A PROGRAM FOR THE AUTOMATIC ANALYSIS OF QUEUEING NETWORK MODELS

by

Tom W. Keller

TR-6

December 1972

This paper constituted in part the author's thesis for the M.A. degree at the University of Texas at Austin, December 1972.

This work was supported in part by National Science Foundation Grant GJ-1084.

Technical Report No. 6
Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712

ACKNOWLEDGEMENTS

The author is indebted to Dr. K. Mani Chandy, who made time to ask and had the patience to answer. Acknowledgement is made to Dr. James C. Browne, without whose support and perspective this thesis would not have been possible. Dr. John Howard is thanked for his suggestions which much improved the tone of this work. The author also wishes to thank Don F. Towsley for adapting some of his work in symbolic manipulation for use in ASQ. The development of ASQ was supported in part by National Science Foundation Grant GJ-1084, "Design and Analysis of Multi-Programmed Computer Operating Systems."

December, 1972

ABSTRACT

This work presents a computer program which automatically analyzes certain types of queueing network models while allowing parameters of the network to be algebraic functions. Queueing networks are important models of a number of important systems, including multiprogrammed computer systems and communications networks. The theoretical work which led to the implementation of the program, the spectrum of queueing networks the program is capable of analyzing, and its importance as an analytical tool are discussed.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I

INTRODUCTION

The richness and complexity of modern computer and communications systems precludes intuitive or abstract analysis of their detailed properties. This thesis describes a computer program which allows the systems analyst to reason at an intuitive and abstract level while still dealing in some detail with systems of great complexity. Computer and communications systems are modeled by systems analysts and designers in order to evaluate the effects of changes in the system and to anticipate the desirability of possible system configurations.

Queueing networks are important models of these systems. A computer network can be modeled as a network of interconnected servers and their queues, through which jobs are routed in a probabilistic fashion. Servers are system resources; central processing units, memory and channels, for example. Jobs in the system compete for service at these finite resources, resulting in jobs queueing up for service at the various devices. Communications networks can be modeled as queueing networks because of similar properties.

Both simulation and analytic techniques are used by the analyst in evaluating queueing network models. During the preliminary stages in analyzing a system simulation techniques are too costly and too slow. The program presented in this work automatically analyzes queueing network models by analytic techniques. Although the class of queueing networks analyzable by the program may not include an exact model of the desired system, an easily and quickly obtained automatic analysis of a network which is a well characterized approximation to the system is frequently preferable to a more detailed and expensive simulation study.

The computer program ASQ, for Algebraic Solutions to Queues, departs notably from previous work in automatic queueing network analysis in two ways. Firstly, the parameters defining the queueing network may be analytic functions or numerical values. The program responds to queries on system performance with the appropriate algebraic or numeric result. Its answers are exact. Secondly, the program allows the analyst to construct an almost arbitrary queueing network for analysis. The network is composed of servers, queues, and their probabilistic interconnection, any of which can be changed by the user in real-time. This gives the analyst considerable freedom in constructing a model best suited to his purposes.

Certain queueing networks may be analyzed as Markov processes. Chapter II describes Markov processes and the classic method of their analysis. The class of queueing networks which can be described as Markov processes and also analyzed by ASQ is defined. The equations characterizing these processes, and an approach to their solution suggested by Chandy [1] are then presented.

Chapter III describes the ASQ system in detail. The format for representing a queueing network to the program and the set of queries evaluating the network the user may address to the program are given. The representations for algebraic information, both for the user and within the program, are discussed.

The last chapter describes related work done in the field of automatic analysis of Markov processes, comparing those approaches with ASQ. Conclusions on the limitations, applications, and extensions of ASQ follow.

Appendix A gives examples of interactive sessions with ASQ which might prove valuable to prospective users. Appendix B is a listing of the program itself.

CHAPTER II

ANALYSIS OF CLOSED QUEUEING NETWORKS

Markov Processes

A finite state continuous parameter (stochastic) process is a representation of a system which, at any time, is in one (and only one) of a finite number of possible states and which, at any time, may change to some other possible state. The transitions between states is governed by probabilistic laws. Figure 1, a state transition graph, depicts such a system. Each state is numbered and the system can move from state i to state j if an arc is drawn from state i to state j.

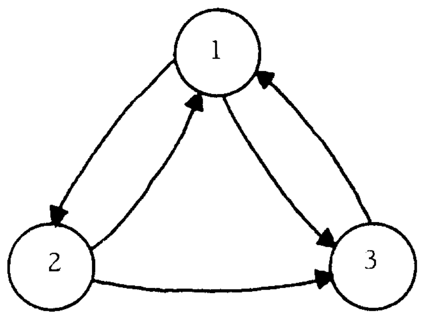

Figure 1: A State Transition Graph

Let us denote the probability of a transition between state i and state j at time t by $P_{ij}(t)$. Consider that at time t_a the system is in state S_a and at time t_b the system is in state S_b. A process is called stationary Markov if it meets two conditions:

1. The probability that the system is in state S_b at time t_a is dependent upon the interval $t_b - t_a$ but independent

of the time t_a, which is to say the probabilistic laws governing the state transitions do not change with time.

2. The above probability depends upon the state S_a, but not upon the manner in which the system reached state S_a. This is the Markov property.

There are four types of Markov processes. We will be concerned with one - the discrete state continuous transition case, as certain types of closed queueing networks can be represented as such a process. Transitions between states may occur at any (continuous) time. A process is Piosson if a transition between states occurs randomly but at some fixed average rate λ. The fact that λ is fixed satisfies condition (1). Condition (2) is satisfied if the probability of finding the system in some state at some random time is not a function of the preceding state of the system. These concepts can be developed mathematically to obtain the probabilities of finding the system in a specific state at some random time.

The condition that the transition from state i to state j occurs randomly at some fixed average rate λ_{ij} implies that if the system is in state i at time t, then the probability that the system transits to state j in some time interval Δt is

$$\lambda_{ij} \Delta t + o(\Delta t)$$

where $o(\Delta t)$ is the probability that the system moves through some intermediate states to reach state j in time Δt. Note that

$$\lim_{\Delta t \to 0} \frac{o(\Delta t)}{\Delta t} = 0. \qquad (2.0)$$

Given that $P_i(t)$ is the probability that the system is in state i at time t, we can obtain the $P_j(t + \Delta t)$ that the system is in state j at time $t + \Delta t$ in the following manner. The system will be in state j at time $t + \Delta t$ if any of the following events occur:

a) The system is in state j at time t and does not transit out of that state during the succeeding time increment Δt.

b) The system is in some state i at time t and transits so as to be in state j at time $t + \Delta t$.

By (2.0) we will ignore higher order terms in Δt. The equations of balance for the m state probabilities are then

$$P_j(t + \Delta t) = P_j(t)(1 - \sum_{i \neq j} \lambda_{ji} \Delta t) + \sum_{i \neq j} P_i \lambda_{ij} \Delta t \qquad (2.1)$$

The first term on the right-hand side expresses the probability of events (a) and the second term the probability of events (b). If we define

$$\lambda_{jj} = - \sum_{i \neq j} \lambda_{ji} \quad \text{for all j}$$

we see:

$$\frac{P_j(t + \Delta t) - P_j(t)}{\Delta t} = \sum_i \lambda_{ij} P_i(t).$$

Letting $\Delta t \to 0$, we take the limit to be

$$\frac{dP_j(t)}{dt} = \sum_{i=1}^{m} \lambda_{ij} P_i(t) \quad j = 1,\ldots,m. \qquad (2.2)$$

The set of differential equations (2.2) together with the set $P_j(0)$ of initial conditions defines the time-dependent state probabilities. These equations may be stated in matrix notation. Let $P(t)$ be the system state row vector where $P_i(t)$ has its usual definition. Let the matrix Q be defined by

$$Q_{ij} = \lambda_{ij} \qquad i \neq j$$
$$Q_{ii} = -\sum_k \lambda_{ik} \quad i \neq k.$$

Thus (2.2) may be expressed as

$$\frac{dP(t)}{dt} = P(t)Q. \tag{2.3}$$

Also,
$$\sum_{i=1}^{m} P_i(t) = 1.$$

If the coefficients of the $P_i(t)$s are bounded and independent of t, then the equations have a solution and the system will eventually reach an equilibrium or steady state [24]. That is, the probabilities

$$P_i = \lim_{t\to\infty} P_i(t)$$

exist, according to the general ergodic theorem for Markov processes, and are independent of the initial conditions of the system. P_i is the fraction of time the system spends in state i at steady state. (2.3) is then

$$\frac{dP(\infty)}{dt} = P(\infty)Q = 0.$$

The set of P_i for the system is the most detailed knowledge of the system's behavior we can obtain. All other desirable equilibrium information about the system can be obtained from these steady state probabilities. Upon getting (2.3) we can find the $P_i(\infty)$ by the following method. Let S_i be the matrix resulting from the replacement of the i^{th} column of Q by the column consisting entirely of ones. This leads to the system of equations

$$P(\infty)\, S_i = [0\ 0\ \ldots\ 0\ 1\ 0\ \ldots\ 0]$$

with the 1 in the i^{th} place. Assuming S_i to be nonsingular we obtain

$$P(\infty) = [0\ 0\ \ldots\ 0\ 1\ 0\ \ldots\ 0] S_i^{-1}.$$

Thus $P(\infty)$ is the i^{th} row of S_i^{-1}.

The inversion of the matrix S_i is an extremely difficult task for non-numerical values of the λ_{ij}, even if the λ_{ij} are simple analytical functions of one variable. The reason for this is the fact that the number of elements in the matrix increases factorially with the complexity of the process. Using symbolic manipulation techniques now available, the inversion may not be tractable. Analytical solutions for $P(\infty)$ have been obtained by inspection [9,10,19,25] if the structure of the process has a regularity that lends itself to easy analysis. Engelman and Kleinman [12] have developed a program, using the MATHLAB system, to invert the analytical matrix S_i. Matrices larger than 4 x 4 can not always be inverted because of storage requirements or singularities. Alternatively, if the coefficients of (2.3) are numerical then the numerical P_i are obtainable. Again, the factorially increasing size of S_i determines a computational limit to the complexity of the processes which can be solved.

Queueing Networks

Queueing theory is a branch of probability theory concerned with the application of mathematical analysis to models composed of servers (or devices) and the queues of customers (or jobs) that await service. A queueing network is a system of interconnected servers and queues. To every server there is associated a queue. For ease in exposition, a server and its queue will be called a node. Upon completion of service at a device a customer leaves that device's queue and enters the queue of another device, there to await service. Customers move from queue to queue in a probabilistic fashion. This is reflected in a set of branching probabilities. The probability that a job enters node j upon completion of service at node i is the branching probability p_{ij}. The amount of time a customer is served at a device is governed by a probability distribution, the service distribution for that device. The manner in which customers enter, wait, and are served in a queue is the queueing discipline associated with that queue. An example of a queueing discipline commonly encountered is the First Come First Served (FCFS) discipline.

A queueing network is closed if jobs neither may enter nor leave the network, thus making the number of jobs in a closed network a constant. A queueing network in which a job enters the network precisely when another job exits can be considered closed, as the number of jobs in the system remains constant. The closed queueing networks we will consider can be drawn so that all jobs must pass through two common points, an origin and destination. A job enters the network from the origin, traverses the nodes, and exits via the destination. A job queues up for and receives service at the nodes.

The branching probabilities define the probabilistic routing of a job through the network. Upon reaching the destination, a job immediately re-enters the network via the origin. The destination to origin path means jobs continually circulate through the queueing network. We shall call a queueing network "loop-free" if a job can obtain service only once at any device before reaching the destination. This means that the only loop in the network is the destination to origin to destination loop.

A closed queueing network of this type can serve as a rough model of some computer systems [7,10,25]. Typically, the model restricts itself to two types of devices, central processing units (CPU) and input/output (I/O) devices. An example taken from Baskett [25] is a closed queueing network composed of one CPU and several I/O units, which have exponential service distributions (Figure 2). The network is labeled according to the conventions described.

Queueing Disciplines

We shall consider three queueing disciplines:

First Come First Served (FCFS)

Jobs enter the queue and receive service in the order of entry. The device services a single job at a time until completion of the job's service need (to completion of its service request).

Processor Sharing (PS)

The PS discipline is the limiting case of the zero-overhead

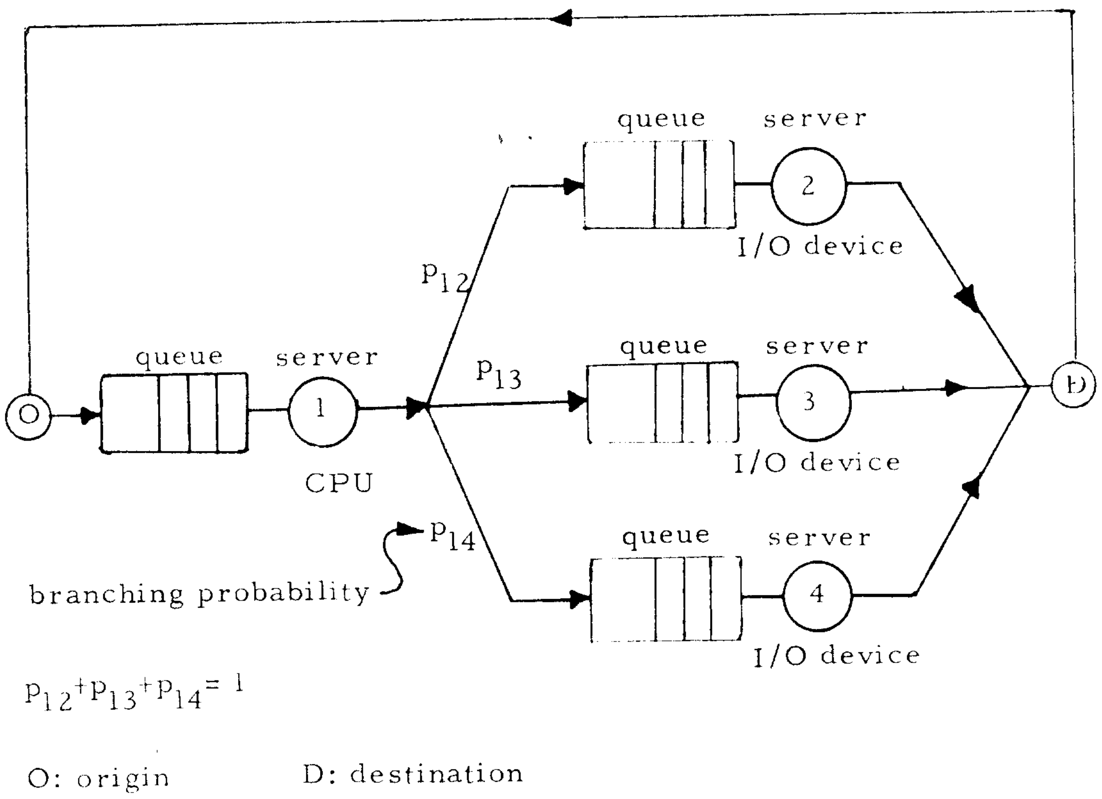

Figure 2. A Classical Computer Model

Round Robin Fixed Quantum discipline as the time quantum becomes arbitrarily small. All jobs in the queue receive service simultaneously - if n jobs are being serviced by the device then each job receives service at 1/n^{th} the rate afforded a single job by the device, $n > 0$.

