MISSION)
   value = 0:
#: 2
   value = 1:
#: 1
level-0 (THE MISSION LEVEL) attribute 2 (SAFETY)
     phase 0 (MISSION)
   value = 0:
#: 4
   value = 1:
#: 3


interlevel translation array product specification

will you want consistency checked?
#: YES

will you want totalness checked?
#: YES

will you want reduction?
#: YES

enter each array product corresponding to each combination of
  level-0 (THE MISSION LEVEL) attribute, phase, and value

  level-0 (THE MISSION LEVEL) attribute 0 (FUEL CONSUMPTION)
   phase 0 (MISSION)
   value = 0
      array product 0

 the level-1 (THE AIRCRAFT FUNCTIONAL TASK LEVEL) attribute 0 (ACTIVE CONTROL)
#FUEL CONSUMPTION=0
 the level-1 (THE AIRCRAFT FUNCTIONAL TASK LEVEL) attribute 0 (ACTIVE CONTROL)
| {0,1} {0,1} |
```

```
 the level-1 (THE AIRCRAFT FUNCTIONAL TASK LEVEL) attribute 1 (WEATHER)
| •     •  |

  level-0 (THE MISSION LEVEL) attribute 0 (FUEL CONSUMPTION)
   phase 0 (MISSION)
   value = 1
      array product 0

 the level-1 (THE AIRCRAFT FUNCTIONAL TASK LEVEL) attribute 0 (ACTIVE CONTROL)
#FUEL CONSUMPTION=1
 the level-1 (THE AIRCRAFT FUNCTIONAL TASK LEVEL) attribute 0 (ACTIVE CONTROL)
| {2,3} • |
 the level-1 (THE AIRCRAFT FUNCTIONAL TASK LEVEL) attribute 1 (WEATHER)
|  •   • |

  level-0 (THE MISSION LEVEL) attribute 0 (FUEL CONSUMPTION)
   phase 0 (MISSION)
   value = 1
      array product 1

 the level-1 (THE AIRCRAFT FUNCTIONAL TASK LEVEL) attribute 0 (ACTIVE CONTROL)
| {0,1} {2,3} |
 the level-1 (THE AIRCRAFT FUNCTIONAL TASK LEVEL) attribute 1 (WEATHER)
|  •     •  |

  level-0 (THE MISSION LEVEL) attribute 1 (DIVERSION)
   phase 0 (MISSION)
   value = 0
      array product 0

 the level-1 (THE AIRCRAFT FUNCTIONAL TASK LEVEL) attribute 0 (ACTIVE CONTROL)
#DIVERSION=0
 the level-1 (THE AIRCRAFT FUNCTIONAL TASK LEVEL) attribute 0 (ACTIVE CONTROL)
| 0 • |
 the level-1 (THE AIRCRAFT FUNCTIONAL TASK LEVEL) attribute 1 (WEATHER)
| • • |

  level-0 (THE MISSION LEVEL) attribute 1 (DIVERSION)
   phase 0 (MISSION)
   value = 0
      array product 1

 the level-1 (THE AIRCRAFT FUNCTIONAL TASK LEVEL) attribute 0 (ACTIVE CONTROL)
| {1,2,3} • |
 the level-1 (THE AIRCRAFT FUNCTIONAL TASK LEVEL) attribute 1 (WEATHER)
|   0    • |

  level-0 (THE MISSION LEVEL) attribute 1 (DIVERSION)
   phase 0 (MISSION)
   value = 1
      array product 0

 the level-1 (THE AIRCRAFT FUNCTIONAL TASK LEVEL) attribute 0 (ACTIVE CONTROL)
#DIVERSION=1
 the level-1 (THE AIRCRAFT FUNCTIONAL TASK LEVEL) attribute 0 (ACTIVE CONTROL)
| {1,2,3} • |
 the level-1 (THE AIRCRAFT FUNCTIONAL TASK LEVEL) attribute 1 (WEATHER)
|   1    • |

  level-0 (THE MISSION LEVEL) attribute 2 (SAFETY)
   phase 0 (MISSION)
   value = 0
      array product 0

 the level-1 (THE AIRCRAFT FUNCTIONAL TASK LEVEL) attribute 0 (ACTIVE CONTROL)
#SAFETY=0
 the level-1 (THE AIRCRAFT FUNCTIONAL TASK LEVEL) attribute 0 (ACTIVE CONTROL)
| 1 • |
 the level-1 (THE AIRCRAFT FUNCTIONAL TASK LEVEL) attribute 1 (WEATHER)
| • • |

  level-0 (THE MISSION LEVEL) attribute 2 (SAFETY)
   phase 0 (MISSION)
   value = 0
      array product 1
```

```
 the level-1 (THE AIRCRAFT FUNCTIONAL TASK LEVEL) attribute 0 (ACTIVE CONTROL)
| 0 {0,2} |
 the level-1 (THE AIRCRAFT FUNCTIONAL TASK LEVEL) attribute 1 (WEATHER)
| •  •  |

  level-0 (THE MISSION LEVEL) attribute 2 (SAFETY)
   phase 0 (MISSION)
   value = 0
      array product 2

 the level-1 (THE AIRCRAFT FUNCTIONAL TASK LEVEL) attribute 0 (ACTIVE CONTROL)
| 0 {1,3} |
 the level-1 (THE AIRCRAFT FUNCTIONAL TASK LEVEL) attribute 1 (WEATHER)
| 0  •  |

  level-0 (THE MISSION LEVEL) attribute 2 (SAFETY)
   phase 0 (MISSION)
   value = 0
      array product 3

 the level-1 (THE AIRCRAFT FUNCTIONAL TASK LEVEL) attribute 0 (ACTIVE CONTROL)
| 2 {0,1} |
 the level-1 (THE AIRCRAFT FUNCTIONAL TASK LEVEL) attribute 1 (WEATHER)
| •  •  |

  level-0 (THE MISSION LEVEL) attribute 2 (SAFETY)
   phase 0 (MISSION)
   value = 1
      array product 0

 the level-1 (THE AIRCRAFT FUNCTIONAL TASK LEVEL) attribute 0 (ACTIVE CONTROL)
#SAFETY=1
 the level-1 (THE AIRCRAFT FUNCTIONAL TASK LEVEL) attribute 0 (ACTIVE CONTROL)
| 3 • |
 the level-1 (THE AIRCRAFT FUNCTIONAL TASK LEVEL) attribute 1 (WEATHER)
| • • |

  level-0 (THE MISSION LEVEL) attribute 2 (SAFETY)
   phase 0 (MISSION)
   value = 1
      array product 1

 the level-1 (THE AIRCRAFT FUNCTIONAL TASK LEVEL) attribute 0 (ACTIVE CONTROL)
| 2 {2,3} |
 the level-1 (THE AIRCRAFT FUNCTIONAL TASK LEVEL) attribute 1 (WEATHER)
| •  •  |

  level-0 (THE MISSION LEVEL) attribute 2 (SAFETY)
   phase 0 (MISSION)
   value = 1
      array product 2

 the level-1 (THE AIRCRAFT FUNCTIONAL TASK LEVEL) attribute 0 (ACTIVE CONTROL)
| 0 {1,3} |
 the level-1 (THE AIRCRAFT FUNCTIONAL TASK LEVEL) attribute 1 (WEATHER)
| 1  •  |
the inverse interlevel translation is already reduced

do you want a list of the array products for the capability function?
#: NO
recommended commands are checkpoint, next, or
#: next

getting the name of the next level (level 2):
What is the name of model hierarchy level-2?
THE COMPUTATIONAL TASK LEVEL

number of level 2 (THE COMPUTATIONAL TASK LEVEL ) attributes?
#: 2

getting the name of each level 2 (THE COMPUTATIONAL TASK LEVEL ) attribute:
What is the name of attribute 0?
FUEL REGULATION COMPUTATIONS
What is the name of attribute 1?
```

```
AUTOLAND COMPUTATIONS

number of values per level 2 (THE COMPUTATIONAL TASK LEVEL ) attribute?
#: 2 2

number of level 2 (THE COMPUTATIONAL TASK LEVEL ) phases?
#: 3
getting the name of each level 2 (THE COMPUTATIONAL TASK LEVEL ) phase:
What is the name of phase 0?
CRUISE I
What is the name of phase 1?
CRUISE II
What is the name of phase 2?
APPROACH AND LANDING

for each value that each level-1 (THE AIRCRAFT FUNCTIONAL TASK LEVEL) attribute and phase can
assume, enter the number of array products to be
input for that value, phase, and attribute.
if an attribute and phase is to be basic, enter BASIC.

level-1 (THE AIRCRAFT FUNCTIONAL TASK LEVEL) attribute 0 (ACTIVE CONTROL)
     phase 0 (TAKEOFF AND CRUISE)
   value = 0:
#: 1
   value = 1:
#: 1
   value = 2:
#: 2
   value = 3:
#: 1
level-1 (THE AIRCRAFT FUNCTIONAL TASK LEVEL) attribute 0 (ACTIVE CONTROL)
     phase 1 (LANDING)
   value = 0:
#: 1
   value = 1:
#: 1
   value = 2:
#: 1
   value = 3:
#: 1
level-1 (THE AIRCRAFT FUNCTIONAL TASK LEVEL) attribute 1 (WEATHER)
     phase 0 (TAKEOFF AND CRUISE)
   value = 0:
#: level-1 (THE AIRCRAFT FUNCTIONAL TASK LEVEL) attribute 1 (WEATHER)
     phase 1 (LANDING)
   value = 0:
#:

interlevel translation array product specification

will you want consistency checked?
#: YES

will you want totalness checked?
#: YES

will you want reduction?
#: YES

enter each array product corresponding to each combination of
  level-1 (THE AIRCRAFT FUNCTIONAL TASK LEVEL) attribute, phase, and value

  level-1 (THE AIRCRAFT FUNCTIONAL TASK LEVEL) attribute 0 (ACTIVE CONTROL)
   phase 0 (TAKEOFF AND CRUISE)
   value = 0
      array product 0

 the level-2 (THE COMPUTATIONAL TASK LEVEL ) attribute 0 (FUEL REGULATION COMPUTATIONS)
# Y(11)=0
 the level-2 (THE COMPUTATIONAL TASK LEVEL ) attribute 0 (FUEL REGULATION COMPUTATIONS)
| 0 0 * |
 the level-2 (THE COMPUTATIONAL TASK LEVEL ) attribute 1 (AUTOLAND COMPUTATIONS)
| * 0 * |

  level-1 (THE AIRCRAFT FUNCTIONAL TASK LEVEL) attribute 0 (ACTIVE CONTROL)
```

```
  phase 0 (TAKEOFF AND CRUISE)
  value = 1
     array product 0

 the level-2 (THE COMPUTATIONAL TASK LEVEL ) attribute 0 (FUEL REGULATION COMPUTATIONS)
# Y(11)=1
 the level-2 (THE COMPUTATIONAL TASK LEVEL ) attribute 0 (FUEL REGULATION COMPUTATIONS)
| 0 0 * |
 the level-2 (THE COMPUTATIONAL TASK LEVEL ) attribute 1 (AUTOLAND COMPUTATIONS)
| * 1 * |

  level-1 (THE AIRCRAFT FUNCTIONAL TASK LEVEL) attribute 0 (ACTIVE CONTROL)
  phase 0 (TAKEOFF AND CRUISE)
  value = 2
     array product 0

 the level-2 (THE COMPUTATIONAL TASK LEVEL ) attribute 0 (FUEL REGULATION COMPUTATIONS)
# Y(11)=2, TRAJ 1
 the level-2 (THE COMPUTATIONAL TASK LEVEL ) attribute 0 (FUEL REGULATION COMPUTATIONS)
| 1 0 * |
 the level-2 (THE COMPUTATIONAL TASK LEVEL ) attribute 1 (AUTOLAND COMPUTATIONS)
| * * * |

  level-1 (THE AIRCRAFT FUNCTIONAL TASK LEVEL) attribute 0 (ACTIVE CONTROL)
  phase 0 (TAKEOFF AND CRUISE)
  value = 2
     array product 1

 the level-2 (THE COMPUTATIONAL TASK LEVEL ) attribute 0 (FUEL REGULATION COMPUTATIONS)
#Y(11)=2, TRAJ 2
 the level-2 (THE COMPUTATIONAL TASK LEVEL ) attribute 0 (FUEL REGULATION COMPUTATIONS)
| 0 1 * |
 the level-2 (THE COMPUTATIONAL TASK LEVEL ) attribute 1 (AUTOLAND COMPUTATIONS)
| * * * |

  level-1 (THE AIRCRAFT FUNCTIONAL TASK LEVEL) attribute 0 (ACTIVE CONTROL)
  phase 0 (TAKEOFF AND CRUISE)
  value = 3
     array product 0

 the level-2 (THE COMPUTATIONAL TASK LEVEL ) attribute 0 (FUEL REGULATION COMPUTATIONS)
# Y(11)=3
 the level-2 (THE COMPUTATIONAL TASK LEVEL ) attribute 0 (FUEL REGULATION COMPUTATIONS)
| 1 1 * |
 the level-2 (THE COMPUTATIONAL TASK LEVEL ) attribute 1 (AUTOLAND COMPUTATIONS)
| * * * |

  level-1 (THE AIRCRAFT FUNCTIONAL TASK LEVEL) attribute 0 (ACTIVE CONTROL)
  phase 1 (LANDING)
  value = 0
     array product 0

 the level-2 (THE COMPUTATIONAL TASK LEVEL ) attribute 0 (FUEL REGULATION COMPUTATIONS)
# Y(12)=4
 the level-2 (THE COMPUTATIONAL TASK LEVEL ) attribute 0 (FUEL REGULATION COMPUTATIONS)
| * * 0 |
 the level-2 (THE COMPUTATIONAL TASK LEVEL ) attribute 1 (AUTOLAND COMPUTATIONS)
| * * 0 |

  level-1 (THE AIRCRAFT FUNCTIONAL TASK LEVEL) attribute 0 (ACTIVE CONTROL)
  phase 1 (LANDING)
  value = 1
     array product 0

 the level-2 (THE COMPUTATIONAL TASK LEVEL ) attribute 0 (FUEL REGULATION COMPUTATIONS)
#Y(12)=5
 the level-2 (THE COMPUTATIONAL TASK LEVEL ) attribute 0 (FUEL REGULATION COMPUTATIONS)
| * * 0 |
 the level-2 (THE COMPUTATIONAL TASK LEVEL ) attribute 1 (AUTOLAND COMPUTATIONS)
| * * 1 |

  level-1 (THE AIRCRAFT FUNCTIONAL TASK LEVEL) attribute 0 (ACTIVE CONTROL)
  phase 1 (LANDING)
  value = 2
     array product 0
```

```
 the level-2 (THE COMPUTATIONAL TASK LEVEL ) attribute 0 (FUEL REGULATION COMPUTATIONS)
# Y(12)=6
 the level-2 (THE COMPUTATIONAL TASK LEVEL ) attribute 0 (FUEL REGULATION COMPUTATIONS)
| * * 1 |
 the level-2 (THE COMPUTATIONAL TASK LEVEL ) attribute 1 (AUTOLAND COMPUTATIONS)
| * * 0 |

  level-1 (THE AIRCRAFT FUNCTIONAL TASK LEVEL) attribute 0 (ACTIVE CONTROL)
   phase 1 (LANDING)
   value = 3
      array product 0

 the level-2 (THE COMPUTATIONAL TASK LEVEL ) attribute 0 (FUEL REGULATION COMPUTATIONS)
# Y(12)=7
 the level-2 (THE COMPUTATIONAL TASK LEVEL ) attribute 0 (FUEL REGULATION COMPUTATIONS)
| * * 1 |
 the level-2 (THE COMPUTATIONAL TASK LEVEL ) attribute 1 (AUTOLAND COMPUTATIONS)
| * * 1 |
the inverse interlevel translation is already reduced

do you want a list of the array products for the capability function?
#: NO
recommended commands are checkpoint, next, or
#: next

getting the name of the next level (level 3):
What is the name of model hierarchy level-3?
THE COMPUTER (BOTTOM) LEVELL

number of level 3 (THE COMPUTER (BOTTOM) LEVELL) attributes?
#: 1

getting the name of each level 3 (THE COMPUTER (BOTTOM) LEVELL) attribute:
What is the name of attribute 0?
HARDWARE STATE

number of values per level 3 (THE COMPUTER (BOTTOM) LEVELL) attribute?
#: 5

number of level 3 (THE COMPUTER (BOTTOM) LEVELL) phases?
#: 3
getting the name of each level 3 (THE COMPUTER (BOTTOM) LEVELL) phase:
What is the name of phase 0?
CRUISE I
What is the name of phase 1?
CRUISE II
What is the name of phase 2?
APPROACH AND LANDING

for each value that each level-2 (THE COMPUTATIONAL TASK LEVEL ) attribute and phase can
assume, enter the number of array products to be
input for that value, phase, and attribute.
if an attribute and phase is to be basic, enter BASIC.

level-2 (THE COMPUTATIONAL TASK LEVEL ) attribute 0 (FUEL REGULATION COMPUTATIONS)
     phase 0 (CRUISE I)
   value = 0:
#: 1
   value = 1:
#: 1
level-2 (THE COMPUTATIONAL TASK LEVEL ) attribute 0 (FUEL REGULATION COMPUTATIONS)
     phase 1 (CRUISE II)
   value = 0:
#: 1
   value = 1:
#: 2
level-2 (THE COMPUTATIONAL TASK LEVEL ) attribute 0 (FUEL REGULATION COMPUTATIONS)
     phase 2 (APPROACH AND LANDING)
   value = 0:
#: 1
   value = 1:
#: 2
level-2 (THE COMPUTATIONAL TASK LEVEL ) attribute 1 (AUTOLAND COMPUTATIONS)
     phase 0 (CRUISE I)
```

```
   value = 0:
#: level-2 (THE COMPUTATIONAL TASK LEVEL ) attribute 1 (AUTOLAND COMPUTATIONS)
     phase 1 (CRUISE II)
   value = 0:
#: 1
   value = 1:
#: 2
level-2 (THE COMPUTATIONAL TASK LEVEL ) attribute 1 (AUTOLAND COMPUTATIONS)
     phase 2 (APPROACH AND LANDING)
   value = 0:
#: 1
   value = 1:
#: 2


interlevel translation array product specification

will you want consistency checked?
#: YES

will you want totalness checked?
#: YES

will you want reduction?
#: YES

enter each array product corresponding to each combination of
  level-2 (THE COMPUTATIONAL TASK LEVEL ) attribute, phase, and value

  level-2 (THE COMPUTATIONAL TASK LEVEL ) attribute 0 (FUEL REGULATION COMPUTATIONS)
   phase 0 (CRUISE I)
   value = 0
      array product 0

 the level-3 (THE COMPUTER (BOTTOM) LEVELL) attribute 0 (HARDWARE STATE)
#X-11 1
 the level-3 (THE COMPUTER (BOTTOM) LEVELL) attribute 0 (HARDWARE STATE)
|{2,3,4} * * |

  level-2 (THE COMPUTATIONAL TASK LEVEL ) attribute 0 (FUEL REGULATION COMPUTATIONS)
   phase 0 (CRUISE I)
   value = 1
      array product 0

 the level-3 (THE COMPUTER (BOTTOM) LEVELL) attribute 0 (HARDWARE STATE)
#X-11 2
 the level-3 (THE COMPUTER (BOTTOM) LEVELL) attribute 0 (HARDWARE STATE)
| {0,1} * * |

  level-2 (THE COMPUTATIONAL TASK LEVEL ) attribute 0 (FUEL REGULATION COMPUTATIONS)
   phase 1 (CRUISE II)
   value = 0
      array product 0

 the level-3 (THE COMPUTER (BOTTOM) LEVELL) attribute 0 (HARDWARE STATE)
#X-12 1
 the level-3 (THE COMPUTER (BOTTOM) LEVELL) attribute 0 (HARDWARE STATE)
| {2,3,4} {2,3,4} * |

  level-2 (THE COMPUTATIONAL TASK LEVEL ) attribute 0 (FUEL REGULATION COMPUTATIONS)
   phase 1 (CRUISE II)
   value = 1
      array product 0

 the level-3 (THE COMPUTER (BOTTOM) LEVELL) attribute 0 (HARDWARE STATE)
#X-12 2 TRAJ1
 the level-3 (THE COMPUTER (BOTTOM) LEVELL) attribute 0 (HARDWARE STATE)
| * {0,1} * |

  level-2 (THE COMPUTATIONAL TASK LEVEL ) attribute 0 (FUEL REGULATION COMPUTATIONS)
   phase 1 (CRUISE II)
   value = 1
      array product 1

the level-3 (THE COMPUTER (BOTTOM) LEVELL) attribute 0 (HARDWARE STATE)
```

```
#X-12 2 TRAJ2
 the level-3 (THE COMPUTER (BOTTOM) LEVELL) attribute 0 (HARDWARE STATE)
| {0,1} {2,3,4} • |

  level-2 (THE COMPUTATIONAL TASK LEVEL ) attribute 0 (FUEL REGULATION COMPUTATIONS)
   phase 2 (APPROACH AND LANDING)
   value = 0
      array product 0

 the level-3 (THE COMPUTER (BOTTOM) LEVELL) attribute 0 (HARDWARE STATE)
#X-13 1
 the level-3 (THE COMPUTER (BOTTOM) LEVELL) attribute 0 (HARDWARE STATE)
| • {4,3} {4,3} |

  level-2 (THE COMPUTATIONAL TASK LEVEL ) attribute 0 (FUEL REGULATION COMPUTATIONS)
   phase 2 (APPROACH AND LANDING)
   value = 1
      array product 0

 the level-3 (THE COMPUTER (BOTTOM) LEVELL) attribute 0 (HARDWARE STATE)
#X-13 2 TRAJ1
 the level-3 (THE COMPUTER (BOTTOM) LEVELL) attribute 0 (HARDWARE STATE)
| • • {0,1,2} |

  level-2 (THE COMPUTATIONAL TASK LEVEL ) attribute 0 (FUEL REGULATION COMPUTATIONS)
   phase 2 (APPROACH AND LANDING)
   value = 1
      array product 1

 the level-3 (THE COMPUTER (BOTTOM) LEVELL) attribute 0 (HARDWARE STATE)
#X-13 2 TRAJ2
 the level-3 (THE COMPUTER (BOTTOM) LEVELL) attribute 0 (HARDWARE STATE)
|• {0,1,2} {4,3} |

  level-2 (THE COMPUTATIONAL TASK LEVEL ) attribute 1 (AUTOLAND COMPUTATIONS)
   phase 1 (CRUISE II)
   value = 0
      array product 0

 the level-3 (THE COMPUTER (BOTTOM) LEVELL) attribute 0 (HARDWARE STATE)
#X-22 1
 the level-3 (THE COMPUTER (BOTTOM) LEVELL) attribute 0 (HARDWARE STATE)
| {3,4} {3,4} • |

  level-2 (THE COMPUTATIONAL TASK LEVEL ) attribute 1 (AUTOLAND COMPUTATIONS)
   phase 1 (CRUISE II)
   value = 1
      array product 0

 the level-3 (THE COMPUTER (BOTTOM) LEVELL) attribute 0 (HARDWARE STATE)
#X-22 2 TRAJ1
 the level-3 (THE COMPUTER (BOTTOM) LEVELL) attribute 0 (HARDWARE STATE)
| {3,4} {0,1,2} • |

  level-2 (THE COMPUTATIONAL TASK LEVEL ) attribute 1 (AUTOLAND COMPUTATIONS)
   phase 1 (CRUISE II)
   value = 1
      array product 1

 the level-3 (THE COMPUTER (BOTTOM) LEVELL) attribute 0 (HARDWARE STATE)
#X-22 2 TRAJ2
 the level-3 (THE COMPUTER (BOTTOM) LEVELL) attribute 0 (HARDWARE STATE)
| {0,1,2} • • |

  level-2 (THE COMPUTATIONAL TASK LEVEL ) attribute 1 (AUTOLAND COMPUTATIONS)
   phase 2 (APPROACH AND LANDING)
   value = 0
      array product 0

 the level-3 (THE COMPUTER (BOTTOM) LEVELL) attribute 0 (HARDWARE STATE)
#X-23 1
 the level-3 (THE COMPUTER (BOTTOM) LEVELL) attribute 0 (HARDWARE STATE)
| • {4,3,2} {4,3,2} |

  level-2 (THE COMPUTATIONAL TASK LEVEL ) attribute 1 (AUTOLAND COMPUTATIONS)
```

```
  phase 2 (APPROACH AND LANDING)
  value = 1
     array product 0

 the level-3 (THE COMPUTER (BOTTOM) LEVELL) attribute 0 (HARDWARE STATE)
#X-23 2 TRAJ1
 the level-3 (THE COMPUTER (BOTTOM) LEVELL) attribute 0 (HARDWARE STATE)
| • {2,3,4} {0,1} |

  level-2 (THE COMPUTATIONAL TASK LEVEL ) attribute 1 (AUTOLAND COMPUTATIONS)
   phase 2 (APPROACH AND LANDING)
   value = 1
      array product 1

 the level-3 (THE COMPUTER (BOTTOM) LEVELL) attribute 0 (HARDWARE STATE)
#X-23 2 TRAJ2
 the level-3 (THE COMPUTER (BOTTOM) LEVELL) attribute 0 (HARDWARE STATE)
|• {0,1} • |
the inverse interlevel translation is already reduced

do you want a list of the array products for the capability function?
#: NO
recommended commands are checkpoint, next, or
#: eval
getting the stochastic model information for attribute 0 (HARDWARE STATE)?

number of stochastic model states for attribute 0 (HARDWARE STATE)?
#: 5
getting the number of stochastic model states corresponding to
value of bottom-level attribute 0 (HARDWARE STATE):
value 0
#: 1
value 1
#: 1
value 2
#: 1
value 3
#: 1
value 4
#: 1
for bottom attribute 0 (HARDWARE STATE),
getting the states corresponding to each value:
value 0 of attribute 0 (HARDWARE STATE)
#: 0
value 1 of attribute 0 (HARDWARE STATE)
#: 1
value 2 of attribute 0 (HARDWARE STATE)
#: 2
value 3 of attribute 0 (HARDWARE STATE)
#: 3
value 4 of attribute 0 (HARDWARE STATE)
#: 4


for bottom level attribute 0 (HARDWARE STATE),
specify the p matrices for each phase, 1 phase at a time

phase 0 (CRUISE I):

what type of p matrix?
#: nfail

enter phase length
#: 2.5
enter component failure rate
#: .001
enter number of component groups
#: 1
enter number of components for each group
#: 4

phase 1 (CRUISE II):

what type of p matrix?
#: nfail
```

```
enter phase length
#: 2.5
enter component failure rate
#: .001
enter number of component groups
#: 1
enter number of components for each group
#: 4

phase 2 (APPROACH AND LANDING):

what type of p matrix?
#: nfail

enter phase length
#: .5
enter component failure rate
#: .001
enter number of component groups
#: 1
enter number of components for each group
#: 4


for bottom level attribute 0 (HARDWARE STATE),
specify the h matrices for each phase, 1 phase at a time

phase 0 (CRUISE I):

what type of h matrix?
#: identity

phase 1 (CRUISE II):

what type of h matrix?
#: identity

for the stochastic model corresponding to
bottom level attribute 0 (HARDWARE STATE)
enter the inital probabilities of each state,
beginning with the lowest numbered state:
#: 1.0 0.0 0.0 0.0 0.0

enter the 2 probabilities for basic variable 0
level-1 (THE AIRCRAFT FUNCTIONAL TASK LEVEL), attribute 1 (WEATHER), phase 0 (TAKEOFF AND CRUISE)
#: 0.981 0.019

enter the 2 probabilities for basic variable 1
level-1 (THE AIRCRAFT FUNCTIONAL TASK LEVEL), attribute 1 (WEATHER), phase 1 (LANDING)
#: 0.0 1.0

enter the 2 probabilities for basic variable 2
level-2 (THE COMPUTATIONAL TASK LEVEL ), attribute 1 (AUTOLAND COMPUTATIONS), phase 0 (CRUISE I)
#: 0.0 1.0


performability for this mission:
 accomplishment level 0 (ALL GOOD (000)): 9.998208e-01

 accomplishment level 1 (BAD FUEL EFFICIENCY (001)): 3.371873e-05

 accomplishment level 2 (DIVERSION (010)): 0.000000e+00

 accomplishment level 3 (BAD FUEL EFFICIENCY AND DIVERSION (011)): 1.449595e-04

 accomplishment level 4 (1 * * (CRASH)): 5.093372e-07

#: exit

bye.
```

APPENDIX H

METAPHOR input data for the SIFT computer example

This appendix contains input to METAPHOR for evaluating the SIFT computer example of Section 4.5.3 [SIFT Computer Example].

```
echo
build
5
ALL GOOD (0000)
ECONOMIC PENALTIES (1000)
OPERATIONAL PENALTIES (•100)
CHANGE IN MISSION PROFILE (••10)
FATALITIES (•••1)
THE MISSION LEVEL
THE AIRCRAFT LEVEL
4
ECONOMICS
OPERATIONS
MISSION PROFILE
SAFETY
2 2 2 2
1
MISSION
9
AIDS
VOR/DME
AIR DATA
INERTIAL
AUTOLAND
ACTIVE FLUTTER CONTROL
ENGINE CONTROL
ATTITUDE CONTROL
WEATHER
2 2 2 2 2 2 2 2 2
4
TAKEOFF AND CRUISE A
CRUISE B
CRUISE C
LANDING
1
1
1
1
4
YES
YES
YES
#THIS IS THE TRAJECTORY SET FOR ACC LEV 0
|0|
|0|
|0|
|0|
#ACC LEV 1
|1|
```

```
|0|
|0|
|0|
#ACC LEV 2
|•|
|1|
|0|
|0|
#ACC LEV 3
|•|
|•|
|1|
|0|
#ACC LEV 4
|1|
|1|
|•|
|1|
#ACC LEV 4
|0|
|0|
|•|
|1|
#ACC LEV 4
|0|
|1|
|•|
|1|
#ACC LEV 4
|1|
|0|
|•|
|1|
NO
1
1
1
1
3
4
4
3
YES
YES
YES
#ECONOMICS=0
|0 0 0 0|
|• • • •|
|• • • •|
|• • • •|
|• • • •|
|• • • •|
|• • • •|
|• • • •|
|• • • •|
#ECONOMICS=1
|(1) (1) (1) (1)|
|• • • •|
|• • • •|
|• • • •|
|• • • •|
|• • • •|
|• • • •|
|• • • •|
|• • • •|
#OPERATION3=0
|• • • •|
|0 0 0 0|
|0 0 0 0|
|• • • •|
|• • • •|
|• • • •|
|• • • •|
|• • • •|
|• • • •|
```

#OPERATIONS=1

```
| • • • • |
| (1) (1) (1) (1) |
| (1) (1) (1) (1) |
| • • • • |
| • • • • |
| • • • • |
| • • • • |
| • • • • |
| • • • • |
```

#MISSION PROFILE=0 TRAJ 1

```
| • • • • |
| • • • • |
| • • • • |
| 0 0 0 • |
| • • 0 • |
| • • • • |
| • • • • |
| • • • • |
| • • • • |
```

#MISSION PROFILE=0 TRAJ 2

```
| • • • • |
| • • 0 • |
| • • 0 • |
| 0 0 1 • |
| • • • • |
| • • • • |
| • • • • |
| • • • • |
| • • 0 • |
```

#MISSION PROFILE=0 TRAJ 3

```
| • • • • |
| • • • • |
| • • • • |
| 0 0 0 • |
| • • 1 • |
| • • • • |
| • • • • |
| • • • • |
| • • 0 • |
```

#MISSION PROFILE=1 TRAJ 1

```
| • • • • |
| • • (1) • |
| • • (1) • |
| 0 0 1 • |
| • • • • |
| • • • • |
| • • • • |
| • • • • |
| • • • • |
```

#MISSION PROFILE=1 TRAJ 2

```
| • • • • |
| • • • • |
| • • • • |
| (1) (1) • • |
| • • • • |
| • • • • |
| • • • • |
| • • • • |
| • • • • |
```

#MISSION PROFILE=1 TRAJ 3

```
| • • • • |
| • • • • |
| • • • • |
| 0 0 0 • |
| • • 1 • |
| • • • • |
| • • • • |
| • • • • |
| • • 1 • |
```

#MISSION PROFILE=1 TRAJ 4

```
| • • • • |
| • • 0 • |
| • • 0 • |
| 0 0 1 • |
```

```
| • • • • |
| • • • • |
| • • • • |
| • • • • |
| • • 1 • |
#SAFETY=0 TRAJ 1
| • • • • |
| • • • • |
| • • • • |
| 0 • • • |
| • • • • |
| 0 0 0 0 |
| 0 0 0 0 |
| 0 0 0 0 |
| • • 0 • |
#SAFETY=0 TRAJ 2
| • • • • |
| • • • • |
| • • • • |
| 0 • (1) • |
| • • (1) • |
| 0 0 0 0 |
| 0 0 0 0 |
| 0 0 0 0 |
| • • 1 • |
#SAFETY=0 TRAJ 3
| • • • • |
| • • • • |
| • • • • |
| 0 • 0 0 |
| • • 0 0 |
| 0 0 0 0 |
| 0 0 0 0 |
| 0 0 0 0 |
| • • 1 • |
#SAFETY=0 TRAJ 4
| • • • • |
| • • • • |
| • • • • |
| 1 • • • |
| • • • • |
| 0 • • • |
| 0 • • • |
| 0 • • • |
| • • • • |
#SAFETY=1 TRAJ 1
| • • • • |
| • • • • |
| • • • • |
| • • • • |
| • • • • |
| (1) • • • |
| (1) • • • |
| (1) • • • |
| • • • • |
#SAFETY=1 TRAJ 2
| • • • • |
| • • • • |
| • • • • |
| 0 • • • |
| • • • • |
| 0 (1) (1) (1) |
| 0 (1) (1) (1) |
| 0 (1) (1) (1) |
| • • • • |
#SAFETY=1 TRAJ 3
| • • • • |
| • • • • |
| • • • • |
| 0 • 0 (1) |
| • • 0 (1) |
| 0 0 0 0 |
| 0 0 0 0 |
| 0 0 0 0 |
| • • 1 • |
```

```
NO
#checkpoint
next
THE COMPUTATIONAL TASK LEVEL
1
COMPUTATIONS
7
8
TAKEOFF
CLIMB
CRUISE I
CRUISE II
CRUISE III
DESCENT
APPROACH
LANDING
# AIDS
1
1
1
1
1
1
1
1
# VOR%DME
1
1
1
1
1
1
1
1
# AIR DATA
1
1
1
1
1
1
2
3
# INERTIAL
1
1
1
1
1
1
1
1
# AUTOLAND
1
1
1
1
1
1
1
1
# ACTIVE FLUTTER CONTROL
1
1
1
1
1
1
1
1
# ENGINE CONTROL
1
1
1
1
```

```
1
1
1
1
# ATTITUDE CONTROL
1
1
1
1
1
1
1
1
# WEATHER  -- JUST NEED 4 OF THESE
BASIC
BASIC
BASIC
BASIC
YES
YES
YES
# AS(1)=0
| 6 6 6 • • • • • |
# AS(1)=1
| (0,1,2,3,4,5) (0,1,2,3,4,5) (0,1,2,3,4,5) • • • • • |
# AS(2)=0
| • • 6 6 • • • • |
# AS(2)=1
| • • (0,1,2,3,4,5) (0,1,2,3,4,5) • • • • |
# AS(3)=0
| • • • 6 6 6 {5,6} • |
#AS(3)=1
| • • • (0,1,2,3,4,5) (0,1,2,3,4,5) (0,1,2,3,4,5) (0,1,2,3,4) • |
# AS(3)=0
| • • • • • • {5,6} {5,6} |
# AS(3)=1
| • • • • • • (0,1,2,3,4) (0,1,2,3,4) |
# VO(1)=0
| {1,2,3,4,5,6} {1,2,3,4,5,6} {1,2,3,4,5,6} • • • • • |
# VO(1)=1
| (0) (0) (0) • • • • • |
# VO(2)=0
| • • {1,2,3,4,5,6} {1,2,3,4,5,6} • • • • |
# VO(2)=1
| • • (0) (0) • • • • |
# VO(3)=0
| • • • {1,2,3,4,5,6} {1,2,3,4,5,6} {1,2,3,4,5,6} {1,2,3,4,5,6} • |
# VO(3)=1
| • • • (0) (0) (0) (0) • |
# VO(4)=0
| • • • • • • {1,2,3,4,5,6} {1,2,3,4,5,6} |
# VO(4)=1
| • • • • • • (0) (0) |
# AD(1)=0
| {1,2,3,4,5,6} {1,2,3,4,5,6} {1,2,3,4,5,6} • • • • • |
# AD(1)=1
| (0) (0) (0) • • • • • |
# AD(2)=0
| • • {1,2,3,4,5,6} {1,2,3,4,5,6} • • • • |
# AD(2)=1
| • • (0) (0) • • • • |
# AD(3)=0
| • • • {1,2,3,4,5,6} {1,2,3,4,5,6} {1,2,3,4,5,6} {1,2,3,4,5,6} • |
# AD(3)=1
| • • • (0) (0) (0) (0) • |
# AD(4)=0
| • • • • • • {4,5,6} {4,5,6} |
| • • • • • • {1,2} 2 |
# AD(4)=1
| • • • • • • {4,5,6} {0,1,2,3} |
| • • • • • • {0,3} • |
| • • • • • • {1,2} {0,1,3,4,5,6} |
# IN(1)=0
| {5,6} {5,6} {5,6} • • • • • |
# IN(1)=1
```

```
| (0,1,2,3,4) (0,1,2,3,4) (0,1,2,3,4) • • • • • |
# IN(2)=0
| • • {5,6} {5,6} • • • • |
# IN(2)=1
| • • (0,1,2,3,4) (0,1,2,3,4) • • • • |
# IN(3)=0
| • • • {5,6} {5,6} {5,6} {3,4,5,6} • |
# IN(3)=1
| • • • (0,1,2,3,4) (0,1,2,3,4) (0,1,2,3,4) (0,1,2) • |
# IN(4)=0
| • • • • • • {3,4,5,6} {1,2,3,4,5,6} |
# IN(4)=1
| • • • • • • (0,1,2) (0) |
# AL(1)=0
| {1,2,3,4,5,6} {1,2,3,4,5,6} {1,2,3,4,5,6} • • • • • |
# AL(1)=1
| (0) (0) (0) • • • • • |
# AL(2)=0
| • • {1,2,3,4,5,6} {1,2,3,4,5,6} • • • • |
# AL(2)=1
| • • (0) (0) • • • • |
# AL(3)=0
| • • • {1,2,3,4,5,6} {1,2,3,4,5,6} {1,2,3,4,5,6} {1,2,3,4,5,6} • |
# AL(3)=1
| • • • (0) (0) (0) (0) • |
# AL(4)=0
| • • • • • • {1,2,3,4,5,6} {1,2,3,4,5,6} |
# AL(4)=1
| • • • • • • (0) (0) |
# AF(1)=0
| {1,2,3,4,5,6} {1,2,3,4,5,6} {1,2,3,4,5,6} • • • • • |
# AF(1)=1
| (0) (0) (0) • • • • • |
# AF(2)=0
| • • {1,2,3,4,5,6} {1,2,3,4,5,6} • • • • |
# AF(2)=1
| • • (0) (0) • • • • |
# AF(3)=0
| • • • {1,2,3,4,5,6} {1,2,3,4,5,6} {1,2,3,4,5,6} {1,2,3,4,5,6} • |
# AF(3)=1
| • • • (0) (0) (0) (0) • |
# AF(4)=0
| • • • • • • {1,2,3,4,5,6} {1,2,3,4,5,6} |
# AF(4)=1
| • • • • • • (0) (0) |
# EC(1)=0
| {1,2,3,4,5,6} {1,2,3,4,5,6} {1,2,3,4,5,6} • • • • • |
# EC(1)=1
| (0) (0) (0) • • • • • |
# EC(2)=0
| • • {1,2,3,4,5,6} {1,2,3,4,5,6} • • • • |
# EC(2)=1
| • • (0) (0) • • • • |
# EC(3)=0
| • • • {1,2,3,4,5,6} {1,2,3,4,5,6} {1,2,3,4,5,6} {1,2,3,4,5,6} • |
# EC(3)=1
| • • • (0) (0) (0) (0) • |
# EC(4)=0
| • • • • • • {2,3,4,5,6} {2,3,4,5,6} |
# EC(4)=1
| • • • • • • (0,1) (0,1) |
# AC(1)=0
| {1,2,3,4,5,6} {1,2,3,4,5,6} {1,2,3,4,5,6} • • • • • |
# AC(1)=1
| (0) (0) (0) • • • • • |
# AC(2)=0
| • • {1,2,3,4,5,6} {1,2,3,4,5,6} • • • • |
# AC(2)=1
| • • (0) (0) • • • • |
# AC(3)=0
| • • • {1,2,3,4,5,6} {1,2,3,4,5,6} {1,2,3,4,5,6} {1,2,3,4,5,6} • |
# AC(3)=1
| • • • (0) (0) (0) (0) • |
# AC(4)=0
| • • • • • • {1,2,3,4,5,6} {1,2,3,4,5,6} |
```