Last Come First Served Preemptive Resume (LCFSPR)

The queue is a stack. The job most recently entering the queue receives service to completion of its service request or until it is preempted by another job entering the queue. When a job is preempted it is "pushed down" to await service.

The state space of the system is the cartesian product of the states of its individual queues. A network state is the number of jobs in each of its nodes.

The implication that all service times of a network be exponentially distributed is frequently too severe a restriction. A recourse is to replace a "non-exponential" service process by an interconnection of several artificial exponential processes [23]. Unfortunately, the cost of this approach is the addition of more state variables in order to make the holding times of the states memoryless. This increases the complexity of the model to the degree that the concept of state is no longer intuitive to the analyst and analytic tractability is lost. A host of service distributions is available by this strategem, namely all with a rational Laplace transform (RLT). Figure 3 shows several derived distributions which find important applications in the modeling of computer systems and their equivalent exponential stages. A job's traversal through this network of stages is probabilistic, reflected by branching probabilities. The time spent in service by a job at each stage is exponentially distributed. The average service time of the derived distribution is the sum of the average service times at each of the stages, weighted by the probability that a job reaches that stage. The additional state variables designate which stage is "active" for each of the jobs receiving service in the network.

Some queueing disciplines allow us to forego the cost of the additional state variables with the "stageing" technique. Chandy showed in [1,5] that queueing networks of LCFSPR and PS nodes behave in the same manner for all RLT service distributions. This result means that FCFS networks with memoryless service distributions and LCFSPR and PS networks with RLT service distributions behave identically for equal network topologies, branching probabilities, and mean service rates, even if they must be analyzed differently. The remainder of this section analyzes a FCFS network with exponential service distributions - which is equivalent in behavior of a network arbitrarily comprised of FCFS queues with exponential service distributions and PS and LCFSPR queues with RLT service distributions.

The Balance Equations for a Closed Queueing Network

In this section we apply the methods of solution of Markov processes developed previously to the class of queueing networks

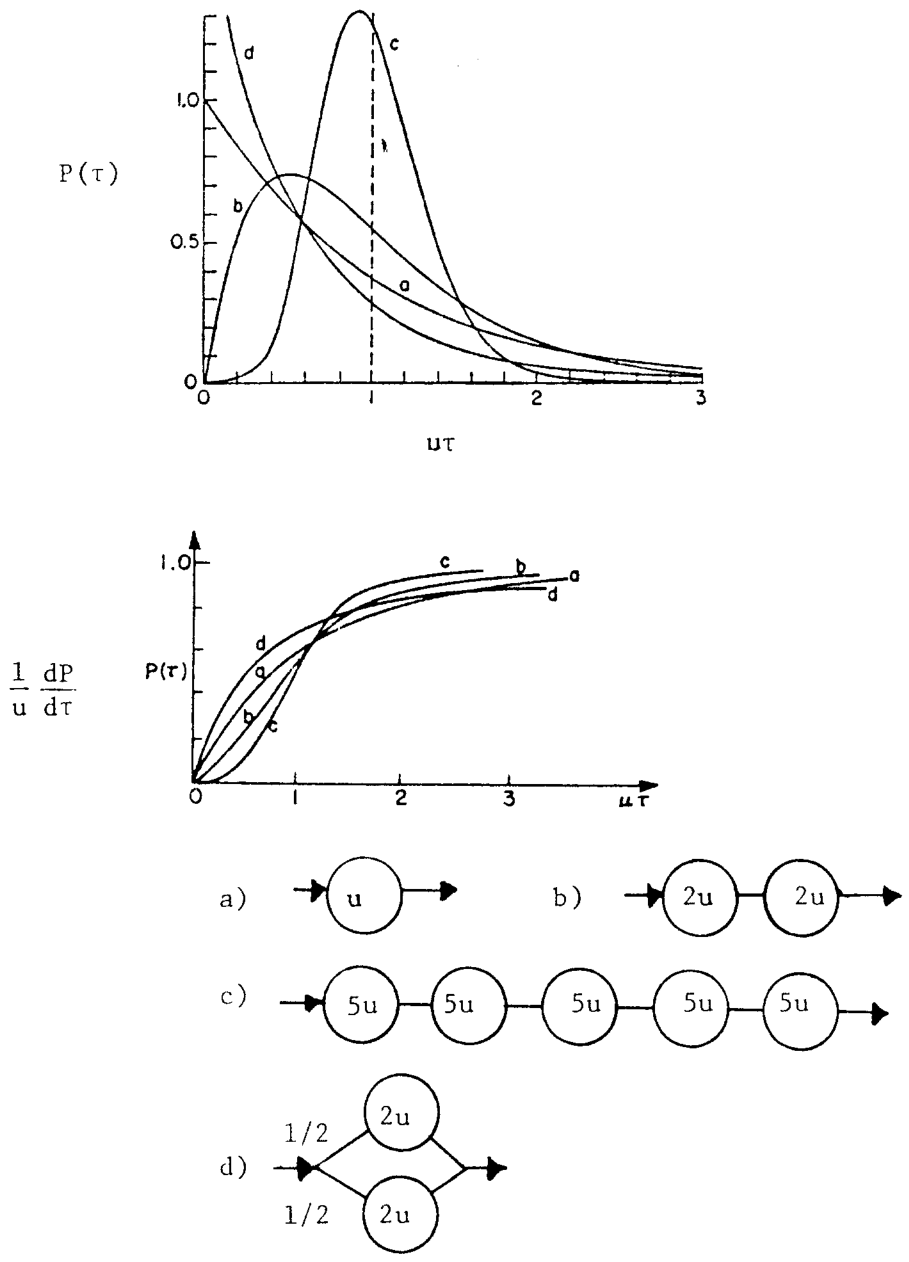

Figure 3 (from Morse). Distribution functions and density functions derived from the exponential and their equivalent exponential stages. The mean of each distribution is 1/u. (a) Exponential (b) Hypoexponential of degree 2 (c) Hypoexponential of degree 5 (d) Hyperexponential, second order.

defined in the preceding section.

For an arbitrary, closed queueing network the set of regular Markov balance equations must be solved in order to obtain the steady state probabilities of the queueing system. A state of the system can be represented by a state vector $(n_1,\ldots,n_L)$ where n_i is the number of customers in the i^{th} node and there are L nodes in the network. The number of customers circulating in the network is at all times the constant N, thus $\sum_{i=1}^{L} n_i = N$.

Each vector is a state in a finite-state Markov chain. Transition paths between Markovian states exist due to the network paths between nodes. The system changes state when a customer moves from one node to another. Consider the queueing system to be in some state $S_i = (n_1,\ldots,n_{i-1},n_i+1,n_{i+1},\ldots,n_{j-1},n_j-1,n_{j+1},\ldots,n_L)$. The system can move to state $S_j = (n_1,\ldots,n_{i-1},n_i-1,n_{i+1},\ldots,n_{j-1},n_j+1,$ $n_{j+1},\ldots,n_L)$ if a customer, finishing service at node i, goes to node j. This transition will be referred to as the transition from state S_i to state S_j due to the movement of a customer from the i^{th} to the j^{th} queue. Note that, in general, S_i can be entered from many states and that the system can transit from S_i to many other states. The system can move out of state S_i in as many ways as customers can exit the nodes populated according to the state vector of S_i. The system can move into some state S_j from any state in the set $\{S_i : p_{ij} > 0;\ i = 1,L\}$, where P_{ij} is the branching probability from node i to node j.

A Markov process achieves steady state if it forms a finite, irreducible, Markov chain [17]. The state space is obviously finite as the total number of jobs in the system is constrained to be a constant. The states form an irreducible chain because all states communicate, i.e., a state transition path exists between any two states because the jobs recirculate through the network. So the system achieves steady state.

At steady state, the rate at which the system transits from some state S must equal the rate at which it transits into S. Denote the steady state probability of state $(n_1,\ldots,n_L)$ by $P(n_1,\ldots,n_L)$. The Markovian equations of balance for a state are thus

$$\sum_{i=1}^{L} \sum_{j} P(n_1,\ldots,n_{j-1},n_j+1,n_{j+1},\ldots,n_{i-1},n_j-1,n_{i+1},\ldots,n_L)u_j p_{ji}$$

$$= \sum_{i=1}^{L} P(n_1,\ldots,n_{i-1},n_i,n_{i+1},\ldots,n_{j-1},n_j,n_{j+1},\ldots,n_L)u_i \qquad (2.4)$$

where P_{ji} is as defined and u_i is the average service rate of node i (and is thus the inverse of the mean service time μ_i of node i). These coupled linear equations can be solved by methods already discussed or similar matrix techniques if the branching probabilities and service rates are restricted to numerical values. Jackson [19] and Gordon and Newell [9] found analytic solutions to (2.4) if the service distributions of the devices are exponential.

For the class of closed queueing networks already defined a much stronger condition holds, the <u>local balance</u> condition that:

The rate of transition into any state, due to the movement of a job into the ith queue is equal to the rate of transition out of that state, due to the movement of a job out of the ith queue. [1]

This condition was discovered in 1971 by Chandy and developed to include many types of queueing networks [1,5], of which the networks considered in this work are a subset. The fact that the local balance condition holds true allows us to equate a single term on the left-hand side of (2.4) with a single term on the right-hand side of (2.4); in effect, "loosening" the coupling of the equations. Following is the proof that (2.5) are sufficient for (2.4) and that solutions of a very simple form exist.

Given the local balance equations for the steady state probabilities:

$$\sum_j P(k_1,\dots,k_{j-1},k_j+1,k_{j+1},\dots,k_{i-1},k_i,k_{i+1},\dots,k_L)u_j p_{ji}$$

$$= P(k_1,\dots,k_{j-1},k_j,k_{j+1},\dots,k_{i-1},k_i+1,k_{i+1},\dots,k_L)u_i \qquad (2.5)$$

which state that the rate of transition into state $(k_1,\dots,k_{j-1},k_j,$ $k_{j+1},\dots,k_{i-1},k_i+1,k_{i+1},\dots,k_L)$ due to a customer moving into queue i equals the rate of transition out of the same state due to a job moving out of queue i. Note $\sum_{i=1}^{L} k_i = N-1$. We assert that (2.5) $\Rightarrow$ (2.4) simply by adding. We can prove solutions (2.5) exist of the form

$$P(n_1,\dots,n_L) = \text{NORM} \cdot \prod_{i=1}^{L} \left(\frac{btm_i}{u_i}\right)^{n_i} \qquad (2.6)$$

where $\text{NORM} = \sum_{\text{all states}} P(n_1,\dots,n_L)$ and btm_i is to be defined.

Define a term E such that

$$E = \text{NORM} \cdot \prod_{i=1}^{L} \left(\frac{btm_i}{u_i}\right)^{k_i}$$

Then the left-hand side of (2.5) is

$$\sum_j P(k_1,\ldots,k_{j-1},k_j+1,k_{j+1},\ldots,k_{i-1},k_i,k_{i+1},\ldots,k_L)u_j p_{ji}$$

$$= \sum_j E \cdot (u_j)^{k_j-1} \cdot btm_j \cdot u_j \cdot p_{ji}$$

$$= E \cdot \sum_j btm_j \cdot p_{ji}$$

and the right-hand side is

$$P(k_1,\ldots,k_{j-1},k_j,k_{j+1},\ldots,k_{i-1},k_i+1,k_{i+1},\ldots,k_L)u_i$$

so if we let $btm_i = E \cdot btm_i$ be any function such that

$$btm_i = \sum_j btm_i \; p_{ij}$$

then (2.6) solves (2.5).

That the local balance equations are sufficient conditions for the usual balance equations can be shown by merely adding all L cases of (2.5), thus

$$\sum_{i=1}^{L} \sum_j P(k_1,\ldots,k_{j-1},k_j+1,k_{j+1},\ldots,k_{i-1},k_i,k_{i+1},\ldots,k_L)u_j \; p_{ji}$$

$$= \sum_{i=1}^{L} P(k_1,\ldots,k_{j-1},k_j,k_{j+1},\ldots,k_{i-1},k_i+1,k_{i+1},\ldots,k_L)u_i$$

By changing notation we obtain:

$$\sum_{i=1}^{L} \sum_j P(n_1,\ldots,n_{j-1},n_j+1,n_{j+1},\ldots,n_{i-1},n_i-1,n_{i+1},\ldots,n_L)u_j \; p_{ji}$$

$$= \sum_{j=1}^{L} P(n_1,\ldots,n_{i-1},n_i,n_{i+1},\ldots,n_{j-1},n_j,n_{j+1},\ldots,n_L)u_j$$

Q.E.D.

In general there are L terms on either side of (2.4). Thus one equation of the form (2.4) gives rise to L equations of the form (2.5). Many of the equations of (2.5) may be redundant. The proof shows that (2.5) are sufficient, but not necessary, conditions for (2.4) to hold. Intuitively, btm_i is the probability that a job reaches node i in a single traversal through the network, with the btm of the origin and destination being 1.

Chandy has shown [1-5] that local balance holds for a wide variety of queueing networks. These queueing networks may be open or closed, with or without the "loop-free" property, and customers may be partitioned into "types". Each type observes different branching probabilities and service distributions, while obeying the same network topology. Solutions of form similar to (2.6) can be obtained to all queueing networks for which local balance holds.

CHAPTER III

ASQ

Using ASQ

The program ASQ accepts a representation of an arbitrary closed queueing network of the class defined in the previous chapter. All jobs are assumed to be of the same type, that is, branching probabilities and service time distributions are independent of the job. Queues are assumed to be of infinite capacity and FCFS, PS, or LCFSPR. Nodes that are FCFS must have exponential distributions. Any service distribution with a rational Laplace transform is accepted for PS and LCFSPR nodes. The service distribution for a node may be dependent upon the number of jobs in its queue. If so, the mean service time for each number must be specified. This can be thought of as there being more than one device servicing the queue, with the devices in parallel. In the following figures this dependency is represented by parallel devices, where each device serves only one job at a time. A node is specified by its queue's label q_i. Devices do not possess labels.