```
# AC(4)==1
| * * * * * * (0) (0) |
NO
# (SASR 4) 2' is represented as 2 herein and
# (SASR 4) 2 is reprseneted as 1

31
1
5
5
5
5
5
5
0
1 7 13 19 25
2 8 14 20 26
3 9 15 21 27
4 10 16 22 28
5 11 17 23 29
6 12 18 24 30
# TAKEOFF
sift
0.0166666666666666666666
6
6
#CLIMB
sift
6
6
#CRUISE I
sift
0.41666666666666666666666666
6
6
#CRUISE II
sift
5
6
6
#CRUISE III
sift
0.41666666666666
6
6
#DESCENT
sift
0.01666666666
6
6
#APPROACH
sift
6
6
#LANDING
sift
0.01666666666666666666
6
6
#TAKEOFF
identity
#CLIMB
identity
#CRUISE I
identity
#CRUISE II
identity
#CRUISE III
identity
#DESCENT
identity
#APPROACH
identity
1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.989 0.011
```

0.989 0.011
0.989 0.011
0,989 0.011

APPENDIX I

Scenario for the dual-dual computer example

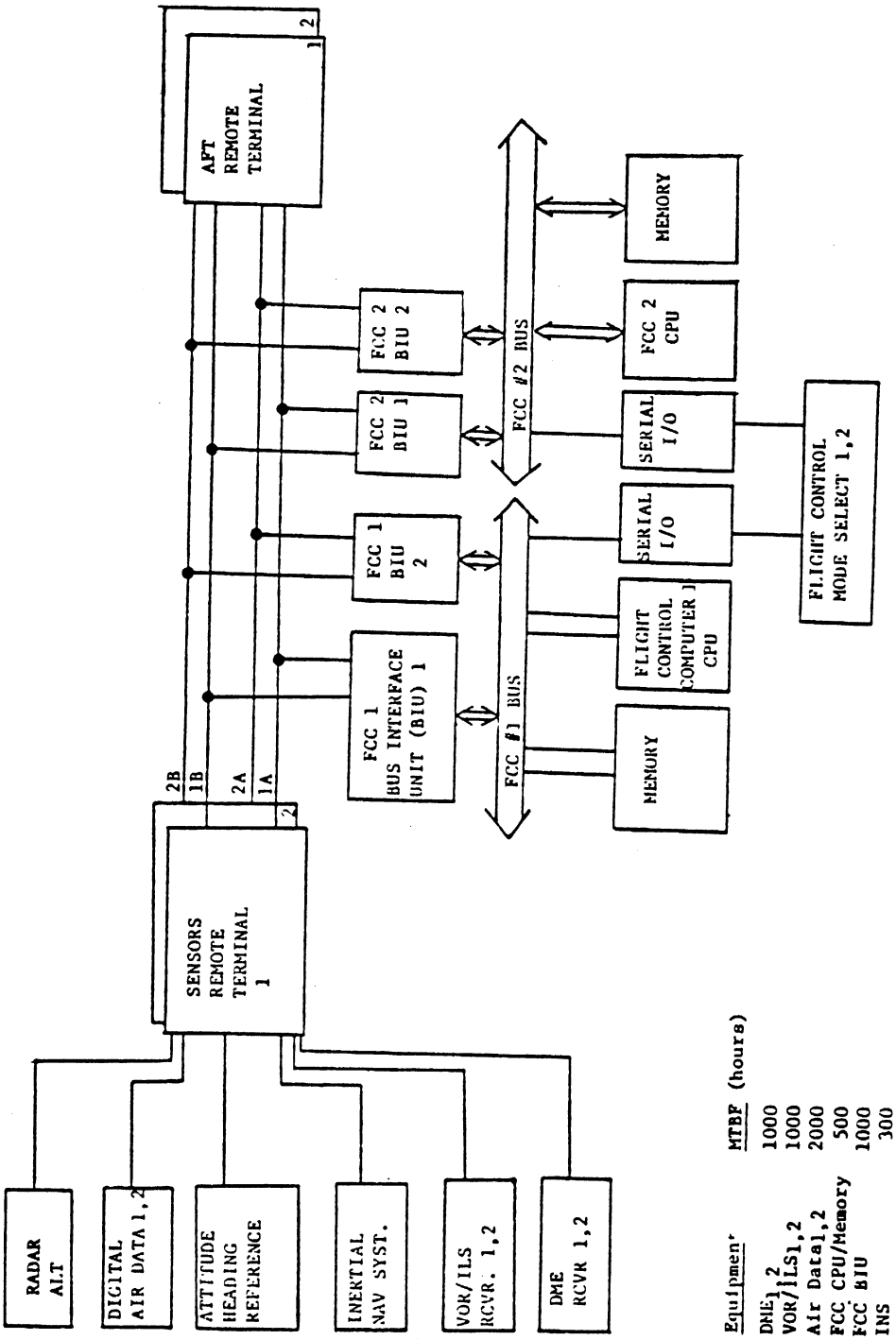
DUAL-DUAL SYSTEM

Figure 8 represents a portion of a digital flight control system which is dual-dual fail-operating. The servo amplifiers and monitor elements and servo sets connected to the actual sensors are not shown to keep the problem within bounds. The sensors are cross-strapped to two remote terminals which convert the sensor signals to digital signals which are tranmitted, on command, over one of the redundant busses for each remote terminal to the flight control computers.

The principal functions to be performed are the state estimation function and the command generation/execution function. Note that a single radar altimeter, attitude heading reference set, and inertial navigation system are carried. Dual-digital air data systems, VOR/ILS receivers and DME receivers are carried and input to both remote terminals. Each remote terminal terminal has a dual redundant bus which interfaces with a bus interface unit that interfaces with the flight control computer bus and hence flight control computer. The dual redundant data bus also interfaces with the remote terminal. In other words, aft remote terminal one and sensor remote terminal one have dual redundant busses 1A and 1B and aft remote terminal two and sensor remote terminal two have busses 2A and 2B. Flight control mode selection is redundant and interfaces with each of the flight control computers through a serial input/output panel.

Scenario

The mission consists of three phases. The first phase is a takeoff/climb phase and is fifteen minutes in duration. The second phase is a cruise phase of forty-five minutes duration. The descent and landing phase consists of fifteen minutes. Assume all equipment is operating at takeoff. During cruise, weather conditions at the scheduled destination develop requiring Category II capability. As stated in FAA Advisory Circular 120-29, Category II conditions require both ILS and glide slope receivers to be operable, the radar altimeter to be operable, both flight control computers to be operable, as well as an attitude reference source such as the attitude heading reference



Equipment	MTBF (hours)
DME 1,2	1000
VOR/ILS1,2	1000
Air Data1,2	2000
FCC CPU/Memory	500
FCC BIU	1000
INS	300
AHR	800
Radar Alt.	700
Remote Terminals	500
Flt. Cntrl Mode Select	2000

FIGURE 8. DUAL-DUAL SYSTEM

or inertial navigation system. Both digital air data systems must also be operable. Table 2 lists the components required for each of the mission phases.

For the purpose of the analysis, the final approach and touchdown phase lasts for two minutes.

Table 2 lists the equipment required for a safe flight, the equipment required to initiate the Category II landing at time equal to 73 minutes, and the equipment required to complete the Category II landing. The analysts calculated the probability of failure to initiate the landing and hence divert to the alternate airport due to loss of equipment required to initiate the landing, probability of successfully landing at the original destination, and probability of loss of the aircraft (unsafe flight) using the data in Figure 8 and Table 2.

At all times, each component is either totally operating or totally failed. The hardware and software associated with detecting component failures and removing failed elements is assumed to be perfectly accurate and perfectly reliable. Failures in each component have an exponential (Poisson) distribution. The Category II Approach and Landing can be aborted any time until T = 75 minutes.

Component	MINIMUM COMPONENT REQUIREMENTS Safe Flight (both phases)	Initiate CAT II Landing (T=73 min)	Complete CAT II Landing (T=75 min)
Radar Alt.		1	1
Digital Air Data	1	2	(1 and 1) or 1
AHRS	1 or 1	1 or 1	
INS			
VOR		2	1
DME		1	
Sensor RT	1	2	1
PU-I	1		1
PU-II		2	
FCMS	1	2	1
Aft RT	1	2	1

where

PU = processing unit
PU-I: one FCC with one associated BIU
PU-II: one FCC with both associated BIUs

TABLE 2. COMPONENT REQUIREMENTS FOR MISSION PERFORMANCE LEVELS

APPENDIX J

METAPHOR input data for the dual-dual computer example

This appendix contains input to METAPHOR for evaluating the dual-dual computer example of Section 4.5.4 [Dual-Dual Example].

```
echo
#brief
build
3
SAFE, DESTINATION
SAFE, ALTERNATE
UNSAFE
THE MISSION LEVEL
THE COMPONENT LEVEL
2
NON-DIVERSION
SAFETY
2 2
1
MISSION
11
RADAR ALTIMETER
DIGITAL AIR DATA
AHRS
INS
VOR
DME
SENSOR RT
FTMP COMPUTER
FCMS
AFT RT
WEATHER
2 3 2 2 3 3 3 6 3 3 2
3
TAKEOFF/CLIMB
CRUISE
APPROACH AND LANDING
1
1
1
YES
YES
YES
#THIS IS THE TRAJECTORY SET FOR ACC SET 0
|0|
|0|
#THIS IS THE TRAJECTORY SET FOR ACC SET 1
|1|
|0|
#THIS IS THE TRAJECTORY SET FOR ACC SET 2
|*|
|1|
3
15
```

```
36
255
YES
YES
YES
# FOR CAT II LANDING =0 (INITIATE)  SET 1
| • • • | RADAR ALT.
| • • • | DAD
| • • • | AHRS
| • • • | INS
| • • • | VOR
| • • • | DME
| • • • | SENSOR RT
| • • • | FTMP COMPUTER
| • • • | FCMS
| • • • | AFT RT
| • 0 • | WEATHER
# FOR CAT II LANDING =0 (INITIATE)  SET 2
| • 1 • | RADAR ALT.
| • 2 • | DAD
| • 1 • | AHRS
| • • • | INS
| • 2 • | VOR
| • {1,2} • | DME
| • 2 • | SENSOR RT
| • 5 • | FTMP COMPUTER
| • 2 • | FCMS
| • 2 • | AFT RT
| • 1 • | WEATHER
# FOR CAT II LANDING =0 (INITIATE)  SET 3
| • 1 • | RADAR ALT.
| • 2 • | DAD
| • 0 • | AHRS
| • 1 • | INS
| • 2 • | VOR
| • {1,2} • | DME
| • 2 • | SENSOR RT
| • 5 • | FTMP COMPUTER
| • 2 • | FCMS
| • 2 • | AFT RT
| • 1 • | WEATHER
# FOR CAT II LANDING =1 (DON'T INITIATE)  SET 1
| • 0 • | RADAR ALT.
| • • • | DAD
| • • • | AHRS
| • • • | INS
| • • • | VOR
| • • • | DME
| • • • | SENSOR RT
| • • • | FTMP COMPUTER
| • • • | FCMS
| • • • | AFT RT
| • 1 • | WEATHER
# FOR CAT II LANDING =1 (DON'T INITIATE)  SET 2
| • 1 • | RADAR ALT.
| • {0,1} • | DAD
| • • • | AHRS
| • • • | INS
| • • • | VOR
| • • • | DME
| • • • | SENSOR RT
| • • • | FTMP COMPUTER
| • • • | FCMS
| • • • | AFT RT
| • 1 • | WEATHER
# FOR CAT II LANDING =1 (DON'T INITIATE)  SET 3
| • 1 • | RADAR ALT.
| • 2 • | DAD
| • 0 • | AHRS
| • 0 • | INS
| • • • | VOR
| • • • | DME
| • • • | SENSOR RT
| • • • | FTMP COMPUTER
| • • • | FCMS
```

```
|• • •| AFT RT
|• 1 •| WEATHER
# FOR CAT II LANDING =1 (DON'T INITIATE) SET 4
|• 1 •| RADAR ALT.
• 2 •| DAD
• 1 •| AHRS
• • •| INS
• {0,1} •| VOR
• • •| DME
• • •| SENSOR RT
• • •| FTMP COMPUTER
• • •| FCMS
• • •| AFT RT
• 1 •| WEATHER
# FOR CAT II LANDING =1 (DON'T INITIATE) SET 5
|• 1 •| RADAR ALT.
• 2 •| DAD
• 0 •| AHRS
• 1 •| INS
• {0,1} •| VOR
• • •| DME
• • •| SENSOR RT
• • •| FTMP COMPUTER
• • •| FCMS
• • •| AFT RT
• 1 •| WEATHER
# FOR CAT II LANDING =1 (DON'T INITIATE) SET 6
|• 1 •| RADAR ALT.
• 2 •| DAD
• 1 •| AHRS
• • •| INS
• 2 •| VOR
• 0 •| DME
• • •| SENSOR RT
• • •| FTMP COMPUTER
• • •| FCMS
• • •| AFT RT
• 1 •| WEATHER
# FOR CAT II LANDING =1 (DON'T INITIATE) SET 7
|• 1 •| RADAR ALT.
• 2 •| DAD
• 0 •| AHRS
• 1 •| INS
• 2 •| VOR
• 0 •| DME
• • •| SENSOR RT
• • •| FTMP COMPUTER
• • •| FCMS
• • •| AFT RT
• 1 •| WEATHER
# FOR CAT II LANDING =1 (DON'T INITIATE) SET 8
|• 1 •| RADAR ALT.
• 2 •| DAD
• 1 •| AHRS
• • •| INS
• 2 •| VOR
• {1,2} •| DME
• {0,1} •| SENSOR RT
• • •| FTMP COMPUTER
• • •| FCMS
• • •| AFT RT
• 1 •| WEATHER
# FOR CAT II LANDING =1 (DON'T INITIATE) SET 9
|• 1 •| RADAR ALT.
• 2 •| DAD
• 0 •| AHRS
• 1 •| INS
• 2 •| VOR
• {1,2} •| DME
• {0,1} •| SENSOR RT
• • •| FTMP COMPUTER
• • •| FCMS
• • •| AFT RT
• 1 •| WEATHER
# FOR CAT II LANDING =1 (DON'T INITIATE) SET 10
```

```
| • 1 • | RADAR ALT.
| • 2 • | DAD
| • 1 • | AHRS
| • • • | INS
| • 2 • | VOR
| • {1,2} • | DME
| • 2 • | SENSOR RT
| • {0,1,2,3,4} • | FTMP COMPUTER
| • • • | FCMS
| • • • | AFT RT
| • 1 • | WEATHER
# FOR CAT II LANDING =1 (DON'T INITIATE) SET 11
| • 1 • | RADAR ALT.
| • 2 • | DAD
| • 0 • | AHRS
| • 1 • | INS
| • 2 • | VOR
| • {1,2} • | DME
| • 2 • | SENSOR RT
| • {0,1,2,3,4} • | FTMP COMPUTER
| • • • | FCMS
| • • • | AFT RT
| • 1 • | WEATHER
# FOR CAT II LANDING =1 (DON'T INITIATE) SET 12
| • 1 • | RADAR ALT.
| • 2 • | DAD
| • 1 • | AHRS
| • • • | INS
| • 2 • | VOR
| • {1,2} • | DME
| • 2 • | SENSOR RT
| • 5 • | FTMP COMPUTER
| • {0,1} • | FCMS
| • • • | AFT RT
| • 1 • | WEATHER
# FOR CAT II LANDING =1 (DON'T INITIATE) SET 13
| • 1 • | RADAR ALT.
| • 2 • | DAD
| • 0 • | AHRS
| • 1 • | INS
| • 2 • | VOR
| • {1,2} • | DME
| • 2 • | SENSOR RT
| • 5 • | FTMP COMPUTER
| • {0,1} • | FCMS
| • • • | AFT RT
| • 1 • | WEATHER
# FOR CAT II LANDING =1 (DON'T INITIATE) SET 14
| • 1 • | RADAR ALT.
| • 2 • | DAD
| • 1 • | AHRS
| • • • | INS
| • 2 • | VOR
| • {1,2} • | DME
| • 2 • | SENSOR RT
| • 5 • | FTMP COMPUTER
| • 2 • | FCMS
| • {0,1} • | AFT RT
| • 1 • | WEATHER
# FOR CAT II LANDING =1 (DON'T INITIATE) SET 15
| • 1 • | RADAR ALT.
| • 2 • | DAD
| • 0 • | AHRS
| • 1 • | INS
| • 2 • | VOR
| • {1,2} • | DME
| • 2 • | SENSOR RT
| • 5 • | FTMP COMPUTER
| • 2 • | FCMS
| • {0,1} • | AFT RT
| • 1 • | WEATHER
# FOR SAFETY=0 SET 1
| • • • | RADAR ALT.
| • • {1,2} | DAD
| • • 1 | AHRS
```

```
| • • • | INS
| • • • | VOR
| • • • | DME
| • • {1,2} | SENSOR RT
| • • {1,2,3,4,5} | FTMP COMPUTER
| • • {1,2} | FCMS
| • • {1,2} | AFT RT
| • 0 • | WEATHER
# FOR SAFETY=0 SET 2
| • • • | RADAR ALT.
| • • {1,2} | DAD
| • • 0 | AHRS
| • • 1 | INS
| • • • | VOR
| • • • | DME
| • • {1,2} | SENSOR RT
| • • {1,2,3,4,5} | FTMP COMPUTER
| • • {1,2} | FCMS
| • • {1,2} | AFT RT
| • 0 • | WEATHER
# FOR SAFETY=0 SET 3
| • 1 1 | RADAR ALT.
| • 2 {1,2} | DAD
| • 1 1 | AHRS
| • • • | INS
| • 2 {1,2} | VOR
| • {1,2} • | DME
| • 2 {1,2} | SENSOR RT
| • 5 {1,2,3,4,5} | FTMP COMPUTER
| • 2 {1,2} | FCMS
| • 2 {1,2} | AFT RT
| • 1 • | WEATHER
# FOR SAFETY=0 SET 4
| • 1 1 | RADAR ALT.
| • 2 {1,2} | DAD
| • 0 1 | AHRS
| • 1 • | INS
| • 2 {1,2} | VOR
| • {1,2} • | DME
| • 2 {1,2} | SENSOR RT
| • 5 {1,2,3,4,5} | FTMP COMPUTER
| • 2 {1,2} | FCMS
| • 2 {1,2} | AFT RT
| • 1 • | WEATHER
# FOR SAFETY=0 SET 5
| • 1 1 | RADAR ALT.
| • 2 {1,2} | DAD
| • 1 0 | AHRS
| • • 1 | INS
| • 2 {1,2} | VOR
| • {1,2} • | DME
| • 2 {1,2} | SENSOR RT
| • 5 {1,2,3,4,5} | FTMP COMPUTER
| • 2 {1,2} | FCMS
| • 2 {1,2} | AFT RT
| • 1 • | WEATHER
# FOR SAFETY=0 SET 6
| • 1 1 | RADAR ALT.
| • 2 {1,2} | DAD
| • 0 0 | AHRS
| • 1 1 | INS
| • 2 {1,2} | VOR
| • {1,2} • | DME
| • 2 {1,2} | SENSOR RT
| • 5 {1,2,3,4,5} | FTMP COMPUTER
| • 2 {1,2} | FCMS
| • 2 {1,2} | AFT RT
| • 1 • | WEATHER
# FOR SAFETY=0 SET 7
| • 0 • | RADAR ALT.
| • • {1,2} | DAD
| • • 1 | AHRS
| • • • | INS
| • • • | VOR
| • • • | DME
```

```
| • • {1,2} | SENSOR RT
| • • {1,2,3,4,5} | FTMP COMPUTER
| • • {1,2} | FCMS
| • • {1,2} | AFT RT
| • 1 • | WEATHER
# FOR SAFETY=0 SET 8
| • 0 • | RADAR ALT.
| • • {1,2} | DAD
| • • 0 | AHRS
| • • 1 | INS
| • • • | VOR
| • • • | DME
| • • {1,2} | SENSOR RT
| • • {1,2,3,4,5} | FTMP COMPUTER
| • • {1,2} | FCMS
| • • {1,2} | AFT RT
| • 1 • | WEATHER
# FOR SAFETY=0 SET 9
| • 1 • | RADAR ALT.
| • {0,1} {1,2} | DAD
| • • 1 | AHRS
| • • • | INS
| • • • | VOR
| • • • | DME
| • • {1,2} | SENSOR RT
| • • {1,2,3,4,5} | FTMP COMPUTER
| • • {1,2} | FCMS
| • • {1,2} | AFT RT
| • 1 • | WEATHER
# FOR SAFETY=0 SET 10
| • 1 • | RADAR ALT.
| • {0,1} {1,2} | DAD
| • • 0 | AHRS
| • • 1 | INS
| • • • | VOR
| • • • | DME
| • • {1,2} | SENSOR RT
| • • {1,2,3,4,5} | FTMP COMPUTER
| • • {1,2} | FCMS
| • • {1,2} | AFT RT
| • 1 • | WEATHER
# FOR SAFETY=0 SET 11
| • 1 • | RADAR ALT.
| • 2 {1,2} | DAD
| • 0 1 | AHRS
| • 0 • | INS
| • • • | VOR
| • • • | DME
| • • {1,2} | SENSOR RT
| • • {1,2,3,4,5} | FTMP COMPUTER
| • • {1,2} | FCMS
| • • {1,2} | AFT RT
| • 1 • | WEATHER
# FOR SAFETY=0 SET 12
| • 1 • | RADAR ALT.
| • 2 {1,2} | DAD
| • 0 0 | AHRS
| • 0 1 | INS
| • • • | VOR
| • • • | DME
| • • {1,2} | SENSOR RT
| • • {1,2,3,4,5} | FTMP COMPUTER
| • • {1,2} | FCMS
| • • {1,2} | AFT RT
| • 1 • | WEATHER
# FOR SAFETY=0 SET 13
| • 1 • | RADAR ALT.
| • 2 {1,2} | DAD
| • 1 1 | AHRS
| • • • | INS
| • {0,1} • | VOR
| • • • | DME
| • • {1,2} | SENSOR RT
| • • {1,2,3,4,5} | FTMP COMPUTER
| • • {1,2} | FCMS
```

```
| • • {1,2} | AFT RT
| • 1 • | WEATHER
# FOR SAFETY=0 SET 14
| • 1 • | RADAR ALT.
| • 2 {1,2} | DAD
| • 1 0 | AHRS
| • • 1 | INS
| • {0,1} • | VOR
| • • • | DME
| • • {1,2} | SENSOR RT
| • • {1,2,3,4,5} | FTMP COMPUTER
| • • {1,2} | FCMS
| • • {1,2} | AFT RT
| • 1 • | WEATHER
# FOR SAFETY=0 SET 15
| • 1 • | RADAR ALT.
| • 2 {1,2} | DAD
| • 0 1 | AHRS
| • 1 • | INS
| • {0,1} • | VOR
| • • • | DME
| • • {1,2} | SENSOR RT
| • • {1,2,3,4,5} | FTMP COMPUTER
| • • {1,2} | FCMS
| • • {1,2} | AFT RT
| • 1 • | WEATHER
# FOR SAFETY=0 SET 16
| • 1 • | RADAR ALT.
| • 2 {1,2} | DAD
| • 0 0 | AHRS
| • 1 1 | INS
| • {0,1} • | VOR
| • • • | DME
| • • {1,2} | SENSOR RT
| • • {1,2,3,4,5} | FTMP COMPUTER
| • • {1,2} | FCMS
| • • {1,2} | AFT RT
| • 1 • | WEATHER
# FOR SAFETY=0 SET 17
| • 1 • | RADAR ALT.
| • 2 {1,2} | DAD
| • 1 1 | AHRS
| • • • | INS
| • 2 • | VOR
| • 0 • | DME
| • • {1,2} | SENSOR RT
| • • {1,2,3,4,5} | FTMP COMPUTER
| • • {1,2} | FCMS
| • • {1,2} | AFT RT
| • 1 • | WEATHER
# FOR SAFETY=0 SET 18
| • 1 • | RADAR ALT.
| • 2 {1,2} | DAD
| • 1 0 | AHRS
| • • 1 | INS
| • 2 • | VOR
| • 0 • | DME
| • • {1,2} | SENSOR RT
| • • {1,2,3,4,5} | FTMP COMPUTER
| • • {1,2} | FCMS
| • • {1,2} | AFT RT
| • 1 • | WEATHER
# FOR SAFETY=0 SET 19
| • 1 • | RADAR ALT.
| • 2 {1,2} | DAD
| • 0 1 | AHRS
| • 1 • | INS
| • 2 • | VOR
| • 0 • | DME
| • • {1,2} | SENSOR RT
| • • {1,2,3,4,5} | FTMP COMPUTER
| • • {1,2} | FCMS
| • • {1,2} | AFT RT
| • 1 • | WEATHER
# FOR SAFETY=0 SET 20
```

| • 1 • | RADAR ALT.
| • 2 {1,2} | DAD
| • 0 0 | AHRS
| • 1 1 | INS
| • 2 • | VOR
| • 0 • | DME
| • • {1,2} | SENSOR RT
| • • {1,2,3,4,5} | FTMP COMPUTER
| • • {1,2} | FCMS
| • • {1,2} | AFT RT
| • 1 • | WEATHER

FOR SAFETY=0 SET 21

| • 1 • | RADAR ALT.
| • 2 {1,2} | DAD
| • 1 1 | AHRS
| • • • | INS
| • 2 • | VOR
| • {1,2} • | DME
| • {0,1} {1,2} | SENSOR RT
| • • {1,2,3,4,5} | FTMP COMPUTER
| • • {1,2} | FCMS
| • • {1,2} | AFT RT
| • 1 • | WEATHER

FOR SAFETY=0 SET 22

| • 1 • | RADAR ALT.
| • 2 {1,2} | DAD
| • 1 0 | AHRS
| • • 1 | INS
| • 2 • | VOR
| • {1,2} • | DME
| • {0,1} {1,2} | SENSOR RT
| • • {1,2,3,4,5} | FTMP COMPUTER
| • • {1,2} | FCMS
| • • {1,2} | AFT RT
| • 1 • | WEATHER

FOR SAFETY=0 SET 23

| • 1 • | RADAR ALT.
| • 2 {1,2} | DAD
| • 0 1 | AHRS
| • 1 • | INS
| • 2 • | VOR
| • {1,2} • | DME
| • {0,1} {1,2} | SENSOR RT
| • • {1,2,3,4,5} | FTMP COMPUTER
| • • {1,2} | FCMS
| • • {1,2} | AFT RT
| • 1 • | WEATHER

FOR SAFETY=0 SET 24

| • 1 • | RADAR ALT.
| • 2 {1,2} | DAD
| • 0 0 | AHRS
| • 1 1 | INS
| • 2 • | VOR
| • {1,2} • | DME
| • {0,1} {1,2} | SENSOR RT
| • • {1,2,3,4,5} | FTMP COMPUTER
| • • {1,2} | FCMS
| • • {1,2} | AFT RT
| • 1 • | WEATHER

FOR SAFETY=0 SET 25

| • 1 • | RADAR ALT.
| • 2 {1,2} | DAD
| • 1 1 | AHRS
| • • • | INS
| • 2 • | VOR
| • {1,2} • | DME
| • 2 {1,2} | SENSOR RT
| • {0,1,2,3,4} {1,2,3,4,5} | FTMP COMPUTER
| • • {1,2} | FCMS
| • • {1,2} | AFT RT
| • 1 • | WEATHER

FOR SAFETY=0 SET 26

| • 1 • | RADAR ALT.
| • 2 {1,2} | DAD
| • 1 0 | AHRS

```
| * * 1 | INS
| * 2 * | VOR
| * {1,2} * | DME
| * 2 {1,2} | SENSOR RT
| * {0,1,2,3,4} {1,2,3,4,5} | FTMP COMPUTER
| * * {1,2} | FCMS
| * * {1,2} | AFT RT
| * 1 * | WEATHER
# FOR SAFETY=0 SET 27
| * 1 * | RADAR ALT.
| * 2 {1,2} | DAD
| * 0 1 | AHRS
| * 1 * | INS
| * 2 * | VOR
| * {1,2} * | DME
| * 2 {1,2} | SENSOR RT
| * {0,1,2,3,4} {1,2,3,4,5} | FTMP COMPUTER
| * * {1,2} | FCMS
| * * {1,2} | AFT RT
| * 1 * | WEATHER
# FOR SAFETY=0 SET 28
| * 1 * | RADAR ALT.
| * 2 {1,2} | DAD
| * 0 0 | AHRS
| * 1 1 | INS
| * 2 * | VOR
| * {1,2} * | DME
| * 2 {1,2} | SENSOR RT
| * {0,1,2,3,4} {1,2,3,4,5} | FTMP COMPUTER
| * * {1,2} | FCMS
| * * {1,2} | AFT RT
| * 1 * | WEATHER
# FOR SAFETY=0 SET 29
| * 1 * | RADAR ALT.
| * 2 {1,2} | DAD
| * 1 1 | AHRS
| * * * | INS
| * 2 * | VOR
| * {1,2} * | DME
| * 2 {1,2} | SENSOR RT
| * 5 {1,2,3,4,5} | FTMP COMPUTER
| * {0,1} {1,2} | FCMS
| * * {1,2} | AFT RT
| * 1 * | WEATHER
# FOR SAFETY=0 SET 30
| * 1 * | RADAR ALT.
| * 2 {1,2} | DAD
| * 1 0 | AHRS
| * * 1 | INS
| * 2 * | VOR
| * {1,2} * | DME
| * 2 {1,2} | SENSOR RT
| * 5 {1,2,3,4,5} | FTMP COMPUTER
| * {0,1} {1,2} | FCMS
| * * {1,2} | AFT RT
| * 1 * | WEATHER
# FOR SAFETY=0 SET 31
| * 1 * | RADAR ALT.
| * 2 {1,2} | DAD
| * 0 1 | AHRS
| * 1 * | INS
| * 2 * | VOR
| * {1,2} * | DME
| * 2 {1,2} | SENSOR RT
| * 5 {1,2,3,4,5} | FTMP COMPUTER
| * {0,1} {1,2} | FCMS
| * * {1,2} | AFT RT
| * 1 * | WEATHER
# FOR SAFETY=0 SET 32
| * 1 * | RADAR ALT.
| * 2 {1,2} | DAD
| * 0 0 | AHRS
| * 1 1 | INS
| * 2 * | VOR
| * {1,2} * | DME
```

```
| • 2 {1,2} | SENSOR RT
| • 5 {1,2,3,4,5} | FTMP COMPUTER
| • {0,1} {1,2} | FCMS
| • • {1,2} | AFT RT
| • 1 • | WEATHER
# FOR SAFETY=0 SET 33
| • 1 • | RADAR ALT.
| • 2 {1,2} | DAD
| • 1 1 | AHRS
| • • • | INS
| • 2 • | VOR
| • {1,2} • | DME
| • 2 {1,2} | SENSOR RT
| • 5 {1,2,3,4,5} | FTMP COMPUTER
| • 2 {1,2} | FCMS
| • {0,1} {1,2} | AFT RT
| • 1 • | WEATHER
# FOR SAFETY=0 SET 34
| • 1 • | RADAR ALT.
| • 2 {1,2} | DAD
| • 1 0 | AHRS
| • • 1 | INS
| • 2 • | VOR
| • {1,2} • | DME
| • 2 {1,2} | SENSOR RT
| • 5 {1,2,3,4,5} | FTMP COMPUTER
| • 2 {1,2} | FCMS
| • {0,1} {1,2} | AFT RT
| • 1 • | WEATHER
# FOR SAFETY=0 SET 35
| • 1 • | RADAR ALT.
| • 2 {1,2} | DAD
| • 0 1 | AHRS
| • 1 • | INS
| • 2 • | VOR
| • {1,2} • | DME
| • 2 {1,2} | SENSOR RT
| • 5 {1,2,3,4,5} | FTMP COMPUTER
| • 2 {1,2} | FCMS
| • {0,1} {1,2} | AFT RT
| • 1 • | WEATHER
# FOR SAFETY=0 SET 36
| • 1 • | RADAR ALT.
| • 2 {1,2} | DAD
| • 0 0 | AHRS
| • 1 1 | INS
| • 2 • | VOR
| • {1,2} • | DME
| • 2 {1,2} | SENSOR RT
| • 5 {1,2,3,4,5} | FTMP COMPUTER
| • 2 {1,2} | FCMS
| • {0,1} {1,2} | AFT RT
| • 1 • | WEATHER
# FOR SAFETY=1 SET 1
#     GOOD WEATHER, THEN CRASH.  10 OF THEM
| • • • | RADAR ALT.
| • • 0 | DAD
| • • • | AHRS
| • • • | INS
| • • • | VOR
| • • • | DME
| • • • | SENSOR RT
| • • • | FTMP COMPUTER
| • • • | FCMS
| • • • | AFT RT
| • 0 • | WEATHER
# FOR SAFETY=1 SET 2
| • • • | RADAR ALT.
| • • {1,2} | DAD
| • • 0 | AHRS
| • • 0 | INS
| • • • | VOR
| • • • | DME
| • • • | SENSOR RT
| • • • | FTMP COMPUTER
```

```
| • • • | FCMS
| • • • | AFT RT
| • 0 • | WEATHER
# FOR SAFETY=1 SET 3
| • • • | RADAR ALT.
| • • {1,2} | DAD
| • • 1 | AHRS
| • • • | INS
| • • • | VOR
| • • • | DME
| • • 0 | SENSOR RT
| • • • | FTMP COMPUTER
| • • • | FCMS
| • • • | AFT RT
| • 0 • | WEATHER
# FOR SAFETY=1 SET 4
| • • • | RADAR ALT.
| • • {1,2} | DAD
| • • 0 | AHRS
| • • 1 | INS
| • • • | VOR
| • • • | DME
| • • 0 | SENSOR RT
| • • • | FTMP COMPUTER
| • • • | FCMS
| • • • | AFT RT
| • 0 • | WEATHER
# FOR SAFETY=1 SET 5
| • • • | RADAR ALT.
| • • {1,2} | DAD
| • • 1 | AHRS
| • • • | INS
| • • • | VOR
| • • • | DME
| • • {1,2} | SENSOR RT
| • • 0 | FTMP COMPUTER
| • • • | FCMS
| • • • | AFT RT
| • 0 • | WEATHER
# FOR SAFETY=1 SET 6
| • • • | RADAR ALT.
| • • {1,2} | DAD
| • • 0 | AHRS
| • • 1 | INS
| • • • | VOR
| • • • | DME
| • • {1,2} | SENSOR RT
| • • 0 | FTMP COMPUTER
| • • • | FCMS
| • • • | AFT RT
| • 0 • | WEATHER
# FOR SAFETY=1 SET 7
| • • • | RADAR ALT.
| • • {1,2} | DAD
| • • 1 | AHRS
| • • • | INS
| • • • | VOR
| • • • | DME
| • • {1,2} | SENSOR RT
| • • {1,2,3,4,5} | FTMP COMPUTER
| • • 0 | FCMS
| • • • | AFT RT
| • 0 • | WEATHER
# FOR SAFETY=1 SET 8
| • • • | RADAR ALT.
| • • {1,2} | DAD
| • • 0 | AHRS
| • • 1 | INS
| • • • | VOR
| • • • | DME
| • • {1,2} | SENSOR RT
| • • {1,2,3,4,5} | FTMP COMPUTER
| • • 0 | FCMS
| • • • | AFT RT
| • 0 • | WEATHER
```

```
# FOR SAFETY=1 SET 9
| • • • | RADAR ALT.
| • • {1,2} | DAD
| • • 1 | AHRS
| • • • | INS
| • • • | VOR
| • • • | DME
| • • {1,2} | SENSOR RT
| • • {1,2,3,4,5} | FTMP COMPUTER
| • • {1,2} | FCMS
| • • 0 | AFT RT
| • 0 • | WEATHER
# FOR SAFETY=1 SET 10
#  BAD WEATHER, NO DIVERSION, CRASH  -- SHOULD BE 26 OF THEM
| • • • | RADAR ALT.
| • • {1,2} | DAD
| • • 0 | AHRS
| • • 1 | INS
| • • • | VOR
| • • • | DME
| • • {1,2} | SENSOR RT
| • • {1,2,3,4,5} | FTMP COMPUTER
| • • {1,2} | FCMS
| • • 0 | AFT RT
| • 0 • | WEATHER
# FOR SAFETY=1 SET 11
| • 1 0 | RADAR ALT.
| • 2 • | DAD
| • 1 • | AHRS
| • • • | INS
| • 2 • | VOR
| • {1,2} • | DME
| • 2 • | SENSOR RT
| • 5 • | FTMP COMPUTER
| • 2 • | FCMS
| • 2 • | AFT RT
| • 1 • | WEATHER
# FOR SAFETY=1 SET 12
| • 1 0 | RADAR ALT.
| • 2 • | DAD
| • 0 • | AHRS
| • 1 • | INS
| • 2 • | VOR
| • {1,2} • | DME
| • 2 • | SENSOR RT
| • 5 • | FTMP COMPUTER
| • 2 • | FCMS
| • 2 • | AFT RT
| • 1 • | WEATHER
# FOR SAFETY=1 SET 13
| • 1 1 | RADAR ALT.
| • 2 0 | DAD
| • 1 • | AHRS
| • • 0 | INS
| • 2 • | VOR
| • {1,2} • | DME
| • 2 • | SENSOR RT
| • 5 • | FTMP COMPUTER
| • 2 • | FCMS
| • 2 • | AFT RT
| • 1 • | WEATHER
# FOR SAFETY=1 SET 14
| • 1 1 | RADAR ALT.
| • 2 0 | DAD
| • 0 • | AHRS
| • 1 0 | INS
| • 2 • | VOR
| • {1,2} • | DME
| • 2 • | SENSOR RT
| • 5 • | FTMP COMPUTER
| • 2 • | FCMS
| • 2 • | AFT RT
| • 1 • | WEATHER
# FOR SAFETY=1 SET 15
| • 1 1 | RADAR ALT.
```

```
| • 2 {1,2} | DAD
| • 1 0 | AHRS
| • • 0 | INS
| • 2 • | VOR
| • {1,2} • | DME
| • 2 • | SENSOR RT
| • 5 • | FTMP COMPUTER
| • 2 • | FCMS
| • 2 • | AFT RT
| • 1 • | WEATHER
# FOR SAFETY=1 SET 16
| • 1 1 | RADAR ALT.
| • 2 {1,2} | DAD
| • 0 0 | AHRS
| • 1 0 | INS
| • 2 • | VOR
| • {1,2} • | DME
| • 2 • | SENSOR RT
| • 5 • | FTMP COMPUTER
| • 2 • | FCMS
| • 2 • | AFT RT
| • 1 • | WEATHER
# FOR SAFETY=1 SET 17
| • 1 1 | RADAR ALT.
| • 2 {1,2} | DAD
| • 1 • | AHRS
| • • 1 | INS
| • 2 0 | VOR
| • {1,2} • | DME
| • 2 • | SENSOR RT
| • 5 • | FTMP COMPUTER
| • 2 • | FCMS
| • 2 • | AFT RT
| • 1 • | WEATHER
# FOR SAFETY=1 SET 18
| • 1 1 | RADAR ALT.
| • 2 {1,2} | DAD
| • 0 • | AHRS
| • 1 1 | INS
| • 2 0 | VOR
| • {1,2} • | DME
| • 2 • | SENSOR RT
| • 5 • | FTMP COMPUTER
| • 2 • | FCMS
| • 2 • | AFT RT
| • 1 • | WEATHER
# FOR SAFETY=1 SET 19
| • 1 1 | RADAR ALT.
| • 2 0 | DAD
| • 1 1 | AHRS
| • • 1 | INS
| • 2 0 | VOR
| • {1,2} • | DME
| • 2 • | SENSOR RT
| • 5 • | FTMP COMPUTER
| • 2 • | FCMS
| • 2 • | AFT RT
| • 1 • | WEATHER
# FOR SAFETY=1 SET 20
| • 1 1 | RADAR ALT.
| • 2 0 | DAD
| • 0 1 | AHRS
| • 1 1 | INS
| • 2 0 | VOR
| • {1,2} • | DME
| • 2 • | SENSOR RT
| • 5 • | FTMP COMPUTER
| • 2 • | FCMS
| • 2 • | AFT RT
| • 1 • | WEATHER
# FOR SAFETY=1 SET 21
| • 1 1 | RADAR ALT.
| • 2 {1,2} | DAD
| • 1 • | AHRS
| • • 1 | INS
```

| • 2 {1,2} | VOR
| • {1,2} • | DME
| • 2 0 | SENSOR RT
| • 5 • | FTMP COMPUTER
| • 2 • | FCMS
| • 2 • | AFT RT
| • 1 • | WEATHER

FOR SAFETY=1 SET 22

| • 1 1 | RADAR ALT.
| • 2 {1,2} | DAD
| • 0 • | AHRS
| • 1 1 | INS
| • 2 {1,2} | VOR
| • {1,2} • | DME
| • 2 0 | SENSOR RT
| • 5 • | FTMP COMPUTER
| • 2 • | FCMS
| • 2 • | AFT RT
| • 1 • | WEATHER

FOR SAFETY=1 SET 23

| • 1 1 | RADAR ALT.
| • 2 0 | DAD
| • 1 1 | AHRS
| • • 1 | INS
| • 2 {1,2} | VOR
| • {1,2} • | DME
| • 2 0 | SENSOR RT
| • 5 • | FTMP COMPUTER
| • 2 • | FCMS
| • 2 • | AFT RT
| • 1 • | WEATHER

FOR SAFETY=1 SET 24

| • 1 1 | RADAR ALT.
| • 2 0 | DAD
| • 0 1 | AHRS
| • 1 1 | INS
| • 2 {1,2} | VOR
| • {1,2} • | DME
| • 2 0 | SENSOR RT
| • 5 • | FTMP COMPUTER
| • 2 • | FCMS
| • 2 • | AFT RT
| • 1 • | WEATHER

FOR SAFETY=1 SET 25

| • 1 1 | RADAR ALT.
| • 2 {1,2} | DAD
| • 1 • | AHRS
| • • 1 | INS
| • 2 {1,2} | VOR
| • {1,2} • | DME
| • 2 {1,2} | SENSOR RT
| • 5 0 | FTMP COMPUTER
| • 2 • | FCMS
| • 2 • | AFT RT
| • 1 • | WEATHER

FOR SAFETY=1 SET 26

| • 1 1 | RADAR ALT.
| • 2 {1,2} | DAD
| • 0 • | AHRS
| • 1 1 | INS
| • 2 {1,2} | VOR
| • {1,2} • | DME
| • 2 {1,2} | SENSOR RT
| • 5 0 | FTMP COMPUTER
| • 2 • | FCMS
| • 2 • | AFT RT
| • 1 • | WEATHER

FOR SAFETY=1 SET 27

| • 1 1 | RADAR ALT.
| • 2 0 | DAD
| • 1 1 | AHRS
| • • 1 | INS
| • 2 {1,2} | VOR
| • {1,2} • | DME
| • 2 {1,2} | SENSOR RT

| • 5 0 | FTMP COMPUTER
| • 2 • | FCMS
| • 2 • | AFT RT
| • 1 • | WEATHER
FOR SAFETY=1 SET 28
| • 1 1 | RADAR ALT.
| • 2 0 | DAD
| • 0 1 | AHRS
| • 1 1 | INS
| • 2 {1,2} | VOR
| • {1,2} • | DME
| • 2 {1,2} | SENSOR RT
| • 5 0 | FTMP COMPUTER
| • 2 • | FCMS
| • 2 • | AFT RT
| • 1 • | WEATHER
FOR SAFETY=1 SET 29
| • 1 1 | RADAR ALT.
| • 2 {1,2} | DAD
| • 1 • | AHRS
| • • 1 | INS
| • 2 {1,2} | VOR
| • {1,2} • | DME
| • 2 {1,2} | SENSOR RT
| • 5 {1,2,3,4,5} | FTMP COMPUTER
| • 2 0 | FCMS
| • 2 • | AFT RT
| • 1 • | WEATHER
FOR SAFETY=1 SET 30
| • 1 1 | RADAR ALT.
| • 2 {1,2} | DAD
| • 0 • | AHRS
| • 1 1 | INS
| • 2 {1,2} | VOR
| • {1,2} • | DME
| • 2 {1,2} | SENSOR RT
| • 5 {1,2,3,4,5} | FTMP COMPUTER
| • 2 0 | FCMS
| • 2 • | AFT RT
| • 1 • | WEATHER
FOR SAFETY=1 SET 31
| • 1 1 | RADAR ALT.
| • 2 0 | DAD
| • 1 1 | AHRS
| • • 1 | INS
| • 2 {1,2} | VOR
| • {1,2} • | DME
| • 2 {1,2} | SENSOR RT
| • 5 {1,2,3,4,5} | FTMP COMPUTER
| • 2 0 | FCMS
| • 2 • | AFT RT
| • 1 • | WEATHER
FOR SAFETY=1 SET 32
| • 1 1 | RADAR ALT.
| • 2 0 | DAD
| • 0 1 | AHRS
| • 1 1 | INS
| • 2 {1,2} | VOR
| • {1,2} • | DME
| • 2 {1,2} | SENSOR RT
| • 5 {1,2,3,4,5} | FTMP COMPUTER
| • 2 0 | FCMS
| • 2 • | AFT RT
| • 1 • | WEATHER
FOR SAFETY=1 SET 33
| • 1 1 | RADAR ALT.
| • 2 {1,2} | DAD
| • 1 • | AHRS
| • • 1 | INS
| • 2 {1,2} | VOR
| • {1,2} • | DME
| • 2 {1,2} | SENSOR RT
| • 5 {1,2,3,4,5} | FTMP COMPUTER
| • 2 {1,2} | FCMS
| • 2 0 | AFT RT

```
| • 1 • | WEATHER
# FOR SAFETY=1 SET 34
| • 1 1 | RADAR ALT.
| • 2 {1,2} | DAD
| • 0 • | AHRS
| • 1 1 | INS
| • 2 {1,2} | VOR
| • {1,2} • | DME
| • 2 {1,2} | SENSOR RT
| • 5 {1,2,3,4,5} | FTMP COMPUTER
| • 2 {1,2} | FCMS
| • 2 0 | AFT RT
| • 1 • | WEATHER
# FOR SAFETY=1 SET 35
| • 1 1 | RADAR ALT.
| • 2 0 | DAD
| • 1 1 | AHRS
| • • 1 | INS
| • 2 {1,2} | VOR
| • {1,2} • | DME
| • 2 {1,2} | SENSOR RT
| • 5 {1,2,3,4,5} | FTMP COMPUTER
| • 2 {1,2} | FCMS
| • 2 0 | AFT RT
| • 1 • | WEATHER
# FOR SAFETY=1 SET 36
| • 1 1 | RADAR ALT.
| • 2 0 | DAD
| • 0 1 | AHRS
| • 1 1 | INS
| • 2 {1,2} | VOR
| • {1,2} • | DME
| • 2 {1,2} | SENSOR RT
| • 5 {1,2,3,4,5} | FTMP COMPUTER
| • 2 {1,2} | FCMS
| • 2 0 | AFT RT
| • 1 1 | WEATHER
#  BAD WEATHER, DIVERSION, CRASH --
#     SHOULD BE 15•2•10=300 OF THEM
#  LET METAPHOR GENERATE AND REDUCE THEM FOR US ...
#  THE FOLLOWING WERE GENERATED BY METAPHOR
#. . . . . .
#
# trajectory set number 0 (out of 218)
#
|      •          0          •     | RADAR ALTIMETER
|      •          •          0     | DIGITAL AIR DATA
|      •          •          •     | AHRS
|      •          •          •     | INS
|      •          •          •     | VOR
|      •          •          •     | DME
|      •          •          •     | SENSOR RT
|      •          •          •     | FTMP COMPUTER
|      •          •          •     | FCMS
|      •          •          •     | AFT RT
|      •          1          •     | WEATHER
#. . . . . .
#
# trajectory set number 1 (out of 218)
#
|      •          0          •     | RADAR ALTIMETER
|      •          •        {1,2}     | DIGITAL AIR DATA
|      •          •          0     | AHRS
|      •          •          0     | INS
|      •          •          •     | VOR
|      •          •          •     | DME
|      •          •          •     | SENSOR RT
|      •          •          •     | FTMP COMPUTER
|      •          •          •     | FCMS
|      •          •          •     | AFT RT
|      •          1          •     | WEATHER
#. . . . . .
#
# trajectory set number 2 (out of 218)
#
```

```
|     •          0          •     | RADAR ALTIMETER
|     •          •        {1,2}     | DIGITAL AIR DATA
|     •          •          0     | AHRS
|     •          •          1     | INS
|     •          •          •     | VOR
|     •          •          •     | DME
|     •          •          0     | SENSOR RT
|     •          •          •     | FTMP COMPUTER
|     •          •          •     | FCMS
|     •          •          •     | AFT RT
|     •          1          •     | WEATHER
#.....................................
#
# trajectory set number 3 (out of 218)
#
|     •          0          •     | RADAR ALTIMETER
|     •          •        {1,2}     | DIGITAL AIR DATA
|     •          •          0     | AHRS
|     •          •          1     | INS
|     •          •          •     | VOR
|     •          •          •     | DME
|     •          •        {1,2}     | SENSOR RT
|     •          •          0     | FTMP COMPUTER
|     •          •          •     | FCMS
|     •          •          •     | AFT RT
|     •          1          •     | WEATHER
#.....................................
#
# trajectory set number 4 (out of 218)
#
|     •          0          •     | RADAR ALTIMETER
|     •          •        {1,2}     | DIGITAL AIR DATA
|     •          •          0     | AHRS
|     •          •          1     | INS
|     •          •          •     | VOR
|     •          •          •     | DME
|     •          •        {1,2}     | SENSOR RT
|     •          •      {1,2,3,4,5} | FTMP COMPUTER
|     •          •          0     | FCMS
|     •          •          •     | AFT RT
|     •          1          •     | WEATHER
#.....................................
#
# trajectory set number 5 (out of 218)
#
|     •          0          •     | RADAR ALTIMETER
|     •          •        {1,2}     | DIGITAL AIR DATA
|     •          •          0     | AHRS
|     •          •          1     | INS
|     •          •          •     | VOR
|     •          •          •     | DME
|     •          •        {1,2}     | SENSOR RT
|     •          •      {1,2,3,4,5} | FTMP COMPUTER
|     •          •        {1,2}     | FCMS
|     •          •          0     | AFT RT
|     •          1          •     | WEATHER
#.....................................
#
# trajectory set number 6 (out of 218)
#
|     •          0          •     | RADAR ALTIMETER
|     •          •        {1,2}     | DIGITAL AIR DATA
|     •          •          1     | AHRS
|     •          •          •     | INS
|     •          •          •     | VOR
|     •          •          •     | DME
|     •          •          0     | SENSOR RT
|     •          •          •     | FTMP COMPUTER
|     •          •          •     | FCMS
|     •          •          •     | AFT RT
|     •          1          •     | WEATHER
#.....................................
#
# trajectory set number 7 (out of 218)
#
```

```
|     •          0          •     | RADAR ALTIMETER
|     •          •        {1,2}     | DIGITAL AIR DATA
|     •          •          1     | AHRS
|     •          •          •     | INS
|     •          •          •     | VOR
|     •          •          •     | DME
|     •          •        {1,2}     | SENSOR RT
|     •          •          0     | FTMP COMPUTER
|     •          •          •     | FCMS
|     •          •          •     | AFT RT
|     •          1          •     | WEATHER
#.............................
#
# trajectory set number 8 (out of 218)
#
|     •          0          •     | RADAR ALTIMETER
|     •          •        {1,2}     | DIGITAL AIR DATA
|     •          •          1     | AHRS
|     •          •          •     | INS
|     •          •          •     | VOR
|     •          •          •     | DME
|     •          •        {1,2}     | SENSOR RT
|     •          •      {1,2,3,4,5} | FTMP COMPUTER
|     •          •          0     | FCMS
|     •          •          •     | AFT RT
|     •          1          •     | WEATHER
#.............................
#
# trajectory set number 9 (out of 218)
#
|     •          0          •     | RADAR ALTIMETER
|     •          •        {1,2}     | DIGITAL AIR DATA
|     •          •          1     | AHRS
|     •          •          •     | INS
|     •          •          •     | VOR
|     •          •          •     | DME
|     •          •        {1,2}     | SENSOR RT
|     •          •      {1,2,3,4,5} | FTMP COMPUTER
|     •          •        {1,2}     | FCMS
|     •          •          0     | AFT RT
|     •          1          •     | WEATHER
#.............................
#
# trajectory set number 10 (out of 218)
#
|     •          1          •     | RADAR ALTIMETER
|     •          2          0     | DIGITAL AIR DATA
|     •          0          •     | AHRS
|     •          0          •     | INS
|     •          •          •     | VOR
|     •          •          •     | DME
|     •          •          •     | SENSOR RT
|     •          •          •     | FTMP COMPUTER
|     •          •          •     | FCMS
|     •          •          •     | AFT RT
|     •          1          •     | WEATHER
#.............................
#
# trajectory set number 11 (out of 218)
#
|     •          1          0     | RADAR ALTIMETER
|     •          2          0     | DIGITAL AIR DATA
|     •          0          •     | AHRS
|     •          1          •     | INS
|     •        {0,1}        •       | VOR
|     •          •          •     | DME
|     •          •          •     | SENSOR RT
|     •          •          •     | FTMP COMPUTER
|     •          •          •     | FCMS
|     •          •          •     | AFT RT
|     •          1          •     | WEATHER
#.............................
#
# trajectory set number 12 (out of 218)
#
```

```
|     •          1          0     | RADAR ALTIMETER
|     •          2          0     | DIGITAL AIR DATA
|     •          0          •     | AHRS
|     •          1          •     | INS
|     •          2          •     | VOR
|     •          0          •     | DME
|     •          •          •     | SENSOR RT
|     •          •          •     | FTMP COMPUTER
|     •          •          •     | FCMS
|     •          •          •     | AFT RT
|     •          1          •     | WEATHER
#.....................................
#
# trajectory set number 13 (out of 218)
#
|     •          1          0     | RADAR ALTIMETER
|     •          2          0     | DIGITAL AIR DATA
|     •          0          •     | AHRS
|     •          1          •     | INS
|     •          2          •     | VOR
|     •        {1,2}          •   |   DME
|     •        {0,1}          •   |   SENSOR RT
|     •          •          •     | FTMP COMPUTER
|     •          •          •     | FCMS
|     •          •          •     | AFT RT
|     •          1          •     | WEATHER
#.....................................
#
# trajectory set number 14 (out of 218)
#
|     •          1          0     | RADAR ALTIMETER
|     •          2          0     | DIGITAL AIR DATA
|     •          0          •     | AHRS
|     •          1          •     | INS
|     •          2          •     | VOR
|     •        {1,2}          •   |   DME
|     •          2          •     | SENSOR RT
|     •      {0,1,2,3,4}        •   | FTMP COMPUTER
|     •          •          •     | FCMS
|     •          •          •     | AFT RT
|     •          1          •     | WEATHER
#.....................................
#
# trajectory set number 15 (out of 218)
#
|     •          1          0     | RADAR ALTIMETER
|     •          2          0     | DIGITAL AIR DATA
|     •          0          •     | AHRS
|     •          1          •     | INS
|     •          2          •     | VOR
|     •        {1,2}          •   |   DME
|     •          2          •     | SENSOR RT
|     •          5          •     | FTMP COMPUTER
|     •        {0,1}          •   |   FCMS
|     •          •          •     | AFT RT
|     •          1          •     | WEATHER
#.....................................
#
# trajectory set number 16 (out of 218)
#
|     •          1          0     | RADAR ALTIMETER
|     •          2          0     | DIGITAL AIR DATA
|     •          0          •     | AHRS
|     •          1          •     | INS
|     •          2          •     | VOR
|     •        {1,2}          •   |   DME
|     •          2          •     | SENSOR RT
|     •          5          •     | FTMP COMPUTER
|     •          2          •     | FCMS
|     •        {0,1}          •   |   AFT RT
|     •          1          •     | WEATHER
#.....................................
#
# trajectory set number 17 (out of 218)
#
```

```
|      •          1          0      | RADAR ALTIMETER
|      •          2          0      | DIGITAL AIR DATA
|      •          1          •      | AHRS
|      •          •          •      | INS
|      •        {0,1}          •    |  | VOR
|      •          •          •      | DME
|      •          •          •      | SENSOR RT
|      •          •          •      | FTMP COMPUTER
|      •          •          •      | FCMS
|      •          •          •      | AFT RT
|      •          1          •      | WEATHER
#.............................
#
# trajectory set number 18 (out of 218)
#
|      •          1          0      | RADAR ALTIMETER
|      •          2          0      | DIGITAL AIR DATA
|      •          1          •      | AHRS
|      •          •          •      | INS
|      •          2          •      | VOR
|      •          0          •      | DME
|      •          •          •      | SENSOR RT
|      •          •          •      | FTMP COMPUTER
|      •          •          •      | FCMS
|      •          •          •      | AFT RT
|      •          1          •      | WEATHER
#.............................
#
# trajectory set number 19 (out of 218)
#
|      •          1          0      | RADAR ALTIMETER
|      •          2          0      | DIGITAL AIR DATA
|      •          1          •      | AHRS
|      •          •          •      | INS
|      •          2          •      | VOR
|      •        {1,2}            •  |  | DME
|      •        {0,1}            •  |  | SENSOR RT
|      •          •          •      | FTMP COMPUTER
|      •          •          •      | FCMS
|      •          •          •      | AFT RT
|      •          1          •      | WEATHER
#.............................
#
# trajectory set number 20 (out of 218)
#
|      •          1          0      | RADAR ALTIMETER
|      •          2          0      | DIGITAL AIR DATA
|      •          1          •      | AHRS
|      •          •          •      | INS
|      •          2          •      | VOR
|      •        {1,2}            •  |  | DME
|      •          2          •      | SENSOR RT
|      •      {0,1,2,3,4}        •  |  | FTMP COMPUTER
|      •          •          •      | FCMS
|      •          •          •      | AFT RT
|      •          1          •      | WEATHER
#.............................
#
# trajectory set number 21 (out of 218)
#
|      •          1          0      | RADAR ALTIMETER
|      •          2          0      | DIGITAL AIR DATA
|      •          1          •      | AHRS
|      •          •          •      | INS
|      •          2          •      | VOR
|      •        {1,2}            •  |  | DME
|      •          2          •      | SENSOR RT
|      •          5          •      | FTMP COMPUTER
|      •        {0,1}            •  |  | FCMS
|      •          •          •      | AFT RT
|      •          1          •      | WEATHER
#.............................
#
# trajectory set number 22 (out of 218)
#
```

```
|      •          1          0      | RADAR ALTIMETER
|      •          2          0      | DIGITAL AIR DATA
|      •          1          •      | AHRS
|      •          •          •      | INS
|      •          2          •      | VOR
|      •        {1,2}          •    |  | DME
|      •          2          •      | SENSOR RT
|      •          5          •      | FTMP COMPUTER
|      •          2          •      | FCMS
|      •        {0,1}          •    |  | AFT RT
|      •          1          •      | WEATHER
#.....................................
#
# trajectory set number 23 (out of 218)
#
|      •          1          •      | RADAR ALTIMETER
|      •          2        {1,2}    |  | DIGITAL AIR DATA
|      •          1          0      | AHRS
|      •          •          0      | INS
|      •        {0,1}          •    |  | VOR
|      •          •          •      | DME
|      •          •          •      | SENSOR RT
|      •          •          •      | FTMP COMPUTER
|      •          •          •      | FCMS
|      •          •          •      | AFT RT
|      •          1          •      | WEATHER
#.....................................
#
# trajectory set number 24 (out of 218)
#
|      •          1          •      | RADAR ALTIMETER
|      •          2        {1,2}    |  | DIGITAL AIR DATA
|      •          1          0      | AHRS
|      •          •          0      | INS
|      •          2          •      | VOR
|      •          0          •      | DME
|      •          •          •      | SENSOR RT
|      •          •          •      | FTMP COMPUTER
|      •          •          •      | FCMS
|      •          •          •      | AFT RT
|      •          1          •      | WEATHER
#.....................................
#
# trajectory set number 25 (out of 218)
#
|      •          1          •      | RADAR ALTIMETER
|      •          2        {1,2}    |  | DIGITAL AIR DATA
|      •          1          0      | AHRS
|      •          •          0      | INS
|      •          2          •      | VOR
|      •        {1,2}          •    |  | DME
|      •        {0,1}          •    |  | SENSOR RT
|      •          •          •      | FTMP COMPUTER
|      •          •          •      | FCMS
|      •          •          •      | AFT RT
|      •          1          •      | WEATHER
#.....................................
#
# trajectory set number 26 (out of 218)
#
|      •          1          •      | RADAR ALTIMETER
|      •          2        {1,2}    |  | DIGITAL AIR DATA
|      •          1          0      | AHRS
|      •          •          0      | INS
|      •          2          •      | VOR
|      •        {1,2}          •    |  | DME
|      •          2          •      | SENSOR RT
|      •      {0,1,2,3,4}      •    |  | FTMP COMPUTER
|      •          •          •      | FCMS
|      •          •          •      | AFT RT
|      •          1          •      | WEATHER
#.....................................
#
# trajectory set number 27 (out of 218)
#
```

```
|     •        1          •     | RADAR ALTIMETER
|     •        2        {1,2}     | DIGITAL AIR DATA
|     •        1          0     | AHRS
|     •        •          0     | INS
|     •        2          •     | VOR
|     •      {1,2}              • | DME
|     •        2          •     | SENSOR RT
|     •        5          •     | FTMP COMPUTER
|     •      {0,1}              • | FCMS
|     •        •          •     | AFT RT
|     •        1          •     | WEATHER
#.....................................
#
# trajectory set number 28 (out of 218)
#
|     •        1          •     | RADAR ALTIMETER
|     •        2        {1,2}     | DIGITAL AIR DATA
|     •        1          0     | AHRS
|     •        •          0     | INS
|     •        2          •     | VOR
|     •      {1,2}              • | DME
|     •        2          •     | SENSOR RT
|     •        5          •     | FTMP COMPUTER
|     •        2          •     | FCMS
|     •      {0,1}              • | AFT RT
|     •        1          •     | WEATHER
#.....................................
#
# trajectory set number 29 (out of 218)
#
|     •        1          •     | RADAR ALTIMETER
|     •        2        {1,2}     | DIGITAL AIR DATA
|     •        1          0     | AHRS
|     •        •          1     | INS
|     •        2          •     | VOR
|     •      {1,2}              • | DME
|     •        2          0     | SENSOR RT
|     •    {0,1,2,3,4}          • | FTMP COMPUTER
|     •        •          •     | FCMS
|     •        •          •     | AFT RT
|     •        1          •     | WEATHER
#.....................................
#
# trajectory set number 30 (out of 218)
#
|     •        1          •     | RADAR ALTIMETER
|     •        2        {1,2}     | DIGITAL AIR DATA
|     •        1          0     | AHRS
|     •        •          1     | INS
|     •        2          •     | VOR
|     •      {1,2}              • | DME
|     •        2          0     | SENSOR RT
|     •        5          •     | FTMP COMPUTER
|     •      {0,1}              • | FCMS
|     •        •          •     | AFT RT
|     •        1          •     | WEATHER
#.....................................
#
# trajectory set number 31 (out of 218)
#
|     •        1          •     | RADAR ALTIMETER
|     •        2        {1,2}     | DIGITAL AIR DATA
|     •        1          0     | AHRS
|     •        •          1     | INS
|     •        2          •     | VOR
|     •      {1,2}              • | DME
|     •        2          0     | SENSOR RT
|     •        5          •     | FTMP COMPUTER
|     •        2          •     | FCMS
|     •      {0,1}              • | AFT RT
|     •        1          •     | WEATHER
#.....................................
#
# trajectory set number 32 (out of 218)
#
```

```
|     •          1          •     | RADAR ALTIMETER
|     •          2        {1,2}     | DIGITAL AIR DATA
|     •          1          0     | AHRS
|     •          •          1     | INS
|     •          2          •     | VOR
|     •        {1,2}          •     | DME
|     •          2        {1,2}     | SENSOR RT
|     •          5          0     | FTMP COMPUTER
|     •        {0,1}          •     | FCMS
|     •          •          •     | AFT RT
|     •          1          •     | WEATHER
#.....
#
# trajectory set number 33 (out of 218)
#
|     •          1          •     | RADAR ALTIMETER
|     •          2        {1,2}     | DIGITAL AIR DATA
|     •          1          0     | AHRS
|     •          •          1     | INS
|     •          2          •     | VOR
|     •        {1,2}          •     | DME
|     •          2        {1,2}     | SENSOR RT
|     •          5          0     | FTMP COMPUTER
|     •          2          •     | FCMS
|     •        {0,1}          •     | AFT RT
|     •          1          •     | WEATHER
#.....
#
# trajectory set number 34 (out of 218)
#
|     •          1          •     | RADAR ALTIMETER
|     •          2        {1,2}     | DIGITAL AIR DATA
|     •          1          0     | AHRS
|     •          •          1     | INS
|     •          2          •     | VOR
|     •        {1,2}          •     | DME
|     •          2        {1,2}     | SENSOR RT
|     •          5      {1,2,3,4,5}   | FTMP COMPUTER
|     •          2          0     | FCMS
|     •        {0,1}          •     | AFT RT
|     •          1          •     | WEATHER
#.....
#
# trajectory set number 35 (out of 218)
#
|     •          1          •     | RADAR ALTIMETER
|     •          2        {1,2}     | DIGITAL AIR DATA
|     •          1          0     | AHRS
|     •          •          1     | INS
|     •          2          •     | VOR
|     •        {1,2}          •     | DME
|     •          2        {1,2}     | SENSOR RT
|     •          5      {1,2,3,4,5}   | FTMP COMPUTER
|     •          2        {1,2}     | FCMS
|     •        {0,1}          0     | AFT RT
|     •          1          •     | WEATHER
#.....
#
# trajectory set number 36 (out of 218)
#
|     •          1          •     | RADAR ALTIMETER
|     •          2        {1,2}     | DIGITAL AIR DATA
|     •          1          0     | AHRS
|     •          •          1     | INS
|     •          2          •     | VOR
|     •        {1,2}          •     | DME
|     •          2        {1,2}     | SENSOR RT
|     •          5      {1,2,3,4,5}   | FTMP COMPUTER
|     •        {0,1}          0     | FCMS
|     •          •          •     | AFT RT
|     •          1          •     | WEATHER
#.....
#
# trajectory set number 37 (out of 218)
#
```

```
|      *          1          *      | RADAR ALTIMETER
|      *          2        {1,2}      | DIGITAL AIR DATA
|      *          1          0      | AHRS
|      *          *          1      | INS
|      *          2          *      | VOR
|      *        {1,2}          *      | DME
|      *          2        {1,2}      | SENSOR RT
|      *          5      {1,2,3,4,5}  | FTMP COMPUTER
|      *        {0,1}        {1,2}      | FCMS
|      *          *          0      | AFT RT
|      *          1          *      | WEATHER
#.....................................
#
# trajectory set number 38 (out of 218)
#
|      *          1          *      | RADAR ALTIMETER
|      *          2        {1,2}      | DIGITAL AIR DATA
|      *          1          0      | AHRS
|      *          *          1      | INS
|      *          2          *      | VOR
|      *        {1,2}          *      | DME
|      *          2        {1,2}      | SENSOR RT
|      *      {0,1,2,3,4}      0      | FTMP COMPUTER
|      *          *          *      | FCMS
|      *          *          *      | AFT RT
|      *          1          *      | WEATHER
#.....................................
#
# trajectory set number 39 (out of 218)
#
|      *          1          *      | RADAR ALTIMETER
|      *          2        {1,2}      | DIGITAL AIR DATA
|      *          1          0      | AHRS
|      *          *          1      | INS
|      *          2          *      | VOR
|      *        {1,2}          *      | DME
|      *          2        {1,2}      | SENSOR RT
|      *      {0,1,2,3,4}  {1,2,3,4,5}  | FTMP COMPUTER
|      *          *          0      | FCMS
|      *          *          *      | AFT RT
|      *          1          *      | WEATHER
#.....................................
#
# trajectory set number 40 (out of 218)
#
|      *          1          *      | RADAR ALTIMETER
|      *          2        {1,2}      | DIGITAL AIR DATA
|      *          1          0      | AHRS
|      *          *          1      | INS
|      *          2          *      | VOR
|      *        {1,2}          *      | DME
|      *          2        {1,2}      | SENSOR RT
|      *      {0,1,2,3,4}  {1,2,3,4,5}  | FTMP COMPUTER
|      *          *        {1,2}      | FCMS
|      *          *          0      | AFT RT
|      *          1          *      | WEATHER
#.....................................
#
# trajectory set number 41 (out of 218)
#
|      *          1          *      | RADAR ALTIMETER
|      *          2        {1,2}      | DIGITAL AIR DATA
|      *          1          0      | AHRS
|      *          *          1      | INS
|      *          2          *      | VOR
|      *        {1,2}          *      | DME
|      *        {0,1}          0      | SENSOR RT
|      *          *          *      | FTMP COMPUTER
|      *          *          *      | FCMS
|      *          *          *      | AFT RT
|      *          1          *      | WEATHER
#.....................................
#
# trajectory set number 42 (out of 218)
#
```

```
|     •          1          •     | RADAR ALTIMETER
|     •          2        {1,2}     | DIGITAL AIR DATA
|     •          1          0     | AHRS
|     •          •          1     | INS
|     •          2          •     | VOR
|     •        {1,2}        •       | DME
|     •        {0,1}      {1,2}       | SENSOR RT
|     •          •          0     | FTMP COMPUTER
|     •          •          •     | FCMS
|     •          •          •     | AFT RT
|     •          1          •     | WEATHER
#.....................................
#
# trajectory set number 43 (out of 218)
#
|     •          1          •     | RADAR ALTIMETER
|     •          2        {1,2}     | DIGITAL AIR DATA
|     •          1          0     | AHRS
|     •          •          1     | INS
|     •          2          •     | VOR
|     •        {1,2}        •       | DME
|     •        {0,1}      {1,2}       | SENSOR RT
|     •          •      {1,2,3,4,5} | FTMP COMPUTER
|     •          •          0     | FCMS
|     •          •          •     | AFT RT
|     •          1          •     | WEATHER
#.....................................
#
# trajectory set number 44 (out of 218)
#
|     •          1          •     | RADAR ALTIMETER
|     •          2        {1,2}     | DIGITAL AIR DATA
|     •          1          0     | AHRS
|     •          •          1     | INS
|     •          2          •     | VOR
|     •        {1,2}        •       | DME
|     •        {0,1}      {1,2}       | SENSOR RT
|     •          •      {1,2,3,4,5} | FTMP COMPUTER
|     •          •        {1,2}       | FCMS
|     •          •          0     | AFT RT
|     •          1          •     | WEATHER
#.....................................
#
# trajectory set number 45 (out of 218)
#
|     •          1          •     | RADAR ALTIMETER
|     •          2        {1,2}     | DIGITAL AIR DATA
|     •          1          0     | AHRS
|     •          •          1     | INS
|     •          2          •     | VOR
|     •          0          •     | DME
|     •          •          0     | SENSOR RT
|     •          •          •     | FTMP COMPUTER
|     •          •          •     | FCMS
|     •          •          •     | AFT RT
|     •          1          •     | WEATHER
#.....................................
#
# trajectory set number 46 (out of 218)
#
|     •          1          •     | RADAR ALTIMETER
|     •          2        {1,2}     | DIGITAL AIR DATA
|     •          1          0     | AHRS
|     •          •          1     | INS
|     •          2          •     | VOR
|     •          0          •     | DME
|     •          •        {1,2}       | SENSOR RT
|     •          •          0     | FTMP COMPUTER
|     •          •          •     | FCMS
|     •          •          •     | AFT RT
|     •          1          •     | WEATHER
#.....................................
#
# trajectory set number 47 (out of 218)
#
```

```
|     •          1          •      | RADAR ALTIMETER
|     •          2        {1,2}      | DIGITAL AIR DATA
|     •          1          0      | AHRS
|     •          •          1      | INS
|     •          2          •      | VOR
|     •          0          •      | DME
|     •          •        {1,2}      | SENSOR RT
|     •          •      {1,2,3,4,5}  | FTMP COMPUTER
|     •          •          0      | FCMS
|     •          •          •      | AFT RT
|     •          1          •      | WEATHER
#.............................
#
# trajectory set number 48 (out of 218)
#
|     •          1          •      | RADAR ALTIMETER
|     •          2        {1,2}      | DIGITAL AIR DATA
|     •          1          0      | AHRS
|     •          •          1      | INS
|     •          2          •      | VOR
|     •          0          •      | DME
|     •          •        {1,2}      | SENSOR RT
|     •          •      {1,2,3,4,5}  | FTMP COMPUTER
|     •          •        {1,2}      | FCMS
|     •          •          0      | AFT RT
|     •          1          •      | WEATHER
#.............................
#
# trajectory set number 49 (out of 218)
#
|     •          1          •      | RADAR ALTIMETER
|     •          2        {1,2}      | DIGITAL AIR DATA
|     •          1          0      | AHRS
|     •          •          1      | INS
|     •        {0,1}          •      | VOR
|     •          •          •      | DME
|     •          •          0      | SENSOR RT
|     •          •          •      | FTMP COMPUTER
|     •          •          •      | FCMS
|     •          •          •      | AFT RT
|     •          1          •      | WEATHER
#.............................
#
# trajectory set number 50 (out of 218)
#
|     •          1          •      | RADAR ALTIMETER
|     •          2        {1,2}      | DIGITAL AIR DATA
|     •          1          0      | AHRS
|     •          •          1      | INS
|     •        {0,1}          •      | VOR
|     •          •          •      | DME
|     •          •        {1,2}      | SENSOR RT
|     •          •          0      | FTMP COMPUTER
|     •          •          •      | FCMS
|     •          •          •      | AFT RT
|     •          1          •      | WEATHER
#.............................
#
# trajectory set number 51 (out of 218)
#
|     •          1          •      | RADAR ALTIMETER
|     •          2        {1,2}      | DIGITAL AIR DATA
|     •          1          0      | AHRS
|     •          •          1      | INS
|     •        {0,1}          •      | VOR
|     •          •          •      | DME
|     •          •        {1,2}      | SENSOR RT
|     •          •      {1,2,3,4,5}  | FTMP COMPUTER
|     •          •          0      | FCMS
|     •          •          •      | AFT RT
|     •          1          •      | WEATHER
#.............................
#
# trajectory set number 52 (out of 218)
#
```

```
|     •          1          •     | RADAR ALTIMETER
|     •          2        {1,2}     | DIGITAL AIR DATA
|     •          1          0     | AHRS
|     •          •          1     | INS
|     •        {0,1}          •     | VOR
|     •          •          •     | DME
|     •          •        {1,2}     | SENSOR RT
|     •          •      {1,2,3,4,5}   | FTMP COMPUTER
|     •          •        {1,2}     | FCMS
|     •          •          0     | AFT RT
|     •          1          •     | WEATHER
# . . . . . .
#
# trajectory set number 53 (out of 218)
#
|     •          1          0     | RADAR ALTIMETER
|     •          2        {1,2}     | DIGITAL AIR DATA
|     •          1          1     | AHRS
|     •          •          •     | INS
|     •          2          •     | VOR
|     •        {1,2}          •     | DME
|     •          2          0     | SENSOR RT
|     •      {0,1,2,3,4}        •     | FTMP COMPUTER
|     •          •          •     | FCMS
|     •          •          •     | AFT RT
|     •          1          •     | WEATHER
# . . . . . .
#
# trajectory set number 54 (out of 218)
#
|     •          1          0     | RADAR ALTIMETER
|     •          2        {1,2}     | DIGITAL AIR DATA
|     •          1          1     | AHRS
|     •          •          •     | INS
|     •          2          •     | VOR
|     •        {1,2}          •     | DME
|     •          2          0     | SENSOR RT
|     •          5          •     | FTMP COMPUTER
|     •        {0,1}          •     | FCMS
|     •          •          •     | AFT RT
|     •          1          •     | WEATHER
# . . . . . .
#
# trajectory set number 55 (out of 218)
#
|     •          1          0     | RADAR ALTIMETER
|     •          2        {1,2}     | DIGITAL AIR DATA
|     •          1          1     | AHRS
|     •          •          •     | INS
|     •          2          •     | VOR
|     •        {1,2}          •     | DME
|     •          2          0     | SENSOR RT
|     •          5          •     | FTMP COMPUTER
|     •          2          •     | FCMS
|     •        {0,1}          •     | AFT RT
|     •          1          •     | WEATHER
# . . . . . .
#
# trajectory set number 56 (out of 218)
#
|     •          1          0     | RADAR ALTIMETER
|     •          2        {1,2}     | DIGITAL AIR DATA
|     •          1          1     | AHRS
|     •          •          •     | INS
|     •          2          •     | VOR
|     •        {1,2}          •     | DME
|     •          2        {1,2}     | SENSOR RT
|     •          5          0     | FTMP COMPUTER
|     •        {0,1}          •     | FCMS
|     •          •          •     | AFT RT
|     •          1          •     | WEATHER
# . . . . . .
#
# trajectory set number 57 (out of 218)
#
```

```
|     •          1          0     | RADAR ALTIMETER
|     •          2        {1,2}     | DIGITAL AIR DATA
|     •          1          1     | AHRS
|     •          •          •     | INS
|     •          2          •     | VOR
|     •        {1,2}          •     | DME
|     •          2        {1,2}     | SENSOR RT
|     •          5          0     | FTMP COMPUTER
|     •          2          •     | FCMS
|     •        {0,1}          •     | AFT RT
|     •          1          •     | WEATHER
# . . . . . .
#
# trajectory set number 58 (out of 218)
#
|     •          1          0     | RADAR ALTIMETER
|     •          2        {1,2}     | DIGITAL AIR DATA
|     •          1          1     | AHRS
|     •          •          •     | INS
|     •          2          •     | VOR
|     •        {1,2}          •     | DME
|     •          2        {1,2}     | SENSOR RT
|     •          5      {1,2,3,4,5}   | FTMP COMPUTER
|     •          2          0     | FCMS
|     •        {0,1}          •     | AFT RT
|     •          1          •     | WEATHER
# . . . . . .
#
# trajectory set number 59 (out of 218)
#
|     •          1          •     | RADAR ALTIMETER
|     •          2        {1,2}     | DIGITAL AIR DATA
|     •          1          1     | AHRS
|     •          •          •     | INS
|     •          2          •     | VOR
|     •        {1,2}          •     | DME
|     •          2        {1,2}     | SENSOR RT
|     •          5      {1,2,3,4,5}   | FTMP COMPUTER
|     •          2        {1,2}     | FCMS
|     •        {0,1}          0     | AFT RT
|     •          1          •     | WEATHER
# . . . . . .
#
# trajectory set number 60 (out of 218)
#
|     •          1          0     | RADAR ALTIMETER
|     •          2        {1,2}     | DIGITAL AIR DATA
|     •          1          1     | AHRS
|     •          •          •     | INS
|     •          2          •     | VOR
|     •        {1,2}          •     | DME
|     •          2        {1,2}     | SENSOR RT
|     •          5      {1,2,3,4,5}   | FTMP COMPUTER
|     •        {0,1}          0     | FCMS
|     •          •          •     | AFT RT
|     •          1          •     | WEATHER
# . . . . . .
#
# trajectory set number 61 (out of 218)
#
|     •          1          •     | RADAR ALTIMETER
|     •          2        {1,2}     | DIGITAL AIR DATA
|     •          1          1     | AHRS
|     •          •          •     | INS
|     •          2          •     | VOR
|     •        {1,2}          •     | DME
|     •          2        {1,2}     | SENSOR RT
|     •          5      {1,2,3,4,5}   | FTMP COMPUTER
|     •        {0,1}        {1,2}       | FCMS
|     •          •          0     | AFT RT
|     •          1          •     | WEATHER
# . . . . . .
#
# trajectory set number 62 (out of 218)
#
```