A probabilistic path between nodes is specified by an edge, the triple (q_i, p_{ij}, q_j) where p_{ij} is the previously defined branching probability. Such an edge specifies the existence of q_i, q_j, and the unique p_{ij}. The service distribution(s) of a node q_i is specified by a list of mean service times $\mu_1, \mu_2, \ldots, \mu_m$, $m \geq 1$, where μ_j is the mean service time when there are j jobs in the queue. For $j \geq m$ the mean service time is μ_m. This corresponds physically to m parallel devices servicing the queue. A state of the network is $(n_1, n_2, \ldots, n_L)$ with n_i the number of jobs in node q_i and L the number of nodes in the network. As the network is closed $\sum_{i=1}^{L} n_i = N$, where N is the total number of jobs. The integer index i for node label q_i is assigned to each node according to its position in the template, which orders all node labels in a list $(q_1, q_2, \ldots, q_L)$. Branching probabilities and mean service times are polynomials, which are discussed in the next section.

A network is described to the program in the following manner. Program responses are merely indented from the left whereas user responses are not. The example is taken from Figure 4. It may be helpful to refer to the program flowchart in Figure 5.

```
        *INPUT EDGES: NODE/PROBABILITY/NODE;

ORIG/X/Q1;
ORIG/1-X/Q2;
Q1/0.43/Q3;
Q1/0.57/Q4;
Q2/2*M/Q3;
Q2/V/Q4;
Q2/1-2*M-V/Q5;
Q3/1/DEST;
Q4/1/DEST;
Q5/1/DEST;
→;
```

The character ";" is the line delimiter for the user, while "→" signals end of user input for that passage.

```
        *INPUT MEAN SERVICE TIMES: NODE/TIME (1 JOB)/TIME (2 JOBS)//;
    Q1/0.183/;
    Q2/W-W^2/0.5*W-0.5*W^2/;
    Q3/W/;
    Q4/Z+5*W/;
    Q5/1.4*W/;
    →;
```

At this point the program checks the network for any nodes not reachable from the origin and any nodes from which the destination cannot be reached. If no such nodes are found the program goes into _command mode_, in which user commands are accepted and evaluated. If such nodes are found the network is not properly connected and an error message is output to the user. When this occurs, only the network changing set of commands is accepted as user input (such as _dedge_ and _amean_). Upon the receipt of a command not in the network changing set, the validity of the network is again checked, and the above procedure repeated if necessary. When ASQ is in command mode any network changing command is accepted, but a valid network must be represented in the program structure before any of the non-network changing commands is evaluated. The network changing commands are useful because they allow the user to more strongly interact with the network model. After the evaluation of some network, the user may change the network on the basis of that evaluation without having to re-input a slightly different network in its entirety. This is a valuable trait for a network design process.

After an analyzable network is obtained, ASQ gives the user the option of specifying the template. This is solely for the

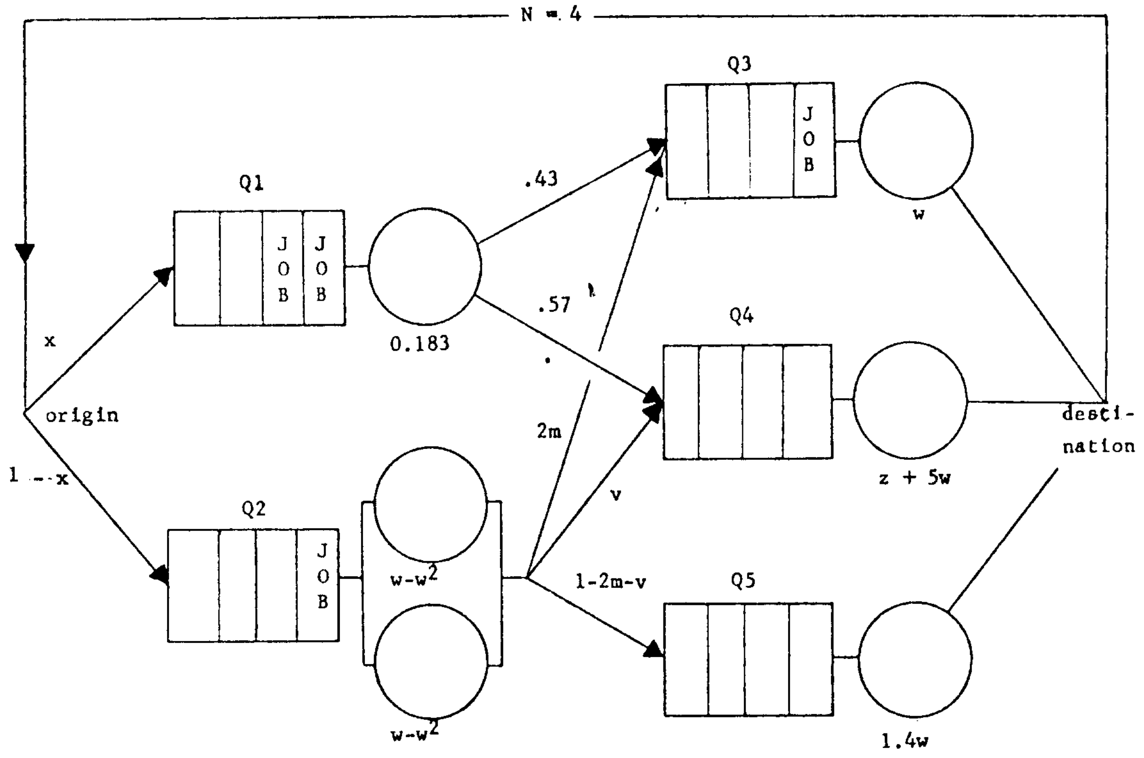

Figure 4. A Closed Queueing Network

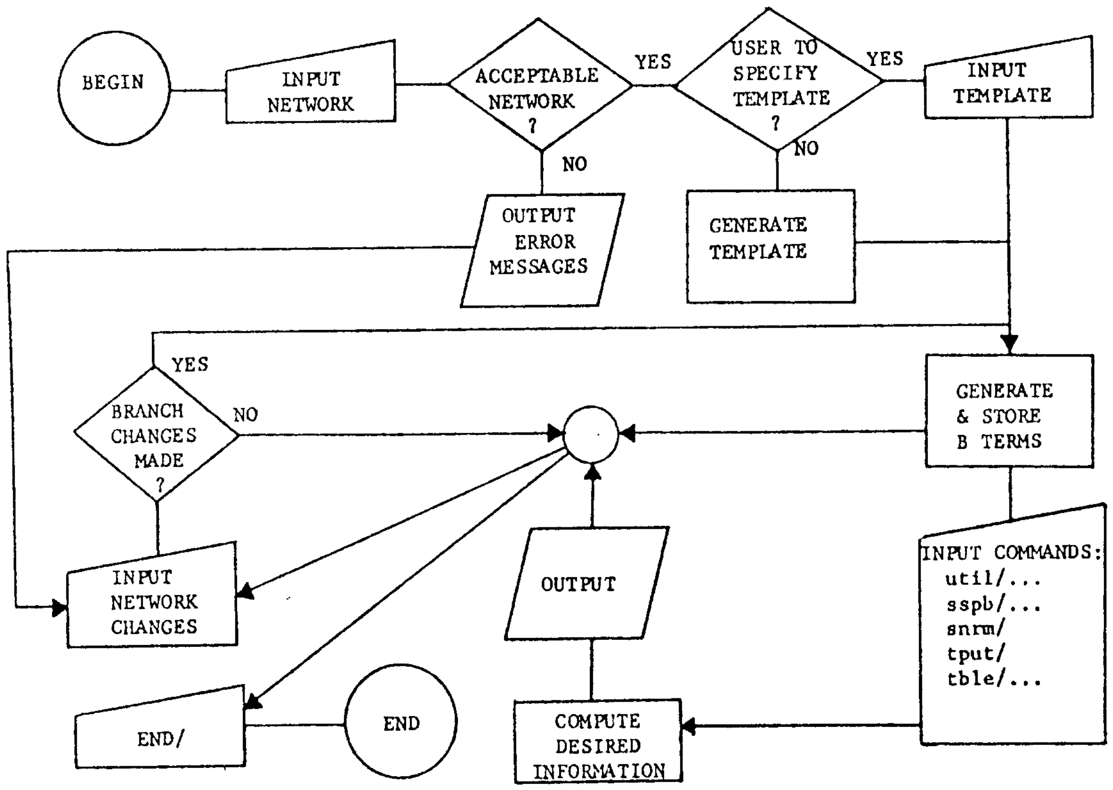

Figure 5: Schematic of Control Flow in ASQ

convenience of the user, who may wish to order the state vector representation to his liking.

```
    *DO YOU WISH TO SPECIFY THE TEMPLATE ?: YES/NO;
```

A "yes" response requires inputting the template vector.

```
    YES;
    T =
    (Q5,Q4,Q3,Q2,Q1);
```

If the user does not wish to specify the template, one is supplied by the program, and output with the same format. The user is then requested to input the origin and destination labels and the total number of jobs in the network. We again use the network in Figure 4 as an example.

```
     *INPUT ORIGIN LABEL
   ORIG;
     *INPUT DESTINATION LABEL
   DEST;
     *INPUT TOTAL NUMBER OF JOBS IN NETWORK
   4;
```

The labels for origin and destination, as for any node, may be arbitrary LISP atoms. The origin and destination may be labeled points, as in the example, or nodes. If they are nodes, then the appropriate node label must be given. The path between the destination and origin is implied and need not be specified.

When the program is in command mode the following commands are available to the user.

Network Changing Commands:

AMEAN/$q_i/\mu_1/\mu_2/\ldots/\mu_m$;

AMEAN specifies a new mean list for a node with label q_i.

If the node already exists then the previous mean list is destroyed. If the node did not previously exist then a new node with label q_i is created in the network.

ANEDGE/$q_i/p_{ij}/q_j$;

ANEDGE creates a new edge (q_i, p_{ij}, q_j).

DEDGE/q_i/q_j;

DEDGE destroyes the edge with nodes q_i and q_j.

JOBS/N;

JOBS changes the total number of jobs in the network to N, an integer.

Evaluation Commands:

SSPB/$(n_1,n_2,\ldots,n_L)/(n_1',n_2',\ldots,n_L')/\ldots/$;

SSPB outputs the unnormalized steady state probability for each of the specified state vectors.

SNRM/;

SNRM outputs the normalization factor for the steady state probabilities.

TPUT/;

TPUT outputs the unnormalized throughput rate for the queueing network.

IDLE/node$_1$/node$_2$/.../;

IDLE returns the unnormalized fraction of time for which no jobs occupy each respective node in the node list, where node$_i$ is a node label.

ASSP/;

ASSP returns a table of all unnormalized steady state probabilities for the network.

Other Commands:

CTPL/;

CTPL results in the program again asking if the user wishes to specify the template.

END;

END terminates command mode, with the control level returning to interactive LISP.

TBLE/ command;

TBLE is a special command, useful when some behavior of the queueing network is desired for study as the function of a single variable. The commands SSPB and one attendant state vector, IDLE and one attendant node label, or TPUT may be used for command. A polynomial is obtained as the answer to the evaluation command given. The user then supplies a single variable label to the program. The user then inputs numerical values for all other variables in the polynomial, which results in the program obtaining the polynomial as a function of a single variable. A lower limit, upper limit, and incremental step are then input for this variable - resulting in a table of polynomial values versus variable values. The maximum and minimum values of the polynomial, and the corresponding variable values are also output.

Examples of the uses of these commands may be found in Appendix A, the examples of interactive sessions with ASQ. ASQ may also be run as a batch program, in which case the user's "responses" to the program prompting must be anticipated and assembled into a data file.

Polynomial Representations in ASQ

The problems of choosing a functional form for algebraic information in ASQ fell squarely in the area of symbolic and algebraic manipulation. Research in that field has clearly emphasized that the form must be chosen to suit the purposes of the application. It was decided that a polynomial in one or more variables was sufficiently general. Rational functions are unnecessary because the algorithms for obtaining the unnormalized steady state probabilities for polynomial service times and branching probabilities can be formulated without division. Normalized steady state probabilities are available to the user by simply performing the division of the unnormalized probabilities by the normalizing factor, which is also obtainable without division. The throughput rate of a network cannot be formulated without division; however, it can be expressed as a sum of fractions on the printed page, the numerators and denominators of the fractions being polynomials. Such a superficial external representation is satisfactory because the throughput rate need never be used in the program as an algebraic entity.

There exist several forms for the internal representation of a polynomial in many variables [11,15,21]. Because the storage for a polynomial must be dynamic, as polynomials can be of arbitrary complexity, and because algebraic manipulations lend themselves to list-structured recursive techniques, a MACSYMA - like representation [11] in the programming language LISP was chosen. A polynomial in a single variable in this representation is a list of (coefficient.exponent) dotted pairs [16]. Thus $3x^2 + 2x + 1$ can be represented as the list ((3 . 2)(2 . 1)(1 . 0)) with the variable x being implicitly understood. A desirable property of any representation is that it be canonical, i.e., unique. Thus a canonical representation of $(x^2 + 2x - 1) + (1 - x)$ would be ((1 . 2)(1 . 1)) and not ((1 . 2)(2 . 1)(-1 . 0)(1 . 0)(-1 . 1)). We can impose uniqueness upon this "list of pairs" representation by requiring that all pairs with identical exponents be combined, and that pairs be ordered by decreasing exponents.

For a polynomial in two or more variables this structure becomes more complex. Uniqueness may be achieved by imposing an ordering upon the variables, such that one variable takes precedence over another. For a precedence relation of $x > y$ the polynomial $3x^2y^2 + 2yx + 1$ can be structured as

(((3 . 2) . 2)((2 . 1) . 1)((1 . 0) . 0))

where each coefficient in an x-pair is a term in the next-ordered variable y; thus ((3 . 2) . 2) corresponds to $[3y^2]x^2$. An important drawback of this method lies in the fact that each algebraic term must be considered to contain as many factors as variables. Thus for a precedence relation $u > v > w > x > y > z$ the term $u \cdot y$ must be structures $u^1 \cdot v^0 \cdot w^0 \cdot x^0 \cdot y^1 \cdot z^0$ which is ((((((1 . 1) . 0) . 0) . 0) . 1) . 0). For "sparse" terms, terms of few variables, a more economical representation is possible by explicitly stating the variables. The above example could be structured as (u((y(1 . 1)) . 1)). More formally, if $P(x_1, x_2, \ldots, x_n)$ is the polynomial P of precedence ordered variables $x_1 > x_2 > \ldots > x_n$ then a canonical form can be recursively defined as

$$P(x_1, \ldots, x_n) = \sum_{i=1}^{r} P_i(x_1, \ldots, x_{n-1}) \cdot x_n^{e_i},$$

where $e_1 > e_2 > \ldots > e_r \geq 0$ and the $P_i(x_1, \ldots, x_{n-1})$ are all non-zero.