```
|      •          1          0      | RADAR ALTIMETER
|      •          2        {1,2}      | DIGITAL AIR DATA
|      •          1          1      | AHRS
|      •          •          •      | INS
|      •          2          •      | VOR
|      •        {1,2}          •      | DME
|      •          2        {1,2}      | SENSOR RT
|      •      {0,1,2,3,4}      0      | FTMP COMPUTER
|      •          •          •      | FCMS
|      •          •          •      | AFT RT
|      •          1          •      | WEATHER
#.....
#
# trajectory set number 63 (out of 218)
#
|      •          1          0      | RADAR ALTIMETER
|      •          2        {1,2}      | DIGITAL AIR DATA
|      •          1          1      | AHRS
|      •          •          •      | INS
|      •          2          •      | VOR
|      •        {1,2}          •      | DME
|      •          2        {1,2}      | SENSOR RT
|      •      {0,1,2,3,4}  {1,2,3,4,5}  | FTMP COMPUTER
|      •          •          0      | FCMS
|      •          •          •      | AFT RT
|      •          1          •      | WEATHER
#.....
#
# trajectory set number 64 (out of 218)
#
|      •          1          •      | RADAR ALTIMETER
|      •          2        {1,2}      | DIGITAL AIR DATA
|      •          1          1      | AHRS
|      •          •          •      | INS
|      •          2          •      | VOR
|      •        {1,2}          •      | DME
|      •          2        {1,2}      | SENSOR RT
|      •      {0,1,2,3,4}  {1,2,3,4,5}  | FTMP COMPUTER
|      •          •        {1,2}      | FCMS
|      •          •          0      | AFT RT
|      •          1          •      | WEATHER
#.....
#
# trajectory set number 65 (out of 218)
#
|      •          1          0      | RADAR ALTIMETER
|      •          2        {1,2}      | DIGITAL AIR DATA
|      •          1          1      | AHRS
|      •          •          •      | INS
|      •          2          •      | VOR
|      •        {1,2}          •      | DME
|      •        {0,1}          0      | SENSOR RT
|      •          •          •      | FTMP COMPUTER
|      •          •          •      | FCMS
|      •          •          •      | AFT RT
|      •          1          •      | WEATHER
#.....
#
# trajectory set number 66 (out of 218)
#
|      •          1          0      | RADAR ALTIMETER
|      •          2        {1,2}      | DIGITAL AIR DATA
|      •          1          1      | AHRS
|      •          •          •      | INS
|      •          2          •      | VOR
|      •        {1,2}          •      | DME
|      •        {0,1}        {1,2}      | SENSOR RT
|      •          •          0      | FTMP COMPUTER
|      •          •          •      | FCMS
|      •          •          •      | AFT RT
|      •          1          •      | WEATHER
#.....
#
# trajectory set number 67 (out of 218)
#
```

```
|     *          1          0      | RADAR ALTIMETER
|     *          2        {1,2}      | DIGITAL AIR DATA
|     *          1          1      | AHRS
|     *          *          *      | INS
|     *          2          *      | VOR
|     *        {1,2}        *        | DME
|     *        {0,1}      {1,2}        | SENSOR RT
|     *          *      {1,2,3,4,5}  | FTMP COMPUTER
|     *          *          0      | FCMS
|     *          *          *      | AFT RT
|     *          1          *      | WEATHER
#.............................
#
# trajectory set number 68 (out of 218)
#
|     *          1          *      | RADAR ALTIMETER
|     *          2        {1,2}      | DIGITAL AIR DATA
|     *          1          1      | AHRS
|     *          *          *      | INS
|     *          2          *      | VOR
|     *        {1,2}        *        | DME
|     *        {0,1}      {1,2}        | SENSOR RT
|     *          *      {1,2,3,4,5}  | FTMP COMPUTER
|     *          *        {1,2}      | FCMS
|     *          *          0      | AFT RT
|     *          1          *      | WEATHER
#.............................
#
# trajectory set number 69 (out of 218)
#
|     *          1          0      | RADAR ALTIMETER
|     *          2        {1,2}      | DIGITAL AIR DATA
|     *          1          1      | AHRS
|     *          *          *      | INS
|     *          2          *      | VOR
|     *          0          *      | DME
|     *          *          0      | SENSOR RT
|     *          *          *      | FTMP COMPUTER
|     *          *          *      | FCMS
|     *          *          *      | AFT RT
|     *          1          *      | WEATHER
#.............................
#
# trajectory set number 70 (out of 218)
#
|     *          1          0      | RADAR ALTIMETER
|     *          2        {1,2}      | DIGITAL AIR DATA
|     *          1          1      | AHRS
|     *          *          *      | INS
|     *          2          *      | VOR
|     *          0          *      | DME
|     *          *        {1,2}      | SENSOR RT
|     *          *          0      | FTMP COMPUTER
|     *          *          *      | FCMS
|     *          *          *      | AFT RT
|     *          1          *      | WEATHER
#.............................
#
# trajectory set number 71 (out of 218)
#
|     *          1          0      | RADAR ALTIMETER
|     *          2        {1,2}      | DIGITAL AIR DATA
|     *          1          1      | AHRS
|     *          *          *      | INS
|     *          2          *      | VOR
|     *          0          *      | DME
|     *          *        {1,2}      | SENSOR RT
|     *          *      {1,2,3,4,5}  | FTMP COMPUTER
|     *          *          0      | FCMS
|     *          *          *      | AFT RT
|     *          1          *      | WEATHER
#.............................
#
# trajectory set number 72 (out of 218)
#
```

```
|      *          1          *      | RADAR ALTIMETER
|      *          2        {1,2}      | DIGITAL AIR DATA
|      *          1          1      | AHRS
|      *          *          *      | INS
|      *          2          *      | VOR
|      *          0          *      | DME
|      *          *        {1,2}      | SENSOR RT
|      *          *      {1,2,3,4,5}  | FTMP COMPUTER
|      *          *        {1,2}      | FCMS
|      *          *          0      | AFT RT
|      *          1          *      | WEATHER
#.............................
#
# trajectory set number 73 (out of 218)
#
|      *          1          *      | RADAR ALTIMETER
|      *          2        {1,2}      | DIGITAL AIR DATA
|      *          1          1      | AHRS
|      *          *          *      | INS
|      *        {0,1}          *      | VOR
|      *          *          *      | DME
|      *          *          0      | SENSOR RT
|      *          *          *      | FTMP COMPUTER
|      *          *          *      | FCMS
|      *          *          *      | AFT RT
|      *          1          *      | WEATHER
#.............................
#
# trajectory set number 74 (out of 218)
#
|      *          1          *      | RADAR ALTIMETER
|      *          2        {1,2}      | DIGITAL AIR DATA
|      *          1          1      | AHRS
|      *          *          *      | INS
|      *        {0,1}          *      | VOR
|      *          *          *      | DME
|      *          *        {1,2}      | SENSOR RT
|      *          *          0      | FTMP COMPUTER
|      *          *          *      | FCMS
|      *          *          *      | AFT RT
|      *          1          *      | WEATHER
#.............................
#
# trajectory set number 75 (out of 218)
#
|      *          1          *      | RADAR ALTIMETER
|      *          2        {1,2}      | DIGITAL AIR DATA
|      *          1          1      | AHRS
|      *          *          *      | INS
|      *        {0,1}          *      | VOR
|      *          *          *      | DME
|      *          *        {1,2}      | SENSOR RT
|      *          *      {1,2,3,4,5}  | FTMP COMPUTER
|      *          *          0      | FCMS
|      *          *          *      | AFT RT
|      *          1          *      | WEATHER
#.............................
#
# trajectory set number 76 (out of 218)
#
|      *          1          *      | RADAR ALTIMETER
|      *          2        {1,2}      | DIGITAL AIR DATA
|      *          1          1      | AHRS
|      *          *          *      | INS
|      *        {0,1}          *      | VOR
|      *          *          *      | DME
|      *          *        {1,2}      | SENSOR RT
|      *          *      {1,2,3,4,5}  | FTMP COMPUTER
|      *          *        {1,2}      | FCMS
|      *          *          0      | AFT RT
|      *          1          *      | WEATHER
#.............................
#
# trajectory set number 77 (out of 218)
#
```

```
|     •          1          •     | RADAR ALTIMETER
|     •          2        {1,2}     | DIGITAL AIR DATA
|     •          0          0     | AHRS
|     •          1          0     | INS
|     •        {0,1}          •     | VOR
|     •          •          •     | DME
|     •          •          •     | SENSOR RT
|     •          •          •     | FTMP COMPUTER
|     •          •          •     | FCMS
|     •          •          •     | AFT RT
|     •          1          •     | WEATHER
#.....................................
#
# trajectory set number 78 (out of 218)
#
|     •          1          •     | RADAR ALTIMETER
|     •          2        {1,2}     | DIGITAL AIR DATA
|     •          0          0     | AHRS
|     •          1          0     | INS
|     •          2          •     | VOR
|     •          0          •     | DME
|     •          •          •     | SENSOR RT
|     •          •          •     | FTMP COMPUTER
|     •          •          •     | FCMS
|     •          •          •     | AFT RT
|     •          1          •     | WEATHER
#.....................................
#
# trajectory set number 79 (out of 218)
#
|     •          1          •     | RADAR ALTIMETER
|     •          2        {1,2}     | DIGITAL AIR DATA
|     •          0          0     | AHRS
|     •          1          0     | INS
|     •          2          •     | VOR
|     •        {1,2}          •     | DME
|     •        {0,1}          •     | SENSOR RT
|     •          •          •     | FTMP COMPUTER
|     •          •          •     | FCMS
|     •          •          •     | AFT RT
|     •          1          •     | WEATHER
#.....................................
#
# trajectory set number 80 (out of 218)
#
|     •          1          •     | RADAR ALTIMETER
|     •          2        {1,2}     | DIGITAL AIR DATA
|     •          0          0     | AHRS
|     •          1          0     | INS
|     •          2          •     | VOR
|     •        {1,2}          •     | DME
|     •          2          •     | SENSOR RT
|     •      {0,1,2,3,4}        •     | FTMP COMPUTER
|     •          •          •     | FCMS
|     •          •          •     | AFT RT
|     •          1          •     | WEATHER
#.....................................
#
# trajectory set number 81 (out of 218)
#
|     •          1          •     | RADAR ALTIMETER
|     •          2        {1,2}     | DIGITAL AIR DATA
|     •          0          0     | AHRS
|     •          1          0     | INS
|     •          2          •     | VOR
|     •        {1,2}          •     | DME
|     •          2          •     | SENSOR RT
|     •          5          •     | FTMP COMPUTER
|     •        {0,1}          •     | FCMS
|     •          •          •     | AFT RT
|     •          1          •     | WEATHER
#.....................................
#
# trajectory set number 82 (out of 218)
#
```

```
|     •          1          •     | RADAR ALTIMETER
|     •          2        {1,2}     | DIGITAL AIR DATA
|     •          0          0     | AHRS
|     •          1          0     | INS
|     •          2          •     | VOR
|     •        {1,2}          •     | DME
|     •          2          •     | SENSOR RT
|     •          5          •     | FTMP COMPUTER
|     •          2          •     | FCMS
|     •        {0,1}          •     | AFT RT
|     •          1          •     | WEATHER
#.....................................
#
# trajectory set number 83 (out of 218)
#
|     •          1          •     | RADAR ALTIMETER
|     •          2        {1,2}     | DIGITAL AIR DATA
|     •          0          0     | AHRS
|     •          1          1     | INS
|     •          2          •     | VOR
|     •        {1,2}          •     | DME
|     •          2          0     | SENSOR RT
|     •      {0,1,2,3,4}        •     | FTMP COMPUTER
|     •          •          •     | FCMS
|     •          •          •     | AFT RT
|     •          1          •     | WEATHER
#.....................................
#
# trajectory set number 84 (out of 218)
#
|     •          1          •     | RADAR ALTIMETER
|     •          2        {1,2}     | DIGITAL AIR DATA
|     •          0          0     | AHRS
|     •          1          1     | INS
|     •          2          •     | VOR
|     •        {1,2}          •     | DME
|     •          2          0     | SENSOR RT
|     •          5          •     | FTMP COMPUTER
|     •        {0,1}          •     | FCMS
|     •          •          •     | AFT RT
|     •          1          •     | WEATHER
#.....................................
#
# trajectory set number 85 (out of 218)
#
|     •          1          •     | RADAR ALTIMETER
|     •          2        {1,2}     | DIGITAL AIR DATA
|     •          0          0     | AHRS
|     •          1          1     | INS
|     •          2          •     | VOR
|     •        {1,2}          •     | DME
|     •          2          0     | SENSOR RT
|     •          5          •     | FTMP COMPUTER
|     •          2          •     | FCMS
|     •        {0,1}          •     | AFT RT
|     •          1          •     | WEATHER
#.....................................
#
# trajectory set number 86 (out of 218)
#
|     •          1          •     | RADAR ALTIMETER
|     •          2        {1,2}     | DIGITAL AIR DATA
|     •          0          0     | AHRS
|     •          1          1     | INS
|     •          2          •     | VOR
|     •        {1,2}          •     | DME
|     •          2        {1,2}     | SENSOR RT
|     •          5          0     | FTMP COMPUTER
|     •        {0,1}          •     | FCMS
|     •          •          •     | AFT RT
|     •          1          •     | WEATHER
#.....................................
#
# trajectory set number 87 (out of 218)
#
```

```
|     *          1          *     | RADAR ALTIMETER
|     *          2        {1,2}     | DIGITAL AIR DATA
|     *          0          0     | AHRS
|     *          1          1     | INS
|     *          2          *     | VOR
|     *        {1,2}          *     | DME
|     *          2        {1,2}     | SENSOR RT
|     *          5          0     | FTMP COMPUTER
|     *          2          *     | FCMS
|     *        {0,1}          *     | AFT RT
|     *          1          *     | WEATHER
#.............................
#
# trajectory set number 88 (out of 218)
#
|     *          1          *     | RADAR ALTIMETER
|     *          2        {1,2}     | DIGITAL AIR DATA
|     *          0          0     | AHRS
|     *          1          1     | INS
|     *          2          *     | VOR
|     *        {1,2}          *     | DME
|     *          2        {1,2}     | SENSOR RT
|     *          5      {1,2,3,4,5}   | FTMP COMPUTER
|     *          2          0     | FCMS
|     *        {0,1}          *     | AFT RT
|     *          1          *     | WEATHER
#.............................
#
# trajectory set number 89 (out of 218)
#
|     *          1          *     | RADAR ALTIMETER
|     *          2        {1,2}     | DIGITAL AIR DATA
|     *          0          0     | AHRS
|     *          1          1     | INS
|     *          2          *     | VOR
|     *        {1,2}          *     | DME
|     *          2        {1,2}     | SENSOR RT
|     *          5      {1,2,3,4,5}   | FTMP COMPUTER
|     *          2        {1,2}     | FCMS
|     *        {0,1}          0     | AFT RT
|     *          1          *     | WEATHER
#.............................
#
# trajectory set number 90 (out of 218)
#
|     *          1          *     | RADAR ALTIMETER
|     *          2        {1,2}     | DIGITAL AIR DATA
|     *          0          0     | AHRS
|     *          1          1     | INS
|     *          2          *     | VOR
|     *        {1,2}          *     | DME
|     *          2        {1,2}     | SENSOR RT
|     *          5      {1,2,3,4,5}   | FTMP COMPUTER
|     *        {0,1}          0     | FCMS
|     *          *          *     | AFT RT
|     *          1          *     | WEATHER
#.............................
#
# trajectory set number 91 (out of 218)
#
|     *          1          *     | RADAR ALTIMETER
|     *          2        {1,2}     | DIGITAL AIR DATA
|     *          0          0     | AHRS
|     *          1          1     | INS
|     *          2          *     | VOR
|     *        {1,2}          *     | DME
|     *          2        {1,2}     | SENSOR RT
|     *          5      {1,2,3,4,5}   | FTMP COMPUTER
|     *        {0,1}        {1,2}       | FCMS
|     *          *          0     | AFT RT
|     *          1          *     | WEATHER
#.............................
#
# trajectory set number 92 (out of 218)
#
```

```
|      *          1          *      | RADAR ALTIMETER
|      *          2        {1,2}      | DIGITAL AIR DATA
|      *          0          0      | AHRS
|      *          1          1      | INS
|      *          2          *      | VOR
|      *        {1,2}          *      | DME
|      *          2        {1,2}      | SENSOR RT
|      *      {0,1,2,3,4}      0      | FTMP COMPUTER
|      *          *          *      | FCMS
|      *          *          *      | AFT RT
|      *          1          *      | WEATHER
#.....................................
#
# trajectory set number 93 (out of 218)
#
|      *          1          *      | RADAR ALTIMETER
|      *          2        {1,2}      | DIGITAL AIR DATA
|      *          0          0      | AHRS
|      *          1          1      | INS
|      *          2          *      | VOR
|      *        {1,2}          *      | DME
|      *          2        {1,2}      | SENSOR RT
|      *      {0,1,2,3,4}    {1,2,3,4,5}  | FTMP COMPUTER
|      *          *          0      | FCMS
|      *          *          *      | AFT RT
|      *          1          *      | WEATHER
#.....................................
#
# trajectory set number 94 (out of 218)
#
|      *          1          *      | RADAR ALTIMETER
|      *          2        {1,2}      | DIGITAL AIR DATA
|      *          0          0      | AHRS
|      *          1          1      | INS
|      *          2          *      | VOR
|      *        {1,2}          *      | DME
|      *          2        {1,2}      | SENSOR RT
|      *      {0,1,2,3,4}    {1,2,3,4,5}  | FTMP COMPUTER
|      *          *        {1,2}      | FCMS
|      *          *          0      | AFT RT
|      *          1          *      | WEATHER
#.....................................
#
# trajectory set number 95 (out of 218)
#
|      *          1          *      | RADAR ALTIMETER
|      *          2        {1,2}      | DIGITAL AIR DATA
|      *          0          0      | AHRS
|      *          1          1      | INS
|      *          2          *      | VOR
|      *        {1,2}          *      | DME
|      *        {0,1}          0      | SENSOR RT
|      *          *          *      | FTMP COMPUTER
|      *          *          *      | FCMS
|      *          *          *      | AFT RT
|      *          1          *      | WEATHER
#.....................................
#
# trajectory set number 96 (out of 218)
#
|      *          1          *      | RADAR ALTIMETER
|      *          2        {1,2}      | DIGITAL AIR DATA
|      *          0          0      | AHRS
|      *          1          1      | INS
|      *          2          *      | VOR
|      *        {1,2}          *      | DME
|      *        {0,1}        {1,2}      | SENSOR RT
|      *          *          0      | FTMP COMPUTER
|      *          *          *      | FCMS
|      *          *          *      | AFT RT
|      *          1          *      | WEATHER
#.....................................
#
# trajectory set number 97 (out of 218)
#
```

```
|     •          1          •      | RADAR ALTIMETER
|     •          2        {1,2}      | DIGITAL AIR DATA
|     •          0          0      | AHRS
|     •          1          1      | INS
|     •          2          •      | VOR
|     •        {1,2}        •        | DME
|     •        {0,1}      {1,2}        | SENSOR RT
|     •          •      {1,2,3,4,5}  | FTMP COMPUTER
|     •          •          0      | FCMS
|     •          •          •      | AFT RT
|     •          1          •      | WEATHER
#.............................
#
# trajectory set number 98 (out of 218)
#
|     •          1          •      | RADAR ALTIMETER
|     •          2        {1,2}      | DIGITAL AIR DATA
|     •          0          0      | AHRS
|     •          1          1      | INS
|     •          2          •      | VOR
|     •        {1,2}        •        | DME
|     •        {0,1}      {1,2}        | SENSOR RT
|     •          •      {1,2,3,4,5}  | FTMP COMPUTER
|     •          •        {1,2}      | FCMS
|     •          •          0      | AFT RT
|     •          1          •      | WEATHER
#.............................
#
# trajectory set number 99 (out of 218)
#
|     •          1          •      | RADAR ALTIMETER
|     •          2        {1,2}      | DIGITAL AIR DATA
|     •          0          0      | AHRS
|     •          1          1      | INS
|     •          2          •      | VOR
|     •          0          •      | DME
|     •          •          0      | SENSOR RT
|     •          •          •      | FTMP COMPUTER
|     •          •          •      | FCMS
|     •          •          •      | AFT RT
|     •          1          •      | WEATHER
#.............................
#
# trajectory set number 100 (out of 218)
#
|     •          1          •      | RADAR ALTIMETER
|     •          2        {1,2}      | DIGITAL AIR DATA
|     •          0          0      | AHRS
|     •          1          1      | INS
|     •          2          •      | VOR
|     •          0          •      | DME
|     •          •        {1,2}      | SENSOR RT
|     •          •          0      | FTMP COMPUTER
|     •          •          •      | FCMS
|     •          •          •      | AFT RT
|     •          1          •      | WEATHER
#.............................
#
# trajectory set number 101 (out of 218)
#
|     •          1          •      | RADAR ALTIMETER
|     •          2        {1,2}      | DIGITAL AIR DATA
|     •          0          0      | AHRS
|     •          1          1      | INS
|     •          2          •      | VOR
|     •          0          •      | DME
|     •          •        {1,2}      | SENSOR RT
|     •          •      {1,2,3,4,5}  | FTMP COMPUTER
|     •          •          0      | FCMS
|     •          •          •      | AFT RT
|     •          1          •      | WEATHER
#.............................
#
# trajectory set number 102 (out of 218)
#
```

```
|     *          1          *      | RADAR ALTIMETER
|     *          2        {1,2}      | DIGITAL AIR DATA
|     *          0          0      | AHRS
|     *          1          1      | INS
|     *          2          *      | VOR
|     *          0          *      | DME
|     *          *        {1,2}      | SENSOR RT
|     *          *      {1,2,3,4,5}  | FTMP COMPUTER
|     *          *        {1,2}      | FCMS
|     *          *          0      | AFT RT
|     *          1          *      | WEATHER
#.............................
#
# trajectory set number 103 (out of 218)
#
|     *          1          *      | RADAR ALTIMETER
|     *          2        {1,2}      | DIGITAL AIR DATA
|     *          0          0      | AHRS
|     *          1          1      | INS
|     *        {0,1}        *      | VOR
|     *          *          *      | DME
|     *          *          0      | SENSOR RT
|     *          *          *      | FTMP COMPUTER
|     *          *          *      | FCMS
|     *          *          *      | AFT RT
|     *          1          *      | WEATHER
#.............................
#
# trajectory set number 104 (out of 218)
#
|     *          1          *      | RADAR ALTIMETER
|     *          2        {1,2}      | DIGITAL AIR DATA
|     *          0          0      | AHRS
|     *          1          1      | INS
|     *        {0,1}        *      | VOR
|     *          *          *      | DME
|     *          *        {1,2}      | SENSOR RT
|     *          *          0      | FTMP COMPUTER
|     *          *          *      | FCMS
|     *          *          *      | AFT RT
|     *          1          *      | WEATHER
#.............................
#
# trajectory set number 105 (out of 218)
#
|     *          1          *      | RADAR ALTIMETER
|     *          2        {1,2}      | DIGITAL AIR DATA
|     *          0          0      | AHRS
|     *          1          1      | INS
|     *        {0,1}        *      | VOR
|     *          *          *      | DME
|     *          *        {1,2}      | SENSOR RT
|     *          *      {1,2,3,4,5}  | FTMP COMPUTER
|     *          *          0      | FCMS
|     *          *          *      | AFT RT
|     *          1          *      | WEATHER
#.............................
#
# trajectory set number 106 (out of 218)
#
|     *          1          *      | RADAR ALTIMETER
|     *          2        {1,2}      | DIGITAL AIR DATA
|     *          0          0      | AHRS
|     *          1          1      | INS
|     *        {0,1}        *      | VOR
|     *          *          *      | DME
|     *          *        {1,2}      | SENSOR RT
|     *          *      {1,2,3,4,5}  | FTMP COMPUTER
|     *          *        {1,2}      | FCMS
|     *          *          0      | AFT RT
|     *          1          *      | WEATHER
#.............................
#
# trajectory set number 107 (out of 218)
#
```

```
|     *          1          *      | RADAR ALTIMETER
|     *          2        {1,2}      | DIGITAL AIR DATA
|     *          0          0      | AHRS
|     *          0          0      | INS
|     *          *          *      | VOR
|     *          *          *      | DME
|     *          *          *      | SENSOR RT
|     *          *          *      | FTMP COMPUTER
|     *          *          *      | FCMS
|     *          *          *      | AFT RT
|     *          1          *      | WEATHER
#.............................
#
# trajectory set number 108 (out of 218)
#
|     *          1          *      | RADAR ALTIMETER
|     *          2        {1,2}      | DIGITAL AIR DATA
|     *          0          0      | AHRS
|     *          0          1      | INS
|     *          *          *      | VOR
|     *          *          *      | DME
|     *          *          0      | SENSOR RT
|     *          *          *      | FTMP COMPUTER
|     *          *          *      | FCMS
|     *          *          *      | AFT RT
|     *          1          *      | WEATHER
#.............................
#
# trajectory set number 109 (out of 218)
#
|     *          1          *      | RADAR ALTIMETER
|     *          2        {1,2}      | DIGITAL AIR DATA
|     *          0          0      | AHRS
|     *          0          1      | INS
|     *          *          *      | VOR
|     *          *          *      | DME
|     *          *        {1,2}      | SENSOR RT
|     *          *          0      | FTMP COMPUTER
|     *          *          *      | FCMS
|     *          *          *      | AFT RT
|     *          1          *      | WEATHER
#.............................
#
# trajectory set number 110 (out of 218)
#
|     *          1          *      | RADAR ALTIMETER
|     *          2        {1,2}      | DIGITAL AIR DATA
|     *          0          0      | AHRS
|     *          0          1      | INS
|     *          *          *      | VOR
|     *          *          *      | DME
|     *          *        {1,2}      | SENSOR RT
|     *          *      {1,2,3,4,5}  | FTMP COMPUTER
|     *          *          0      | FCMS
|     *          *          *      | AFT RT
|     *          1          *      | WEATHER
#.............................
#
# trajectory set number 111 (out of 218)
#
|     *          1          *      | RADAR ALTIMETER
|     *          2        {1,2}      | DIGITAL AIR DATA
|     *          0          0      | AHRS
|     *          0          1      | INS
|     *          *          *      | VOR
|     *          *          *      | DME
|     *          *        {1,2}      | SENSOR RT
|     *          *      {1,2,3,4,5}  | FTMP COMPUTER
|     *          *        {1,2}      | FCMS
|     *          *          0      | AFT RT
|     *          1          *      | WEATHER
#.............................
#
# trajectory set number 112 (out of 218)
#
```

```
|     •          1          0     | RADAR ALTIMETER
|     •          2        {1,2}     | DIGITAL AIR DATA
|     •          0          1     | AHRS
|     •          1          •     | INS
|     •          2          •     | VOR
|     •        {1,2}          •     | DME
|     •          2          0     | SENSOR RT
|     •      {0,1,2,3,4}        •     | FTMP COMPUTER
|     •          •          •     | FCMS
|     •          •          •     | AFT RT
|     •          1          •     | WEATHER
#.............................
#
# trajectory set number 113 (out of 218)
#
|     •          1          0     | RADAR ALTIMETER
|     •          2        {1,2}     | DIGITAL AIR DATA
|     •          0          1     | AHRS
|     •          1          •     | INS
|     •          2          •     | VOR
|     •        {1,2}          •     | DME
|     •          2          0     | SENSOR RT
|     •          5          •     | FTMP COMPUTER
|     •        {0,1}          •     | FCMS
|     •          •          •     | AFT RT
|     •          1          •     | WEATHER
#.............................
#
# trajectory set number 114 (out of 218)
#
|     •          1          0     | RADAR ALTIMETER
|     •          2        {1,2}     | DIGITAL AIR DATA
|     •          0          1     | AHRS
|     •          1          •     | INS
|     •          2          •     | VOR
|     •        {1,2}          •     | DME
|     •          2          0     | SENSOR RT
|     •          5          •     | FTMP COMPUTER
|     •          2          •     | FCMS
|     •        {0,1}          •     | AFT RT
|     •          1          •     | WEATHER
#.............................
#
# trajectory set number 115 (out of 218)
#
|     •          1          0     | RADAR ALTIMETER
|     •          2        {1,2}     | DIGITAL AIR DATA
|     •          0          1     | AHRS
|     •          1          •     | INS
|     •          2          •     | VOR
|     •        {1,2}          •     | DME
|     •          2        {1,2}     | SENSOR RT
|     •          5          0     | FTMP COMPUTER
|     •        {0,1}          •     | FCMS
|     •          •          •     | AFT RT
|     •          1          •     | WEATHER
#.............................
#
# trajectory set number 116 (out of 218)
#
|     •          1          0     | RADAR ALTIMETER
|     •          2        {1,2}     | DIGITAL AIR DATA
|     •          0          1     | AHRS
|     •          1          •     | INS
|     •          2          •     | VOR
|     •        {1,2}          •     | DME
|     •          2        {1,2}     | SENSOR RT
|     •          5          0     | FTMP COMPUTER
|     •          2          •     | FCMS
|     •        {0,1}          •     | AFT RT
|     •          1          •     | WEATHER
#.............................
#
# trajectory set number 117 (out of 218)
#
```

```
|     •          1          0     | RADAR ALTIMETER
|     •          2        {1,2}     | DIGITAL AIR DATA
|     •          0          1     | AHRS
|     •          1          •     | INS
|     •          2          •     | VOR
|     •        {1,2}          •     | DME
|     •          2        {1,2}     | SENSOR RT
|     •          5      {1,2,3,4,5} | FTMP COMPUTER
|     •          2          0     | FCMS
|     •        {0,1}          •     | AFT RT
|     •          1          •     | WEATHER
#.............................
#
# trajectory set number 118 (out of 218)
#
|     •          1          •     | RADAR ALTIMETER
|     •          2        {1,2}     | DIGITAL AIR DATA
|     •          0          1     | AHRS
|     •          1          •     | INS
|     •          2          •     | VOR
|     •        {1,2}          •     | DME
|     •          2        {1,2}     | SENSOR RT
|     •          5      {1,2,3,4,5} | FTMP COMPUTER
|     •          2        {1,2}     | FCMS
|     •        {0,1}          0     | AFT RT
|     •          1          •     | WEATHER
#.............................
#
# trajectory set number 119 (out of 218)
#
|     •          1          0     | RADAR ALTIMETER
|     •          2        {1,2}     | DIGITAL AIR DATA
|     •          0          1     | AHRS
|     •          1          •     | INS
|     •          2          •     | VOR
|     •        {1,2}          •     | DME
|     •          2        {1,2}     | SENSOR RT
|     •          5      {1,2,3,4,5} | FTMP COMPUTER
|     •        {0,1}          0     | FCMS
|     •          •          •     | AFT RT
|     •          1          •     | WEATHER
#.............................
#
# trajectory set number 120 (out of 218)
#
|     •          1          •     | RADAR ALTIMETER
|     •          2        {1,2}     | DIGITAL AIR DATA
|     •          0          1     | AHRS
|     •          1          •     | INS
|     •          2          •     | VOR
|     •        {1,2}          •     | DME
|     •          2        {1,2}     | SENSOR RT
|     •          5      {1,2,3,4,5} | FTMP COMPUTER
|     •        {0,1}        {1,2}       | FCMS
|     •          •          0     | AFT RT
|     •          1          •     | WEATHER
#.............................
#
# trajectory set number 121 (out of 218)
#
|     •          1          0     | RADAR ALTIMETER
|     •          2        {1,2}     | DIGITAL AIR DATA
|     •          0          1     | AHRS
|     •          1          •     | INS
|     •          2          •     | VOR
|     •        {1,2}          •     | DME
|     •          2        {1,2}     | SENSOR RT
|     •      {0,1,2,3,4}      0     | FTMP COMPUTER
|     •          •          •     | FCMS
|     •          •          •     | AFT RT
|     •          1          •     | WEATHER
#.............................
#
# trajectory set number 122 (out of 218)
#
```

```
|     *          1          0      | RADAR ALTIMETER
|     *          2        {1,2}      | DIGITAL AIR DATA
|     *          0          1      | AHRS
|     *          1          *      | INS
|     *          2          *      | VOR
|     *        {1,2}          *      | DME
|     *          2        {1,2}      | SENSOR RT
|     *      {0,1,2,3,4}  {1,2,3,4,5}  | FTMP COMPUTER
|     *          *          0      | FCMS
|     *          *          *      | AFT RT
|     *          1          *      | WEATHER
# . . . . . .
#
# trajectory set number 123 (out of 218)
#
|     *          1          *      | RADAR ALTIMETER
|     *          2        {1,2}      | DIGITAL AIR DATA
|     *          0          1      | AHRS
|     *          1          *      | INS
|     *          2          *      | VOR
|     *        {1,2}          *      | DME
|     *          2        {1,2}      | SENSOR RT
|     *      {0,1,2,3,4}  {1,2,3,4,5}  | FTMP COMPUTER
|     *          *        {1,2}      | FCMS
|     *          *          0      | AFT RT
|     *          1          *      | WEATHER
# . . . . . .
#
# trajectory set number 124 (out of 218)
#
|     *          1          0      | RADAR ALTIMETER
|     *          2        {1,2}      | DIGITAL AIR DATA
|     *          0          1      | AHRS
|     *          1          *      | INS
|     *          2          *      | VOR
|     *        {1,2}          *      | DME
|     *        {0,1}          0      | SENSOR RT
|     *          *          *      | FTMP COMPUTER
|     *          *          *      | FCMS
|     *          *          *      | AFT RT
|     *          1          *      | WEATHER
# . . . . . .
#
# trajectory set number 125 (out of 218)
#
|     *          1          0      | RADAR ALTIMETER
|     *          2        {1,2}      | DIGITAL AIR DATA
|     *          0          1      | AHRS
|     *          1          *      | INS
|     *          2          *      | VOR
|     *        {1,2}          *      | DME
|     *        {0,1}        {1,2}      | SENSOR RT
|     *          *          0      | FTMP COMPUTER
|     *          *          *      | FCMS
|     *          *          *      | AFT RT
|     *          1          *      | WEATHER
# . . . . . .
#
# trajectory set number 126 (out of 218)
#
|     *          1          0      | RADAR ALTIMETER
|     *          2        {1,2}      | DIGITAL AIR DATA
|     *          0          1      | AHRS
|     *          1          *      | INS
|     *          2          *      | VOR
|     *        {1,2}          *      | DME
|     *        {0,1}        {1,2}      | SENSOR RT
|     *          *      {1,2,3,4,5}  | FTMP COMPUTER
|     *          *          0      | FCMS
|     *          *          *      | AFT RT
|     *          1          *      | WEATHER
# . . . . . .
#
# trajectory set number 127 (out of 218)
#
```

```
|     •          1          •      | RADAR ALTIMETER
|     •          2        {1,2}      | DIGITAL AIR DATA
|     •          0          1      | AHRS
|     •          1          •      | INS
|     •          2          •      | VOR
|     •        {1,2}        •        | DME
|     •        {0,1}      {1,2}        | SENSOR RT
|     •          •      {1,2,3,4,5}  | FTMP COMPUTER
|     •          •        {1,2}      | FCMS
|     •          •          0      | AFT RT
|     •          1          •      | WEATHER
# . . . . . .
#
# trajectory set number 128 (out of 218)
#
|     •          1          0      | RADAR ALTIMETER
|     •          2        {1,2}      | DIGITAL AIR DATA
|     •          0          1      | AHRS
|     •          1          •      | INS
|     •          2          •      | VOR
|     •          0          •      | DME
|     •          •          0      | SENSOR RT
|     •          •          •      | FTMP COMPUTER
|     •          •          •      | FCMS
|     •          •          •      | AFT RT
|     •          1          •      | WEATHER
# . . . . . .
#
# trajectory set number 129 (out of 218)
#
|     •          1          0      | RADAR ALTIMETER
|     •          2        {1,2}      | DIGITAL AIR DATA
|     •          0          1      | AHRS
|     •          1          •      | INS
|     •          2          •      | VOR
|     •          0          •      | DME
|     •          •        {1,2}      | SENSOR RT
|     •          •          0      | FTMP COMPUTER
|     •          •          •      | FCMS
|     •          •          •      | AFT RT
|     •          1          •      | WEATHER
# . . . . . .
#
# trajectory set number 130 (out of 218)
#
|     •          1          0      | RADAR ALTIMETER
|     •          2        {1,2}      | DIGITAL AIR DATA
|     •          0          1      | AHRS
|     •          1          •      | INS
|     •          2          •      | VOR
|     •          0          •      | DME
|     •          •        {1,2}      | SENSOR RT
|     •          •      {1,2,3,4,5}  | FTMP COMPUTER
|     •          •          0      | FCMS
|     •          •          •      | AFT RT
|     •          1          •      | WEATHER
# . . . . . .
#
# trajectory set number 131 (out of 218)
#
|     •          1          •      | RADAR ALTIMETER
|     •          2        {1,2}      | DIGITAL AIR DATA
|     •          0          1      | AHRS
|     •          1          •      | INS
|     •          2          •      | VOR
|     •          0          •      | DME
|     •          •        {1,2}      | SENSOR RT
|     •          •      {1,2,3,4,5}  | FTMP COMPUTER
|     •          •        {1,2}      | FCMS
|     •          •          0      | AFT RT
|     •          1          •      | WEATHER
# . . . . . .
#
# trajectory set number 132 (out of 218)
#
```

```
|     •          1          •     | RADAR ALTIMETER
|     •          2        {1,2}     | DIGITAL AIR DATA
|     •          0          1     | AHRS
|     •          1          •     | INS
|     •        {0,1}          •     | VOR
|     •          •          •     | DME
|     •          •          0     | SENSOR RT
|     •          •          •     | FTMP COMPUTER
|     •          •          •     | FCMS
|     •          •          •     | AFT RT
|     •          1          •     | WEATHER
# . . . . . .
#
# trajectory set number 133 (out of 218)
#
|     •          1          •     | RADAR ALTIMETER
|     •          2        {1,2}     | DIGITAL AIR DATA
|     •          0          1     | AHRS
|     •          1          •     | INS
|     •        {0,1}          •     | VOR
|     •          •          •     | DME
|     •          •        {1,2}     | SENSOR RT
|     •          •          0     | FTMP COMPUTER
|     •          •          •     | FCMS
|     •          •          •     | AFT RT
|     •          1          •     | WEATHER
# . . . . . .
#
# trajectory set number 134 (out of 218)
#
|     •          1          •     | RADAR ALTIMETER
|     •          2        {1,2}     | DIGITAL AIR DATA
|     •          0          1     | AHRS
|     •          1          •     | INS
|     •        {0,1}          •     | VOR
|     •          •          •     | DME
|     •          •        {1,2}     | SENSOR RT
|     •          •      {1,2,3,4,5}   | FTMP COMPUTER
|     •          •          0     | FCMS
|     •          •          •     | AFT RT
|     •          1          •     | WEATHER
# . . . . . .
#
# trajectory set number 135 (out of 218)
#
|     •          1          •     | RADAR ALTIMETER
|     •          2        {1,2}     | DIGITAL AIR DATA
|     •          0          1     | AHRS
|     •          1          •     | INS
|     •        {0,1}          •     | VOR
|     •          •          •     | DME
|     •          •        {1,2}     | SENSOR RT
|     •          •      {1,2,3,4,5}   | FTMP COMPUTER
|     •          •        {1,2}     | FCMS
|     •          •          0     | AFT RT
|     •          1          •     | WEATHER
# . . . . . .
#
# trajectory set number 136 (out of 218)
#
|     •          1          •     | RADAR ALTIMETER
|     •          2        {1,2}     | DIGITAL AIR DATA
|     •          0          1     | AHRS
|     •          0          •     | INS
|     •          •          •     | VOR
|     •          •          •     | DME
|     •          •          0     | SENSOR RT
|     •          •          •     | FTMP COMPUTER
|     •          •          •     | FCMS
|     •          •          •     | AFT RT
|     •          1          •     | WEATHER
# . . . . . .
#
# trajectory set number 137 (out of 218)
#
```