In the internal polynomial list, x_n is the first element in a list of dotted pairs, the first element of a pair being $P_i(x_1, \ldots, x_{n-1})$ and the second element the integer e_i.

Obviously there is a tradeoff between implicit and explicit internal representation. A polynomial in many variables but with sparse terms is better represented in the explicit manner, while polynomials in few variables or with non-sparse terms are better represented in the implicit manner. The explicit representation was chosen for ASQ as the former type of polynomial was anticipated to be the more common. A formal syntax is given in Figure 6 for the external polynomial representation, which is the representation for user/program interaction, and the internal program representation of polynomials. Routines in ASQ make the appropriate translations between external and internal representations. Examples of these representations can be found in Figure 7.

The fact that a polynomial can be recursively defined implies that the operations of addition and multiplication upon two polynomials can be achieved by recursive algorithms.

As $P(x_1,\ldots,x_n) = \sum_{i=1}^{r} P_i(x_1,\ldots,x_{n-1}) \cdot x_n^{e_i}$

then $P_1(x_1,\ldots,x_n)+P_2(x_1,\ldots,x_n) = \sum_{i=1}^{r} [P_{1i}(x_1,\ldots,x_{n-1}) + P_{2i}(x_1,\ldots,x_{n-1})] \cdot x_n^{e_i}$

and $P_1(x_1,\ldots,x_n)*P_2(x_1,\ldots,x_n) = \sum_{i=1}^{r} \sum_{k=1}^{s} P_{1r}(x_1,\ldots,x_{n-1}) * P_{2k}(x_1,\ldots,x_{n-1})]\ x_n^{e_i+e_k}$

where $e_1 > e_2 > \ldots > e_r \geq 0$ and $e_1 > e_2 > \ldots > e_s \geq 0$.

Again, the recursive features in the programming language LISP make the implementation of these algorithms straightforward.

External Representation

```
<polynomial> ::= <term>{<addop><term>}*
<term> ::= <factor>{<mulop><factor>}*
<factor> ::= <primary>{↑<integer>}
<primary> ::= <number>|<variable>|(<polynomial>)
<number> ::= <integer>|<integer>.<integer>
<variable> ::= <letter><char>*
<char> ::= <letter>|<digit>
<integer> ::= <digit><digit>*
<addop> ::= +|-
<mulop> ::= *
```

Blanks may appear anywhere within a polynomial .

Internal Representation

```
<polynomial> ::=  <number>|(<variable><term><term>*)
<term> ::= (<coefficient>·<exponent>)
<coefficient> ::= <polynomial>|<real number>
<exponent> ::= <integer>
<variable> ::= <letter><char>*
<char> ::= <letter>|<digit>
```

Figure 6. Syntax of the Polynomial Representations in BACKUS-NAUR Form

Precedence Relation: u>v>w>x>y>z

```
X

(X (1.0000000000 . 1))


1.565*X↑2

(X (1.5650000000 . 2))


1.5*X↑2*Y↑2 + (1.4 + Y)*X

(X ((Y (1.5000000000 . 2)) . 2) ((Y (1.0000000000 . 1)
(1.400000000 . 0)) . 1))


2*X↑3+0.9*X↑2*Y-2*Y*Z+1

(X (2.000000000 . 3) ((Y (0.9000000000 . 1)) . 2) ((Y ((Z (
-2.0000000000 . 1)) . 1)(1 . 0)) . 0))


X↑2 + Y↑2 + Z↑2 + U↑3 + V↑4

(U (1.0000000000 . 3) ((V (1.0000000000 . 4) ((X (1.0000000000 . 2) ((
Y (1.0000000000 . 2) ((Z (1.0000000000 . 2)) . 0)) . 0)) . 0)) . 0))


X*Y*Z*U*V*W

(U ((V ((W ((X ((Y ((Z (1.0000000000 . 1)) . 1)) . 1)) . 1)) . 1)) . 1
))


1.3*X↑5 + 2.9*X↑3 + 0.33*X↑2 + X + 1
(X (1.3000000000 . 5) (2.9000000000 . 3) (0.3300000000 . 2) (
1.0000000000 . 1) (1 . 0))
```

Figure 7. Examples of the Polynomial Representations

CHAPTER IV

COMPARISONS AND CONCLUSIONS

Other Work in Automatic Analysis of Markov Processes

Wallace and Rosenberg [14] presented a program in 1966 capable of analyzing any closed queueing network that could be modeled as a finite state Markov process. Their approach was to obtain the numerical matrix characterizing the Markov balance equations of the network and then manipulate the matrix by a power-iteration technique to obtain the steady state probabilities. Storage for the elements comprising the matrix was the limiting factor of their program. A sparse matrix representation was used and some of the program implemented in machine code, with the result that 1,000 state networks could be analyzed numerically. The solutions were not exact, but iterations were made until the solutions were as exact as desired. The thrust of their work was to provide a cheaper alternative to the analysis of computer systems than simulation. They argued that as computer systems become more complex and more susceptible to interference between tasks, simulations become either too expensive or their probability estimates too imprecise, since accuracy incurs the expense of very large samples.

Engleman and Kleinman [12] developed a program in 1972 that obtains analytic solutions of finite state Markov processes. Input to their program consists of the state transition matrix characterizing the Markov process, the elements of the matrix being analytic functions.

Solutions to the steady state probabilities are obtained by standard matrix techniques, with the operations being supported by MATHLAB, a system for symbolic computation. This approach is similar to that of Wallace and Rosenberg, only symbolic rather than arithmetic operations are performed. Thus the usual storage problem is present, only made worse by the additional storage requirement of the symbolic coefficients. Also, the symbolic coefficients make the advantageous power-iteration technique impossible. These limitations make the analytic solution of all but the most trivial closed queueing networks infeasible.

ASQ attacks the analysis differently. For those closed queueing networks for which local balance holds, the steady state probabilities are obtained from the network topology, rather than from a transition matrix. Thus a large matrix need not be constructed nor manipulated. Even the states of the network need not be stored, only enumerated. The user has the choice of numerical or algebraic solution. For the later approach a network topology need be analyzed only once by the program, as the solutions are functions of the algebraic network parameters. The solution of the network for any particular values of branching probabilities and mean service times is obtained by a simple substitution into the polynomial.

Whether it is to the advantage of the user to use values or functions or a combination of both for the network parameters depends upon the problem. Suppose it is desired to find the maximum throughput rate of a network with n unknown branching probabilities. Assume the values of the mean service times are known. The analyst must therefore search an n-variable space for the throughput maximum. He may assign numerical values to the probabilities and obtain a value for the throughput every time the value of a parameter is changed. Taking this approach means "solving" the network at each iteration of the search. The other approach is to assign simple variables $p_1, p_2, \ldots, p_n$ to the probabilities, and then obtain the throughput polynomial $P(p_1, p_2, \ldots, p_n)$ by solving the network once. The polynomial is then evaluated for its maximum. Which approach is the more practical depends upon the nature of the polynomial, a function of the complexity of the network topology and the number of jobs in the network. Also, the number of iterations necessary in the first approach, must be considered. The choice of which course of analysis to pursue in general is in the realm of numerical versus symbolic analysis. The advantage of algebraic solutions is that they contain a much greater amount of information about system behavior than a set of solved numerical "network configurations". This additional information allows the asymptotic behavior of the network to be easily derived from a polynomial. The disadvantage of algebraic solutions in this case is that the particular information the user might desire can too often be buried in pages of lengthy polynomials. Where the tradeoff lies between numerical and symbolic evaluation of the class of equations characterizing queueing networks is a question for further research.

Applications

ASQ was envisioned as a recourse to the time-consuming efforts required in constructing and evaluating models of multi-programmed computer systems. Before the realization of local balance the system of linear equations characterizing a queueing network of the type in Appendix A with PS and LCFSPR disciplined queues could require a great deal of work toward a solution. Local balance reduced the complexity of the system of equations, transforming the nature of the analysis into still tedious algorithms. ASQ implements the algorithms. The analysis of models now requires little more of the analyst's time than the description of the network. Freeing the analyst from the characteristic equations raises the level of abstraction in the analysis. ASQ is then an analytical tool, and importantly, one that can be used by that naive analyst -- the student. As a convenient and interactive teaching tool, ASQ allows students to develop an intuition for network analysis -- and at a much lower cost than simulation.

Another application for the program is the verification of simulation studies. Analytic techniques are very helpful in simulation validation in at least two ways. The simulation model must be constrained to be analytically tractiable or a simpler model used for validation. Firstly, when analytical results and simulation results do not agree an error in code can be suspected in the simulation program. Secondly, analytical results are useful in the justification of running times (i.e., has the simulation run long enough to produce statistically reliable results?) An estimation of valid times can be obtained by running the simulation until the simulation results agree. Lasseter [20] used ASQ to verify a simulation of a multiprogrammed computer system. Foster [22] verified a simulation study of a computer memory hierarchy with an analysis of his model performed by ASQ. (A simplified version of Foster's study may be found in Appendix A.)

Limitations

A severe limitation of the program is the storage requirement for algebraic analysis. Even simple network topologies generate large polynomials due to the many products in the algorithms. There is also the classic nagging doubt that "90% of the computation is spent on 10% of the important information", in our case, the important characteristics of system behavior. One recourse would be to implement decision routines to discard "non-significant" terms during the polynomial multiplications. This is impractical unless bounds are placed upon the ranges of the variables in the analytic functions by the user -- which begins to defeat the purpose of algebraic analysis, its generality. An obvious recourse is to constrain all network parameters to numerical values, and settle for a set of numerical points on a graph. This, of course, is what has been done previously. The ASQ algorithms are still faster than the matrix methods currently available.

The most important limitation is in the generality of acceptable queueing networks. Many important queueing network models lie outside the realm of "local balance." In some instances it is possible to develop an analyzable network which behaves in almost the same manner as the unanalyzable network and apply the results. In general, this is not the case.

Extensions

ASQ has proved a useful tool in aiding the systems analyst in studying certain kinds of queueing systems. This work clears the way for the construction of a program to analyze all queueing network models satisfying local balance, one in which the punitive effect of some limitations in ASQ can be avoided with care. Work has begun on this extended version of ASQ [26], implemented in the programming language FORTRAN. Use of a compiled, rather than interpreted, language will result in shorter running times for analyses. Another run-time advantage results from the network evaluation routines in the FORTRAN ASQ being partitioned so that evaluation proceeds in either a numeric or an algebraic mode. This means that the algebraic manipulation routines need be loaded only if necessary, resulting in considerable storage savings when only numerical results are required. Algebraic operations are handled by a slightly modified SAC-1 system [15]. The usual wealth of trivial improvements gained by hindsight is being incorporated into the extension. Improvements in processing efficiency for the FORTRAN ASQ and its ability to analyze a more complex class of models will result in solutions richer in detail being made available to the analyst. Further extensions in analytic capability are anticipated as new developments in queueing theory result in new algorithms for model evaluation.

It is important to note that the implementation of a practical analytical tool was possible only because of a theoretical advance. The author feels that the scope and direction of future systems for automatic analysis in the field of queueing theory are tightly bound to theoretical developments. The Gordian knot of factorially increasing complexity can not be resolved by new programming techniques nor improved machinery, but only by unifying assumptions that raise the level of abstraction in which we consider the problem. Furthermore, breakthroughs in theory must be made available to the practicing systems analyst such that they can be put to use simply and effectively. Theory is a means to an end: the better analysis of systems, and theoretical advances must be rendered into readily usable forms. ASQ is a step in that direction.

APPENDIX A

EXAMPLES OF INTERACTIVE SESSIONS WITH ASQ

Two examples are provided. In the first, a queueing network model of a memory hierarchy is analyzed (Figure 8). The purpose of the analysis is to determine the optimal partitioning of accessable data for a maximum "fetch" rate; that is, the partition of data between the two drums so that fetches from main memory proceed at an optimal rate. This corresponds to obtaining the value of x in the model for which the throughput rate is maximized. The branching probability x is the probability that a fetch is aimed at the fast drum. The numbers in the figure are the unitless mean service times for each device.

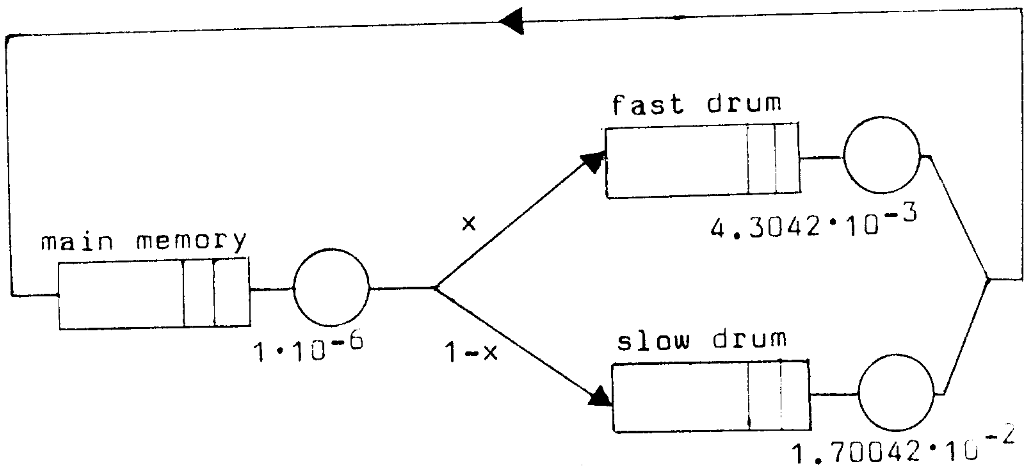

Figure 8. Model of a Memory Hierarchy

ASQ determines the throughput of the closed network as a function of x for degrees of multiprogramming 3 and 4 during the session. The model's throughput rates as a function of x for degrees of multiprogramming 1,2,3, and 4 are plotted in Figure 9.

In the second example a queueing network model with the same network topology pictured in Figure 4 is described to ASQ. All network parameters are numeric. Various commands are exercised throughout the interactive sessions for the purpose of demonstrating their usage.

```
NETIN()

 *INPUT EDGES: NODE/PROBABILITY/NODE;

ORIG/1/MAINMEM;
MAINMEM/X/FASTDRUM;
MAINMEM/1-X/SLOWDRUM;
FASTDRUM/1/DEST;
SLOWDRUM/1/DEST;
⌐;

 *INPUT MEAN SERVICE TIMES: NODE/TIME (1 JOB) /TIME (2 JOBS) / /;


MAINMEM/0.000001/;
FASTDRUM/0.0043042/;
SLOWDRUM/0.0170042/;
⌐;
 TEMPLATE=(MAINMEM FASTDRUM SLOWDRUM)
 *INPUT ORIGIN LABEL

ORIG;

 *INPUT DESTINATION LABEL

DEST;

 *INPUT TOTAL NUMBER JOBS IN NETWORK

3;

 *DO YOU WISH TO SPECIFY THE TEMPLATE >: YES/NO;

NO;

 T= (MAINMEM FASTDRUM SLOWDRUM)

*COMMAND

SSPR/(1,1,1)/(2,0,1)/(3,0,0)/;

 (1 1 1)
 -0.7318947764E-10*X↑2 + 0.7318947764E-10*X

 (2 0 1)
 -0.1700420000E-13*X + 0.1700420000E-13

 (3 0 0)
 1.0000000000E-18
*COMMAND

SNRM/;
 1/ -0.3907395732E-5*X↑3 + 0.1257612650E-4*X↑2 + -0.1350590349E-4*X +
0.4916931460E-5
```

```
*COMMAND

TPUT/;
 UNNORMALIZED THROUGHPUT RATE
 +[ 0.2344794776E-9*X↑2 + -0.5051088576E-9*X + 0.2891598228E-9]/[
1.0000000000E-6]
*COMMAND

IDLE/MAINMEM/FASTDRUM/SLOWDRUM/;

 UNNORMALIZED FRACTION OF IDLE TIME OF NODE MAINMEM
 -0.3907395732E-5*X↑3 + 0.1257589202E-4*X↑2 + -0.1350539838E-4*X +
0.4916642300E-5
 UNNORMALIZED FRACTION OF IDLE TIME OF NODE FASTDRUM
 -0.4916642300E-5*X↑3 + 0.1475021604E-4*X↑2 + -0.1475050520E-4*X +
0.4916931460E-5
 UNNORMALIZED FRACTION OF IDLE TIME OF NODE SLOWDRUM
 0.7974020163E-7*X↑3 + 0.1852613764E-10*X↑2 + 0.4304200000E-14*X +
1.0000000000E-18
*COMMAND

ASSP/;



 (0 0 3)
 -0.4916642300E-5*X↑3 + 0.1474992690E-4*X↑2 + -0.1474992690E-4*X +
0.4916642300E-5

 (0 1 2)
 0.1244528516E-5*X↑3 + -0.2489057031E-5*X↑2 + 0.1244528516E-5*X

 (1 0 2)
 0.2891428176E-9*X↑2 + -0.5782856353E-9*X + 0.2891428176E-9

 (0 2 1)
 -0.3150221497E-6*X↑3 + 0.3150221497E-6*X↑2

 (1 1 1)
 -0.7318947764E-10*X↑2 + 0.7318947764E-10*X

 (2 0 1)
 -0.1700420000E-13*X + 0.1700420000E-13

 (0 3 0)
 0.7974020163E-7*X↑3

 (1 2 0)
 0.1852613764E-10*X↑2

 (2 1 0)
 0.4304200000E-14*X

 (3 0 0)
 1.0000000000E-18
```

```
*COMMAND

TBLE/TPUT/;

*VAR =
X;

 LOWLIM/HIGHLIM/STEP =
0.1/0.9/0.05;

1.0000000000E-1                    6.5341917439E+1
0.1500000000                       6.9181159014E+1
0.2000000000                       7.3493781022E+1
0.2500000000                       7.8368834672E+1
0.3000000000                       8.3917156832E+1
0.3500000000                       9.0277338850E+1
0.4000000000                       9.7622968551E+1
0.4500000000                       1.0617065107E+2
0.5000000000                       1.1618690683E+2
0.5500000000                       1.2798857267E+2
0.6000000000                       1.4192328034E+2
0.6500000000                       1.5829930786E+2
0.7000000000                       1.7720217386E+2
0.7500000000                       1.9809705507E+2
0.8000000000                       2.1915269434E+2
0.8500000000                       2.3660179385E+2
0.9000000000                       2.4541940106E+2
*MAX 0.9000000000                  2.4541940106E+2
*MIN 1.0000000000E-1               6.5341917439E+1
*COMMAND

TBLE/TPUT/;

*VAR =
X;

 LOWLIM/HIGHLIM/STEP =
0.9/1.00/0.01;

0.9000000000                       2.4541940106E+2
0.9100000000                       2.4580523842E+2
0.9200000000                       2.4571469464E+2
0.9300000000                       2.4516411502E+2
0.9400000000                       2.4418285703E+2
0.9500000000                       2.4281295376E+2
0.9600000000                       2.4110815989E+2
0.9700000000                       2.3913245342E+2
0.9800000000                       2.3695812083E+2
0.9900000000                       2.3466359227E+2
1.0000000000                       2.3233121137E+2
*MAX 0.9100000000                  2.4580523842E+2
*MIN 1.0000000000                  2.3233121137E+2
```

```
*COMMAND

JOBS/4;

*COMMAND

TPUT/;

 UNNORMALIZED THROUGHPUT RATE
 +[ -0.3907395732E-11*X↑3 + 0.1257612650E-10*X↑2 + -0.1350590349E-10*X
 + 0.4916931460E-11]/[ 1.0000000000E-6]
*COMMAND

TBLE/TPUT/;

*VAR =
X;

 LOWLIM/HIGHLIM/STEP =
0.86/0.93/0.01;

0.8600000000                    2.5161748442E+2
0.8700000000                    2.5273286869E+2
0.8800000000                    2.5326518742E+2
0.8900000000                    2.5323426666E+2
0.9000000000                    2.5267779072E+2
0.9100000000                    2.5164894471E+2
0.9200000000                    2.5021283974E+2
0.9300000000                    2.4844211402E+2
*MAX 0.8800000000               2.5326518742E+2
*MIN 0.9300000000               2.4844211402E+2
*COMMAND

DEDGE/MAINMEM/FASTDRUM;
*COMMAND

DEDGE/MAINMEM/SLOWDRUM;
*COMMAND

ANEDGE/MAINMEM/0.88/FASTDRUM;
*COMMAND

ANEDGE/MAINMEM/0.12/SLOWDRUM;
*COMMAND

TPUT/;

 UNNORMALIZED THROUGHPUT RATE
 +[ 0.1079079612E-12]/[ 1.0000000000E-6]
*COMMAND

SNRM/;

 1/ 0.4260670892E-9
*COMMAND

END/;
```

```
NETIN()

 *INPUT EDGES: NODE/PROBABILITY/NODE;

ORIGIN/0.5/Q1;
ORIGIN/0.5/Q2;
Q1/0.43/Q3;
Q1/0.57/Q4;
Q2/0.1/Q3;
Q2/0.6/Q4;
Q2/0.3/Q5;
Q3/1/DESTINATION;
Q4/1/DESTINATION/;--;
Q5/1/DESTINATION;
";

 *INPUT MEAN SERVICE TIMES: NODE/TIME (1 JOB) /TIME (2 JOBS) / /;


Q1/0.183/;
Q2/0.183/0.0915/;
Q3/0.001/;
Q4/0.017/;
Q5/0.0014/;
";
 TEMPLATE=(Q1 Q2 Q3 Q4 Q5)
 *INPUT ORIGIN LABEL

ORIGIN;

 *INPUT DESTINATION LABEL

DESTINATION;

 *INPUT TOTAL NUMBER JOBS IN NETWORK

4;

 *DO YOU WISH TO SPECIFY THE TEMPLATE >: YES/NO;

NO;
 T= (Q1 Q2 Q3 Q4 Q5)

*COMMAND

SSPB/(0,1,1,1,1)/;

 (0 1 1 1 1)
 0.5063969137E-10
*COMMAND

SSPB/(0,2,2,0,0)/;

 (0 2 2 0 0)
 0.2939706281E-9
```

```
*COMMAND

TPUT/;
 UNNORMALIZED THROUGHPUT RATE
 +[ 0.2145484370E-3]/[ 0.1830000000]
 +[ 0.7904039452E-4]/[ 0.1830000000]
 +[ 0.6775402124E-4]/[ 0.9150000000E-1]
*COMMAND

SNRM/;
 1/ 0.2258450767E-3
*COMMAND

AMEAN/Q2/0.02/;
*COMMAND

IDLE/Q1/Q2/Q3/Q4/Q5/;

 UNNORMALIZED FRACTION OF IDLE TIME OF NODE Q1
 0.5138921108E-7
 UNNORMALIZED FRACTION OF IDLE TIME OF NODE Q2
 0.7905066090E-4
 UNNORMALIZED FRACTION OF IDLE TIME OF NODE Q3
 0.8848695427E-4
 UNNORMALIZED FRACTION OF IDLE TIME OF NODE Q4
 0.7910397329E-4
 UNNORMALIZED FRACTION OF IDLE TIME OF NODE Q5
 0.8854026666E-4
*COMMAND

SNRM/;
 1/ 0.8874382307E-4
*COMMAND

JOBS/1;
*COMMAND

TPUT/;
 UNNORMALIZED THROUGHPUT RATE
 +[ 0.9150000000E-1]/[ 0.1830000000]
 +[ 1.0000000000E-2]/[ 0.2000000000E-1]
*COMMAND

SNRM/;
 1/ 0.1119200000
*COMMAND

END/;
```