```
|     •          1          •     | RADAR ALTIMETER
|     •          2        {1,2}     | DIGITAL AIR DATA
|     •          0          1     | AHRS
|     •          0          •     | INS
|     •          •          •     | VOR
|     •          •          •     | DME
|     •          •        {1,2}     | SENSOR RT
|     •          •          0     | FTMP COMPUTER
|     •          •          •     | FCMS
|     •          •          •     | AFT RT
|     •          1          •     | WEATHER
#.....
#
# trajectory set number 138 (out of 218)
#
|     •          1          •     | RADAR ALTIMETER
|     •          2        {1,2}     | DIGITAL AIR DATA
|     •          0          1     | AHRS
|     •          0          •     | INS
|     •          •          •     | VOR
|     •          •          •     | DME
|     •          •        {1,2}     | SENSOR RT
|     •          •      {1,2,3,4,5} | FTMP COMPUTER
|     •          •          0     | FCMS
|     •          •          •     | AFT RT
|     •          1          •     | WEATHER
#.....
#
# trajectory set number 139 (out of 218)
#
|     •          1          •     | RADAR ALTIMETER
|     •          2        {1,2}     | DIGITAL AIR DATA
|     •          0          1     | AHRS
|     •          0          •     | INS
|     •          •          •     | VOR
|     •          •          •     | DME
|     •          •        {1,2}     | SENSOR RT
|     •          •      {1,2,3,4,5} | FTMP COMPUTER
|     •          •        {1,2}     | FCMS
|     •          •          0     | AFT RT
|     •          1          •     | WEATHER
#.....
#
# trajectory set number 140 (out of 218)
#
|     •          1          •     | RADAR ALTIMETER
|     •        {0,1}          0     | DIGITAL AIR DATA
|     •          •          •     | AHRS
|     •          •          •     | INS
|     •          •          •     | VOR
|     •          •          •     | DME
|     •          •          •     | SENSOR RT
|     •          •          •     | FTMP COMPUTER
|     •          •          •     | FCMS
|     •          •          •     | AFT RT
|     •          1          •     | WEATHER
#.....
#
# trajectory set number 141 (out of 218)
#
|     •          1          •     | RADAR ALTIMETER
|     •        {0,1}        {1,2}     | DIGITAL AIR DATA
|     •          •          0     | AHRS
|     •          •          0     | INS
|     •          •          •     | VOR
|     •          •          •     | DME
|     •          •          •     | SENSOR RT
|     •          •          •     | FTMP COMPUTER
|     •          •          •     | FCMS
|     •          •          •     | AFT RT
|     •          1          •     | WEATHER
#.....
#
# trajectory set number 142 (out of 218)
#
```

```
|     •          1          •     | RADAR ALTIMETER
|     •        {0,1}       {1,2}       | DIGITAL AIR DATA
|     •          •          0     | AHRS
|     •          •          1     | INS
|     •          •          •     | VOR
|     •          •          •     | DME
|     •          •          0     | SENSOR RT
|     •          •          •     | FTMP COMPUTER
|     •          •          •     | FCMS
|     •          •          •     | AFT RT
|     •          1          •     | WEATHER
#. . . . . .
#
# trajectory set number 143 (out of 218)
#
|     •          1          •     | RADAR ALTIMETER
|     •        {0,1}       {1,2}       | DIGITAL AIR DATA
|     •          •          0     | AHRS
|     •          •          1     | INS
|     •          •          •     | VOR
|     •          •          •     | DME
|     •          •        {1,2}       | SENSOR RT
|     •          •          0     | FTMP COMPUTER
|     •          •          •     | FCMS
|     •          •          •     | AFT RT
|     •          1          •     | WEATHER
#. . . . . .
#
# trajectory set number 144 (out of 218)
#
|     •          1          •     | RADAR ALTIMETER
|     •        {0,1}       {1,2}       | DIGITAL AIR DATA
|     •          •          0     | AHRS
|     •          •          1     | INS
|     •          •          •     | VOR
|     •          •          •     | DME
|     •          •        {1,2}       | SENSOR RT
|     •          •      {1,2,3,4,5}   | FTMP COMPUTER
|     •          •          0     | FCMS
|     •          •          •     | AFT RT
|     •          1          •     | WEATHER
#. . . . . .
#
# trajectory set number 145 (out of 218)
#
|     •          1          •     | RADAR ALTIMETER
|     •        {0,1}       {1,2}       | DIGITAL AIR DATA
|     •          •          0     | AHRS
|     •          •          1     | INS
|     •          •          •     | VOR
|     •          •          •     | DME
|     •          •        {1,2}       | SENSOR RT
|     •          •      {1,2,3,4,5}   | FTMP COMPUTER
|     •          •        {1,2}       | FCMS
|     •          •          0     | AFT RT
|     •          1          •     | WEATHER
#. . . . . .
#
# trajectory set number 146 (out of 218)
#
|     •          1          •     | RADAR ALTIMETER
|     •        {0,1}       {1,2}       | DIGITAL AIR DATA
|     •          •          1     | AHRS
|     •          •          •     | INS
|     •          •          •     | VOR
|     •          •          •     | DME
|     •          •          0     | SENSOR RT
|     •          •          •     | FTMP COMPUTER
|     •          •          •     | FCMS
|     •          •          •     | AFT RT
|     •          1          •     | WEATHER
#. . . . . .
#
# trajectory set number 147 (out of 218)
#
```

```
|     •          1          •     | RADAR ALTIMETER
|     •        {0,1}       {1,2}      | DIGITAL AIR DATA
|     •          •          1     | AHRS
|     •          •          •     | INS
|     •          •          •     | VOR
|     •          •          •     | DME
|     •          •        {1,2}       | SENSOR RT
|     •          •          0     | FTMP COMPUTER
|     •          •          •     | FCMS
|     •          •          •     | AFT RT
|     •          1          •     | WEATHER
# . . . . . .
#
# trajectory set number 148 (out of 218)
#
|     •          1          •     | RADAR ALTIMETER
|     •        {0,1}       {1,2}      | DIGITAL AIR DATA
|     •          •          1     | AHRS
|     •          •          •     | INS
|     •          •          •     | VOR
|     •          •          •     | DME
|     •          •        {1,2}       | SENSOR RT
|     •          •      {1,2,3,4,5}   | FTMP COMPUTER
|     •          •          0     | FCMS
|     •          •          •     | AFT RT
|     •          1          •     | WEATHER
# . . . . . .
#
# trajectory set number 149 (out of 218)
#
|     •          1          •     | RADAR ALTIMETER
|     •        {0,1}       {1,2}      | DIGITAL AIR DATA
|     •          •          1     | AHRS
|     •          •          •     | INS
|     •          •          •     | VOR
|     •          •          •     | DME
|     •          •        {1,2}       | SENSOR RT
|     •          •      {1,2,3,4,5}   | FTMP COMPUTER
|     •          •        {1,2}       | FCMS
|     •          •          0     | AFT RT
|     •          1          •     | WEATHER
# . . . . . .
#
# trajectory set number 150 (out of 218)
#
|     •          1          1     | RADAR ALTIMETER
|     •          2          0     | DIGITAL AIR DATA
|     •          0          0     | AHRS
|     •          1          1     | INS
|     •          •          •     | VOR
|     •          •          •     | DME
|     •          •          •     | SENSOR RT
|     •          •          •     | FTMP COMPUTER
|     •          •          •     | FCMS
|     •          •          •     | AFT RT
|     •          1          •     | WEATHER
# . . . . . .
#
# trajectory set number 151 (out of 218)
#
|     •          1          1     | RADAR ALTIMETER
|     •          2          0     | DIGITAL AIR DATA
|     •          0          1     | AHRS
|     •          1          •     | INS
|     •        {0,1}        •       | VOR
|     •          •          •     | DME
|     •          •          •     | SENSOR RT
|     •          •          •     | FTMP COMPUTER
|     •          •          •     | FCMS
|     •          •          •     | AFT RT
|     •          1          •     | WEATHER
# . . . . . .
#
# trajectory set number 152 (out of 218)
#
```

```
|      •          1          1      | RADAR ALTIMETER
|      •          2          0      | DIGITAL AIR DATA
|      •          0          1      | AHRS
|      •          1          •      | INS
|      •          2          •      | VOR
|      •          0          •      | DME
|      •          •          •      | SENSOR RT
|      •          •          •      | FTMP COMPUTER
|      •          •          •      | FCMS
|      •          •          •      | AFT RT
|      •          1          •      | WEATHER
#.....................................
#
# trajectory set number 153 (out of 218)
#
|      •          1          1      | RADAR ALTIMETER
|      •          2          0      | DIGITAL AIR DATA
|      •          0          1      | AHRS
|      •          1          •      | INS
|      •          2          •      | VOR
|      •        {1,2}        •      |  | DME
|      •        {0,1}        •      |  | SENSOR RT
|      •          •          •      | FTMP COMPUTER
|      •          •          •      | FCMS
|      •          •          •      | AFT RT
|      •          1          •      | WEATHER
#.....................................
#
# trajectory set number 154 (out of 218)
#
|      •          1          1      | RADAR ALTIMETER
|      •          2          0      | DIGITAL AIR DATA
|      •          0          1      | AHRS
|      •          1          •      | INS
|      •          2          •      | VOR
|      •        {1,2}        •      |  | DME
|      •          2          •      | SENSOR RT
|      •      {0,1,2,3,4}    •      |  | FTMP COMPUTER
|      •          •          •      | FCMS
|      •          •          •      | AFT RT
|      •          1          •      | WEATHER
#.....................................
#
# trajectory set number 155 (out of 218)
#
|      •          1          1      | RADAR ALTIMETER
|      •          2          0      | DIGITAL AIR DATA
|      •          0          1      | AHRS
|      •          1          •      | INS
|      •          2          •      | VOR
|      •        {1,2}        •      |  | DME
|      •          2          •      | SENSOR RT
|      •          5          •      | FTMP COMPUTER
|      •        {0,1}        •      |  | FCMS
|      •          •          •      | AFT RT
|      •          1          •      | WEATHER
#.....................................
#
# trajectory set number 156 (out of 218)
#
|      •          1          1      | RADAR ALTIMETER
|      •          2          0      | DIGITAL AIR DATA
|      •          0          1      | AHRS
|      •          1          •      | INS
|      •          2          •      | VOR
|      •        {1,2}        •      |  | DME
|      •          2          •      | SENSOR RT
|      •          5          •      | FTMP COMPUTER
|      •          2          •      | FCMS
|      •        {0,1}        •      |  | AFT RT
|      •          1          •      | WEATHER
#.....................................
#
# trajectory set number 157 (out of 218)
#
```

```
|      •          1           1      | RADAR ALTIMETER
|      •          2           0      | DIGITAL AIR DATA
|      •          0           1      | AHRS
|      •          1           1      | INS
|      •          2         {1,2}    |   VOR
|      •        {1,2}         •      |   DME
|      •          2         {1,2}    |   SENSOR RT
|      •          5       {1,2,3,4,5}| FTMP COMPUTER
|      •          2         {1,2}    |   FCMS
|      •          2         {1,2}    |   AFT RT
|      •          1           •      | WEATHER
# . . . . . .
#
# trajectory set number 158 (out of 218)
#
|      •          1           1      | RADAR ALTIMETER
|      •          2           0      | DIGITAL AIR DATA
|      •          0           1      | AHRS
|      •          1           1      | INS
|      •          2         {1,2}    |   VOR
|      •        {1,2}         •      |   DME
|      •          2         {1,2}    |   SENSOR RT
|      •          5       {1,2,3,4,5}| FTMP COMPUTER
|      •          2         {1,2}    |   FCMS
|      •          2           0      | AFT RT
|      •          1           0      | WEATHER
# . . . . . .
#
# trajectory set number 159 (out of 218)
#
|      •          1           1      | RADAR ALTIMETER
|      •          2           0      | DIGITAL AIR DATA
|      •          0           0      | AHRS
|      •          1           0      | INS
|      •        {0,1}         •      |   VOR
|      •          •           •      | DME
|      •          •           •      | SENSOR RT
|      •          •           •      | FTMP COMPUTER
|      •          •           •      | FCMS
|      •          •           •      | AFT RT
|      •          1           •      | WEATHER
# . . . . . .
#
# trajectory set number 160 (out of 218)
#
|      •          1           1      | RADAR ALTIMETER
|      •          2           0      | DIGITAL AIR DATA
|      •          0           0      | AHRS
|      •          1           0      | INS
|      •          2           •      | VOR
|      •          0           •      | DME
|      •          •           •      | SENSOR RT
|      •          •           •      | FTMP COMPUTER
|      •          •           •      | FCMS
|      •          •           •      | AFT RT
|      •          1           •      | WEATHER
# . . . . . .
#
# trajectory set number 161 (out of 218)
#
|      •          1           1      | RADAR ALTIMETER
|      •          2           0      | DIGITAL AIR DATA
|      •          0           0      | AHRS
|      •          1           0      | INS
|      •          2           •      | VOR
|      •        {1,2}         •      |   DME
|      •        {0,1}         •      |   SENSOR RT
|      •          •           •      | FTMP COMPUTER
|      •          •           •      | FCMS
|      •          •           •      | AFT RT
|      •          1           •      | WEATHER
# . . . . . .
#
# trajectory set number 162 (out of 218)
#
```

```
|      •          1          1      | RADAR ALTIMETER
|      •          2          0      | DIGITAL AIR DATA
|      •          0          0      | AHRS
|      •          1          0      | INS
|      •          2          •      | VOR
|      •        {1,2}          •    |  | DME
|      •          2          •      | SENSOR RT
|      •      {0,1,2,3,4}      •    |  | FTMP COMPUTER
|      •          •          •      | FCMS
|      •          •          •      | AFT RT
|      •          1          •      | WEATHER
# . . . . . .
#
# trajectory set number 163 (out of 218)
#
|      •          1          1      | RADAR ALTIMETER
|      •          2          0      | DIGITAL AIR DATA
|      •          0          0      | AHRS
|      •          1          0      | INS
|      •          2          •      | VOR
|      •        {1,2}          •    |  | DME
|      •          2          •      | SENSOR RT
|      •          5          •      | FTMP COMPUTER
|      •        {0,1}          •    |  | FCMS
|      •          •          •      | AFT RT
|      •          1          •      | WEATHER
# . . . . . .
#
# trajectory set number 164 (out of 218)
#
|      •          1          1      | RADAR ALTIMETER
|      •          2          0      | DIGITAL AIR DATA
|      •          0          0      | AHRS
|      •          1          0      | INS
|      •          2          •      | VOR
|      •        {1,2}          •    |  | DME
|      •          2          •      | SENSOR RT
|      •          5          •      | FTMP COMPUTER
|      •          2          •      | FCMS
|      •        {0,1}          •    |  | AFT RT
|      •          1          •      | WEATHER
# . . . . . .
#
# trajectory set number 165 (out of 218)
#
|      •          1          1      | RADAR ALTIMETER
|      •          2          0      | DIGITAL AIR DATA
|      •          1          0      | AHRS
|      •          •          1      | INS
|      •          •          •      | VOR
|      •          •          •      | DME
|      •          •          •      | SENSOR RT
|      •          •          •      | FTMP COMPUTER
|      •          •          •      | FCMS
|      •          •          •      | AFT RT
|      •          1          •      | WEATHER
# . . . . . .
#
# trajectory set number 166 (out of 218)
#
|      •          1          1      | RADAR ALTIMETER
|      •          2          0      | DIGITAL AIR DATA
|      •          1          1      | AHRS
|      •          •          •      | INS
|      •        {0,1}          •    |  | VOR
|      •          •          •      | DME
|      •          •          •      | SENSOR RT
|      •          •          •      | FTMP COMPUTER
|      •          •          •      | FCMS
|      •          •          •      | AFT RT
|      •          1          •      | WEATHER
# . . . . . .
#
# trajectory set number 167 (out of 218)
#
```

```
|      •          1          1      | RADAR ALTIMETER
|      •          2          0      | DIGITAL AIR DATA
|      •          1          1      | AHRS
|      •          •          •      | INS
|      •          2          •      | VOR
|      •          0          •      | DME
|      •          •          •      | SENSOR RT
|      •          •          •      | FTMP COMPUTER
|      •          •          •      | FCMS
|      •          •          •      | AFT RT
|      •          1          •      | WEATHER
#.....................................
#
# trajectory set number 168 (out of 218)
#
|      •          1          1      | RADAR ALTIMETER
|      •          2          0      | DIGITAL AIR DATA
|      •          1          1      | AHRS
|      •          •          •      | INS
|      •          2          •      | VOR
|      •        {1,2}              •   | DME
|      •        {0,1}              •   | SENSOR RT
|      •          •          •      | FTMP COMPUTER
|      •          •          •      | FCMS
|      •          •          •      | AFT RT
|      •          1          •      | WEATHER
#.....................................
#
# trajectory set number 169 (out of 218)
#
|      •          1          1      | RADAR ALTIMETER
|      •          2          0      | DIGITAL AIR DATA
|      •          1          1      | AHRS
|      •          •          •      | INS
|      •          2          •      | VOR
|      •        {1,2}              •   | DME
|      •          2          •      | SENSOR RT
|      •      {0,1,2,3,4}          •   | FTMP COMPUTER
|      •          •          •      | FCMS
|      •          •          •      | AFT RT
|      •          1          •      | WEATHER
#.....................................
#
# trajectory set number 170 (out of 218)
#
|      •          1          1      | RADAR ALTIMETER
|      •          2          0      | DIGITAL AIR DATA
|      •          1          1      | AHRS
|      •          •          •      | INS
|      •          2          •      | VOR
|      •        {1,2}              •   | DME
|      •          2          •      | SENSOR RT
|      •          5          •      | FTMP COMPUTER
|      •        {0,1}              •   | FCMS
|      •          •          •      | AFT RT
|      •          1          •      | WEATHER
#.....................................
#
# trajectory set number 171 (out of 218)
#
|      •          1          1      | RADAR ALTIMETER
|      •          2          0      | DIGITAL AIR DATA
|      •          1          1      | AHRS
|      •          •          •      | INS
|      •          2          •      | VOR
|      •        {1,2}              •   | DME
|      •          2          •      | SENSOR RT
|      •          5          •      | FTMP COMPUTER
|      •          2          •      | FCMS
|      •        {0,1}              •   | AFT RT
|      •          1          •      | WEATHER
#.....................................
#
# trajectory set number 172 (out of 218)
#
```

```
|     •          1          1      | RADAR ALTIMETER
|     •          2          0      | DIGITAL AIR DATA
|     •          1          1      | AHRS
|     •          •          1      | INS
|     •          2        {1,2}      | VOR
|     •        {1,2}        •        | DME
|     •          2        {1,2}      | SENSOR RT
|     •          5      {1,2,3,4,5}  | FTMP COMPUTER
|     •          2        {1,2}      | FCMS
|     •          2        {1,2}      | AFT RT
|     •          1          •      | WEATHER
#.............................
#
# trajectory set number 173 (out of 218)
#
|     •          1          1      | RADAR ALTIMETER
|     •          2          0      | DIGITAL AIR DATA
|     •          1          0      | AHRS
|     •          •          0      | INS
|     •        {0,1}        •        | VOR
|     •          •          •      | DME
|     •          •          •      | SENSOR RT
|     •          •          •      | FTMP COMPUTER
|     •          •          •      | FCMS
|     •          •          •      | AFT RT
|     •          1          •      | WEATHER
#.............................
#
# trajectory set number 174 (out of 218)
#
|     •          1          1      | RADAR ALTIMETER
|     •          2          0      | DIGITAL AIR DATA
|     •          1          0      | AHRS
|     •          •          0      | INS
|     •          2          •      | VOR
|     •          0          •      | DME
|     •          •          •      | SENSOR RT
|     •          •          •      | FTMP COMPUTER
|     •          •          •      | FCMS
|     •          •          •      | AFT RT
|     •          1          •      | WEATHER
#.............................
#
# trajectory set number 175 (out of 218)
#
|     •          1          1      | RADAR ALTIMETER
|     •          2          0      | DIGITAL AIR DATA
|     •          1          0      | AHRS
|     •          •          0      | INS
|     •          2          •      | VOR
|     •        {1,2}        •        | DME
|     •        {0,1}        •        | SENSOR RT
|     •          •          •      | FTMP COMPUTER
|     •          •          •      | FCMS
|     •          •          •      | AFT RT
|     •          1          •      | WEATHER
#.............................
#
# trajectory set number 176 (out of 218)
#
|     •          1          1      | RADAR ALTIMETER
|     •          2          0      | DIGITAL AIR DATA
|     •          1          0      | AHRS
|     •          •          0      | INS
|     •          2          •      | VOR
|     •        {1,2}        •        | DME
|     •          2          •      | SENSOR RT
|     •      {0,1,2,3,4}    •        | FTMP COMPUTER
|     •          •          •      | FCMS
|     •          •          •      | AFT RT
|     •          1          •      | WEATHER
#.............................
#
# trajectory set number 177 (out of 218)
#
```

```
|    •          1          1      | RADAR ALTIMETER
|    •          2          0      | DIGITAL AIR DATA
|    •          1          0      | AHRS
|    •          •          0      | INS
|    •          2          •      | VOR
|    •        {1,2}              • |   DME
|    •          2          •      | SENSOR RT
|    •          5          •      | FTMP COMPUTER
|    •        {0,1}              • |   FCMS
|    •          •          •      | AFT RT
|    •          1          •      | WEATHER
#.....
#
# trajectory set number 178 (out of 218)
#
|    •          1          1      | RADAR ALTIMETER
|    •          2          0      | DIGITAL AIR DATA
|    •          1          0      | AHRS
|    •          •          0      | INS
|    •          2          •      | VOR
|    •        {1,2}              • |   DME
|    •          2          •      | SENSOR RT
|    •          5          •      | FTMP COMPUTER
|    •          2          •      | FCMS
|    •        {0,1}              • |   AFT RT
|    •          1          •      | WEATHER
#.....
#
# trajectory set number 179 (out of 218)
#
|    •          1          1      | RADAR ALTIMETER
|    •          2        {1,2}    |   DIGITAL AIR DATA
|    •          0          1      | AHRS
|    •          1          0      | INS
|    •          2          •      | VOR
|    •          •          •      | DME
|    •          •          0      | SENSOR RT
|    •          •          •      | FTMP COMPUTER
|    •          •          •      | FCMS
|    •          •          •      | AFT RT
|    •          1          •      | WEATHER
#.....
#
# trajectory set number 180 (out of 218)
#
|    •          1          1      | RADAR ALTIMETER
|    •          2        {1,2}    |   DIGITAL AIR DATA
|    •          0          1      | AHRS
|    •          1          1      | INS
|    •          2          •      | VOR
|    •        {1,2}              • |   DME
|    •          2          0      | SENSOR RT
|    •      {0,1,2,3,4}          • |   FTMP COMPUTER
|    •          •          •      | FCMS
|    •          •          •      | AFT RT
|    •          1          •      | WEATHER
#.....
#
# trajectory set number 181 (out of 218)
#
|    •          1          1      | RADAR ALTIMETER
|    •          2        {1,2}    |   DIGITAL AIR DATA
|    •          0          1      | AHRS
|    •          1          1      | INS
|    •          2          •      | VOR
|    •        {1,2}              • |   DME
|    •          2          0      | SENSOR RT
|    •          5          •      | FTMP COMPUTER
|    •        {0,1}              • |   FCMS
|    •          •          •      | AFT RT
|    •          1          •      | WEATHER
#.....
#
# trajectory set number 182 (out of 218)
#
```

```
|     •        1          1      | RADAR ALTIMETER
|     •        2        {1,2}      | DIGITAL AIR DATA
|     •        0          1      | AHRS
|     •        1          1      | INS
|     •        2          •      | VOR
|     •      {1,2}        •        | DME
|     •        2          0      | SENSOR RT
|     •        5          •      | FTMP COMPUTER
|     •        2          •      | FCMS
|     •      {0,1}        •        | AFT RT
|     •        1          •      | WEATHER
#. . . . . .
#
# trajectory set number 183 (out of 218)
#
|     •        1          1      | RADAR ALTIMETER
|     •        2        {1,2}      | DIGITAL AIR DATA
|     •        0          1      | AHRS
|     •        1          0      | INS
|     •        2          •      | VOR
|     •        •          •      | DME
|     •        •        {1,2}      | SENSOR RT
|     •        •          0      | FTMP COMPUTER
|     •        •          •      | FCMS
|     •        •          •      | AFT RT
|     •        1          •      | WEATHER
#. . . . . .
#
# trajectory set number 184 (out of 218)
#
|     •        1          1      | RADAR ALTIMETER
|     •        2        {1,2}      | DIGITAL AIR DATA
|     •        0          1      | AHRS
|     •        1          1      | INS
|     •        2          •      | VOR
|     •      {1,2}        •        | DME
|     •        2        {1,2}      | SENSOR RT
|     •        5          0      | FTMP COMPUTER
|     •      {0,1}        •        | FCMS
|     •        •          •      | AFT RT
|     •        1          •      | WEATHER
#. . . . . .
#
# trajectory set number 185 (out of 218)
#
|     •        1          1      | RADAR ALTIMETER
|     •        2        {1,2}      | DIGITAL AIR DATA
|     •        0          1      | AHRS
|     •        1          1      | INS
|     •        2          •      | VOR
|     •      {1,2}        •        | DME
|     •        2        {1,2}      | SENSOR RT
|     •        5          0      | FTMP COMPUTER
|     •        2          •      | FCMS
|     •      {0,1}        •        | AFT RT
|     •        1          •      | WEATHER
#. . . . . .
#
# trajectory set number 186 (out of 218)
#
|     •        1          1      | RADAR ALTIMETER
|     •        2        {1,2}      | DIGITAL AIR DATA
|     •        0          1      | AHRS
|     •        1          0      | INS
|     •        2          •      | VOR
|     •        •          •      | DME
|     •        •        {1,2}      | SENSOR RT
|     •        •      {1,2,3,4,5}  | FTMP COMPUTER
|     •        •          0      | FCMS
|     •        •          •      | AFT RT
|     •        1          •      | WEATHER
#. . . . . .
#
# trajectory set number 187 (out of 218)
#
```

```
|     •          1          1     | RADAR ALTIMETER
|     •          2        {1,2}     | DIGITAL AIR DATA
|     •          0          1     | AHRS
|     •          1          1     | INS
|     •          2          •     | VOR
|     •        {1,2}          •     | DME
|     •          2        {1,2}     | SENSOR RT
|     •          5      {1,2,3,4,5}  | FTMP COMPUTER
|     •          2          0     | FCMS
|     •        {0,1}          •     | AFT RT
|     •          1          •     | WEATHER
#.....................
#
# trajectory set number 188 (out of 218)
#
|     •          1          1     | RADAR ALTIMETER
|     •          2        {1,2}     | DIGITAL AIR DATA
|     •          0          1     | AHRS
|     •          1          0     | INS
|     •          2          0     | VOR
|     •        {1,2}          •     | DME
|     •          2        {1,2}     | SENSOR RT
|     •          5      {1,2,3,4,5}  | FTMP COMPUTER
|     •          2        {1,2}     | FCMS
|     •          2          •     | AFT RT
|     •          1          •     | WEATHER
#.....................
#
# trajectory set number 189 (out of 218)
#
|     •          1          1     | RADAR ALTIMETER
|     •          2        {1,2}     | DIGITAL AIR DATA
|     •          0          1     | AHRS
|     •          1          1     | INS
|     •          2          •     | VOR
|     •        {1,2}          •     | DME
|     •          2        {1,2}     | SENSOR RT
|     •          5      {1,2,3,4,5}  | FTMP COMPUTER
|     •        {0,1}          0     | FCMS
|     •          •          •     | AFT RT
|     •          1          •     | WEATHER
#.....................
#
# trajectory set number 190 (out of 218)
#
|     •          1          1     | RADAR ALTIMETER
|     •          2        {1,2}     | DIGITAL AIR DATA
|     •          0          1     | AHRS
|     •          1          1     | INS
|     •          2          •     | VOR
|     •        {1,2}          •     | DME
|     •          2        {1,2}     | SENSOR RT
|     •      {0,1,2,3,4}      0     | FTMP COMPUTER
|     •          •          •     | FCMS
|     •          •          •     | AFT RT
|     •          1          •     | WEATHER
#.....................
#
# trajectory set number 191 (out of 218)
#
|     •          1          1     | RADAR ALTIMETER
|     •          2        {1,2}     | DIGITAL AIR DATA
|     •          0          1     | AHRS
|     •          1          1     | INS
|     •          2          •     | VOR
|     •        {1,2}          •     | DME
|     •          2        {1,2}     | SENSOR RT
|     •      {0,1,2,3,4}  {1,2,3,4,5}  | FTMP COMPUTER
|     •          •          0     | FCMS
|     •          •          •     | AFT RT
|     •          1          •     | WEATHER
#.....................
#
# trajectory set number 192 (out of 218)
#
```

```
|     •          1          1      | RADAR ALTIMETER
|     •          2        {1,2}      | DIGITAL AIR DATA
|     •          0          1      | AHRS
|     •          1          1      | INS
|     •          2          •      | VOR
|     •        {1,2}        •        | DME
|     •        {0,1}        0        | SENSOR RT
|     •          •          •      | FTMP COMPUTER
|     •          •          •      | FCMS
|     •          •          •      | AFT RT
|     •          1          •      | WEATHER
#.....................................
#
# trajectory set number 193 (out of 218)
#
|     •          1          1      | RADAR ALTIMETER
|     •          2        {1,2}      | DIGITAL AIR DATA
|     •          0          1      | AHRS
|     •          1          1      | INS
|     •          2          •      | VOR
|     •        {1,2}        •        | DME
|     •        {0,1}      {1,2}        | SENSOR RT
|     •          •          0      | FTMP COMPUTER
|     •          •          •      | FCMS
|     •          •          •      | AFT RT
|     •          1          •      | WEATHER
#.....................................
#
# trajectory set number 194 (out of 218)
#
|     •          1          1      | RADAR ALTIMETER
|     •          2        {1,2}      | DIGITAL AIR DATA
|     •          0          1      | AHRS
|     •          1          1      | INS
|     •          2          •      | VOR
|     •        {1,2}        •        | DME
|     •        {0,1}      {1,2}        | SENSOR RT
|     •          •      {1,2,3,4,5}  | FTMP COMPUTER
|     •          •          0      | FCMS
|     •          •          •      | AFT RT
|     •          1          •      | WEATHER
#.....................................
#
# trajectory set number 195 (out of 218)
#
|     •          1          1      | RADAR ALTIMETER
|     •          2        {1,2}      | DIGITAL AIR DATA
|     •          0          1      | AHRS
|     •          1          1      | INS
|     •          2          •      | VOR
|     •          0          •      | DME
|     •          •          0      | SENSOR RT
|     •          •          •      | FTMP COMPUTER
|     •          •          •      | FCMS
|     •          •          •      | AFT RT
|     •          1          •      | WEATHER
#.....................................
#
# trajectory set number 196 (out of 218)
#
|     •          1          1      | RADAR ALTIMETER
|     •          2        {1,2}      | DIGITAL AIR DATA
|     •          0          1      | AHRS
|     •          1          1      | INS
|     •          2          •      | VOR
|     •          0          •      | DME
|     •          •        {1,2}      | SENSOR RT
|     •          •          0      | FTMP COMPUTER
|     •          •          •      | FCMS
|     •          •          •      | AFT RT
|     •          1          •      | WEATHER
#.....................................
#
# trajectory set number 197 (out of 218)
#
```

```
|      •          1          1      | RADAR ALTIMETER
|      •          2        {1,2}      | DIGITAL AIR DATA
|      •          0          1      | AHRS
|      •          1          1      | INS
|      •          2          •      | VOR
|      •          0          •      | DME
|      •          •        {1,2}      | SENSOR RT
|      •          •      {1,2,3,4,5}  | FTMP COMPUTER
|      •          •          0      | FCMS
|      •          •          •      | AFT RT
|      •          1          •      | WEATHER
#.............................
#
# trajectory set number 198 (out of 218)
#
|      •          1          1      | RADAR ALTIMETER
|      •          2        {1,2}      | DIGITAL AIR DATA
|      •          0          1      | AHRS
|      •          1          0      | INS
|      •          2        {1,2}      | VOR
|      •        {1,2}        •      | DME
|      •          2        {1,2}      | SENSOR RT
|      •          5      {1,2,3,4,5}  | FTMP COMPUTER
|      •          2        {1,2}      | FCMS
|      •          2          0      | AFT RT
|      •          1          •      | WEATHER
#.............................
#
# trajectory set number 199 (out of 218)
#
|      •          1          1      | RADAR ALTIMETER
|      •          2        {1,2}      | DIGITAL AIR DATA
|      •          1          1      | AHRS
|      •          •          0      | INS
|      •          2          •      | VOR
|      •          •          •      | DME
|      •          •          0      | SENSOR RT
|      •          •          •      | FTMP COMPUTER
|      •          •          •      | FCMS
|      •          •          •      | AFT RT
|      •          1          •      | WEATHER
#.............................
#
# trajectory set number 200 (out of 218)
#
|      •          1          1      | RADAR ALTIMETER
|      •          2        {1,2}      | DIGITAL AIR DATA
|      •          1          1      | AHRS
|      •          •          1      | INS
|      •          2          •      | VOR
|      •        {1,2}        •      | DME
|      •          2          0      | SENSOR RT
|      •      {0,1,2,3,4}    •      | FTMP COMPUTER
|      •          •          •      | FCMS
|      •          •          •      | AFT RT
|      •          1          •      | WEATHER
#.............................
#
# trajectory set number 201 (out of 218)
#
|      •          1          1      | RADAR ALTIMETER
|      •          2        {1,2}      | DIGITAL AIR DATA
|      •          1          1      | AHRS
|      •          •          1      | INS
|      •          2          •      | VOR
|      •        {1,2}        •      | DME
|      •          2          0      | SENSOR RT
|      •          5          •      | FTMP COMPUTER
|      •        {0,1}        •      | FCMS
|      •          •          •      | AFT RT
|      •          1          •      | WEATHER
#.............................
#
# trajectory set number 202 (out of 218)
#
```

```
|     •          1          1     | RADAR ALTIMETER
|     •          2        {1,2}     | DIGITAL AIR DATA
|     •          1          1     | AHRS
|     •          •          1     | INS
|     •          2          •     | VOR
|     •        {1,2}          •     | DME
|     •          2          0     | SENSOR RT
|     •          5          •     | FTMP COMPUTER
|     •          2          •     | FCMS
|     •        {0,1}          •     | AFT RT
|     •          1          •     | WEATHER
#.....
#
# trajectory set number 203 (out of 218)
#
|     •          1          1     | RADAR ALTIMETER
|     •          2        {1,2}     | DIGITAL AIR DATA
|     •          1          1     | AHRS
|     •          •          0     | INS
|     •          2          •     | VOR
|     •          •          •     | DME
|     •          •        {1,2}     | SENSOR RT
|     •          •          0     | FTMP COMPUTER
|     •          •          •     | FCMS
|     •          •          •     | AFT RT
|     •          1          •     | WEATHER
#.....
#
# trajectory set number 204 (out of 218)
#
|     •          1          1     | RADAR ALTIMETER
|     •          2        {1,2}     | DIGITAL AIR DATA
|     •          1          1     | AHRS
|     •          •          1     | INS
|     •          2          •     | VOR
|     •        {1,2}          •     | DME
|     •          2        {1,2}     | SENSOR RT
|     •          5          0     | FTMP COMPUTER
|     •        {0,1}          •     | FCMS
|     •          •          •     | AFT RT
|     •          1          •     | WEATHER
#.....
#
# trajectory set number 205 (out of 218)
#
|     •          1          1     | RADAR ALTIMETER
|     •          2        {1,2}     | DIGITAL AIR DATA
|     •          1          1     | AHRS
|     •          •          1     | INS
|     •          2          •     | VOR
|     •        {1,2}          •     | DME
|     •          2        {1,2}     | SENSOR RT
|     •          5          0     | FTMP COMPUTER
|     •          2          •     | FCMS
|     •        {0,1}          •     | AFT RT
|     •          1          •     | WEATHER
#.....
#
# trajectory set number 206 (out of 218)
#
|     •          1          1     | RADAR ALTIMETER
|     •          2        {1,2}     | DIGITAL AIR DATA
|     •          1          1     | AHRS
|     •          •          0     | INS
|     •          2          •     | VOR
|     •          •          •     | DME
|     •          •        {1,2}     | SENSOR RT
|     •          •      {1,2,3,4,5}   | FTMP COMPUTER
|     •          •          0     | FCMS
|     •          •          •     | AFT RT
|     •          1          •     | WEATHER
#.....
#
# trajectory set number 207 (out of 218)
#
```

```
|     •        1          1      | RADAR ALTIMETER
|     •        2        {1,2}      | DIGITAL AIR DATA
|     •        1          1      | AHRS
|     •        •          1      | INS
|     •        2          •      | VOR
|     •      {1,2}          •      | DME
|     •        2        {1,2}      | SENSOR RT
|     •        5      {1,2,3,4,5}  | FTMP COMPUTER
|     •        2          0      | FCMS
|     •      {0,1}          •      | AFT RT
|     •        1          •      | WEATHER
#.............................
#
# trajectory set number 208 (out of 218)
#
|     •        1          1      | RADAR ALTIMETER
|     •        2        {1,2}      | DIGITAL AIR DATA
|     •        1          1      | AHRS
|     •        •          0      | INS
|     •        2          0      | VOR
|     •      {1,2}          •      | DME
|     •        2        {1,2}      | SENSOR RT
|     •        5      {1,2,3,4,5}  | FTMP COMPUTER
|     •        2        {1,2}      | FCMS
|     •        2          •      | AFT RT
|     •        1          •      | WEATHER
#.............................
#
# trajectory set number 209 (out of 218)
#
|     •        1          1      | RADAR ALTIMETER
|     •        2        {1,2}      | DIGITAL AIR DATA
|     •        1          1      | AHRS
|     •        •          1      | INS
|     •        2          •      | VOR
|     •      {1,2}          •      | DME
|     •        2        {1,2}      | SENSOR RT
|     •        5      {1,2,3,4,5}  | FTMP COMPUTER
|     •      {0,1}          0      | FCMS
|     •        •          •      | AFT RT
|     •        1          •      | WEATHER
#.............................
#
# trajectory set number 210 (out of 218)
#
|     •        1          1      | RADAR ALTIMETER
|     •        2        {1,2}      | DIGITAL AIR DATA
|     •        1          1      | AHRS
|     •        •          1      | INS
|     •        2          •      | VOR
|     •      {1,2}          •      | DME
|     •        2        {1,2}      | SENSOR RT
|     •    {0,1,2,3,4}      0      | FTMP COMPUTER
|     •        •          •      | FCMS
|     •        •          •      | AFT RT
|     •        1          •      | WEATHER
#.............................
#
# trajectory set number 211 (out of 218)
#
|     •        1          1      | RADAR ALTIMETER
|     •        2        {1,2}      | DIGITAL AIR DATA
|     •        1          1      | AHRS
|     •        •          1      | INS
|     •        2          •      | VOR
|     •      {1,2}          •      | DME
|     •        2        {1,2}      | SENSOR RT
|     •    {0,1,2,3,4}  {1,2,3,4,5}  | FTMP COMPUTER
|     •        •          0      | FCMS
|     •        •          •      | AFT RT
|     •        1          •      | WEATHER
#.............................
#
# trajectory set number 212 (out of 218)
#
```