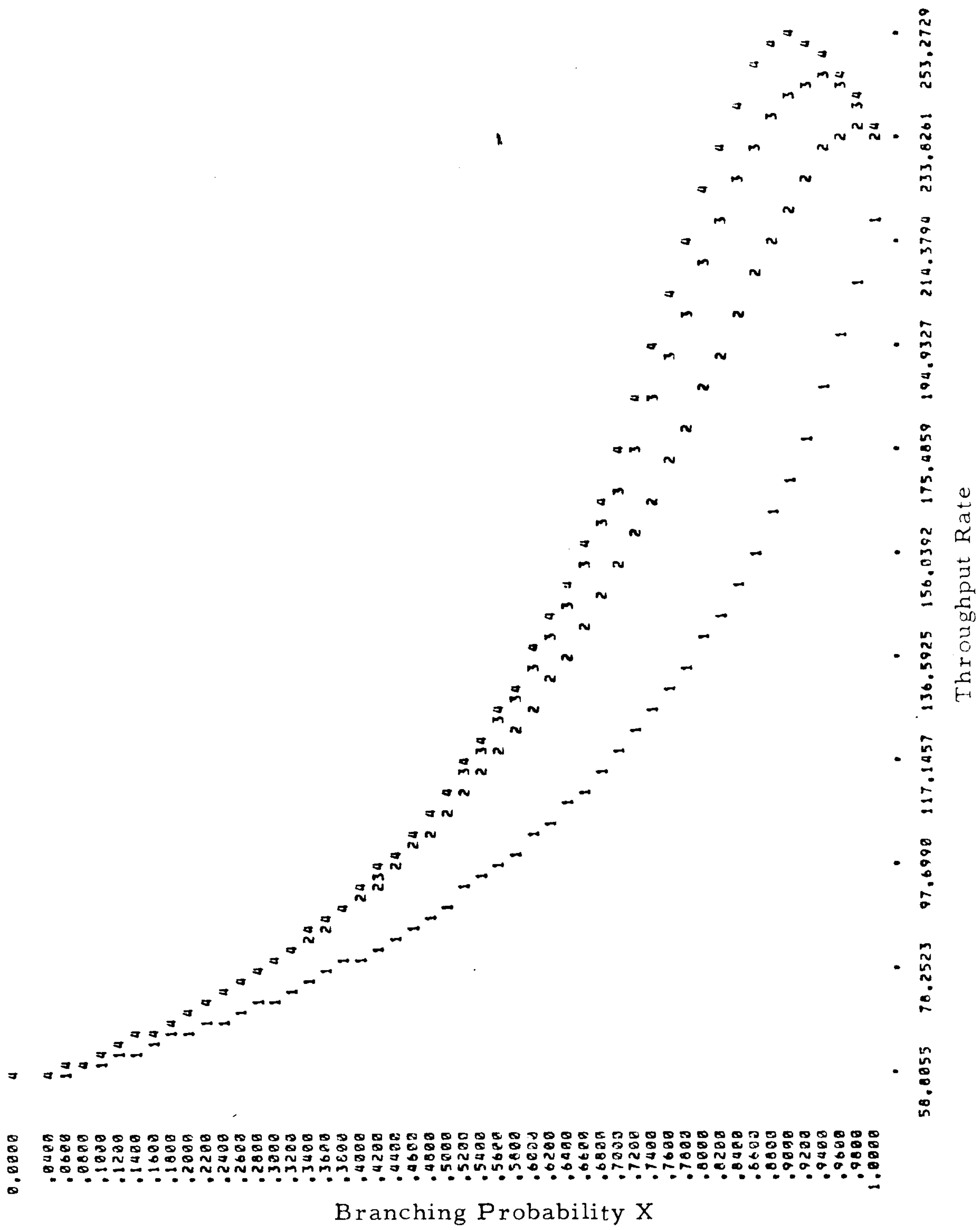

Figure 9. A Graph of System Behavior

APPENDIX B

THE PROGRAM ASQ

```
↓ ASQ: ALGEBRAIC SOLUTIONS FOR QUEUEING NETWORKS          ↓

↓ NOTE: THE DOCUMENTATION IS DELIBERATELY BRIEF,
        FOR IN MOST CASES THE VARIABLE AND FUNCTION LABELS ARE
        SELF EXPLANATORY. ↓

↓ GLOBAL VARIABLES
    VLIST: THE LIST OF VARIABLE LABELS DEFINING PRECEDENCE
    VERTICES: THE LIST OF DEVICE LABELS, SET BY ↱MEANREAD↱
                 OR ↱TEMPLATE↱.
    EDGES: THE LIST OF EDGES, SET BY ↱EDGREAD↱
    MLOP: THE LIST OF PAIRS OF THE FORM (VERTEX LABEL,MEAN)
              SET BY ↱MEANREAD↱
    BTLOP: THE LIST OF PAIRS OF THE FORM (VERTEX
              LABEL,BTERM), SET BY ↱TOPSORT↱
    ORIGIN: THE LABEL OF THE ORIGIN VERTEX
    DESTINATION: THE LABEL OF THE DESTINATION VERTEX
    DEGMULPROG: THE TOTAL NUMBER OF JOBS IN THE SYSTEM, IE.
                  THE DEGREE OF MULTIPROGRAMMING  ↓

↓ COMMANDS IS THE INTERACTIVE DRIVER PROGRAM, INTERPRETING AND
EXECUTING USER COMMANDS.  ↓
(COMMANDS(LAMBDA()(PROG(LEXLINE COM VECTOR N DUM
         END TPUT SSPB IDLE SNRM ASSP TFLAG)
      (TERPRI)
      (SETQ CTPL(QUOTE CTPL))(SETQ TBLE(QUOTE TBLE))
      (CSETQ EDGEFLAG 0)(SETQ TFLAG 0)
      (SETQ END (QUOTE END))(SETQ TPUT (QUOTE TPUT))
      (SETQ SSPB (QUOTE SSPB))(SETQ IDLE (QUOTE IDLE))
     (SETQ AMEAN(QUOTE AMEAN))(SETQ DEDGE(QUOTE DEDGE))
     (SETQ ANEDGE(QUOTE ANEDGE))(SETQ JOBS(QUOTE JOBS))
      (SETQ SNRM (QUOTE SNRM))(SETQ ASSP (QUOTE ASSP))
READLINE  (PRINT (QUOTE *COMMAND))
         (SETQ LEXLINE (LEXICAL (READER)))
T     (SETQ COM(CAAR LEXLINE))
      (SETQ LEXLINE (CDR LEXLINE))
A     (COND
           ((EQ COM DEDGE)(GO DEDGE))
            ((EQ COM ANEDGE)(GO ANEDGE))
            ((EQN EDGEFLAG 1)(GO S))
           ((EQ COM END)(RETURN NIL))
           ((EQ COM SSPB)(GO SSPB))
           ((EQ COM TPUT)(GO TPUT))
           ((EQ COM IDLE)(GO IDLE))
           ((EQ COM SNRM)(GO SNRM))
           ((EQ COM ASSP)(GO ASSP))
            ((EQ COM CTPL)(GO CTPL))
           ((EQ COM AMEAN)(GO AMEAN))
           ((EQ COM JOBS)(GO JOBS))
           ((EQ COM TBLE)(GO TBLE))
        (T (GO READLINE))
     )
S     (CHECK1 ORIGIN EDGES VERTICES)
```

```
      (CHECK2 DESTINATION EDGES VERTICES)
      (TOPSORT ORIGIN DESTINATION)
      (CSETQ EDGEFLAG 0)
      (GO A)
SSPB  (COND((NULL(CDR LEXLINE))(COND((ZEROP TFLAG)(GO READLINE))
         (T(PROG2(SETQ P COM)(GO V)))) ))
      (SETQ LEXLINE (CDDR LEXLINE))
      (SETQ VECTOR NIL)
SSPB1 (SETQ N (CAR LEXLINE))
      (SETQ LEXLINE (CDR LEXLINE))
      (COND((EQ N RPAR)(GO B3)))
      (SETQ VECTOR (APPEND VECTOR (LIST N)))
      (GO SSPB1)
B3    (SETQ COM (SSPROB VECTOR))
      (PRINT BLANK)(PRIN1 BLANK)
      (PRINT VECTOR)(SETPR COM)(TERPRI)
      (GO SSPB)
SNRM  (SETQ DUM (NORMS DEGMULPROG (LENGTH VERTICES)))
      (PRIN1 BLANK)(PRIN1 (QUOTE 1))(PRIN1 (QUOTE /))
      (SETPR RECNORM)(TERPRI)
      (GO READLINE)
ASSP  (SETQ DUM (NORMSPR DEGMULPROG (LENGTH VERTICES)))
      (GO READLINE)
TPUT (COND((ZEROP TFLAG)(GO TPUT1)))
     (SETQ TFLAG 0)(TBLETPUT)(GO READLINE)
TPUT1 (SETQ DUM (TPUT))
      (PRQUOTE *(UNNORMALIZED THROUGHPUT RATE))
      (PRTPUT DUM)(GO READLINE)
CTPL(SETQ DUM(TEMPLATE))(GO READLINE)
AMEAN(SETQ DUM(AMEAN(CDR LEXLINE)))(GO READLINE)
ANEDGE(SETQ DUM(ANEDGE(CDR LEXLINE)))(GO READLINE)
DEDGE(SETQ DUM(DEDGE(CDR LEXLINE)))(GO READLINE)
JOBS(CSETQ DEGMULPROG (CADR LEXLINE))(GO READLINE)
TBLE  (SETQ TFLAG 1)(SETQ LEXLINE(CDR LEXLINE))(GO T)
V (TBLE P)(SETQ TFLAG 0)(GO READLINE)
IDLE  (COND((NULL(CDR LEXLINE))(GO READLINE)))
      (SETQ LEXLINE(CDR LEXLINE))
      (SETQ N(CAAR LEXLINE))
      (SETQ LEXLINE(CDR LEXLINE))
      (SETQ DUM(IDLE N DEGMULPROG (LENGTH VERTICES)))
      (PRQUOTE(APPEND(QUOTE(UNNORMALIZED FRACTION OF
        IDLE TIME OF NODE))(LIST N)))
      (SETPR DUM)(SETQ TFLAG 0)(TERPRI)
      (GO IDLE)
      ) ))


↓ MEANREAD PROMPTS THE USER TO INPUT THE MEAN SERVICE TIME(S) FOR
EACH DEVICE AND SETS GLOBAL VARIABLES *MLOP* AND *VERTICES*. ↓
(MEANREAD(LAMBDA()(PROG(MEANS QMEANS LINE LEXLINE LEXPOLY
                        POLY VERTS)
      (TERPRI)
      (PRQUOTE (QUOTE (*INPUT MEAN SERVICE TIMES:
```

```
                  NODE/TIME(1 JOB)/TIME(2 JOBS)/ /;
                   )))
       (PRIN1 BLANK)
       (PRINT BLANK)
       (SETQ VERTS NIL)
       (SETQ MEANS NIL)
READLINE (SETQ LINE (READER))
       (COND ((EQ (CAR LINE) RARROW)(GO OUT)))
       (SETQ LEXLINE (LEXICAL LINE))
       (COND((NOT(MEMBER (CAAR LEXLINE) VERTS))
             (SETQ VERTS (CONS (CAAR LEXLINE) VERTS))))
       (SETQ QMEANS (CAR LEXLINE))
       (SETQ LEXLINE (CDDR LEXLINE))
AA     (SETQ LEXPOLY NIL)
A      (COND ((EQUAL (CAR LEXLINE) SLASH)(GO B)))
       (SETQ LEXPOLY (APPEND LEXPOLY (LIST (CAR LEXLINE))))
       (SETQ LEXLINE (CDR LEXLINE))(GO A)
B      (SETQ POLY (EXPRESSION LEXPOLY))
       (SETQ QMEANS (APPEND QMEANS (LIST POLY)))
       (COND ((NULL (CDR LEXLINE))(GO C)))
       (SETQ LEXLINE (CDR LEXLINE))
       (GO AA)
C      (SETQ MEANS (APPEND MEANS (LIST QMEANS)))
       (GO READLINE)
OUT    (CSETQ MLOP MEANS)
         (CSETQ VERTICES(REVERSE VERTS))
         (PRIN1 BLANK)(PRIN1 (QUOTE TEMPLATE*))
                 (PRINT VERTICES)
       (RETURN (REVERSE VERTS)) ) ))

↓ EDGREAD PROMPTS THE USER TO INPUT THE EDGES, WHICH IT ASSEMBLES
INTO THE GLOBAL VARIABLE *EDGES*.  ↓
(EDGREAD(LAMBDA()(PROG(EDGS LINE    LEXLINE NI NO LEXPOLY POLY)
       (TERPRI)
       (PRQUOTE (QUOTE (*INPUT EDGES: NODE/PROBABILITY/NODE;)))
       (SETQ EDGS NIL)
READLINE (SETQ LINE (READER))
       (SETQ LEXPOLY NIL)
       (COND((EQ (CAR LINE) RARROW)(GO OUT)))
       (SETQ LEXLINE (LEXICAL LINE))
       (SETQ NI (CAAR LEXLINE))
       (SETQ LEXLINE (CDDR LEXLINE))
SKIM   (COND((EQUAL(CAR LEXLINE) SLASH)(GO ASSMBL)))
       (SETQ LEXPOLY (APPEND LEXPOLY (LIST (CAR LEXLINE))))
       (SETQ LEXLINE (CDR LEXLINE))(GO SKIM)
ASSMBL (SETQ NO (CAADR LEXLINE))
       (SETQ POLY (EXPRESSION LEXPOLY))
       (SETQ EDGS (CONS(LIST NI POLY NO) EDGS))
       (GO READLINE)
OUT    (CSETQ EDGES EDGS)
       (RETURN NIL) ) ))
```

```
↓ INOD PROMPTS THE USER FOR THE ORIGIN LABEL, DESTINATION LABEL,
AND DEGREE OF MULTIPROGRAMMING; SETTING THE GLOBAL VARIABLES
↱ORIGIN↱, ↱DESTINATION↱, AND ↱DEGMULPROG↱.   ↓
(INOD(LAMBDA()(PROG(O D)
      (PRQUOTE (QUOTE (*INPUT ORIGIN LABEL)))
      (CSETQ ORIGIN (READATOM))
      (PRQUOTE (QUOTE (*INPUT DESTINATION LABEL)))
      (CSETQ DESTINATION (READATOM))
      (CHECK1 ORIGIN EDGES VERTICES)
      (CHECK2 DESTINATION EDGES VERTICES)
      (TOPSORT ORIGIN DESTINATION)
      (PRQUOTE (QUOTE(*INPUT TOTAL NUMBER JOBS IN NETWORK)))
      (CSETQ DEGMULPROG (READATOM))
     (RETURN NIL) ) ))

↓ FILENETIN READS A NETWORK DESCRIPTION FROM FILE ↱FNAME. ↓
(FILENETIN(LAMBDA(FNAME)(PROG()
      (CSETQ VLIST NIL)
      (REWIND FNAME)
      (RDS FNAME)
      (EDGREAD)(MEANREAD)(INOD)
      (RDS (QUOTE TTY))
      (COMMANDS)
      (RETURN NIL) ) ))

↓ NETIN PROMPTS A NETWORK DESCRIPTION FROM THE USER. ↓
(NETIN(LAMBDA()(PROG()
      (CSETQ VLIST NIL)
      (EDGREAD)(MEANREAD)
      (INOD)(TEMPLATE)(COMMANDS)
      (RETURN NIL) ) ))

↓ PRQUOTE PRINTS THE CHARACTERS OF LIST ↱L↱.  ↓
(PRQUOTE(LAMBDA(L)(PROG()
A     (COND((NULL L)(PROG2 (TERPRI) (RETURN NIL))))
      (PRIN1 BLANK)
      (PRIN1 (CAR L))
      (SETQ L (CDR L)) (GO A) ) ))


↓ TBLE DOES THE ↱TBLE↱ OPERATION UPON A POLYNOMIAL ↱P↱ ↓
(TBLE(LAMBDA(P)(PROG(V L VALS H S R M N M1 N1 NORM)
   (SETQ M1 -1.0E300)(SETQ N1 1.0E300)
   (SETQ M 0)(SETQ N 0)
   (PRIN1 ≡*)(PRIN1 ≡VAR)(PRIN1 BLANK)(PRIN1 ≡=)
   (SETQ V(READATOM))(CSETQ TBLVARS(LIST(CONS V(LIST V
    (CONS 1 1)))))
   (SETQ P(SUBS P))(NORMS DEGMULPROG (LENGTH VERTICES))
   (SETQ NORM RECNORM)
   (SETQ NORM (SUBS NORM))
   (PRQUOTE ≡(LOWLIM/HIGHLIM/STEP ≡))
   (SETQ L(READATOMS))(SETQ H(CADR L))(SETQ S(CADDR L))
    (SETQ L(CAR L))
    (TERPRI)
A   (COND((LESSP H L)(GO B)))(SETQ R(SUB(CDR P)L))(PRIN1 L)
   (OSPACE (DIFFERENCE 30 (OSPACE 1)))
   (SETQ R (QUOTIENT R (SUB(CDR NORM)L)))
   (PRIN1 R)(TERPRI)
   (COND((GREATERP R M1)(PROG2(SETQ M L)(SETQ M1 R))))
        (COND((LESSP R N1)(PROG2(SETQ N L)(SETQ N1 R))) )
    (SETQ L(PLUS L S))(GO A)
B   (PRIN1 ≡*)(PRIN1 ≡MAX)(PRIN1 BLANK)(PRIN1 M)
   (OSPACE (DIFFERENCE 30 (OSPACE 1)))
   (PRIN1 M1)(TERPRI)(PRIN1 ≡*)(PRIN1 ≡MIN)(PRIN1 BLANK)
   (PRIN1 N)(OSPACE (DIFFERENCE 30(OSPACE 1)))
   (PRIN1 N1)(TERPRI)(RETURN NIL) ) ))
(SUB(LAMBDA(P V)(PROG(S T)(SETQ S 0)
A  (COND((NULL P)(RETURN S)))
   (SETQ S (PLUS (TIMES(TOTHEN V (CDAR P))(CAAR P))S))
   (SETQ P(CDR P))(GO A) ) ))
(SUBS(LAMBDA(P)(PROG(V VL PRS S)
   (SETQ S NIL)
   (COND((NUMBERP P)(RETURN P)))
   (SETQ V(CAR P))(SETQ PRS(CDR P))
   (SETQ VL (GV V))
A  (COND((NULL PRS)(RETURN S)))
   (SETQ P (RATPD(TOTHEN VL(CDAR PRS))(SUBS(CAAR PRS))))
   (SETQ S (RATAD P S))
   (SETQ PRS(CDR PRS))(GO A) ) ))
(GVAL(LAMBDA(V)(PROG(A)
   (PRIN1 ≡*)(PRIN1 BLANK)(PRIN1 V)(PRIN1 BLANK)(PRIN1 ≡=)
   (SETQ A(READATOM))(RETURN(LIST(CONS V A))) ) ))
(GV(LAMBDA(V)(PROG(Z)(SETQ Z TBLVARS)
A   (COND((NULL Z)(GO B))
  ((EQ(CAAR Z)V)(RETURN(CDAR Z))))(SETQ Z(CDR Z))(GO A)
B  (SETQ Z(GVAL V))
    (CSETQ TBLVARS(APPEND Z TBLVARS))
    (RETURN (CDAR Z)) ) ))
↓ TBLETPUT DOES THE ↗TABLE↗ OPERATION UPON THROUGHPUT -
WHICH IS THE SUM OF RATIOS OF POLYNOMIALS. ↓
(TBLETPUT(LAMBDA()(PROG(V L VALS H S R P M N M1 NI NORM)
   (SETQ M1 -1.0E300)(SETQ N1 1.0E300)
   (SETQ M 0)(SETQ N 0)
   (PRIN1 ≡*)(PRIN1 ≡VAR)(PRIN1 BLANK)(PRIN1 ≡=)
   (SETQ V(READATOM))(CSETQ TBLVARS(LIST(CONS V(LIST V
      (CONS 1 1))))))
   (SETQ P(TPUT))(NORMS DEGMULPROG (LENGTH VERTICES))
   (SETQ P(BREAKOUT P))(SETQ P(TPUTSUBS P))
   (SETQ NORM (SUBS RECNORM))
   (PRQUOTE ≡(LOWLIM/HIGHLIM/STEP ≡))
   (SETQ L(READATOMS))(SETQ H(CADR L))(SETQ S(CADDR L))
    (SETQ L(CAR L)) (TERPRI)
A   (COND((LESSP H L)(GO B)))(SETQ R(SUBTPUT P L))(PRIN1 L)
   (OSPACE (DIFFERENCE 30 (OSPACE 1)))
   (SETQ R (QUOTIENT R (SUB (CDR NORM)L)))
   (PRIN1 R)(TERPRI)
   (COND((GREATERP R M1)(PROG2(SETQ M L)(SETQ M1 R))))
        (COND((LESSP R N1)(PROG2(SETQ N L)(SETQ N1 R))) )
     (SETQ L(PLUS L S))(GO A)
B   (PRIN1 ■*)(PRIN1 ■MAX)(PRIN1 BLANK)(PRIN1 M)
   (OSPACE (DIFFERENCE 30 (OSPACE 1)))
   (PRIN1 M1)(TERPRI)(PRIN1 ■*)(PRIN1 ■MIN)(PRIN1 BLANK)
   (PRIN1 N)(OSPACE (DIFFERENCE 30(OSPACE 1)))
   (PRIN1 N1)(TERPRI)(RETURN NIL) ) ))
(SUBTPUT(LAMBDA(X L)(PROG(S N D)
   (SETQ S 0)
A  (COND((NULL X)(RETURN S)))
   (SETQ N(CAAR X))(SETQ D(CADAR X))(SETQ X(CDR X))
   (COND((NOT(NUMBERP N))(SETQ N(SUB(CDR N)L))))
   (COND((NOT(NUMBERP D))(SETQ D(SUB(CDR D)L))))
   (SETQ S(PLUS S (QUOTIENT N D)))
   (GO A) ) ))
(TPUTSUBS(LAMBDA(P)(PROG(S C D)
A  (COND((NULL P)(RETURN S)))
   (SETQ C(CAAR P))(SETQ D(CADAR P))(SETQ P(CDR P))
   (SETQ S(CONS(LIST(SUBS C)(SUBS D))S))
   (GO A) )))
(BREAKOUT(LAMBDA(P)(PROG(S C D)
A  (COND((NULL P)(RETURN S)))
   (SETQ C(CAR P))(SETQ P(CDR P))
B  (COND((NULL C)(GO A)))
   (SETQ D(CAR C))(SETQ C(CDR C))
   (SETQ S(CONS D S))(GO B) )))


↓ EXPRESSION ACCEPTS AS AN ARGUMENT AN EXTERNAL POLYNOMIAL
REPRESENTATION, WHERE EACH SYNTACTIC TYPE OF THE REPRESENTATION
IS SURROUNDED BY PARENTHESIS.  IT RETURNS THE INTERNAL POLY-
NOMIAL REPRESENTATION.  ↓
 (EXPRESSION(LAMBDA(E)(PROG(EXP X Y)
      (COND((NULL E)(RETURN()))
         ((EQ(CAR E)PLUSS)(COND((NOT(SETQ X(TERM(CDR E))))
          (RETURN())) (T X)))
         ((EQ(CAR E)DASH)(GO M))
         ((NULL(SETQ X(TERM E)))(RETURN ())))
      (SETQ EXP(CAR X))
    E (COND((NULL(CDR X))(RETURN(COND((NULL EXP)0)(T EXP))))
         ((MEMBER(CADR X)(QUOTE(+ -)))T)
         (T(RETURN(CONS EXP(CDR X)))))
      (COND((NULL(SETQ Y(TERM(CDDR X))))(RETURN())))
      (SETQ EXP(RATAD(COND((EQ(CADR X)DASH)(NEG(CAR Y)))
          (T(CAR Y)))EXP))  (SETQ X Y)(GO E)
    M (COND((NOT(SETQ X(TERM(CDR E))))(RETURN())))
      (SETQ EXP(NEG(CAR X)))(GO E) ) ))
 (TERM(LAMBDA(E)(PROG(X Y Z)
      (SETQ X(SECONDARY E))
    A (COND((OR(NULL X)(NULL(CDR X)))(RETURN X)))
      (SETQ Z(CDDR X))
      (COND((EQ(CADR X)STAR)())(T(RETURN X)))
```

```
       (COND((SETQ Z(SECONDARY Z))
               (SETQ X(CONS(RATPD(CAR X)(CAR Z))(CDR Z))))
          (T(RETURN X)))
       (GO A) ) ))
  (SECONDARY(LAMBDA(E)(PROG(X Y)
       (COND((NULL(SETQ X(PRIMARY E)))(RETURN()))
          ((NULL(CDR X))(RETURN X))
          ((NOT(EQ(CADR X)UPARROW))(RETURN X))
          ((OR(NULL(CDDR X))(NOT(NUMBERP(CADDR X))))(RETURN()))
          (T(RETURN(CONS(LIST(CAAR X)(CONS 1.0(CADDR X)))
        (CDDDR X))))) )))
  (PRIMARY(LAMBDA(E)(PROG(X)
       (COND((SETQ X(VARIABLE E))(RETURN X))
          ((SETQ X(CONSTANT E))(RETURN X))
          ((NOT(EQ(CAR E)LPAR))(RETURN()))
          ((NOT(SETQ X(EXPRESSION(CDR E))))(RETURN()))
          ((NULL(CDR X))(RETURN()))
          ((EQ(CADR X)RPAR)(RETURN(CONS(CAR X)(CDDR X))))
          (T())) ) ))
 (VARIABLE(LAMBDA(E)(PROG(A)
      (COND((OR(NUMBERP(CAR E))(ATOM(CAR E)))(RETURN())))
       (SETQ A (CAAR E))
       (COND((MEMBER A VLIST)())
             (T(CSETQ VLIST(CONC VLIST(LIST A)))))
       (RETURN(CONS(LIST A(CONS 1.0 1))(CDR E))) ) ))


↓ RATAD: R1,R2 INTERNAL POLYNOMIALS
            RETURNS (R1 + R2)               ↓
 (RATAD(LAMBDA(R1 R2)(COND((NULL R1)R2)((NULL R2)R1)
       ((AND(NUMBERP R1)(NUMBERP R2))(COND((EQN R1(MINUS R2))())
                                (T(PLUS R1 R2))))
       ((NUMBERP R1)(POLAD1 R2 R1))
       ((NUMBERP R2)(POLAD1 R1 R2))
   ((EQ (CAR R1)(CAR R2))(CONS(CAR R1)(POLAD2(CDR R1)(CDR R2))))
       ((LESSP(POSIT(CAR R1))(POSIT(CAR R2)))(POLAD1 R1 R2))
       (T(POLAD1 R2 R1)) ) ))
(POLAD1(LAMBDA(R1 R2)(PROG(X Y)(SETQ X(CDR R1))
 (SETQ Y(LIST(CAR R1)))
    A (COND((NULL X )(RETURN(CONC Y(LIST(CONS R2 0)))))
 ((ZEROP(CDAR X))(RETURN(CONC Y(LIST(CONS(RATAD(CAAR X)R2)0))))))
 (SETQ Y(CONC Y(LIST(CAR X ))))
 (SETQ X(CDR X))(GO A) ) ))
(POLAD2(LAMBDA(R1 R2)(COND((NULL R1)R2)
 ((NULL R2)R1)
 ((LESSP(CDAR R1)(CDAR R2))(CONS(CAR R2)(POLAD2 R1(CDR R2))))
 ((LESSP(CDAR R2)(CDAR R1))(CONS(CAR R1)(POLAD2 R2(CDR R1))))
 ((NULL(CSETQ XX(RATAD(CAAR R1)(CAAR R2))))(POLAD2(CDR R1)(CDR R2)
 ))
 (T(CONS(CONS XX(CDAR R1))(POLAD2(CDR R1)(CDR R2)))) ) ))
 (RATPD(LAMBDA(R1 R2)(PROG(I J K A B TM X M N)
      (COND((OR(NULL R1)(NULL R2))(RETURN())))
      (COND((AND(NUMBERP R1)(NUMBERP R2))(RETURN(TIMES R1 R2)))
          ((AND(NUMBERP R1)(ONEP R1))(RETURN R2))
          ((NUMBERP R1)(RETURN(NUMPD R1 R2)))
          (T(SETQ I(POSIT(CAR R1)))))
       (COND((AND(NUMBERP R2)(ONEP R2))(RETURN R1))
          ((NUMBERP R2)(RETURN(NUMPD R2 R1)))
          (T(SETQ J(POSIT(CAR R2)))))
       (COND((LESSP I J)(GO A)))
       (SETQ N I)(SETQ I J)(SETQ J N)(SETQ A(CDR R2))(SETQ B R1)
       (SETQ K R1)(SETQ M(CAR R2))(GO B)
     A (SETQ A(CDR R1))(SETQ B R2)(SETQ K R2)(SETQ M(CAR R1))
     B (SETQ B(COND((NUMBERP B)B)(T(CDR B))))(SETQ N B)
     F (COND((NULL A)(RETURN(CONS M X))))
       (SETQ TM(CAR A))
       (COND((LESSP I J)(GO E)))
     C(COND((NULL B)(GO D)))
       (SETQ X(POLAD2 X(LIST (CONS(RATPD(CAR TM)(CAAR B))
          (PLUS(CDR TM) (CDAR B))))))
       (SETQ B(CDR B))(GO C)
     E (SETQ X(POLAD2 X(LIST (CONS(RATPD(CAR TM)K)(CDR TM)))))
     D (SETQ A(CDR A))(SETQ B N)(GO F)
       ) ))
  (NUMPD(LAMBDA(N R)(PROG(X R1 TM)(SETQ R1(CDR R))
     A (COND((NULL R1)(RETURN(CONS(CAR R)X))))
       (SETQ TM(CAR R1))
       (SETQ X(CONC X(COND((NUMBERP(LIST(CONS(TIMES N
           (CAR TM))(CDR TM))))
          (T(LIST(CONS(NUMPD N(CAR TM))(CDR TM)))))))
       (SETQ R1(CDR R1))(GO A) ) ))
  (NEG(LAMBDA(P)(COND((NULL P)())
       ((NUMBERP P)(MINUS P))
       ((ATOM(CAR P))(CONS(CAR P)(NEG(CDR P))))
       (T(CONS(CONS(NEG(CAAR P))(CDAR P))(NEG(CDR P)))) ) ))
  (POSIT(LAMBDA(A)(PROG(CTR VL)
       (SETQ CTR 1)(SETQ VL VLIST)
     A (COND((EQ A(CAR VL))(RETURN CTR)))
       (SETQ CTR(ADD1 CTR))(SETQ VL (CDR VL))(GO A) ) ))


↓ READATOM(S) READS ONE LINE.  THE LINE IS ASSUMED
TO BE AN ATOM(S) FOLLOWED BY A SEMICOLON.  IT RETURNS
THE (LIST OF THE) NUMBER(S) OR ATOM(S) FOUND BEFORE
THE SEMICOLON.   ↓
(READATOM(LAMBDA()(PROG(X Y)
       (SETQ Y NIL)
A      (SETQ X (ADVANCE))
       (COND((EQ X BLANK)(GO A))
       ((OR(EQ X EOR)(EQ X SEMICOLON))(GO ON)))
       (SETQ Y (CONS X Y))(GO A)
ON     (SETQ Y (REVERSE Y))
       (ENDREAD)
       (COND((LITER (CAR Y))(SETQ Y (CAAR (LEXICAL Y))))
            (T (SETQ Y (CAR (LEXICAL Y)))) )
       (RETURN Y) ) ))
```

```
(READATOMS(LAMBDA()(PROG(X Y F Z)
      (SETQ F Ø)
A   (SETQ X(ADVANCE))
    (COND((EQ X BLANK)(GO A))((EQ X SEMICOLON)(PROG2(SETQ F 1)
               (GO I)))
     ((EQ X SLASH)(GO I)))(SETQ Y(CONS X Y))(GO A)
    I (SETQ Y(REVERSE Y))(COND((LITER(CAR Y))(SETQ Y
         (CAAR(LEXICAL Y))))
     (T(SETQ Y(CAR(LEXICAL Y)))))
    (SETQ Z(CONS Y Z))(SETQ Y NIL)
O   (COND((ONEP F)(RETURN (REVERSE Z))))(GO A))))


↓ LEXICAL ASSEMBLES A STRING OF SINGLE CHARACTERS
INTO A LIST OF ELEMENTS, EACH ELEMENT A SINGLETON LIST
OF EITHER AN ATOM OR NUMBER, OR LIST OF SAME.  ↓
 (LEXICAL(LAMBDA(S)(COND((NULL S)())
      (T(LEX*(CAR S)(NCONC(CDR S)(QUOTE(EOR)))(CLEARBUFF))) ) ))
 (LEX*(LAMBDA(X Y Z)(COND((NULL Y)())
     ((EQ X COMMA)(LEX*(CAR Y)(CDR Y)()))
      ((LITER X)(LEXV(CAR Y)(CDR Y)(PACK X)))
      ((DIGIT X)(LEXC(CAR Y)(CDR Y)(PACK X)))
      (T(CONS X(LEX* (CAR Y)(CDR Y)()))) ) ))
(LEXR(LAMBDA(X Y Z)(PROG(COUNT NOB NORM)