```
|     •          1          1      | RADAR ALTIMETER
|     •          2        {1,2}      | DIGITAL AIR DATA
|     •          1          1      | AHRS
|     •          •          1      | INS
|     •          2          •      | VOR
|     •        {1,2}        •        | DME
|     •        {0,1}        0        | SENSOR RT
|     •          •          •      | FTMP COMPUTER
|     •          •          •      | FCMS
|     •          •          •      | AFT RT
|     •          1          •      | WEATHER
# . . . . . .
#
# trajectory set number 213 (out of 218)
#
|     •          1          1      | RADAR ALTIMETER
|     •          2        {1,2}      | DIGITAL AIR DATA
|     •          1          1      | AHRS
|     •          •          1      | INS
|     •          2          •      | VOR
|     •        {1,2}        •        | DME
|     •        {0,1}      {1,2}        | SENSOR RT
|     •          •          0      | FTMP COMPUTER
|     •          •          •      | FCMS
|     •          •          •      | AFT RT
|     •          1          •      | WEATHER
# . . . . . .
#
# trajectory set number 214 (out of 218)
#
|     •          1          1      | RADAR ALTIMETER
|     •          2        {1,2}      | DIGITAL AIR DATA
|     •          1          1      | AHRS
|     •          •          1      | INS
|     •          2          •      | VOR
|     •        {1,2}        •        | DME
|     •        {0,1}      {1,2}        | SENSOR RT
|     •          •      {1,2,3,4,5}  | FTMP COMPUTER
|     •          •          0      | FCMS
|     •          •          •      | AFT RT
|     •          1          •      | WEATHER
# . . . . . .
#
# trajectory set number 215 (out of 218)
#
|     •          1          1      | RADAR ALTIMETER
|     •          2        {1,2}      | DIGITAL AIR DATA
|     •          1          1      | AHRS
|     •          •          1      | INS
|     •          2          •      | VOR
|     •          0          •      | DME
|     •          •          0      | SENSOR RT
|     •          •          •      | FTMP COMPUTER
|     •          •          •      | FCMS
|     •          •          •      | AFT RT
|     •          1          •      | WEATHER
# . . . . . .
#
# trajectory set number 216 (out of 218)
#
|     •          1          1      | RADAR ALTIMETER
|     •          2        {1,2}      | DIGITAL AIR DATA
|     •          1          1      | AHRS
|     •          •          1      | INS
|     •          2          •      | VOR
|     •          0          •      | DME
|     •          •        {1,2}      | SENSOR RT
|     •          •          0      | FTMP COMPUTER
|     •          •          •      | FCMS
|     •          •          •      | AFT RT
|     •          1          •      | WEATHER
# . . . . . .
#
# trajectory set number 217 (out of 218)
#
```

```
|     •          1          1      | RADAR ALTIMETER
|     •          2        {1,2}      | DIGITAL AIR DATA
|     •          1          1      | AHRS
|     •          •          1      | INS
|     •          2          •      | VOR
|     •          0          •      | DME
|     •          •        {1,2}      | SENSOR RT
|     •          •      {1,2,3,4,5}  | FTMP COMPUTER
|     •          •          0      | FCMS
|     •          •          •      | AFT RT
|     •          1          •      | WEATHER
# . . . . . .
#
# trajectory set number 218 (out of 218)
#
|     •          1          1      | RADAR ALTIMETER
|     •          2        {1,2}      | DIGITAL AIR DATA
|     •          1          1      | AHRS
|     •          •          0      | INS
|     •          2        {1,2}      | VOR
|     •        {1,2}          •      | DME
|     •          2        {1,2}      | SENSOR RT
|     •          5      {1,2,3,4,5}  | FTMP COMPUTER
|     •          2        {1,2}      | FCMS
|     •          2          0      | AFT RT
|     •          1          •      | WEATHER
```

APPENDIX K

Closed-form solution of a simple 3-state, acyclic, nonrecoverable process

This appendix contains the symbolic derivation of the distribution of reward for a simple markovian three-state, acyclic, nonrecoverable process, where $r(u_2) > r(u_1) > r(u_0)$. See Section 5.3.9.2 [Three State Markovian Acyclic Process] and Fig. 5.4. The regions $C^l_{y,j}$ are explicitly stated and the appropriate cases of Theorem 5.25 iii) are indicated in brackets.

For the example presented in this appendix, we carry the full notation throughout the derivation. This is done, in part, to emphasize the complexity of the integrations. As can be seen from the intricacy of the equations in this appendix, such a "brute force" technique quickly becomes difficult to follow. Therefore, in Appendix L [Closed-form solution of a simple 4-state, acyclic, nonrecoverable process], we use intermediate variables to denote partial results.

Let $y \in [r(u_0)t, r(u_1)t)$ (thus, $l = 1$):

$i = 1 = n-1$ (i.e., checking for 1-resolvability):

$j = n = 2$:

$$C^1_{y,2} = \left(\underset{\geq 0}{}, \frac{y - r(u_1)t}{r(u_2) - r(u_1)} \right] \quad \text{[case } b\text{-}i\text{)]} \tag{K.1}$$

and since $y < r(u_1)t$:

$$= \left[0, \frac{y - r(u_0)t}{r(u_2) - r(u_0)} \right] \tag{K.2}$$

$j = 1 = i$:

$$C^1_{y,1} = \left[0, \frac{y - r(u_0)t - (r(u_2) - r(u_0))v_2}{r(u_1) - r(u_0)} \right] \qquad [case\ b\text{–}iii)] \tag{K.3}$$

$j = 0$:

$$C^1_{y,0} = [0, \infty) \qquad [case\ b\text{–}iv)] \tag{K.4}$$

$$F_Y(y) = \int_{C^1_{y,2}} \int_{C^1_{y,1}} f_2(v_2) f_1(v_1) dv_1 dv_2 \tag{K.5}$$

$$= \int_0^{\frac{y - r(u_0)t}{r(u_2) - r(u_0)}} \int_0^{\frac{y - r(u_0)t - (r(u_2) - r(u_0))v_2}{r(u_1) - r(u_0)}} \lambda_2 e^{-\lambda_2 v_2} \lambda_1 e^{-\lambda_1 v_1} dv_1 dv_2 \tag{K.6}$$

$$= \int_0^{\frac{y - r(u_0)t}{r(u_2) - r(u_0)}} \lambda_2 e^{-\lambda_2 v_2} \left[1 - e^{-\frac{\lambda_1 \left[y - r(u_0)t - (r(u_2) - r(u_0))v_2 \right]}{r(u_1) - r(u_0)}} \right] dv_2 \tag{K.7}$$

$$= \left[- e^{-\lambda_2 v_2} + \frac{\lambda_2 (r(u_1) - r(u_0))}{\lambda_1 (r(u_2) - r(u_0)) + \lambda_2 (r(u_1) - r(u_0))} \right.$$

$$\left. e^{-\frac{\lambda_1 (y - r(u_0)t)}{r(u_1) - r(u_0)} - \frac{\left[\lambda_1 (r(u_2) - r(u_0)) + \lambda_2 (r(u_1) - r(u_0)) \right] v_2}{r(u_1) - r(u_0)}} \right]_0^{\frac{y - r(u_0)t}{r(u_2) - r(u_0)}} \tag{K.8}$$

If $y \in [r(u_0)t, r(u_1)t)$ (K.9)

$$F_Y(y) = 1 - e^{-\frac{\lambda_2(y - r(u_0)t)}{r(u_2) - r(u_0)}} + \frac{\lambda_2(r(u_1) - r(u_0))}{\lambda_1(r(u_2) - r(u_0)) + \lambda_2(r(u_1) - r(u_0))}$$

$$e^{-\frac{\lambda_1(y - r(u_0)t)}{r(u_1) - r(u_0)}} \left[1 - e^{-\frac{\left[\lambda_1(r(u_2) - r(u_0)) + \lambda_2(r(u_1) - r(u_0))\right](y - r(u_0)t)}{r(u_1) - r(u_0)}} \right].$$

Let $y \in [r(u_1)t, r(u_2)t)$ (thus, $l = 2$):

$i = 2 = n$ (i.e., checking for 2-resolvability):

$j = n = 2$:

$$C^2_{y,2} = \left[0, \frac{y - r(u_1)t}{r(u_2) - r(u_1)} \right) \quad \text{[case } a\text{-}i\text{)]} \tag{K.10}$$

$j = n-1 = 1$:

$$C^2_{y,1} = [0, \infty) \quad \text{[case } a\text{-}ii\text{)]} \tag{K.11}$$

$j = 0$:

$$C^2_{y,1} = [0, \infty) \quad \text{[case } a\text{-}ii\text{)]} \tag{K.12}$$

$$\int_{C^2_{y,2}} f_2(v_2)dv_2 = \int_0^{\frac{y - r(u_1)t}{r(u_2) - r(u_1)}} \lambda_2 e^{-\lambda_2 v_2} dv_2 \tag{K.13}$$

$$\int_{C^2_{y,2}} f_2(v_2)dv_2 = 1 - e^{\frac{\lambda_2(y - r(u_1)t)}{r(u_2) - r(u_1)}} \tag{K.14}$$

$i = n - 1 = l - 1 = 1$ (i.e., checking for 1-resolvability):

$j = 2$:

$$C^1_{y,2} = \underset{\geq 0}{\left(\right.} \frac{y - r(u_1)t}{r(u_2) - r(u_1)} \; , \; \frac{y - r(u_0)t}{r(u_2) - r(u_0)} \left. \right) \quad \text{[case } b\text{–}i\text{)]} \tag{K.15}$$

$y \geq r(u_1)t$ so

$$= \left[\frac{y - r(u_1)t}{r(u_2) - r(u_1)} \; , \; \frac{y - r(u_0)t}{r(u_2) - r(u_0)} \right) \tag{K.16}$$

$j = 1$:

$$C^1_{y,1} = \underset{\geq 0}{\left(\right.} \frac{y - r(u_1)t - (r(u_2) - r(u_1))v_2}{(\, r(u_1) - r(u_0)\,)} \; ,$$

$$\frac{y - r(u_0)t - (r(u_2) - r(u_0))v_2}{(\, r(u_1) - r(u_0)\,)} \left. \right] \tag{K.17}$$

[case b–ii)]

We know from Eq. K.16 that $v_2 \geq \dfrac{y - r(u_1)t}{r(u_2) - r(u_1)}$

$$\Rightarrow \quad \frac{y - r(u_1)t - (r(u_2) - r(u_1))v_2}{(\, r(u_1) - r(u_0)\,)}$$

$$\leq \frac{y - r(u_1)t - \dfrac{(r(u_2) - r(u_1))(y - r(u_1)t)}{r(u_2) - r(u_1)}}{(\, r(u_1) - r(u_0)\,)} \tag{K.18}$$

$$= 0, \text{ so}$$

$$C^1_{y,1} = \left[\, 0, \frac{y - r(u_0)t - (r(u_2) - r(u_0))v_2}{r(u_1) - r(u_0)} \,\right] \tag{K.19}$$

$j = 0$:

$$C^1_{y,0} = [0, \infty) \qquad [\text{case } b\text{-}iii)] \tag{K.20}$$

$$\int_{C^2_{y,1}} \int_{C^2_{y,1}} f_2(v_2) f_1(v_1) dv_1 dv_2 \tag{K.21}$$

$$= \int_{\frac{y - r(u_1)t}{r(u_2) - r(u_1)}}^{\frac{y - r(u_0)t}{r(u_2) - r(u_0)}} \int_0^{\frac{y - r(u_0)t - (r(u_2) - r(u_0))v_2}{r(u_1) - r(u_0)}} \lambda_2 e^{-\lambda_2 v_2} \lambda_1 e^{-\lambda_1 v_1} dv_1 dv_2 \tag{K.22}$$

$$= \int_{\frac{y - r(u_1)t}{r(u_2) - r(u_1)}}^{\frac{y - r(u_0)t}{r(u_2) - r(u_0)}} \lambda_2 e^{-\lambda_2 v_2} \left[1 - e^{-\frac{\lambda_1}{r(u_1) - r(u_0)} \left[y - r(u_0)t - (r(u_2) - r(u_0))v_2 \right]} \right] dv_2 \tag{K.23}$$

$$= \int_{\frac{y - r(u_1)t}{r(u_2) - r(u_1)t}}^{\frac{y - r(u_0)t}{r(u_2) - r(u_0)}} \left[\lambda_2 e^{-\lambda_2 v_2} - \lambda_2 e^{-\frac{v_2}{r(u_1) - r(u_0)} - \frac{\lambda_1 (y - r(u_0)t)}{r(u_1) - r(u_0)}} \right] dv_2 \tag{K.24}$$

$$= \left[- e^{-\lambda_2 v_2} + \frac{(r(u_1) - r(u_0))\lambda_2}{\lambda_2 (r(u_1) - r(u_0)) - \lambda_1 (r(u_2) - r(u_0))} \right. \tag{K.25}$$

$$\left. e^{-\frac{v_2}{r(u_1) - r(u_0)} - \frac{\lambda_1 (y - r(u_0)t)}{r(u_1) - r(u_0)}} \right]_{\frac{y - r(u_1)t}{r(u_2) - r(u_1)}}^{\frac{y - r(u_0)t}{r(u_2) - r(u_0)}}$$

$$\int_{C^2_{y,1}} \int_{C^2_{y,1}} f_2(v_2) f_1(v_1) dv_1 dv_2 = e^{-\frac{\lambda_2(y - r(u_1)t)}{r(u_2) - r(u_1)}} - e^{-\frac{\lambda_2(y - r(u_0)t)}{r(u_2) - r(u_0)}} \tag{K.26}$$

$$+ \frac{(r(u_1) - r(u_0))\lambda_2}{\lambda_2(r(u_1) - r(u_0)) - \lambda_1(r(u_2) - r(u_0))} e^{-\frac{\lambda_1(y - r(u_0)t)}{r(u_2) - r(u_0)}}$$

$$\left[e^{-\frac{(y - r(u_0)t)\left[\lambda_2(r(u_1) - r(u_0)) - \lambda_1(r(u_2) - r(u_0))\right]}{(r(u_1) - r(u_0))(r(u_2) - r(u_0))}} \right.$$

$$\left. - e^{-\frac{(y - r(u_1)t)\left[\lambda_2(r(u_1) - r(u_0)) - \lambda_1(r(u_2) - r(u_0))\right]}{(r(u_1) - r(u_0))(r(u_2) - r(u_0))}} \right]$$

Therefore,

$$F_Y(y) = \int_{C^2_{y,2}} f_1(v_1) dv_1 + \int_{C^1_{y,2}} \int_{C^1_{y,1}} f_2(v_2) f_1(v_1) dv_1 dv_2 \tag{K.27}$$

$$= 1 - e^{-\frac{\lambda_2(y - r(u_1)t)}{r(u_2) - r(u_1)}} + e^{-\frac{\lambda_2(y - r(u_1)t)}{r(u_2) - r(u_1)}} - e^{-\frac{\lambda_2(y - r(u_0)t)}{r(u_2) - r(u_0)}}$$

$$+ \frac{(r(u_1) - r(u_0))\lambda_2}{\lambda_2(r(u_1) - r(u_0)) - \lambda_1(r(u_2) - r(u_0))} e^{-\frac{\lambda_1(y - r(u_0)t)}{r(u_2) - r(u_0)}}$$

$$\left[e^{-\frac{(y - r(u_0)t)\left[\lambda_2(r(u_1) - r(u_0)) - \lambda_1(r(u_2) - r(u_0))\right]}{(r(u_1) - r(u_0))(r(u_2) - r(u_0))}} \right. \tag{K.28}$$

$$\left. - e^{-\frac{(y - r(u_1)t)\left[\lambda_2(r(u_1) - r(u_0)) - \lambda_1(r(u_2) - r(u_0))\right]}{(r(u_1) - r(u_0))(r(u_2) - r(u_0))}} \right]$$

If $y \in [r(u_1)t, r(u_2)t)$ (K.29)

$$F_Y(y) = 1 - e^{-\frac{\lambda_2(y - r(u_0)t)}{r(u_2) - r(u_0)}} + \frac{(r(u_1) - r(u_0))\lambda_2}{\lambda_2(r(u_1) - r(u_0)) - \lambda_1(r(u_2) - r(u_0))}$$

$$e^{-\frac{\lambda_1(y - r(u_0)t)}{r(u_2) - r(u_0)}} \left[e^{-\frac{(y - r(u_0)t)\left[\lambda_2(r(u_1) - r(u_0)) - \lambda_1(r(u_2) - r(u_0))\right]}{(r(u_1) - r(u_0))(r(u_2) - r(u_0))}} \right.$$

$$\left. - e^{-\frac{(y - r(u_1)t)\left[\lambda_2(r(u_1) - r(u_0)) - \lambda_1(r(u_2) - r(u_0))\right]}{(r(u_1) - r(u_0))(r(u_2) - r(u_0))}} \right]$$

For $y \geq r(u_2)t$:

If $y \in [r(u_2)t, \infty)$ (K.30)

$$F_Y(y) = 1$$

In summary,

if $y \in [r(u_0)t, r(u_1)t)$:

$$F_Y(y) = 1 - e^{-\frac{\lambda_2(y - r(u_0)t}{r(u_2) - r(u_0)}} + \frac{(r(u_1) - r(u_0))\lambda_2}{\lambda_1(r(u_2) - r(u_0)) + \lambda_2(r(u_1) - r(u_0))}$$

$$e^{-\frac{\lambda_1(y - r(u_0)t)}{r(u_1) - r(u_0)}} \left[1 - e^{-\frac{\left[\lambda_1(r(u_2) - r(u_0)) + \lambda_2(r(u_1) - r(u_0))\right](y - r(u_0)t)}{r(u_1) - r(u_0)}} \right] \quad \text{(K.31)}$$

if $y \in [r(u_1)t, r(u_2)t)$:

$$F_Y(y) = 1 - e^{-\frac{\lambda_2(y - r(u_0)t)}{r(u_2) - r(u_0)}} + \frac{(r(u_1) - r(u_0))\lambda_2}{\lambda_2(r(u_1) - r(u_0)) - \lambda_1(r(u_2) - r(u_0))}$$

$$e^{-\frac{\lambda_1(y - r(u_0)t)}{r(u_2) - r(u_0)}} \left[e^{-\frac{(y - r(u_0)t)\left[\lambda_2(r(u_1) - r(u_0)) - \lambda_1(r(u_2) - r(u_0))\right]}{(r(u_1) - r(u_0))(r(u_2) - r(u_0))}} \right. \tag{K.32}$$

$$\left. - e^{-\frac{(y - r(u_1)t)\left[\lambda_2(r(u_1) - r(u_0)) - \lambda_1(r(u_2) - r(u_0))\right]}{(r(u_1) - r(u_0))(r(u_2) - r(u_0))}} \right]$$

if $y \in [r(u_2)t, \infty)$:

$$F_Y(y) = 1 \tag{K.33}$$

Finally, note that the above are sums of exponentials:

if $y \in [r(u_0)t, r(u_1)t)$:

$$F_Y(y) = 1 - B_1 e^{-A_1 y} + B_2 e^{-A_2 y} - B_3 e^{-A_3 y} \tag{K.34}$$

if $y \in [r(u_1)t, r(u_2)t)$:

$$F_Y(y) = 1 - B_1 e^{-A_1 y} + B_4 e^{-A_4 y} - B_3 e^{-A_3 y} \tag{K.35}$$

if $y \in [r(u_2)t, \infty)$:

$$F_Y(y) = 1 . \tag{K.36}$$

where

$$A_1 = \frac{\lambda_2}{r(u_2) - r(u_0)} \tag{K.37}$$

$$A_2 = \frac{\lambda_1}{r(u_2) - r(u_0)} \tag{K.38}$$

$$A_3 = \frac{\lambda_1}{r(u_2) - r(u_0)} + \frac{\lambda_1(r(u_2) - r(u_0)) + \lambda_2(r(u_1) - r(u_0))}{r(u_1) - r(u_0)} \tag{K.39}$$

$$A_4 = A_3 \tag{K.40}$$

$$B_1 = e^{\frac{\lambda_2 r(u_0)t}{r(u_2) - r(u_0)}} \tag{K.41}$$

$$B_2 = \frac{(r(u_1) - r(u_0))\lambda_2}{\lambda_1(r(u_2) - r(u_0)) + \lambda_2(r(u_1) - r(u_0))} e^{\frac{\lambda_1 r(u_0)t}{r(u_1) - r(u_0)}} \tag{K.42}$$

$$B_3 = \frac{(r(u_1) - r(u_0))\lambda_2}{\lambda_1(r(u_2) - r(u_0)) + \lambda_2(r(u_1) - r(u_0))} e^{\frac{\lambda_1 r(u_0)t}{r(u_1) - r(u_0)} + \frac{\left[\lambda_1(r(u_2) - r(u_0)) + \lambda_2(r(u_1) - r(u_0))\right] r(u_0)t}{(r(u_1) - r(u_0))(r(u_2) - r(u_0))}} \tag{K.43}$$

$$B_4 = \frac{(r(u_1) - r(u_0))\lambda_2}{\lambda_1(r(u_2) - r(u_0)) + \lambda_2(r(u_1) - r(u_0))} e^{\frac{\lambda_1 r(u_0)t}{r(u_1) - r(u_0)} + \frac{\left[\lambda_1(r(u_2) - r(u_0)) + \lambda_2(r(u_1) - r(u_0))\right] r(u_1)t}{(r(u_1) - r(u_0))(r(u_2) - r(u_0))}} \tag{K.44}$$

APPENDIX L

Closed-form solution of a simple 4-state, acyclic, nonrecoverable process

This appendix contains the symbolic derivation of the distribution of reward for a simple markovian four-state, acyclic, nonrecoverable process, where $r(u_3) > r(u_2) > r(u_1) > r(v_0)$. See Section 5.3.9.3 [Four State Markovian Acyclic Process] and Fig. 5.5. The regions $C^l_{y,j}$ are explicitly stated and the appropriate cases of Theorem 5.25 are indicated in brackets.

To help simplify the manipulations, we use the notation:

$$K_{ij}(y) = \frac{y - r(u_j)t}{r(u_i) - r(u_j)} \tag{L.1}$$

and

$$\xi_{ijk} = \frac{r(u_k) - r(u_i)}{r(u_j) - r(u_i)} \tag{L.2}$$

Let $y \in [r(u_0)t, r(u_1)t)$ (thus, $l = 1$):

$i = 1 = n-2$ (i.e., checking for 1-resolvability):

$j = n = 3$:

$$C^1_{y,3} = \underset{\geq 0}{\left(\right.} \frac{y - r(u_2)t}{r(u_3) - r(u_2)}, \frac{y - r(u_0)t}{r(u_3) - r(u_0)} \left.\right] \quad \text{[case } b\text{-}i\text{)]} \tag{L.3}$$

and since $y < r(u_2)t$:

$$= \left[0, \frac{y - r(u_0)t}{r(u_3) - r(u_0)}\right] \tag{L.4}$$

$$= \left[0, K_{3c}(y) \right] \tag{L.5}$$

$j = n - 1 = 2$:

$$C^1_{y,2} = \left(\underset{\geq 0}{} \frac{y - r(u_1)t - (r(u_3) - r(u_1))v_3}{r(u_2) - r(u_1)}, \frac{y - r(u_0)t - (r(u_3) - r(u_0))v_3}{r(u_2) - r(u_0)} \right] \tag{L.6}$$

[case *b–ii*)]

and since $y < r(u_1)t$:

$$= \left[0, \frac{y - r(u_0)t - (r(u_3) - r(u_0))v_3}{r(u_2) - r(u_0)} \right] \tag{L.7}$$

$$= \left[0, K_{20}(y) - \xi_{023}v_3 \right] \tag{L.8}$$

$j = 1 = i$:

$$C^1_{y,1} = \left[0, \frac{y - r(u_0)t - (r(u_2) - r(u_0))v_2 - (r(u_3) - r(u_0))v_3}{r(u_1) - r(u_0)} \right] \quad \text{[case } b\text{–}iii)] \tag{L.9}$$

$$= \left[0, K_{10}(y) - \xi_{012}v_2 - \xi_{013}v_3 \right] \tag{L.10}$$

$j = 0$:

$$C^1_{y,0} = \infty \qquad \text{[case } b\text{–}iv)] \tag{L.11}$$

$$F_Y(y) = \int_{C^1_{y,3}} \int_{C^1_{y,2}} \int_{C^1_{y,1}} f_3(v_3) f_2(v_2) f_1(v_1) dv_1 dv_2 dv_3 \tag{L.12}$$

$$= \int_0^{K_{30}(y)} \int_0^{K_{20}(y)-\xi_{023}v_3} \int_0^{K_{10}(y)-\xi_{012}v_2-\xi_{013}v_3} \lambda_3 e^{-\lambda_3 v_3} \lambda_2 e^{-\lambda_2 v_2} \lambda_1 e^{-\lambda_1 v_1} dv_1 dv_2 dv_3 \tag{L.13}$$

$$= \int_0^{K_{30}(y)} \int_0^{K_{20}(y)-\xi_{023}v_3} \lambda_3 e^{-\lambda_3 v_3} \lambda_2 e^{-\lambda_2 v_2} \left[1 - e^{-\lambda_1 [K_{10}(y)-\xi_{012}v_2-\xi_{013}v_3]}\right] dv_2 dv_3 \tag{L.14}$$

$$= \int_0^{K_{30}(y)} \int_0^{K_{20}(y)-\xi_{023}v_3} \lambda_3 e^{-\lambda_3 v_3} \left[\lambda_2 e^{-\lambda_2 v_2} - \lambda_2 e^{-\lambda_2 v_2 - \lambda_1 (K_{10}(y)-\xi_{012}v_2-\xi_{013}v_3)}\right] dv_2 dv_3 \tag{L.15}$$

$$= \int_0^{K_{30}(y)} \int_0^{K_{20}(y)-\xi_{023}v_3} \lambda_3 e^{-\lambda_3 v_3} \left[\lambda_2 e^{-\lambda_2 v_2} - B_1'(y) e^{-\lambda_1 \xi_{013} v_3} e^{-A_1' v_2}\right] dv_2 dv_3 \tag{L.16}$$

where

$$A_1' = \lambda_2 - \lambda_1 \xi_{012} \tag{L.17}$$

and

$$B_1'(y) = \lambda_2 e^{-\lambda_1 K_{10}(y)}. \tag{L.18}$$

Then,

$$F_Y(y) = \int_0^{K_{30}(y)} \lambda_3 e^{-\lambda_3 v_3} \left[1 - e^{-\lambda_2 (K_{20}(y)-\xi_{023}v_3)} - \frac{B_1'(y)}{A_1'} e^{-\lambda_1 \xi_{013} v_3} \left[1 - e^{-A_1' (K_{20}(y)-\xi_{023}v_3)}\right]\right] dv_3 \tag{L.19}$$

$$= \int_0^{K_{30}(y)} \lambda_3 e^{-\lambda_3 v_3} - \lambda_3 e^{-\lambda_3 v_3 - \lambda_2 (K_{20}(y) - \xi_{023} v_3)} - \frac{B_1'(y)\lambda_3}{A_1'} e^{-\lambda_3 v_3 - \lambda_1 \xi_{013} v_3} \tag{L.20}$$

$$+ \frac{B_1'(y)\lambda_3}{A_1'} e^{-\lambda_3 v_3 - \lambda_1 \xi_{013} v_3 - A_1' (K_{20}(y) - \xi_{023} v_3)} \, dv_3$$

$$= \int_0^{K_{30}(y)} e^{-\lambda_3 v_3} - B_2''(y) e^{-A_2'' v_3} \tag{L.21}$$

$$- B_3''(y) e^{-A_3'' v_3} + B_4''(y) e^{-A_4'' v_3} \, dv_3$$

where

$$A_2'' = \lambda_3 - \lambda_2 \xi_{023} \tag{L.22}$$

$$A_3'' = \lambda_3 + \lambda_1 \xi_{013} \tag{L.23}$$

$$A_4'' = \lambda_3 + \lambda_1 \xi_{013} - A_1' \xi_{023} \tag{L.24}$$

$$B_2''(y) = \lambda_2 \lambda_3 e^{-\lambda_2 K_{20}(y)} \tag{L.25}$$

$$B_3''(y) = \frac{B_1'(y)\lambda_3}{A_1'} = \frac{\lambda_2 \lambda_3}{A_1'} e^{-\lambda_1 K_{10}(y)} \tag{L.26}$$

and

$$B_4''(y) = \frac{B_1'(y)\lambda_3}{A_1'} e^{-A_1' K_{20}(y)} = \frac{\lambda_2 \lambda_3}{A_1'} e^{-\lambda_1 K_{10}(y) - A_1' K_{20}(y)} \tag{L.27}$$

Thus,

$$F_Y(y) = \left[1 - e^{-\lambda_3 K_{30}(y)}\right] - \frac{B_2''(y)}{A_2''}\left[1 - e^{-A_2'' K_{30}(y)}\right] - \frac{B_3''(y)}{A_3''}\left[1 - e^{-A_3'' K_{30}(y)}\right] + \frac{B_4''(y)}{A_4''}\left[1 - e^{-A_4'' K_{30}(y)}\right] \tag{L.28}$$

$$= 1 - B_1 e^{-A_1 y} - B_2(y) e^{-A_2 y} + B_3(y) e^{-A_3 y} - B_4(y) e^{-A_4 y} + B_5(y) e^{-A_5 y} + B_6(y) e^{-A_6 y} - B_7(y) e^{-A_7 y} \tag{L.29}$$

where

$$A_1 = \frac{\lambda_3}{r(u_3) - r(u_0)} \tag{L.30}$$

$$A_2 = \frac{\lambda_2}{r(u_2) - r(u_0)} \tag{L.31}$$

$$A_3 = \frac{\lambda_2}{r(u_2) - r(u_0)} + \frac{A_2''}{r(u_3) - r(u_0)} \tag{L.32}$$

$$A_4 = \frac{\lambda_1}{r(u_1) - r(u_0)} \tag{L.33}$$

$$A_5 = \frac{\lambda_1}{r(u_1) - r(u_0)} + \frac{A_3''}{r(u_3) - r(u_0)} \tag{L.34}$$

$$A_6 = \frac{\lambda_1}{r(u_1) - r(u_0)} + \frac{\lambda_2}{r(u_2) - r(u_0)} \tag{L.35}$$

$$A_7 = \frac{\lambda_1}{r(u_1) - r(u_0)} + \frac{A_1' r(u_0)}{r(u_2) - r(u_0)} + \frac{A_4''}{r(u_3) - r(u_0)} \tag{L.36}$$

$$B_1(y) = \frac{B_1''}{\lambda_3}\, e^{-\frac{\lambda_3 r(u_0)t}{r(u_3)-r(u_2)}} \tag{L.37}$$

$$B_2 = \frac{\lambda_2\lambda_3}{A_2''}\, e^{\frac{\lambda_2 r(u_0)t}{r(u_2)-r(u_0)}} \tag{L.38}$$

$$B_3(y) = \frac{\lambda_3}{A_2''}\, e^{-\frac{A_2'' r(u_0)t}{r(u_3)-r(u_2)} + \frac{\lambda_2 r(u_0)t}{r(u_2)-r(u_0)}} \tag{L.39}$$

$$B_4 = \frac{\lambda_2\lambda_3}{A_1' A_3''}\, e^{\frac{\lambda_1 r(u_0)t}{r(u_1)-r(u_0)}} \tag{L.40}$$

$$B_5(y) = \frac{\lambda_2\lambda_3}{A_1' A_3''}\, e^{-\frac{A_3'' r(u_0)t}{r(u_3)-r(u_2)} + \frac{\lambda_1 r(u_0)t}{r(u_1)-r(u_0)}} \tag{L.41}$$

$$B_6 = \frac{\lambda_2\lambda_3}{A_1' A_4''}\, e^{\frac{\lambda_1 r(u_0)t}{r(u_1)+r(u_0)} - \frac{\lambda_2 r(u_0)t}{r(u_2)-r(u_0)}} \tag{L.42}$$

$$B_7(y) = \frac{\lambda_2\lambda_3}{A_1' A_4''}\, e^{-\frac{A_4'' r(u_0)t}{r(u_3)-r(u_2)} + \frac{\lambda_1 r(u_0)t}{r(u_1)-r(u_0)} + \frac{A_{10}' r(u_0)t}{r(u_2)-r(u_0)}} \tag{L.43}$$

So, $F_Y(y)$ is the sum of exponentials:

If $y \in [r(u_0)t, r(u_1)t)$: (L.44)

$$F_Y(y) = 1 - B_1 e^{-A_1 y} - B_2 e^{-A_2 y} + B_3 e^{-A_3 y} - B_4 e^{-A_4 y} + B_5 e^{-A_5 y} + B_6 e^{-A_6 y} - B_7 e^{-A_7 y}.$$

Let $y \in [r(u_1)t, r(u_2)t)$ (thus, $l = 2$):

$i = 1 = n-2$ (i.e., checking for 1-resolvability):

$j = n = 3$:

$$C^1_{y,3} = \left(\underset{\geq 0}{} \frac{y - r(u_2)t}{r(u_3) - r(u_2)} , \frac{y - r(u_0)t}{r(u_3) - r(u_0)} \right] \quad \text{[case } b\text{-}i\text{)]} \tag{L.45}$$

and since $y < r(u_2)t$:

$$= \left[0, \frac{y - r(u_0)t}{r(u_3) - r(u_0)} \right] \tag{L.46}$$

$$= \left[0, K_{30}(y) \right] \tag{L.47}$$

$j = n - 1 = 2$:

$$C^1_{y,2} = \left(\underset{\geq 0}{} \frac{y - r(u_1)t - (r(u_3) - r(u_1))v_3}{r(u_2) - r(u_1)} , \frac{y - r(u_0)t - (r(u_3) - r(u_0))v_3}{r(u_2) - r(u_0)} \right] \tag{L.48}$$

[case b-ii)]

Consider the left-hand term above and assume it is less than 0. Then

$$\frac{y - r(u_1)t - (r(u_3) - r(u_1))v_3}{r(u_2) - r(u_1)} < 0 \Rightarrow v_3 < \frac{y - r(u_1)t}{r(u_2) - r(u_1)} \tag{L.49}$$

but $y \geq r(u_1)t$, so no such v_3 is possible. Thus, the above term is greater than or equal to zero, and so

$$C^1_{y,2} = \left(\frac{y - r(u_1)t - (r(u_3) - r(u_1))v_3}{r(u_2) - r(u_1)} , \frac{y - r(u_0)t - (r(u_3) - r(u_0))v_3}{r(u_2) - r(u_0)} \right] \tag{L.50}$$

$$= \left(K_{21}(y) - \xi_{123}v_3, K_{20}(y) - \xi_{023}v_3 \right] \tag{L.51}$$

$j = 1 = i$:

$$C^1_{y,1} = \left[0, \frac{y - r(u_0)t - (r(u_2) - r(u_0))v_2 - (r(u_3) - r(u_0))v_3}{r(u_1) - r(u_0)} \right] \text{ [case } b\text{–}iii)\text{]} \tag{L.52}$$

$$= \left[0, K_{10}(y) - \xi_{012}v_2 - \xi_{013}v_3 \right] \tag{L.53}$$

$j = 0$:

$$C^1_{y,0} = \infty \qquad [case\ b\text{–}iv)] \tag{L.54}$$

$$\int_{C^1_{y,3}} \int_{C^1_{y,2}} \int_{C^1_{y,1}} f_3(v_3) f_2(v_2) f_1(v_1) dv_1 dv_2 dv_3 \tag{L.55}$$

$$= \int_0^{K_{30}(y)} \int_{K_{21}(y) - \xi_{023}v_3}^{K_{20}(y) - \xi_{023}v_3} \int_0^{K_{10}(y) - \xi_{012}v_2 - \xi_{013}v_3} \lambda_3 e^{-\lambda_3 v_3} \lambda_2 e^{-\lambda_2 v_2} \lambda_1 e^{-\lambda_1 v_1} dv_1 dv_2 dv_3 \tag{L.56}$$

$$= \int_0^{K_{30}(y)} \int_{K_{21}(y) - \xi_{023}v_3}^{K_{20}(y) - \xi_{023}v_3} \lambda_3 e^{-\lambda_3 v_3} \lambda_2 e^{-\lambda_2 v_2} \left[1 - e^{-\lambda_2 K_{10}(y) - \xi_{012}v_2 - \xi_{013}v_3} \right] dv_2 dv_3 \tag{L.57}$$

$$= \int_0^{K_{30}(y)} \int_{K_{21}(y) - \xi_{023}v_3}^{K_{20}(y) - \xi_{023}v_3} \lambda_3 e^{-\lambda_3 v_3} \left[\lambda_2 e^{-\lambda_2 v_2} - \lambda_2 e^{-\lambda_2 v_2 - \lambda_1 (K_{10}(y) - \xi_{012}v_2 - \xi_{013}v_3)} \right] dv_2 dv_3 \tag{L.58}$$

$$= \int_0^{K_{30}(y)} \int_{K_{21}(y)-\xi_{023}v_3}^{K_{20}(y)-\xi_{023}v_3} \lambda_3 e^{-\lambda_3 v_3}\left[\lambda_2 e^{-\lambda_2 v_2} - B_1'(y) e^{-\lambda_1 \xi_{013} v_3} e^{-A_1' v_2}\right] dv_2 dv_3 \tag{L.59}$$

$$= \int_0^{K_{30}(y)} \lambda_3 e^{-\lambda_3 v_3}\left[e^{-\lambda_2(K_{21}(y)-\xi_{023}v_3)} - e^{-\lambda_2(K_{20}(y)-\xi_{023}v_3)}\right.$$

$$\left. - \frac{B_1'(y)}{A_1'} e^{-\lambda_1 \xi_{013} v_3}\left[e^{-A_1'(K_{21}(y)-\xi_{023}v_3)} - e^{-A_1'(K_{20}(y)-\xi_{023}v_3)}\right]\right] dv_3 \tag{L.60}$$

$$= \int_0^{K_{30}(y)} \lambda_3 e^{-\lambda_3 v_3 - \lambda_2(K_{21}(y)-\xi_{023}v_3)} - \lambda_3 e^{-\lambda_3 v_3 - \lambda_2(K_{20}(y)-\xi_{023}v_3)}$$

$$- \frac{B_1'(y)\lambda_3}{A_1'} e^{-\lambda_3 v_3 - A_1'(K_{21}(y)-\xi_{023}v_3) - \lambda_1 \xi_{013} v_3} \tag{L.61}$$

$$+ \frac{B_1'(y)\lambda_3}{A_1'} e^{-\lambda_3 v_3 - \lambda_1 \xi_{013} v_3 - A_1'(K_{20}(y)-\xi_{023}v_3)} dv_3$$

$$= \int_0^{K_{30}(y)} B_5''(y) e^{-A_5'' v_3} - B_2''(y) e^{-A_2'' v_3}$$

$$- B_6''(y) e^{-A_6'' v_3} + B_4''(y) e^{-A_4'' v_3} dv_3 \tag{L.62}$$

where

$$A_5'' = \lambda_3 - A_1' \xi_{023} \tag{L.63}$$

$$A_6'' = \lambda_3 + \lambda_1 \xi_{013} - A_1' \xi_{023} \tag{L.64}$$

$$B_5''(y) = \lambda_3 e^{-\lambda_2 K_{21}(y)} \tag{L.65}$$

and

$$B_6''(y) = \frac{B_1'(y)\lambda_3}{A_1'} e^{-A_1' K_{21}(y)} = \frac{\lambda_2 \lambda_3}{A_1'} e^{-\lambda_1 K_{10}(y) - A_1' K_{21}(y)} \tag{L.66}$$

Thus,

$$\int_{C_{y,3}^1} \int_{C_{y,2}^1} \int_{C_{y,1}^1} f_3(v_3) f_2(v_2) f_1(v_1) dv_1 dv_2 dv_3$$

$$= \frac{B_5''}{A_5''} \left[1 - e^{-A_5'' K_{30}(y)}\right] - \frac{B_2''(y)}{A_2''} \left[1 - e^{-A_2'' K_{30}(y)}\right] \tag{L.67}$$

$$- \frac{B_6''(y)}{A_6''} \left[1 - e^{-A_6'' K_{30}(y)}\right] + \frac{B_4''(y)}{A_4''} \left[1 - e^{-A_4'' K_{30}(y)}\right]$$

$$= B_8 e^{-A_8 y} - B_9 e^{-A_9 y} - B_2 e^{-A_2 y} + B_3 e^{-A_3 y}$$

$$- B_{10} e^{-A_{10} y} + B_{11} e^{-A_{11} y} + B_6 e^{-A_6 y} - B_7 e^{-A_7 y} \tag{L.68}$$

where

$$A_8 = \frac{\lambda_2}{r(u_2) - r(u_1)} \tag{L.69}$$

$$A_9 = \frac{\lambda_2}{r(u_2) - r(u_1)} + \frac{A_5''}{r(u_3) - r(u_0)} \tag{L.70}$$

$$A_{10} = \frac{\lambda_1}{r(u_1) - r(u_0)} + \frac{A_1'}{r(u_2) - r(u_1)} \tag{L.71}$$

$$A_{11} = \frac{\lambda_1}{r(u_1) - r(u_0)} + \frac{A_1'}{r(u_3) - r(u_0)} + \frac{A_6''}{r(u_3) - r(u_0)} \tag{L.72}$$

$$B_8 = \frac{\lambda_3}{A_2''} e^{\frac{\lambda_2 r(u_1) t}{r(u_2) - r(u_1)}} \tag{L.73}$$

$$B_9 = \frac{\lambda_3}{A_5'' A_1'} e^{\frac{A_5'' r(u_0) t}{r(u_3) - r(u_2)} + \frac{\lambda_2 r(u_1) t}{r(u_2) - r(u_1)}} \tag{L.74}$$

$$B_{10} = \frac{\lambda_2 \lambda_3}{A_1' A_4''} e^{\frac{\lambda_1 r(u_0) t}{r(u_1) - r(u_0)} + \frac{A_1' r(u_1) t}{r(u_2) - r(u_1)}} \tag{L.75}$$

$$B_{11} = \frac{\lambda_2 \lambda_3}{A_6'' A_1'} e^{\frac{A_6'' r(u_0)}{r(u_3) - r(u_2)} + \frac{\lambda_1 r(u_2)}{r(u_3) - r(u_2)} + \frac{A_1' r(u_1)}{r(u_2) - r(u_1)}} . \tag{L.76}$$

So, $\int_{C^1_{y,3}} \int_{C^1_{y,2}} \int_{C^1_{y,1}} f_3(v_3) f_2(v_2) f_1(v_1) dv_1 dv_2 dv_3$ is the sum of exponentials:

$$\int_{C^1_{y,3}} \int_{C^1_{y,2}} \int_{C^1_{y,1}} f_3(v_3) f_2(v_2) f_1(v_1) dv_1 dv_2 dv_3 \tag{L.77}$$
$$= B_7 e^{-A_8 y} - B_9 e^{-A_9 y} - B_2 e^{-A_2 y} + B_3 e^{-A_3 y}$$
$$- B_{10} e^{-A_{10} y} + B_{11} e^{-A_{11} y} + B_6 e^{-A_6 y} - B_7 e^{-A_7 y} .$$

$i = 2 = n-1$ (i.e., checking for 2-resolvability):

$j = n = 3$:

$$C^2_{y,3} = \left(\frac{y - r(u_2)t}{r(u_3) - r(u_2)}, \frac{y - r(u_1)t}{r(u_3) - r(u_1)} \right]_{\geq 0} \quad [\text{case } b\text{-}i)] \tag{L.78}$$

and since $y < r(u_2)t$:

$$= \left[0, \frac{y - r(u_1)t}{r(u_3) - r(u_1)} \right] \tag{L.79}$$

$$= \left[0, K_{31}(y) \right] \tag{L.80}$$

$j = n - 1 = 2$:

$$C^2_{y,2} = \left[0, \frac{y - r(u_1)t - (r(u_3) - r(u_1))v_3}{r(u_2) - r(u_1)} \right] \quad [\text{case } b\text{-}iii)] \tag{L.81}$$

$$= \left[0, K_{21}(y) - \xi_{123}v_3 \right] \tag{L.82}$$

$j = 1$:

$$C^2_{y,1} = [0, \infty) \quad [\text{case } b\text{-}iv)] \tag{L.83}$$

$j = 0$:

$$C^2_{y,0} = \infty \qquad [\text{case } b\text{-}iv)] \tag{L.84}$$

$$\int_{C^2_{y,3}} \int_{C^2_{y,2}} f_3(v_3) f_2(v_2) dv_2 dv_3 = \int_0^{K_{31}(y)} \int_0^{K_{21}(y) - \xi_{123}v_3} \lambda_3 e^{-\lambda_3 v_3} \lambda_2 e^{-\lambda_2 v_2} dv_2 dv_3 \tag{L.85}$$

$$= \int_0^{K_{31}(y)} \lambda_3 e^{-\lambda_3 v_3}\left[1 - e^{-\lambda_2(K_{21}(y) - \xi_{123} v_3)}\right] dv_3 \tag{L.86}$$

$$= \int_0^{K_{31}(y)} \lambda_3 e^{-\lambda_3 v_3} - B_2'(y) e^{-A_2' v_3} dv_3 \tag{L.87}$$

where

$$A_2' = \lambda_3 - \lambda_2 \xi_{123} \tag{L.88}$$

$$B_2'(y) = \lambda_3 e^{-\lambda_2 K_{21}(y)} \tag{L.89}$$

Then,

$$\int_{C^2_{y,3}} \int_{C^2_{y,2}} f_3(v_3) f_2(v_2) dv_2 dv_3 = 1 - e^{-\lambda_3 K_{31}(y)} - \frac{B_2'(y)}{A_2'}\left[1 - e^{-A_2' K_{31}(y)}\right] \tag{L.90}$$

$$= 1 - B_{12}(y) e^{-A_{12} y} - B_{13}(y) e^{-A_{13} y} + B_{14}(y) e^{-A_{14} y} \tag{L.91}$$

where

$$A_{12} = \frac{\lambda_3}{r(u_3) - r(u_1)} \tag{L.92}$$

$$A_{13} = \frac{\lambda_2}{r(u_2) - r(u_1)} \tag{L.93}$$

$$A_{14} = \frac{\lambda_2}{r(u_2) - r(u_1)} + \frac{A_2}{r(u_3) - r(u_1)} \tag{L.94}$$

$$B_{12} = e^{\frac{\lambda_3 r(u_1)t}{r(u_3) - r(u_1)}} \tag{L.95}$$

$$B_{13} = \frac{\lambda_3}{A_2'} e^{\frac{\lambda_2 r(u_1)t}{r(u_2) - r(u_1)}} \tag{L.96}$$

$$B_{14} = \frac{\lambda_2}{A_2'} e^{\frac{\lambda_2 r(u_1)t}{r(u_2) - r(u_1)} + \frac{A_2' r(u_1)t}{r(u_3) - r(u_1)t}} \tag{L.97}$$

So, $\int_{C^2_{y,3}} \int_{C^2_{y,2}} f_3(v_3) f_2(v_2) dv_2 dv_3$ is the sum of exponentials:

$$\int_{C^2_{y,3}} \int_{C^2_{y,2}} f_3(v_3) f_2(v_2) dv_2 dv_3 = 1 - B_{12} e^{-A_{12}y} - B_{13} e^{-A_{13}y} + B_{14} e^{-A_{14}y} . \tag{L.98}$$

If $y \in [r(u_1)t, r(u_2)t)$: (L.99)

$$F_Y(y) = 1 - B_2 e^{-A_2 y} + B_3 e^{-A_3 y} + B_8 e^{-A_8 y} - B_9 e^{-A_9 y} + B_{10} e^{-A_{10} y} - B_{11} e^{-A_{11} y} - B_{12} e^{-A_{12} y} - B_{13} e^{-A_{13} y} + B_{14} e^{-A_{14} y} .$$

Let $y \in [r(u_2)t, r(u_3)t)$ (thus, $l = 3$):

$i = 1 = n-2$ (i.e., checking for 1-resolvability):

$i = n = 3$:

$$C^1_{y,3} = \left(\underset{\geq 0}{} \frac{y - r(u_2)t}{r(u_3) - r(u_2)}, \frac{y - r(u_0)t}{r(u_3) - r(u_0)} \right] \quad \text{[case } b\text{–}i\text{)]} \tag{L.100}$$

and since $y \geq r(u_2)t$:

$$= \left[\frac{y - r(u_2)t}{r(u_3) - r(u_2)}, \frac{y - r(u_0)t}{r(u_3) - r(u_0)} \right] \tag{L.101}$$

$$= \left[K_{32}(y),\ K_{30}(y) \right] \tag{L.102}$$

$j = n - 1 = 2$:

$$C^1_{y,2} = \left(\underset{\geq 0}{} \frac{y - r(u_1)t - (r(u_3) - r(u_1))v_3}{r(u_2) - r(u_1)}, \frac{y - r(u_0)t - (r(u_3) - r(u_0))v_3}{r(u_2) - r(u_0)} \right] \tag{L.103}$$

[case b–ii)]

and from Eq. L.49

$$C^1_{y,2} = \left(\frac{y - r(u_1)t - (r(u_3) - r(u_1))v_3}{r(u_2) - r(u_1)}, \frac{y - r(u_0)t - (r(u_3) - r(u_0))v_3}{r(u_2) - r(u_0)} \right] \tag{L.104}$$

$$= \left(K_{21}(y) - \xi_{123}v_3,\ K_{20}(y) - \xi_{023}v_3 \right] \tag{L.105}$$

$j = 1 = i$:

$$C^1_{y,1} = \left[0, \frac{y - r(u_0)t - (r(u_2) - r(u_0))v_2 - (r(u_3) - r(u_0))v_3}{r(u_1) - r(u_0)} \right] \quad \text{[case } b\text{–}iii\text{)]} \tag{L.106}$$

$$= \left[0, K_{10}(y) - \xi_{012}v_2 - \xi_{013}v_3 \right] \tag{L.107}$$

$j = 0$:

$$C^1_{y,0} = \infty \qquad [case\ b\text{–}iv)] \tag{L.108}$$

$$\int_{C^1_{y,3}} \int_{C^1_{y,2}} \int_{C^1_{y,1}} f_3(v_3) f_2(v_2) f_1(v_1) dv_1 dv_2 dv_3 \tag{L.109}$$

$$= \int_{K_{32}(y)}^{K_{30}(y)} \int_{K_{21}(y) - \xi_{023} v_3}^{K_{20}(y) - \xi_{023} v_3} \int_0^{K_{10}(y) - \xi_{012} v_2 - \xi_{013} v_3} \lambda_3 e^{-\lambda_3 v_3} \lambda_2 e^{-\lambda_2 v_2} \lambda_1 e^{-\lambda_1 v_1} dv_1 dv_2 dv_3 \tag{L.110}$$

and following the pattern of Eqs. L.57-L.62,

$$= \int_{K_{32}(y)}^{K_{30}(y)} B_5''(y) e^{-A_5'' v_3} - B_2''(y) e^{-A_2'' v_3} - B_6''(y) e^{-A_6'' v_3} + B_4''(y) e^{-A_4'' v_3} \, dv_3 \tag{L.111}$$

$$= \frac{B_5''}{A_5''}\left[e^{-A_5'' K_{32}(y)} - e^{-A_5'' K_{30}(y)}\right]$$

$$- \frac{B_2''(y)}{A_2''}\left[e^{-A_2'' K_{32}(y)} - e^{-A_2'' K_{30}(y)}\right]$$

$$- \frac{B_6''(y)}{A_6''}\left[e^{-A_6'' K_{32}(y)} - e^{-A_6'' K_{30}(y)}\right] \tag{L.112}$$

$$+ \frac{B_4''(y)}{A_4''}\left[e^{-A_4'' K_{32}(y)} - e^{-A_4'' K_{30}(y)}\right]$$

$$= B_{15}e^{-A_{15}''' y} - B_9 e^{-A_9''' y} - B_{16}e^{-A_{16}''' y} + B_3 e^{-A_3''' y}$$

$$- B_{17}e^{-A_{17}''' y} + B_{11}e^{-A_{11}''' y} + B_{18}e^{-A_{18}''' y} - B_7 e^{-A_7''' y} \tag{L.113}$$

where

$$A_{15} = \frac{\lambda_2}{r(u_2) - r(u_1)} + \frac{A_5''}{r(u_3) - r(u_2)} \tag{L.114}$$

$$A_{16} = \frac{\lambda_2}{r(u_2) - r(u_1)} + \frac{A_2''}{r(u_3) - r(u_0)} \tag{L.115}$$

$$A_{17} = \frac{\lambda_1}{r(u_1) - r(u_0)} + \frac{A_1'}{r(u_2) - r(u_1)} + \frac{A_6''}{r(u_3) - r(u_2)} \tag{L.116}$$

$$A_{18} = \frac{\lambda_1}{r(u_1) - r(u_0)} + \frac{A_1'}{r(u_2) - r(u_0)} + \frac{A_4''}{r(u_3) - r(u_2)} \tag{L.117}$$

$$B_{15} = \frac{\lambda_3}{A_5''} e^{\frac{A_5'' r(u_2)t}{r(u_3)-r(u_2)} + \frac{\lambda_2 r(u_1)t}{r(u_2)-r(u_1)}} \tag{L.118}$$

$$B_{16} = \frac{\lambda_3}{A_2''} e^{\frac{A_2'' r(u_2)t}{r(u_3)-r(u_2)} + \frac{\lambda_2 r(u_0)t}{r(u_2)-r(u_0)}} \tag{L.119}$$

$$B_{17} = \frac{\lambda_2\lambda_3}{A_1' A_6''} e^{\frac{A_6'' r(u_2)t}{r(u_3)-r(u_2)} + \frac{\lambda_1 r(u_0)t}{r(u_1)-r(u_0)} + \frac{A_1' r(u_1)t}{r(u_2)-r(u_1)}} \tag{L.120}$$

and

$$B_{18} = \frac{\lambda_2\lambda_3}{A_1' A_4''} e^{\frac{A_4'' r(u_2)t}{r(u_3)-r(u_2)} + \frac{\lambda_1 r(u_0)t}{r(u_1)-r(u_0)} + \frac{A_1' r(u_0)t}{r(u_2)-r(u_0)}} \tag{L.121}$$

So, $\int_{C^1_{y,3}} \int_{C^1_{y,2}} \int_{C^1_{y,1}} f_3(v_3) f_2(v_2) f_1(v_1) dv_1 dv_2 dv_3$ is the sum of exponentials:

$$\int_{C^1_{y,3}} \int_{C^1_{y,2}} \int_{C^1_{y,1}} f_3(v_3) f_2(v_2) f_1(v_1) dv_1 dv_2 dv_3 \tag{L.122}$$
$$= B_{15} e^{-A_{15}y} - B_9 e^{-A_9 y} - B_{16} e^{-A_{16}y} + B_3 e^{-A_3 y}$$
$$- B_{17} e^{-A_{17}y} + B_{11} e^{-A_{11}y} + B_{18} e^{-A_{18}y} - B_7 e^{-A_7 y} .$$

$i = 2 = n-1$ (i.e., checking for 2-resolvability):

$j = n = 3$:

$$C^2_{y,3} = \left(\underset{\geq 0}{} \frac{y - r(u_2)t}{r(u_3) - r(u_2)} , \frac{y - r(u_1)t}{r(u_3) - r(u_1)} \right] \quad \text{[case } b\text{-}i\text{)]} \tag{L.123}$$

and since $y \geq r(u_2)t$:

$$= \left(\frac{y - r(u_2)t}{r(u_3) - r(u_2)}, \frac{y - r(u_1)t}{r(u_3) - r(u_1)} \right] \tag{L.124}$$

$$= \left(K_{32}(y), K_{31}(y) \right] \tag{L.125}$$

$j = n - 1 = 2$:

$$C^2_{y,2} = \left[0, \frac{y - r(u_1)t - (r(u_3) - r(u_1))v_3}{r(u_2) - r(u_1)} \right] \quad [\text{case } b\text{–}iii)] \tag{L.126}$$

$$= \left[0, K_{21}(y) - \xi_{123}v_3 \right] \tag{L.127}$$

$j = 1$:

$$C^2_{y,1} = [0,\infty) \quad [\text{case } b\text{–}iv)] \tag{L.128}$$

$j = 0$:

$$C^2_{y,0} = \infty \qquad [\text{case } b\text{–}iv)] \tag{L.129}$$

$$\int_{C^2_{y,3}} \int_{C^2_{y,2}} f_3(v_3) f_2(v_2) dv_2 dv_3 = \int_{K_{32}(y)}^{K_{31}(y)} \int_0^{K_{21}(y) - \xi_{123}v_3} \lambda_3 e^{-\lambda_3 v_3} \lambda_2 e^{-\lambda_2 v_2} dv_2 dv_3 \tag{L.130}$$

$$= \int_{K_{32}(y)}^{K_{31}(y)} \lambda_3 e^{-\lambda_3 v_3} \left[1 - e^{-\lambda_2 (K_{21}(y) - \xi_{123}v_3)} \right] dv_3 \tag{L.131}$$

$$= \int_{K_{32}(y)}^{K_{31}(y)} \lambda_3 e^{-\lambda_3 v_3} - B_2'(y) e^{-A_2' v_3} dv_3 \tag{L.132}$$

$$= e^{-\lambda_3 K_{32}(y)} - e^{-\lambda_3 K_{31}(y)} - \frac{B_2'(y)}{A_2'} \left[e^{-\lambda_3 K_{32}(y)} - e^{-A_2' K_{31}(y)} \right] \tag{L.133}$$

$$B_{19}(y) e^{-A_{19} y} - B_{12}(y) e^{-A_{12} y} - B_{20}(y) e^{-A_{20} y} + B_{14}(y) e^{-A_{14} y} \tag{L.134}$$

where

$$A_{19} = \frac{\lambda_3}{r(u_3) - r(u_2)} \tag{L.135}$$

$$A_{20} = \frac{\lambda_3}{r(u_3) - r(u_2)} + \frac{\lambda_2}{r(u_2) - r(u_1)} \tag{L.136}$$

$$B_{19} = e^{\frac{\lambda_3 r(u_2) t}{r(u_3) - r(u_2)}} \tag{L.137}$$

$$B_{20} = \frac{\lambda_3}{A_2'} e^{\frac{\lambda_2 r(u_1) t}{r(u_2) - r(u_1)}} \tag{L.138}$$

So, $\int_{C_{y,3}^2} \int_{C_{y,2}^2} f_3(v_3) f_2(v_2) dv_2 dv_3$ is the sum of exponentials:

$$\int_{C_{y,3}^2} \int_{C_{y,2}^2} f_3(v_3) f_2(v_2) dv_2 dv_3 = B_{19} e^{-A_{19} y} - B_{12} e^{-A_{12} y} - B_{20} e^{-A_{20} y} + B_{14} e^{-A_{14} y} . \tag{L.139}$$

$i = 3 = n$ (i.e., checking for 3-resolvability):

$j = n = 3$:

$$C^3_{y,3} = \left[0, \frac{y - r(u_2)t}{r(u_3) - r(u_2)} \right] \qquad [\text{case } a\text{–}i)] \tag{L.140}$$

$j = 2$:

$$C^3_{y,2} = [0,\infty) \quad [\text{case } a\text{–}ii)] \tag{L.141}$$

$j = 1$:

$$C^3_{y,1} = [0,\infty) \quad [\text{case } a\text{–}ii)] \tag{L.142}$$

$j = 0$:

$$C^3_{y,0} = \infty \qquad [\text{case } a\text{–}iii)] \tag{L.143}$$

$$\int_{C^3_{y,3}} f_3(v_3)dv_3 = \int_0^{K_{32}(y)} \lambda_3 e^{-\lambda_3 v_3} dv_3 \tag{L.144}$$

$$= 1 - e^{-\lambda_3 K_{32}(y)} \tag{L.145}$$

$$= 1 - A_{19} e^{-A_{19} y} \tag{L.146}$$

So, $\int_{C^3_{y,2}} C^2_{y,2} \; f_3(v_3)v_3$ is the sum of exponentials:

$$\int_{\mathcal{C}^3_{y,3}} f_3(v_3)v_3 = 1 - B_{19}e^{-A_{19}y} . \tag{L.147}$$

If $y \in [r(u_2)t, r(u_3)t)$: (L.148)

$$\begin{aligned} F_Y(y) = 1 &+ B_3e^{-A_3y} - B_7e^{-A_7y} - B_9e^{-A_9y} \\ &+ B_{11}e^{-A_{11}y} - B_{12}e^{-A_{12}y} + B_{14}e^{-A_{14}y} - B_{16}e^{-A_{16}y} \\ &- B_{17}e^{-A_{17}y} + B_{18}e^{-A_{18}y} - B_{20}e^{-A_{20}y} . \end{aligned}$$

Finally, let $y \in [r(u_3)t, \infty)$. Then

If $y \in [r(u_3)t, \infty)$: (L.149)

$$F_Y(y) = 1 .$$

APPENDIX M

Recursively derived solution of a simple 3-state, acyclic, nonrecoverable proce

This appendix contains the recursive derivation of the distribution of reward for a simple markovian three-state, acyclic, nonrecoverable process, where $r(u_2) > r(u_1) > r(u_0)$. See Sections 5.3.10 [Recursive Formulation of $F_{Y|U}$], 5.3.9.2 [Three State Markovian Acyclic Process], and 5.3.10.1.2 [Three State Markovian Acyclic Process], and Fig. 5.4. The regions $C_{y,j}^{1,2;r(u_2)}$ are explicitly stated and the appropriate cases of Theorem 5.25 iii) are indicated in brackets. Many derivations are detailed in Appendix [Closed-form solution of a simple 3-state; numbers of the corresponding Appendix K equations are also indicated in brackets.

Let $y \in [r(u_0)t, r(u_1)t)$ (thus, $l = 1$):

$j = n = 2$:

$$C_{y,2}^{1,2;r(u_2)} = \left[\; 0, \; \frac{y - r(u_0)t}{r(u_2) - r(u_0)} \; \right] \text{ [Eq. } K.2\text{]} \tag{M.1}$$

$j = 1 = i$:

$$C_{y,1}^{1,2;r(u_2)} = \left[\; 0, \; \frac{y - r(u_0)t - (r(u_2) - r(u_0))v_2}{r(u_1) - r(u_0)} \; \right] \quad [case\ b\text{–}iii)] \tag{M.2}$$

$j = 0$:

$$C_{y,0}^{1,2;r(u_2)} = [0, \infty) \quad [case\ b\text{–}iv)] \tag{M.3}$$

$$F_{Y|U}^{2;\boldsymbol{\lambda}^1}(y|\ (u_2,u_1,u_0)) = F_Y(y) = \int\limits_{C_{y,2}^{1,2;r(u_2)}} \int\limits_{C_{y,1}^{1,2;r(u_2)}} f_2^{\boldsymbol{\lambda}^2}(v_2) f_1^{\boldsymbol{\lambda}^2}(v_1)\, dv_1 dv_2 \tag{M.4}$$

$$= \int\limits_0^{\frac{y-r(u_0)t}{r(u_2)-r(u_0)}} \int\limits_0^{\frac{y-r(u_0)t-(r(u_2)-r(u_0))v_2}{r(u_1)-r(u_0)}} \lambda_2 e^{-\lambda_2 v_2} \lambda_1 e^{-\lambda_1 v_1} dv_1 dv_2 \tag{M.5}$$

and from Eq. FYS1i1d:

If $y \in [r(u_0)t, r(u_1)t)$ (M.6)

$$F_{Y|U}^{1;\boldsymbol{\lambda}^2}(y|\ (u_2,u_1,u_0)) = F_Y(y)$$

$$= 1 - e^{-\frac{\lambda_2(y-r(u_0)t)}{r(u_2)-r(u_0)}} + \frac{\lambda_2(r(u_1)-r(u_0))}{\lambda_1(r(u_2)-r(u_0)) + \lambda_2(r(u_1)-r(u_0))}$$

$$e^{-\frac{\lambda_1(y-r(u_0)t)}{r(u_1)-r(u_0)}} \left[1 - e^{-\frac{\left[\lambda_1(r(u_2)-r(u_0)) + \lambda_2(r(u_1)-r(u_0))\right](y-r(u_0)t)}{r(u_1)-r(u_0)}}\right].$$

Let $y \in [r(u_1)t, r(u_2)t)$ (thus, $l = 2$):

$j = 2$:

$$C_{y,2}^{1,2;r(u_2)} = \left[\ \frac{y-r(u_1)t}{r(u_2)-r(u_1)}\ ,\ \frac{y-r(u_0)t}{r(u_2)-r(u_0)}\ \right) \text{ [Eq. } K.16\text{]} \tag{M.7}$$

$j = 1$:

$$C_{y,1}^{1,2;r(u_2)} = \left[\ 0,\ \frac{y-r(u_0)t-(r(u_2)-r(u_0))v_2}{r(u_1)-r(u_0)}\ \right] \text{ [Eq. } K.19\text{]} \tag{M.8}$$

$j = 0$:

$$C_{y,0}^{1,2;r(u_2)} = [0,\infty) \qquad [\text{case } b\text{-}iii)] \tag{M.9}$$

$$\int_{C_{y,1}^{2,2;r(u_2)}} \int_{C_{y,1}^{2,2;r(u_2)}} f_2^{\lambda^2}(v_2) f_1^{\lambda^2}(v_1) dv_1 dv_2 \tag{M.10}$$

$$= \int_{\frac{y - r(u_1)t}{r(u_2) - r(u_1)}}^{\frac{y - r(u_0)t}{r(u_2) - r(u_0)}} \int_{0}^{\frac{y - r(u_0)t - (r(u_2) - r(u_0))v_2}{r(u_1) - r(u_0)}} \lambda_2 e^{-\lambda_2 v_2} \lambda_1 e^{-\lambda_1 v_1} dv_1 dv_2 \tag{M.11}$$

$$= e^{-\frac{\lambda_2(y - r(u_1)t)}{r(u_2) - r(u_1)}} - e^{-\frac{\lambda_2(y - r(u_0))}{r(u_2) - r(u_0)}}$$

$$+ \frac{(r(u_1) - r(u_0))\lambda_2}{\lambda_2(r(u_1) - r(u_0)) - \lambda_1(r(u_2) - r(u_0))} e^{-\frac{\lambda_1(y - r(u_0))}{r(u_2) - r(u_0)}}$$

$$\left[e^{-\frac{(y - r(u_0))\left[\lambda_2(r(u_1) - r(u_0)) - \lambda_1(r(u_2) - r(u_0))\right]}{(r(u_1) - r(u_0))(r(u_2) - r(u_0))}} \right. \tag{M.12}$$

$$\left. - e^{-\frac{(y - r(u_1))\left[\lambda_2(r(u_1) - r(u_0)) - \lambda_1(r(u_2) - r(u_0))\right]}{(r(u_1) - r(u_0))(r(u_2) - r(u_0))}} \right]$$

[Eq. *Tx220integratione*]

$$F_{Y|U}^{1;\lambda^2}(y|\ (u_2, u_1, u_0)) = F_Y(y) = 1 - e^{-\frac{\lambda_2(y - r(u_1)t)}{r(u_2) - r(u_1)}} \tag{M.13}$$

Therefore,

$$F^{2;\boldsymbol{\lambda}^2}_{Y|U}(y|\ (u_2,u_1,u_0)) = F_Y(y)$$

$$= \int_{C^{1,2;r(u_2)}_{y,2}} \int_{C^{1,2;r(u_2)}_{y,1}} f_2^{\boldsymbol{\lambda}^2}(v_2) f_1^{\boldsymbol{\lambda}^2}(v_1)\,dv_1\,dv_2 + F^{2;\boldsymbol{\lambda}^1}_{Y|U}(y|\ (u_2,u_1,u_0)) \tag{M.14}$$

$$= 1 - e^{-\frac{\lambda_2(y - r(u_1)t)}{r(u_2)-r(u_1)}} + e^{-\frac{\lambda_2(y - r(u_1)t)}{r(u_2)-r(u_1)}} - e^{-\frac{\lambda_2(y - r(u_0))}{r(u_2)-r(u_0)}}$$

$$+ \frac{(r(u_1)-r(u_0))\lambda_2}{\lambda_2(r(u_1)-r(u_0)) - \lambda_1(r(u_2)-r(u_0))}\ e^{-\frac{\lambda_1(y - r(u_0))}{r(u_2)-r(u_0)}}$$

$$\left[e^{-\frac{(y - r(u_0))\left[\lambda_2(r(u_1)-r(u_0)) - \lambda_1(r(u_2)-r(u_0))\right]}{(r(u_1)-r(u_0))(r(u_2)-r(u_0))}} \right. \tag{M.15}$$

$$\left. - e^{-\frac{(y - r(u_1))\left[\lambda_2(r(u_1)-r(u_0)) - \lambda_1(r(u_2)-r(u_0))\right]}{(r(u_1)-r(u_0))(r(u_2)-r(u_0))}} \right]$$

$$= 1 - e^{-\frac{\lambda_2(y - r(u_0))}{r(u_2)-r(u_0)}} + \frac{(r(u_1)-r(u_0))\lambda_2}{\lambda_2(r(u_1)-r(u_0)) - \lambda_1(r(u_2)-r(u_0))}$$

$$e^{-\frac{\lambda_1(y - r(u_0))}{r(u_2)-r(u_0)}} \left[e^{-\frac{(y - r(u_0))\left[\lambda_2(r(u_1)-r(u_0)) - \lambda_1(r(u_2)-r(u_0))\right]}{(r(u_1)-r(u_0))(r(u_2)-r(u_0))}} \right. \tag{M.16}$$

$$\left. - e^{-\frac{(y - r(u_1))\left[\lambda_2(r(u_1)-r(u_0)) - \lambda_1(r(u_2)-r(u_0))\right]}{(r(u_1)-r(u_0))(r(u_2)-r(u_0))}} \right]$$

For $y \geq r(u_2)t$:

$$F^{2;\boldsymbol{\lambda}^2}_{Y|U}(y|\ (u_2,u_1,u_0)) = F_Y(y) = 1 \tag{M.17}$$

APPENDIX N

Recursively derived solution of a simple 4-state, acyclic, nonrecoverable proc

This appendix contains the recursive derivation of the distribution of reward for a simple markovian four-state, acyclic, nonrecoverable process, where $r(u_3) > r(u_2) > r(u_1) > r(u_0)$. See Sections 5.3.10 [Recursive Formulation of $F_{Y|U}$], 5.3.9.3 [Four State Markovian Acyclic Process], and 5.3.10.1.3 [Four State Markovian Acyclic Process], and Fig. 5.5. The regions $C_{y,j}^{1,3;r(u_3)}$ are explicitly stated and the appropriate cases of Theorem 5.25 iii) are indicated in brackets. Many derivations are detailed in Appendix L [Closed-form solution of a simple 4-state; numbers of the corresponding Appendix L equations are also indicated in brackets. Intermediate variables such as A_1 and B_1 are as defined in Appendix L.

To help simplify the manipulations, we use the notation:

$$K_{ij}(y) = \frac{y - r(u_j)t}{r(u_i) - r(u_j)} \tag{N.1}$$

and

$$\xi_{ijk} = \frac{r(u_k) - r(u_i)}{r(u_j) - r(u_i)} \quad . \tag{N.2}$$

Let $y \in [r(u_0)t, r(u_1)t)$ (thus, $l = 1$):

$j = n = 3$:

$$C_{y,3}^{1,3;r(u_3)} = \left[\, 0, K_{30}(y) \,\right] \quad \text{[Eq } L.5\text{]} \tag{N.3}$$

$j = n - 1 = 2$:

$$C_{y,2}^{1,3;r(u_3)} = \left[\, 0, K_{20}(y) - \xi_{023} v_3 \,\right] \quad \text{[Eq. } L.8\text{]} \tag{N.4}$$

$j = 1 = i$:

$$C_{y,1}^{1,3;r(u_3)} = \left[\, 0, K_{10}(y) - \xi_{012} v_2 - \xi_{013} v_3 \,\right] \quad \text{[Eq. } L.10\text{]} \tag{N.5}$$

$j = 0$:

$$C_{y,0}^{1,3;r(u_3)} = \infty \qquad [case\ b\text{–}iv)] \tag{N.6}$$

If $y \in [r(u_0)t, r(u_1)t)$: (N.7)

$$F_{\eta\, U}^{3;\,\lambda^3}(y|\ (u_3, u_2, u_1, u_0) = 1 - B_1 e^{-A_1 y} - B_2 e^{-A_2 y} + B_3 e^{-A_3 y}) \\ - B_4 e^{-A_4 y} + B_5 e^{-A_5 y} + B_6 e^{-A_6 y} - B_7 e^{-A_7 y}\,.$$

Let $y \in [r(u_1)t\#_{\alpha(N\,8)}$ (thus, $l = 2$):

$j = n = 3$:

$$C_{y,3}^{1,3;r(u_3)} = \left[\, 0, K_{30}(y) \,\right] \quad \text{[Eq. } L.47\text{]} \tag{N.9}$$

$j = n - 1 = 2$:

$$C_{y,2}^{1,3;r(u_3)} = \left(\, K_{21}(y) - \xi_{123} v_3,\ K_{20}(y) - \xi_{023} v_3 \,\right] \quad \text{[Eq. } L.51\text{]} \tag{N.10}$$

$j = 1 = i$:

$$C_{y,1}^{1,3;r(u_3)} = \left[0, K_{10}(y) - \xi_{012}v_2 - \xi_{013}v_3 \right] \text{ [Eq. } L.53\text{]} \tag{N.11}$$

$j = 0$:

$$C_{y,0}^{1,3;r(u_3)} = \infty \qquad [case\ b\text{-}iv)] \tag{N.12}$$

$$\int_{C_{y,3}^{1,3;r(u_3)}} \int_{C_{y,2}^{1,3;r(u_3)}} \int_{C_{y,1}^{1,3;r(u_3)}} f_3(v_3)f_2(v_2)f_1(v_1)dv_1dv_2dv_3 \tag{N.13}$$
$$= B_8e^{-A_8y} - B_9e^{-A_9y} - B_2e^{-A_2y} + B_3e^{-A_3y}$$
$$- B_{10}e^{-A_{10}y} + B_{11}e^{-A_{11}y} + B_6e^{-A_6y} - B_7e^{-A_7y}.$$

$$F_{Y|\,U}^{2;\,\boldsymbol{\lambda}^3}(y|\ (u_3, u_2, u_1, u_0)$$

$$= 1 - e^{-\frac{\lambda_3(y - r(u_1)t)}{r(u_3) - r(u_1)}} + \frac{\lambda_3(r(u_2) - r(u_1))}{\lambda_2(r(u_3) - r(u_1)) + \lambda_3(r(u_2) - r(u_1))} \tag{N.14}$$
$$e^{-\frac{\lambda_2(y - r(u_1)t)}{r(u_2) - r(u_1)}} \left[1 - e^{-\frac{\left[\lambda_2(r(u_3) - r(u_1)) + \lambda_3(r(u_2) - r(u_1))\right](y - r(u_1)t)}{r(u_2) - r(u_1)}} \right]$$

$$F^{2;\,\lambda^3}_{\gamma|\,U}(y|\ (u_3,u_2,u_1,u_0) = 1 - B_{12}e^{-A_{12}y} - B_{13}e^{-A_{13}y} + B_{14}e^{-A_{14}y}\ . \qquad \text{(N.15)}$$

If $y \in [r(u_1)t, r(u_2)t)$: (N.16)

$$\begin{aligned} F^{3;\,\lambda^3}_{\gamma|\,U}(y|\ (u_3,u_2,u_1,u_0) = {} & 1 - B_2e^{-A_2y} + B_3e^{-A_3y} \\ & + B_8e^{-A_8y} - B_9e^{-A_9y} + B_{10}e^{-A_{10}y} - B_{11}e^{-A_{11}y} \\ & - B_{12}e^{-A_{12}y} - B_{13}e^{-A_{13}y} + B_{14}e^{-A_{14}y}\ . \end{aligned}$$

Let $y \in [r(u_2)t, r(u_3)t)$ (thus, $l = 3$):

$j = n = 3$:

$$C^{1,3;r(u_3)}_{y,3} = \Big[\, K_{32}(y),\ K_{30}(y) \Big] \quad \text{[Eq. } L.102\text{]} \qquad \text{(N.17)}$$

$j = n - 1 = 2$:

$$C^{1,3;r(u_3)}_{y,2} = \big(\, K_{21}(y) - \xi_{123}v_3,\ K_{20}(y) - \xi_{023}v_3 \,\big] \quad \text{[Eq. } L.105\text{]} \qquad \text{(N.18)}$$

$j = 1 = i$:

$$C^{1,3;r(u_3)}_{y,1} = \quad = \Big[\, 0, K_{10}(y) - \xi_{012}v_2 - \xi_{013}v_3 \,\Big] \quad \text{[Eq. } L.107\text{]} \qquad \text{(N.19)}$$

$j = 0$:

$$C^{1,3;r(u_3)}_{y,0} = \infty \qquad [\textit{case } b\text{–}iv)] \qquad \text{(N.20)}$$

$$\int_{C_{y,3}^{1,3;r(u_3)}} \int_{C_{y,2}^{1,3;r(u_3)}} \int_{C_{y,1}^{1,3;r(u_3)}} f_3(v_3) f_2(v_2) f_1(v_1) dv_1 dv_2 dv_3 \tag{N.21}$$

$$= B_{15} e^{-A_{15} y} - B_9 e^{-A_9 y} - B_{16} e^{-A_{16} y} + B_3 e^{-A_3 y}$$

$$- B_{17} e^{-A_{17} y} + B_{11} e^{-A_{11} y} + B_{18} e^{-A_{18} y} - B_7 e^{-A_7 y} .$$

$$F_{\gamma|U}^{2;\boldsymbol{\lambda}^3}(y|\ (u_3, u_2, u_1, u_0)$$

$$= 1 - e^{-\frac{\lambda_3 (y - r(u_1) t)}{r(u_3) - r(u_1)}} + \frac{(r(u_2) - r(u_1))\lambda_3}{\lambda_1 (r(u_2) - r(u_1)) - \lambda_2 (r(u_3) - r(u_1))}$$

$$e^{-\frac{\lambda_2 (y - r(u_1) t)}{r(u_3) - r(u_1)}} \left[e^{-\frac{(y - r(u_1) t)\left[\lambda_3 (r(u_2) - r(u_1)) - \lambda_2 (r(u_3) - r(u_1))\right]}{(r(u_2) - r(u_1))(r(u_3) - r(u_1))}} \right. \tag{N.22}$$

$$\left. - e^{-\frac{(y - r(u_2) t)\left[\lambda_3 (r(u_2) - r(u_1)) - \lambda_2 (r(u_3) - r(u_1))\right]}{(r(u_2) - r(u_1))(r(u_3) - r(u_1))}} \right]$$

$$F_{\gamma|U}^{2;\boldsymbol{\lambda}^3}(y|\ (u_3, u_2, u_1, u_0) \tag{N.23}$$

$$= 1 - B_{12} e^{-A_{12} y} - B_{20} e^{-A_{20} y} + B_{14} e^{-A_{14} y}$$

Therefore,

$$F^{3;\,\lambda^3}_{\gamma|\,U}(y|\ (u_3,u_2,u_1,u_0) = \int\limits_{C^{1,3;r(u_3)}_{y,3}} \int\limits_{C^{1,3;r(u_3)}_{y,2}} \int\limits_{C^{1,3;r(u_3)}_{y,1}} f_3(v_3)f_2(v_2)f_1(v_1)dv_1dv_2dv_3 \qquad \text{(N.24)}$$

$$+ F^{2;\,\lambda^3}_{\gamma|\,U}(y|\ (u_3,u_2,u_1,u_0)$$

If $y \in [r(u_2)t, r(u_3)t)$: (N.25)

$$F^{2;\,\lambda^3}_{\gamma|\,U}(y|\ (u_3,u_2,u_1,u_0) = 1 + B_3e^{-A_3y} - B_7e^{-A_7y} - B_9e^{-A_9y}$$
$$+ B_{11}e^{-A_{11}y} - B_{12}e^{-A_{12}y} + B_{14}e^{-A_{14}y} - B_{16}e^{-A_{16}y}$$
$$- B_{17}e^{-A_{17}y} + B_{18}e^{-A_{18}y} - B_{20}e^{-A_{20}y}\ .$$

Finally, let $y \in [r(u_3)t, \infty)$. Then

If $y \in [r(u_3)t, \infty)$: (N.26)

$$F^{2;\,\lambda^3}_{\gamma|\,U}(y|\ (u_3,u_2,u_1,u_0) = 1\ .$$

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] L. Svobodova, *Computer Performance Measures and Evaluation Methods: Analysis and Applications.* New York, NY: American Elsevier, 1976.

[2] D. Ferrari, *Computer Systems Performance Evaluation.* Englewood Cliffs, NJ: Prentice-Hall, 1978.

[3] H. Kobayashi, *Modeling and Analysis: An Introduction to System Performance Evaluation Methodology.* Reading, MA: Addison-Wesely, 1978.

[4] C. H. Sauer and K. M. Chandy, *Computer Systems Performance Modeling.* Englewood Cliffs, NJ: Prentice-Hall, 1981.

[5] M. L. Shooman, *Probabilistic Reliability: An Engineering Approach.* New York, NY: McGraw-Hill, 1968.

[6] B. V. Gnedenko, Yu. K. Belyayev, and A. D. Solovyev, *Mathematical Methods of Reliability Theory.* New York, NY: Academic Press, 1969.

[7] R. E. Barlow and F. Proschan, *Statistical Theory of Reliability and Life Testing: Probability Models.* New York, NY: Holt, Rinehart and Winston, Inc., 1975.

[8] A. Coppola, "A mathematical model for the prediction of sytem effectiveness," in *Proc. 2nd New York Conf. Electronic Reliability,* Oct. 1961.

[9] Weapon System Effectiveness Industry Advisory Committee (WSEIAC), Chairman's Final Report, AFSC-TR-65-6, US Air Force Systems Command, available from Defense Documentation Center (DDC), Cameron Station, Alexandria, VA 22314 USA, Jan. 1965.

[10] F. A. Tillman, C. L. Hwang, and W. Kuo, "System effectiveness models: An annotated bibliography," *IEEE Trans. Reliability,* vol. R-29, pp. 295-304, Oct. 1980.

[11] J. F. Meyer, D. G. Furchtgott, and A. Movaghar, "A bibliography on formal methods for system specification, design, and validation," SEL Report No. 163, Systems Engineering Lab, The University of Michigan, Ann Arbor, MI, Jan. 1982.

[12] J. F. Meyer, "Computation-based reliability analysis," *IEEE Trans. Comput.,* vol. C-25, pp. 578-584, June 1976.

[13] B. R. Borgerson and R. F. Freitas, "A Reliability model for gracefully degrading and standby-sparing systems," *IEEE Trans. Comput.,* vol. C-24, pp. 517-525, May 1975.

[14] H. B. Baskin, B. R. Borgerson, and R. Roberts, "PRIME–A modular architecture for terminal oriented systems," in *1972 Spring Joint Computer Conf., AFIPS Conf. Proc. 40*, pp. 431-437, 1972.

[15] R. Troy, "Dynamic reconfiguration: An algorithm and its efficiency evaluation," in *Proc. 1977 Int. Symp. Fault-Tolerant Computing*, Los Angeles, CA, pp. 44-49, June 1977.

[16] J. Losq, "Effects of failures on gracefully degradable systems," in *Proc. 1977 Int. Symp. Fault-Tolerant Computing*, Los Angeles, CA, pp. 29-34, June 1977.

[17] M. D. Beaudry, "Performance-related reliability measures for computing systems," *IEEE Trans. Comput.*, vol. C-27, pp. 540-547, June 1978.

[18] H. Mine and K. Hatayama, "Performance related reliability measures for computing systems," in *Proc. 1979 Int. Symp. on Fault-Tolerant Computing*, Madison, WI, pp. 59-62, June 1979.

[19] F. A. Gay and M. L. Ketelsen, "Performance evaluation for gracefully degrading systems," in *Proc. 1979 Int. Symp. on Fault-Tolerant Computing*, Madison, Wisconsin, pp. 51-58, June 1979.

[20] T. C. K. Chou and J. A. Abraham, "Performance/availabilty model of shared resource multiprocessors," *IEEE Trans. Reliability*, vol. R-29, pp. 70-76, April 1980.

[21] X. Castillo and D. P. Siewiorek, "A performance-reliability model for computing systems," in *Proc. 1980 Int. Symp. Fault-Tolerant Computing*, Kyoto, Japan, pp. 187-192, Oct. 1980.

[22] J. M. De Souza, "A unified method for the benefit analysis of fault-tolerance," in *Proc. 1980 Int. Symp. on Fault-Tolerant Computing*, Kyoto, Japan, pp. 201-203, Oct. 1980.

[23] X. Castillo and D. P. Siewiorek, "Workload, performance, and reliability of digital computing systems," in *Proc. 1981 Int. Symp. Fault-Tolerant Computing*, Portland, ME, pp. 84-89, June 1981.

[24] R. Huslende, "A combined evaluation of performance and reliability for degradable systems," in *ACM/SIGMETRICS Conf. on Measurement and Modeling of Computer Systems*, Las Vegas, NV, pp. 157-164, Sept. 1981.

[25] A. Pedar and V. V. S. Sarma , "Phased-mission analysis for revaluationg the effectiveness of aerospace computing-systems ," *IEEE Trans. Reliability* , vol. R-30 , Dec. 1981 .

[26] C. M. Krishna and K. G. Shin, "Performance measures for multiprocessor controllers," CRL-TR-1-82, Computing Research Laboratory, The University of Michigan, Ann Arbor, MI, Oct. 1982.

[27] J. Arlat and J. C. Laprie , "Performance-related dependability evaluation of supercomputer systems ," in *Proc. 1983 Int. Symp. on Fault-Tolerant Computing* , Milano, Italy, June, 1983 .

[28] E. Gai and M. Adams , "Measures of merit for fault-tolerant systems ," in *Proc. Guidance and Control Conf.* , Gatlinburg, TN , Aug. 1983 .

[29] J. F. Meyer, "On evaluating the performability of degradable computing systems," *IEEE Trans. Comput.*, vol. C-29, pp. 720-731, Aug. 1980.

[30] J. F. Meyer, R. A. Ballance, D. G. Furchtgott, and L. T. Wu, "Models and techniques for evaluating the effectiveness of aircraft computing systems," Semi-Annual Status Report Number 3, NASA Grant NSG 1306, SEL Report No. 116, Systems Engineering Lab, The University of Michigan, Ann Arbor, MI, Jan. 1978.

[31] ——, "Models and techniques for evaluating the effectiveness of aircraft computing systems," Semi-Annual Status Report Number 4, NASA Grant NSG 1306, SEL Report No. 121, Systems Engineering Lab, The University of Michigan, Ann Arbor, MI, July 1978.

[32] J. F. Meyer, D. G. Furchtgott, and L. T. Wu, "Performability evaluation of the SIFT computer," *IEEE Trans. Comput.*, vol. C-29, pp. 501-509, June 1980.

[33] J. F. Meyer, "Closed-form solutions of performability," *IEEE Trans. Comput.*, vol. C-31, pp. 648-657, July 1982.

[34] ——, "A model hierarchy for evaluating the effectiveness of computing systems," in *Texte des Conferences III-e Congres National de Fiabilite*, Perros-Guirec, France, pp. 539-555, Sept. 1976, Tome II.

[35] J. F. Meyer, D. G. Furchtgott, and L. T. Wu, "Models and techniques for evaluating the effectiveness of aircraft computing systems," Semi-Annual Status Report Number 1, NASA Grant NSG 1306, SEL Report No. 106-1, Systems Engineering Lab, The University of Michigan, Ann Arbor, MI, Nov. 1976.

[36] J. F. Meyer, R. A. Ballance, D. G. Furchtgott, and L. T. Wu, "Models and techniques for evaluating the effectiveness of aircraft computing systems," Semi-Annual Status Report Number 2, NASA Grant NSG 1306, SEL Report No. 111, Systems Engineering Lab, The University of Michigan, Ann Arbor, MI, July 1977.

[37] R. A. Ballance and J. F. Meyer, "Functional dependence and its application to system evaluation," in *Proc. of the 1978 Johns Hopkins Conf. on Information Sciences and Systems*, Baltimore, MD, pp. 280-285, March 1978.

[38] J. F. Meyer, "On evaluating the performability of degradable computing systems," in *Proc. 1978 Int. Symp. on Fault-Tolerant Computing*, Toulouse, France, pp. 44-49, June 1978.

[39] D. G. Furchtgott and J. F. Meyer, "Performability evaluation of fault-tolerant multiprocessors," in *1978 Government Microcircuit Applications Conference Digest of Papers*, Monterey, California, pp. 362-365, Nov. 1978.

[40] J. F. Meyer, D. G. Furchtgott, and L. T. Wu, "Performability evaluation of the SIFT computer," SEL Report No. 127, Systems Engineering Lab, The University of Michigan, Ann Arbor, MI, Jan. 1979.

[41] D. G. Furchtgott, "METAPHOR (Version 1) Programmer's Guide," SEL Report No. 128, Systems Engineering Lab, The University of Michigan, Ann Arbor, MI, Jan. 1979.

[42] J. F. Meyer, D. G. Furchtgott, and L. T. Wu, "Models and techniques for evaluating the effectiveness of aircraft computing systems," Semi-Annual Status Report Number 5, NASA Grant NSG 1306, SEL Report No. 129, Systems Engineering Lab, The University of Michigan, Ann Arbor, MI, Jan. 1979.

[43] L. T. Wu and J. F. Meyer, "Phased models for evaluating the performability of computing systems," in *Proc. of the 1979 Johns Hopkins Conf. on Information Sciences and Systems*, Baltimore, MD, pp. 426-431, March 1979.

[44] J. F. Meyer, D. G. Furchtgott, and L. T. Wu, "Performability evaluation of the SIFT computer," in *Proc. 1979 Int. Symp. on Fault-Tolerant Computing*, Madison, Wisconsin, pp. 43-50, June 1979.

[45] L. T. Wu and J. F. Meyer, "Phased models for evaluating the performability of computing systems," SEL Report No. 135, Systems Engineering Lab, The University of Michigan, Ann Arbor, MI, July 1979.

[46] D. G. Furchtgott, "METAPHOR (Version 1) User's Guide," SEL Report No. 136, Systems Engineering Lab, The University of Michigan, Ann Arbor, MI, July 1979.

[47] J. F. Meyer, "Performability modeling with continuous accomplishment sets," SEL Report No. 137, Systems Engineering Lab, The University of Michigan, Ann Arbor, MI, July 1979.

[48] J. F. Meyer, D. G. Furchtgott, and L. T. Wu, "Models and techniques for evaluating the effectiveness of aircraft computing systems," Semi-Annual Status Report Number 6, NASA Grant NSG 1306, SEL Report No. 138, Systems Engineering Lab, The University of Michigan, Ann Arbor, MI, July 1979.

[49] ——, "Models and techniques for evaluating the effectiveness of aircraft computing systems," Semi-Annual Status Report Number 7, NASA Grant NSG 1306, SEL Report No. 141, Systems Engineering Lab, The University of Michigan, Ann Arbor, MI, January 1980.

[50] J. F. Meyer and L. T. Wu, "Evaluation of computing systems using functionals of a stochastic process," SEL Report No. 140, Systems Engineering Lab, The University of Michigan, Ann Arbor, MI, Jan. 1980.

[51] J. F. Meyer, "Performability models and solutions for continuous performance variables," SEL Report No. 139, Systems Engineering Lab, The University of Michigan, Ann Arbor, MI, July 1980.

[52] J. F. Meyer, D. G. Furchtgott, and L. T. Wu, "Models and techniques for evaluating the effectiveness of aircraft computing systems," Semi-Annual Status Report Number 8, NASA Grant NSG 1306, SEL Report No. 145, Systems Engineering Lab, The University of Michigan, Ann Arbor, MI, July 1980.

[53] J. F. Meyer, "Closed-form solutions of performability," SEL Report No. 147, Systems Engineering Lab, The University of Michigan, Ann Arbor, MI, January 1981.

[54] J. F. Meyer, D. G. Furchtgott, A. Movaghar, and L. T. Wu, "Models and techniques for evaluating the effectiveness of aircraft computing systems," Semi-Annual Status Report Number 9, NASA Grant NSG 1306, SEL Report No. 148, Systems Engineering Lab, The University of Michigan, Ann Arbor, MI, January 1981.

[55] J. F. Meyer and L. T. Wu, "Evaluation of computing systems using functionals of a Markov process," in *Proc. 14th Annual Hawaii Int. Conf. on System Sciences*, Honolulu, HI, pp. 74-83, Jan. 1981.

[56] J. F. Meyer, "Closed-form solutions of performability," in *Proc. 1981 Int. Symp. on Fault-Tolerant Computing*, Portland, ME, pp. 66-71, June 1981.

[57] J. F. Meyer, D. G. Furchtgott, and A. Movaghar, "Models and techniques for evaluating the effectiveness of aircraft computing systems," Semi-Annual Status Report Number 10, NASA Grant NSG 1306, SEL Report No. 155, Systems Engineering Lab, The University of Michigan, Ann Arbor, MI, July 1981.

[58] ——, "Models and techniques for evaluating the effectiveness of aircraft computing systems," Semi-Annual Status Report Number 11, NASA Grant NSG 1306, SEL Report No. 164, Systems Engineering Lab, The University of Michigan, Ann Arbor, MI, January 1982.

[59] L. T. Wu, "Models for evaluating the performability of degradable computing systems", Ph.D. Thesis, University of Michigan, Ann Arbor, MI, 1982.

[60] ——, "Models for evaluating the performability of degradable computing systems," CRL-TR-7-82 (also issued as SEL Report No. 169), Computing Research Lab, The University of Michigan, Ann Arbor, MI, June 1982.

[61] J. F. Meyer, D. G. Furchtgott, and A. Movaghar, "Models and techniques for evaluating the effectiveness of aircraft computing systems," Final Report, NASA Grant NSG 1306, SEL Report No. 170, Systems Engineering Lab, The University of Michigan, Ann Arbor, MI, July 1982.

[62] L. T. Wu, "Operational models for the evaluation of degradable computing systems," in *ACM/SIGMETRICS Conf. Measurement and Modeling of Computer Systems*, Seattle, WA, pp. 179-185, Aug. 1982.

[63] D. G. Furchtgott and J. F. Meyer , "Performability evaluation of computing systems using reward models ," CRL-TR-27-83 , Computing Research Lab, Univ. of Mich., Ann Arbor, MI , Aug. 1983 .

[64] J. F. Meyer , "Performability modelling of distributed real-time systems ," in *Proc. Int. Workshop on Applied Mathematics and Performance/Reliability Models of Computer/Communication Systems* , Pisa, Italy , Sept. 1983 .

[65] A. Movaghar, "Models for validation of degradable systems," Thesis proposal, The University of Michigan, Ann Arbor, MI, Jan. 1982.

[66] J. F. Meyer , "Performability modeling of distributed real-time systems ," Computing Research Lab, CRL-TR-28-83 , The University of Michigan, Ann Arbor, MI , Aug. 1983 .

[67] J. H. Wensley, L. Lamport, J.Goldberg, M. W. Green, K. N. Levitt, P. M. Melliar-Smith, R. E. Shostak, and C. B. Weinstock, "SIFT: Design and analysis of a fault-tolerant computer for aircraft control," *Proc. of the IEEE*, vol. 66, no. 10, pp. 1240-1255, Oct. 1978.

[68] J. H. Wensley, J. Goldberg, M. W. Green, W. H. Kautz, K. N. Levitt, M. E. Mills, R. E. Shostak, P. M. Whiting-O'Keefe, and H. M. Zeidler, "Design study of software implemented fault tolerance (SIFT) computer," Interim Technical Report No. 1, NASA Contract NAS1-13792, Stanford Research Institute, June 1978.

[69] A. L. Hopkins, Jr., T. B. Smith, III, and J. H. Lala, "FTMP--A highly reliable fault-tolerant multiprocessor for aircraft," *Proc. of the IEEE*, vol. 66, no. 10, pp. 1221-1239, Oct. 1978.

[70] T. B. Smith, A. L. Hopkins, W. Taylor, R. A. Ausrotas, J. H. Lala, L. D. Hanley, J. H. Martin, E. C. Hall, and J. R. Howatt, "A fault tolerant multiprocessor architecture for aircraft," vols I-III, Technical Report, NASA Contract NAS1-13782, The Charles Stark Draper Laboratory, Inc, Cambridge, MA, July 1976, April 1977, and Nov. 1978.

[71] S. K. Kachhal and S. R. Arora, "Seeking configurational optimization in computer systems," in *Proc. ACM Ann. Conf.*, pp. 96-101, 1975.

[72] K. S. Trivedi and T. M. Sigmon, "A performance comparison of optimally designed computer systems with and without virtual memory," in *Proc. 6th Annual Int. Symp. on Computer Architecture*, pp. 117-121, Apr. 1979.

[73] K. S. Trivedi, *Probability and Statistics with Reliability, Queuing, and Computer Science Applications*. Englewood Cliffs, NJ: Prentice-Hall, 1982.

[74] A. L. Scherr, "An analysis of time shared computer systems," Doctoral Thesis, Department of Electrical Engineering, MIT, Cambridge, MA, June, 1965.

[75] V. L. Wallace and R. S. Rosenberg, "Markovian modelss and numerical analysis of computer system behavior," in *AFIPS Conf. Proc.*, vol. 28, pp. 141-148, 1966.

[76] J. L. Smith, "An analysis of time sharing computer systems using Markov models," in *AFIPS Conf. Proc.*, vol. 28, pp. 87-95, 1966.

[77] J. von Neumann, "Probabilistic logics and the synthesis of reliable organisms from unreliable components," in *Automata Studies*, Annals of Math Studies No. 34, C. E. Shannon and J. McCarthy, Ed. Princeton, NJ: Princeton University Press, pp. 43-98, 1956.

[78] E. F. Moore and C. E. Shannon, "Reliable circuits using less reliable relays," *J. of the Franklin Institute*, vol. 262, part I, pp. 191-208, and part II, pp. 281-297, 1956.

[79] Z. W. Birnbaum, J. D. Esary, and S. C. Saunders, "Multi-component systems and structures and their reliability," *Technometrics*, vol. 3, pp. 55-57, Feb. 1961.

[80] W. G. Bouricius, W. C. Carter, and P. R. Schneider, "Reliability modeling techniques for self-repairing computer systems," in *Proc. ACM 1969 Annual Conf.*, pp. 295-309, Aug. 1969.

[81] W. G. Bouricius, W. C. Carter, D. C. Jessep, P. R. Schneider, and A. B. Wadia, "Reliability modeling for fault-tolerant computers," *IEEE Trans. Comput.*, vol. C-20, no. 11, pp. 1306-1311, Nov. 1971.

[82] W. M. Hirsch, M. Meisner, and C. Boll, "Cannibalization in multicomponent systems and the theory of reliability," *Naval Research Logistics Quarterly*, vol. 15, no. 3, pp. 331-359, 1968.

[83] V. Postelnicu, "Nondichotomic multi-component structures," *Bull. Math. de la Soc. Sci. Math. de la R. S. de Roumanie*, vol. 14, no. 2, pp. 209-217, 1970.

[84] J. D. Murchland, "Fundamental concepts and relations for reliablity analysis of multistate systems," in *Reliability and Fault-Tree Analysis*, R. E. Barlow, J. B. Fussell, and N. D. Singpurwalla, Ed. Philadelphia, PA: SIAM, 1975.

[85] F. A. Tillman, C. H. Lie, and C. L. Hwang, "Analysis of pseudo-reliability of a combat tank system and its optimal design," *IEEE Trans. Reliability*, vol. R-25, pp. 239-242, Oct. 1976.

[86] J. C. Laprie and K. Medhaffer-Kanoun , "Dependability modeling of safety systems ," in *Proc. 1980 Int. Symp. on Fault Tolerant Computing* , Kyoto, Japan , Oct. 1980 .

[87] X. Castillo and D. P. Siewiorek, "A workload dependent software reliability prediction model," in *Proc. 1982 Int. Symp. on Fault-Tolerant Computing*, Los Angeles, CA, pp. 279-286, June 1982.

[88] H. S. Winokur, Jr. and L. J. Goldstein, "Analysis of mission-oriented systems," *IEEE Trans. Reliability*, vol. R-18, no. 4, Nov. 1969.

[89] D. G. Furchtgott, "February Monthly Report," Internal memorandum, Systems Engineering Lab, The University of Michigan, Ann Arbor, MI, Feb. 1977.

[90] M. D. Beaudry, "Performance related reliability measures for computing systems," in *Proc. 1977 Int. Symp. on Fault-Tolerant Computing*, Los Angeles, CA, pp. 16-21, June 1977.

[91] F. A. Gay, "Performance Modeling for Gracefully Degrading Systems", Ph. D. Dissertation, Northwestern Universtity, Evanston, IL, June 1979.

[92] G. N. Cherkesov, "Semi-markovian models of reliability of multichannel systems with unreplenishable reserve of time," *Engineering Cybernetics*, vol. 18, March 1981.

[93] P. J. Denning and J. P. Buzen, "The operational analysis of queueing network models," *Computing Surveys*, vol. 10, pp. 225-261, Sept. 1978.

[94] D. F. Haasl, "Advanced concepts in fault tree analysis," System Safety Symposium, The University of Washington and The Boeing Company, Seattle, WA, June 1965.

[95] S. A. Lapp and G. J. Powers, "Computer-aided synthesis of fault trees," *IEEE Trans. Reliability*, vol. R-26, pp. 2-13, April 1977.

[96] G. E. Apostolakis, S. L. Salem, and J. S. Wu, "CAT: A computer code for the automated construction of fault trees," Rep. EPRI NP-705, Electric Power Research Institute, March 1978.

[97] J. D. Esary and H. Ziehms, "Reliability of phased missions," in *Reliablity and Fault Tree Analysis.* Philadelphia, PA: SIAM, pp. 213-236, 1975.

[98] H. Kumamoto and E. J. Henley, "Top-down algorithm for obtaining prime implicant sets of non-coherent fault trees," *IEEE Trans. Reliability,* vol. R-27, pp. 242-249, Oct. 1978.

[99] R. L. Williams and W. Y. Gateley, "GO methodology—an overview," EPRI NP-765, Electric Power Research Institute, May 1978.

[100] L. Caldarla, "Fault tree analysis of multi-state systems with multistate components," in *Proc. American Nuclear Society Topical Meeting on Probabilistic Analysis of Nuclear Reactor Safety,* Los Angeles, CA, vol. VIII-Paper 1, May 1978.

[101] D. G. Furchtgott, "June Monthly Report," Internal memorandum, Systems Engineering Lab, The University of Michigan, Ann Arbor, MI, June 1981.

[102] ——, "August Monthly Report," Internal memorandum, Systems Engineering Lab, The University of Michigan, Ann Arbor, MI, Aug. 1981.

[103] E. R. Woodcock, "The calculation of reliabilty systems: The program NOTED," Authority Health and Safety Branch, U.K.A.E.A., 11 Charles II Street, London SW1 England, 1968.

[104] Y. H. Kim, K. E. Case, and P. M. Ghare, "A method for computing complex system reliability," *IEEE Trans. Reliability,* vol. R-21, pp. 215-219, Nov. 1972.

[105] K. P. Parker and E. J. McCluskey, "Probabilistic treatment of general combinational networks," *IEEE Trans. Comput.,* vol. C-24, pp. 668-670, 1975.

[106] A. Satyanarayana and A. Prabhakar, "New topological formula and rapid algorithm for reliability analysis of complex networks," *IEEE Trans. Reliability,* vol. R-27, pp. 82-100, June 1978.

[107] J. M. Hammersley and D. C. Handscomb, *Monte Carlo Method.* London, England: Methuen and Co., Ltd., 1964.

[108] J. B. Fussell and J. S. Arendt, "System reliability engineering methodology: A discussion of the state of the art," *Nucl. Safety,* vol. 20, 1979.

[109] N. J. McCormick, *Reliability and Risk Analysis.* New York, NY: Academic Press, 1981.

[110] H. E. Kongso , "RELY4: A Monte Carlo computer program for system reliability analysis ," Report RISO-M-1500 , Danish Atomic Energy Commission , June 1972 .

[111] M. O. Locks, "Monte Carolo Bayesian system reliability-and-MTBF-Confidence assessment," Air Force Flight Dynamics Laboratiory, Wright-Patterson AFB, Ohio, AFFDL-TR-75-144. Available from NTIS; Springfield, VA 22161, 1975.

[112] . "Reactor Safety Study: An assessment of accident rates in US commercial nuclear power plants," WASH-1400 (NUREG 75/014), US Nuclear Regulatory Commission, available from NTIS, Springfield, VA 22161, Oct. 1975.

[113] S. D. Matthews, "MOCARS: A Monte Carlo simulation code for determining the distribution and simulation limits," ERDA Rep. TREE-1138, EG&G Idaho, July 1977.

[114] R. C. Erdmann, et al., "Probabilistic safety analysis III," EPRI NP-749, Electric Power Research Institute, April 1978.

[115] P. A. Jensen and M. Bellmore, "An algorithm to determine the reliability of a complex system," *IEEE Trans. Reliability*, vol. R-18, pp. 169-174, Nov. 1969.

[116] W. E. Vesely and R. E. Narum, "PREP and KITT: Computer codes for the automatic evaluation of a fault tree," USAEC Rep. IN-1349, Idaho Nuclear Corporation, available from NTIS; Springfield, VA 22151, Aug. 1970.

[117] R. H. Blazek , R. E. Thomas , R. K. Thatcher , and J. L. Easterday , "TASRA: TAbular System Reliability Analysis ," AFFDL-TR-71-128 , Battelle, Columbus Lab. , 1971 .

[118] S. N Semanderes, "ELRAFT–A computer program for the efficient logic reduction analysis of fault trees," *IEEE Trans. Nucl. Sci.*, vol. NS-18, pp. 481-487, Feb. 1971.

[119] J. B. Fussell, E. B. Henry, and N. H. Marshall, "MOCUS--A computer program to obtain minimal sets from fault trees," USAEC Rep ANCR-1156, Aerojet Nuclear Company, Aug. 1974.

[120] Boeing Commercial Airplane Co. , "ARMM: Automatic Reliability Mathematical Model ," Boeing Document No. D6A-10500-1 .

[121] P. K. Pande, M. E. Spector. and P. Chatterjee, "Computerized fault tree analysis: TREEL and MICSUP," Rep. ORC-75-3 (AD-A010 146), Operations Research Center, Univ. of California, Berkeley, CA, April 1975.

[122] W. J. Van Slyke and D. E. Griffing, "ALLCUTS–A fast comprehensive fault tree analysis code," ERDA Rep. ARH-ST-112, Atlantic Richfield Hanford Company, July 1975.

[123] B. E. Bjurman , G. M. Jenkins , C. J. Masreliez , K. L. McClellan , and J. E. Templeman , "CARSRA: A reliability estimation tool for redundant systems ," in *Airborne Advanced Reconfigurable Computer System (ARCS)* , August 1976 .

[124] G. R. Burdick, N. H. Marshall, and J. R. Wilson, "COMCAN--A computer program for common cause failure analysis," Rep ANCR-1314, Aerojet Nuclear Company, May 1976.

[125] F. L. Leverenz and H. Kirch, "Users guide for the WAM-BAM computer code," Electric Power Research Institute Report Rep. 217-2-5, Jan. 1976.

[126] O. Platz and J. V. Olsen , "FAUNET: A program package for evaluation of fault trees and networks ," Report RISO-348 , Danish Atomic Energy Commission , Sept. 1976 .

[127] C. L. Cate and J. B. Fussell, "BACFIRE--A computer code for common cause failure analysis," Rep. NERS-77-02, Nuclear Engineering Department, University of Tennessee, Knoxville, TN, Feb. 1977.

[128] J. B. Fussell , D. M. Rasmuson , and D. Wagner , "SUPERPOCUS--A computer program for calculaing system probabilistic reliability and safety characteristics ," Report NERS-77-01 , Nuclear Engineering Department, Univ. Tennessee, Knoxville, TN , May 1977 .

[129] H. E. Lambert and F. M. Gilman, "The IMPORTANCE computer code," ERDA Rep. UCRL-79269, Lawrence Livermore Laboratory, Univ. of California, March 1977.

[130] J. Olmos and L. Wolf, "A modular approach to fault tree and reliabilty analysis," Rep. MITNE-209, Dept. Nuclear Engineering, Massachusetts Institute of Technology, Cambridge, MA, Aug. 1977.

[131] P. J. Pelto and W. L. Purcell , "MFAULT: A computer program for analyzing fault trees ," USDOE Report BNWL-2145, Battelle Pacific Northwest Laboriitoires, NTIS , Nov. 1977 .

[132] W. E. Vesely and F. F. Goldberg, "FRANTIC--A computer code for time-dependent unavailability analysis," U. S. Nuclear Regulatory Commission Rep NUREG-0193, Oct. 1977.

[133] D. B. Wheeler, J. S. Hsuan, R. R. Duersch, and G. M. Roe, "Fault tree analysis using bit manipulation," *IEEE Trans. Reliability*, vol. R-26, pp. 95-99, Jun. 1977.

[134] R. C. Erdmann, F. L. Leverenz, and H. Kirch, "WAMCUT: A computer code for fault tree evaluation," Electric Power Research Institute Rep. EPRNI-NP-803, June 1978.

[135] W. Y. Gateley and R. L. Williams, "GO Methodology–System reliability assessment and computer code manual," EPRI NP-766, Electric Power Research Institute, May 1978.

[136] D. M. Rasmuson, et al., "COMCAN-II: A computer program for common cause failure analysis," Rep. TREE-1289, Idaho National Engineering Laboratory, Sept. 1978.

[137] D. M. Rasmuson and N. H. Marshall, "FATRAM--A core efficient cut-set algorithm," *IEEE Trans. Reliability*, vol. R-27, pp. 250-253, Oct 1978.

[138] R. E. Worrell and D. W. Stack, "A SETS user's manual for the fault tree analyst," Rep. SAND-77-2051, Sandia Laboratories, Nov. 1978.

[139] J. P. Roth, W. G. Bouricius, W. C. Carter, and P. R. Schneider, "Phase II of an architectrual study of a sellf-repairing computer," SAMSO Report, no. TR-67-106, Los Angeles, CA, Nov. 1967.

[140] J. L. Fleming, "RELCOMP: A computer program for calculating system reliability and MTBF," *IEEE Trans. Rel.*, vol. R-20, no. 3, pp. 102-107, Aug. 1971.

[141] F. P. Mathur and A. Avizienis, "Relibility analysis and architecture of a hybrid-redundant digital system: Generalized triple modular redndancy with self-repair," in *1970 Spring Joint Computer Conf., AFIPS Conf. Proc. 36*, pp. 375-383, 1970.

[142] D. A. Rennels and A. Avizienis, "RMS: A reliability modeling system for self-repairing computers," in *Proc. 3rd Int. Symp. Fault-Tolerant Computing*, pp. 131-135, June 1973.

[143] J. J. Stiffler, L. Bryant, and L. J. Guccione, "An engineering treatise on the CARE II dual mode and coverage models," Final Report, NASA Contract No. L-18084, Nov. 1975.

[144] J. J. Stiffler, L. A. Bryant, and L. J. Guccione, "CARE III final report," Phase I, Vol I and II, Raytheon Co., prepared for the NASA Langley Research Center, Nov. 1979.

[145] Y. W. Ng and A. Avizienis, "ARIES 76 User's Guide," Tech. Rep. No. UCLA-ENG-7894, Comp. Sci. Dept., Univ. of California at Los Angeles, Los Angeles, CA, Dec. 1978.

[146] S. V. Makam and A. Avizienis, in *Proc. 1982 Int. Symp. on Fault-Tolerant Computing*, Los Angeles, CA, pp. 267-274, June 1982.

[147] A. Costes, J. E. Doucet, C. Landrault, and J. C. Laprie, "SURF: A program for dependability evaluation of complex fault-tolerant computing systems," in *Proc. 1981 Int. Symp. on Fault-Tolerant Computing*, Portland, ME, pp. 72-78, June 1981.

[148] J. B. Fussell, G. J. Powers, and R. G. Bennetts, "Fault-trees: A state of the art discussion," *IEEE Trans. Reliability*, vol. R-23, pp. 51-55, Apr. 1974.

[149] J. D. Esary and F. Proschan, "A reliability bound for systems of maintained, interdependent components," *J. Amer. Statist. Assoc.*, vol. 65, pp. 329-338, 1970.

[150] J. L. Bricker, "A unified method for analyzing mission reliability for fault tolerant computer systems," *IEEE Trans. Reliability*, vol. R-22, no. 2, June 1973.

[151] A. Pedar, "Reliability modeling and architecture optimization of aerospace computing systems", Ph.D. Thesis, Indian Institute of Science, Bangalore-560 012, India, 1981.

[152] J. C. Laprie, "Reliability and availability of repairable structures," in *Proc. 1975 Int. Symp. Fault-Tolerant Computing*, Paris, France, pp. 87-92, June 1975.

[153] Y.-W. Ng and A. Avizienis, "A reliability model for gracefully degrading and repairable fault-tolerant systems," in *Proc. 1977 Int. Symp. Fault-Tolerant Computing*, Los Angeles, CA, pp. 22-28, June 1977.

[154] A. Costes, C. Landrault, and J. C. Laprie, "Reliability and availability models for maintained systems featuring hardware failures and design faults," *IEEE Trans. Comput.*, vol. C-27, pp. 548-560, June 1978.

[155] R. A. Howard, *Dynamic Probabilisitc Systems, Vol. II: Semi-Markov and Decision Processes.* New York, NY: Wiley, 1971.

[156] E. F. Hitt, M. S. Bridgman, and A. C. Robinson, "Comparative analysis of techniques for evaluating the effectiveness of aircraft computing systems," NASA contract NAS1-15760, NASA Contractor Report 159358, Battelle Columbus Laboratories, Columbus, OH, June 1980.

[157] "Models and techniques for evaluating the effectiveness of aircraft computing systems," Proposal for NASA Grant NSG 1306, submitted to NASA Langley Research Center, March 1976.

[158] T. F. Arnold, "The concept of coverage and its effect on the reliability of a repairable system," *IEEE Trans. Rel.*, vol. R-22, no. 3, pp. 251-251, March 1973.

[159] "Models and techniques for evaluating the effectiveness of aircraft computing systems," Proposal for extension of NASA Grant NSG 1306, submitted to NASA Langley Research Center, March 1977.

[160] D. G. Furchtgott, "January Monthly Report," Internal memorandum, Systems Engineering Lab, The University of Michigan, Ann Arbor, MI, Jan. 1977.

[161] J. F. Meyer and A. W. Naylor, "Logically consistent task sets for system evaluation," in *Proc. Seminaire sur l'Approach Systemes*, ENSAE, Toulouse, France, vol. I, pp. 65-99, November 1973.

[162] P. E. Pfeiffer, *Concepts of Probability Theory*. New York, NY: McGraw-Hill, 1965.

[163] H. L. Royden , *Real Analysis* . New York, NY : Macmillan , 1968 .

[164] E. Wong, *Stochastic Processes in Information and Dynamical Systems*. New York, NY: McGraw-Hill, 1971.

[165] M. Davio, J. P. Deschamps, and A. Thayse, *Discrete and Switching Functions*. New York, NY: McGraw-Hill, 1978.

[166] E. Phibbs and S. H. Kuwamoto, "An efficient map method for processing multistate logic treees," *IEEE Trans. Reliabilty*, vol. R-23, pp. 93-98, May 1972.

[167] R. G. Bennetts, "On the analysis of fault trees," *IEEE Trans. on Reliability*, vol. R-24, pp. 175-185, Aug. 1975.

[168] N. K. Nanda, "Applicantion of a Boolean identity for fault trees," *IEEE Trans. Reliability*, vol. R-29, p. 12, April 1980.

[169] C. L. Hwang, Frank A. Tillman, and M. H. Lee, "System-reliability evaluation techniques for complex/large systems—A review," *IEEE Trans. Reliability*, vol. R-30, pp. 416-423, Dec. 1981.

[170] L. Fratta and U. G. Montanari, "A Boolean algebra method for computing the terminal reliability in a communication network," *IEEE Trans. Circuit Theory*, vol. C-20, pp. 203-211, May 1973.

[171] K. Gopal, K. K. Aggarwal, and J. S. Gupta, "Reliability analysis of multisate device networks," *IEEE Trans. Reliability*, vol. R-27, pp. 233-236, Aug. 1978.

[172] J. A. Abraham, "An improved algorithm for network reliability," *IEEE Trans. Reliability*, vol. R-28, pp. 58-61, April 1979.

[173] M. O. Locks, "Recursive disjoint products, inclusion-exclusion, and min-cut approximations," *IEEE Trans. Reliability*, vol. R-29, pp. 361-367, Dec. 1980.

[174] R. K. Tiwari and M. Verma, "An algebraic technique for reliability evaluation," *IEEE Trans. Reliability*, vol. R-29, pp. 311-313, Oct. 1980.

[175] K. E. Iverson, A Programming Language, New York, NY: Wiley, 1962.

[176] B. W. Kernighan and D. M. Ritchie, *The C Programming Language.* Englewood Cliffs, New Jersey: Prentice-Hall, 1978.

[177] R. Miller, *Switching Theory, Volume I: Combinational Circuits.* New York: Wiley, 1965.

[178] Z. Kohavi , *Switching and Finite Automata Theory* . New York, NY : McGraw-Hill , 1970 .

[179] F. Preparata and R. Yeh , *Introduction to Discrete Structures for Computer Science and Engineering* . Reading, MA : Addison-Wesley , 1973 .

[180] E. Veitch, "A chart method for simplifying truth functions," *Proc. ACM,* pp. 127-133, 1952.

[181] M. Karnaugh, "The map method for synthesis of combinational logic circuits," *Trans. AIEE, Part I, Communication and Electronics,* vol. 72, pp. 593-599, 1953.

[182] G. Grätzer , *Lattice Theory: First Concepts and Distributive Lattices* . San Francisco, CA : W. H. Freeman and Co. , 1971 .

[183] S. MacLane and G. Birkhoff , *Algebra* . New York : MacMillan , 1967 .

[184] C. Shannon , "The synthesis of two-terminal switching circuits ," *Bell System Tech. J.* , vol. 57 , 1949 .

[185] W. V. Quine , "The problem of simplifying truth functions ," *Am Math. Monthly* , vol. 59 , Oct. 1952 .

[186] E. J. McCluskey , "Minimization of Boolean functions ," *Bell System Tech. J.* , vol. 35 , Nov. 1956 .

[187] P. Tison , "Generlization of consensus theory and application to the minimization of Boolean functions ," *IEEE Trans. Electron. Comput.* , vol. EC-5 , 1967.

[188] P. L. Tison, "An Algebra for Logic Systems–Switching Circuits Application," *IEEE Trans. Comput.*, vol. C-20, pp. 339-351, Nov. 1971.

[189] I. Koren and M. Berg, "A module replacement policy for dynamic redundancy fault-tolerant computing systems," in *Proc. 1981 Int. Symp. Fault-Tolerant Computing,* Portland, ME, pp. 90-95, June 1981.

[190] S. V. Makam and A. Avizienis, "Modeling and analysis of periodically renewed closed fault-tolerant systems," in *Proc. 1981 Int. Symp. Fault-Tolerant Computing,* Portland, ME, pp. 134-141, June 1981.

[191] T. Nakagawa, K. Yasui, and S. Osaki, "Optimum maintenance policies for a computer system with restart," in *Proc. 1981 Int. Symp. Fault-Tolerant Computing*, Portland, ME, pp. 148-150, June 1981.

[192] Y. Oda, Y. Tohma, and K. Furuya, "Reliability and performance evaluation of self-reconfigurable systems with periodic maintenance," in *Proc. 1981 Int. Symp. Fault-Tolerant Computing*, Portland, ME, pp. 142-147, June 1981.

[193] R. Huslende , "Optimal cost/reliability allocation in communication networks ," in *Proc. 1983 Int. Symp. on Fault-Tolerant Computing* , Milano, Italy, June, 1983 .

[194] J. A. Munarin , "Dynamic workload model for performance/reliability analysis of gracefullly degrading systems ," in *Proc. 1983 Int. Symp. on Fault-Tolerant Computing* , Milano, Italy, pp. 290-295, June, 1983 .

[195] E. Cinlar, *Introduction to Stochastic Processes.* Englewood Cliffs, NJ: Prentice-Hall, 1975.

[196] V. A. Ditkin and A. P. Prudnikov, *Operational Calculus in Two Variables and its Applications.* New York: Pergamon Press, 1962.

[197] D. Gross and C. M. Harris, *Fundamentals of Queueing Theory.* New York, NY: Wiley, 1974.

[198] L. Kleinrock, *Queuing Systems, Volume I: Theory.* New York, NY: Wiley, 1975.

[199] H. Raiffa, *Decision Analysis: Introductory Lectures on Choices under Uncertainty.* Reading, MA: Addison Wesley, 1968.

[200] J. Moses, "Symbolic integration: The stormy decade," *Comm. ACM*, vol. 14, Aug. 1971.

[201] A. C. Hearn, "REDUCE 2: A system and language for algebraic manipulation," in *Proc. Second Symp. Symbolic and Algebraic Manipulation.*