         (SETQ COUNT Ø)  (SETQ NOB (NUMOB)) (CLEARBUFF)
       A  (SETQ X (CAR Y))
          (SETQ Y (CDR Y))
      (COND((NOT(DIGIT X))(GO B)))
      (PACK X)
          (SETQ COUNT (ADD1 COUNT))(GO A)
       B  (SETQ NORM (EX1Ø COUNT))
      (SETQ NORM (QUOTIENT (NUMOB) NORM))
      (SETQ NOB (PLUS NOB NORM))
      (RETURN (CONS NOB (LEX* X Y (CLEARBUFF)))) ) ))
(EX1Ø(LAMBDA(N)(PROG(SUM C)
      (SETQ SUM 1)(SETQ C Ø)
 A    (COND((EQN C N)(RETURN SUM)))
      (SETQ SUM (TIMES SUM 1Ø.Ø))
      (SETQ C (ADD1 C)) (GO A) ) ))
 (LEXV(LAMBDA(X Y Z)(COND((NULL Y)(LIST(LIST(INTERN(MKNAM)))))
      ((OR(LITER X)(DIGIT X))(LEXV(CAR Y)(CDR Y)(PACK X)))
      (T(CONS(LIST(INTERN(MKNAM)))(LEX* X Y(CLEARBUFF)))) ) ))
 (LEXC(LAMBDA(X Y Z)(COND((NULL Y)(LIST(NUMOB)))
      ((DIGIT X)(LEXC(CAR Y)(CDR Y)(PACK X)))
      ((EQ X PERIOD)(LEXR () Y () ))
      (T(CONS(NUMOB)(LEX* X Y(CLEARBUFF)))) ) ))
 (READER(LAMBDA()(PROG(X Y)
    A (SETQ X(ADVANCE))
      (COND((EQ X BLANK)(GO A))
         ((EQ X EOR)(GO A))
         ((EQ X SEMICOLON)(PROG2(ENDREAD)(RETURN(REVERSE Y)))))
      (SETQ Y(CONS X Y))(GO A) ) ))
```

```
↓ SETPR PRINTS THE CHARACTER STRING OF THE EXTERNAL
REPRESENTATION OF THE INTERNALLY REPRESENTED POLYNOMIAL
↱P↱.      ↓
(SETPR(LAMBDA(P)(PROG()(PRIN1 BLANK)
      (COND((NULL P)(PRIN1 Ø))(T(SPRI1 P()()T))) ) ))
 (SPRI1(LAMBDA(P N M L)(PROG(X Y A)(COND((NUMBERP P)(GO A)))
      (SETQ A(CAR P))(SETQ P(CDR P))
      (COND((SETQ X(AND N(NOT(NULL(CDR P)))))(PRIN1 LPAR)))
    B (SETQ Y(ZEROP(CDAR P)))
      (SPRI1(CAAR P)T(COND(Y M)(T T))(COND((AND Y X)())(T T)))
      (COND(Y(GO G)))(PRIN1 A)
      (COND((ONEP(CDAR P))(GO F)))(PRIN1 UPARROW)(PRIN1(CDAR P))
    F (COND((AND(NOT X)M L)(PRIN1 STAR)))
    G (COND((NULL(CDR P))(GO H)))
      (PRIN1 BLANK)(PRIN1 PLUSS)(PRIN1 BLANK)(SETQ P(CDR P))
      (SETQ M())(GO B)
    H (COND(X(PRIN1 RPAR)))(RETURN())
   A (COND((NULL M)(COND((ZEROP(CADR(DIVIDE P 1)))
      (PRIN1(FIX P)))(T(PRIN1 P))))((ONEP P)())
        ((ONEP(MINUS P))(PRIN1 DASH))(T(PROG2
   (COND((ZEROP(CADR(DIVIDE P 1)))(PRIN1(FIX P)))(T(PRIN1 P)))
    (PRIN1 STAR))))
      ) ))


↓ TOPSORT PERFORMS A TOPOLOGICAL SORT UPON THE VERTICES
OF THE NETWORK, CREATES BTERMS BY SETTING GLOBAL VARIABLE
↱BTLOP↱.      ↓
(TOPSORT(LAMBDA(GN TN)
      (PROG(SORTS   SIQS SOQS REM BTS  )
      (SETQ SORTS NIL)(SETQ BTS NIL)
      (SETQ REM NIL)
TEST  (COND ((NOTIN (SETQ SIQS (SIQ GN)) SORTS  )(GO BACK))
      ((MEMBER GN SORTS)(GO BACK)))
      (SETQ SORTS (APPEND SORTS (LIST GN)))
      (SETQ BTS (APPEND BTS (LIST(CRBTM GN SIQS EDGES BTS))))
      (COND((EQ GN TN)(GO OUT)))
      (SETQ GN (CAR (SETQ SOQS (SOQ GN))))
      (SETQ REM (APPEND REM (CDR SOQS)))
      (GO TEST)
BACK  (SETQ GN (CAR REM))
      (SETQ REM (CDR REM))
      (GO TEST)
OUT   (CSETQ BTLOP BTS) (CSETQ SORTLAT SORTS)
      (RETURN NIL) ) ))

↓ SOQ: SET OF OUTGOING QUEUES
    SIQ: SET OF INCOMING QUEUES
    NOTIN: SET MEMBERSHIP TEST
    CHECK1,CHECK2: CONTINUITY TESTS OF NETWORK
     GRAPH FROM ORIGIN AND DESTINATION
    UNION: SET UNION   ↓
```

```
(SOQ(LAMBDA(VTEMP)
      (SOQ* VTEMP EDGES)     ))
(SIQ(LAMBDA(VTEMP)
      (SIQ* VTEMP EDGES) ))
(SOQ*(LAMBDA(VTEMP LEDGES)
      (COND
      ((NULL LEDGES) NIL)
      ((EQ (CAAR LEDGES) VTEMP)
       (APPEND (CDDAR LEDGES) (SOQ* VTEMP (CDR LEDGES))))
      (T (SOQ* VTEMP (CDR LEDGES))) ) ))
(SIQ*(LAMBDA(VTEMP LEDGES)
      (COND
      ((NULL LEDGES) NIL)
      ((EQ (CADDAR LEDGES) VTEMP) (APPEND (LIST(CAAR LEDGES))
                                  (SIQ* VTEMP (CDR LEDGES))))
      (T (SIQ* VTEMP (CDR LEDGES))) ) ))
(NOTIN(LAMBDA(X Y)
      (COND
      ((NULL X) NIL)
      ((MEMBER (CAR X) Y)
       (NOTIN (CDR X) Y))
      (T  (CONS (CAR X) (NOTIN (CDR X) Y))) ) ))
(ONEP(LAMBDA(N)(COND((ZEROP(SUB1 N))T)(T NIL)) ))

(CHECK1(LAMBDA(SOURCE EDGS VERTS)
      (PROG (STACK VL VTEMP SUCCESSORS)
            (SETQ VL (LIST SOURCE))
            (SETQ STACK VL)
LOC1  (COND ((NULL STACK)(GO L1)))
      (SETQ VTEMP (CAR STACK))
      (SETQ SUCCESSORS (SOQ* VTEMP EDGS))
      (SETQ STACK (APPEND (NOTIN SUCCESSORS VL) (CDR STACK)))
      (SETQ VL (UNION SUCCESSORS VL))
      (GO LOC1)
L1    (SETQ STACK (NOTIN VERTS VL))
      (COND((NULL STACK)(GO A)))
      (PRQUOTE(APPEND(QUOTE(* NODES NOT CONNECTED FROM
      ORIGIN -))(LIST STACK)))
A     (RETURN NIL) )))
(CHECK2(LAMBDA(SINK EDGS VERTS)
      (PROG(STACK VL VTEMP PREDECESSORS)
        (SETQ VL (LIST SINK))
        (SETQ STACK VL)
LOC1  (COND((NULL STACK)(GO L1)))
      (SETQ VTEMP (CAR STACK))
      (SETQ PREDECESSORS (SIQ* VTEMP EDGS))
      (SETQ STACK (APPEND (NOTIN PREDECESSORS VL)(CDR STACK)))
      (SETQ VL (UNION PREDECESSORS VL))
      (GO LOC1)
L1    (SETQ STACK (NOTIN VERTS VL))
      (COND((NULL STACK)(GO A)))
      (PRQUOTE(APPEND(QUOTE(* NODES NOT CONNECTED TO
      DESTINATION -))(LIST STACK)))
```

```
A      (RETURN NIL) ) ))
(UNION(LAMBDA(LAT1 LAT2)
       (COND
       ((NULL LAT1) LAT2)
       ((MEMBER (CAR LAT1) LAT2) (UNION (CDR LAT1) LAT2))
       (T  (CONS (CAR LAT1) (UNION (CDR LAT1) LAT2))) ) ))


↓ SSPROB RETURNS UNNORMALIZED STEADY-STATE
PROBABILITY OF STATE ↱VECTOR↱,  ↓
(SSPROB(LAMBDA(VECTOR )(PROG(S PDT NI V L)
       (SETQ S VERTICES)(SETQ PDT 1)(SETQ L (LENGTH VECTOR))
A      (COND((NOT S)(RETURN PDT)))
       (SETQ V (CAR S))(SETQ S (CDR S))
       (SETQ NI (CAR VECTOR))(SETQ VECTOR (CDR VECTOR))
       (SETQ PDT (RATPD PDT (TRM V NI BTLOP MLOP)))
       (GO A) ) ))

↓ TRM(I)=(BTERM(I)*PTERM(I))↑N(I)  ↓
(TRM(LAMBDA(I NI BLOP MLOP)(PROG(BTERM PTM)
       (SETQ PTM (PTERM I NI MLOP))
       (SETQ BTERM (GSLOP I BLOP))
       (SETQ BTERM (TOTHEN BTERM NI))
       (RETURN (RATPD BTERM PTM)) ) ))

↓ PTERM(I)=PTERM OF NODE IDENTIFIED BY INTEGER I  ↓
(PTERM(LAMBDA(I NI LOP)(PROG(M S C)
      (SETQ S 1)
A      (COND((ZEROP NI)(RETURN S)))
       (SETQ S(RATPD S (GETMEAN I NI LOP)))(SETQ NI(SUB1 NI))
       (GO A) ) ))

↓ TOTHEN(TERM N)= (TERM)↑N  ↓
(TOTHEN(LAMBDA(TERM N)(PROG(SUM PDT)
       (SETQ SUM 0)
       (SETQ PDT 1)
LOOP   (COND((EQN SUM N)(GO OUT)) )
       (SETQ PDT (RATPD PDT TERM))
       (SETQ SUM (ADD1 SUM))
       (GO LOOP)
OUT    (RETURN PDT) ) ))

↓ CRBTM RETURNS THE CALCULATED BTERM OF NEW NODE
↱NODE↱, GIVEN NODE PLACEMENT IN NETWORK  ↓
(CRBTM(LAMBDA(NODE SIQS EDGS BTMS)
       (PROG(SIQ        SUM PDT BTM ONE)
       (SETQ ONE 1)
       (SETQ SUM ())
       (COND((NULL BTMS)(GO C)))
A      (COND((NULL SIQS)(GO B)))
       (SETQ SIQ (CAR SIQS))
       (SETQ SIQS (CDR SIQS))
       (SETQ PDT (RATPD(GTRPROB SIQ NODE EDGS)(GSLOP SIQ BTMS)))
       (SETQ SUM (RATAD SUM PDT))
       (GO A)
B      (RETURN (LIST NODE SUM))
C      (RETURN(LIST NODE ONE)) ) ))


↓ GTRPROB RETURNS THE BRANCHING PROBABILITY GIVEN TWO
NODES.   ↓
(GTRPROB(LAMBDA(NODE1 NODE2 EDGS)
       (PROG(EDG)
A      (COND((NULL EDGS)(GO C)))
       (SETQ EDG (CAR EDGS))
       (SETQ EDGS (CDR EDGS))
       (COND((AND(EQ (CAR EDG) NODE1)(EQ (CADDR EDG) NODE2))
         (GO B))) (GO A)
B      (RETURN (CADR EDG))
C      (PRINT (QUOTE EDGESEXHAUSTED))
       (RETURN NIL)) ))

↓ GSLOP GETS THE SECOND ELEMENT FROM A LIST OF PAIRS GIVEN
THE FIRST ELEMENT OF A PAIR.   ↓
(GSLOP(LAMBDA(MATCH LIST)(PROG()
A      (COND((NULL LIST)(GO C))
            ((EQUAL MATCH (CAAR LIST))(RETURN (CADAR LIST))))
       (SETQ LIST (CDR LIST))
       (GO A)
C      (PRINT(QUOTE GSLOP*FINDS*NULL*LIST))
       (RETURN NIL)) ))

↓ GETMEAN SEARCHES *MLOP* FOR SERVICE MEAN GIVEN VERTEX
IDENTIFYING LABEL *Q* AND THE NUMBER OF JOBS *NBR* IN QUEUE. ↓
(GETMEAN(LAMBDA(Q NBR MEANS)(PROG(CTR MEAN)
         (SETQ CTR 0)
A        (COND((NULL MEANS)(GO E))
              ((EQ Q (CAAR MEANS))(GO B)))
         (SETQ MEANS (CDR MEANS))(GO A)
B        (SETQ MEANS (CDAR MEANS))
C       (COND((NOT MEANS)(GO D)))
        (SETQ MEAN (CAR MEANS))
        (SETQ MEANS (CDR MEANS))
        (SETQ CTR (ADD1 CTR))
        (COND((EQN CTR NBR)(RETURN MEAN)))
        (GO C)
D       (RETURN MEAN)
E       (PRINT(QUOTE ERROR*GETMEAN))(RETURN NIL) ) ))


↓ NORMS GENERATES AND RETURNS THE SUM OF ALL THE UNNORMALIZED
STEADY-STATE PROBABILITIES.   ↓
(NORMS(LAMBDA(N L)(PROG(VECTOR P C K VECTLIST CV CVECT T1 T2
          SPROB)
       (SETQ K 0)(SETQ VECTOR (LIST N))(SETQ P L)
       (SETQ CVECT NIL)(SETQ T1 ())(SETQ T2 ())
```

```
SK      (SETQ K (ADD1 K))
        (COND((EQN K L)(GO C1)))
        (SETQ VECTOR (CONS 0 VECTOR))
        (GO SK)
C1      (SETQ VECTLIST (LIST VECTOR))
        (SETQ T1 (RATAD T1 (SSPROB VECTOR)))
CO      (SETQ C 0)
        (COND((EQN N (CAR VECTOR))(GO OUT)))
        (SETQ VECTOR (PAE VECTOR P (SUB1 (GAE VECTOR P))))
        (SETQ C (ADD1 C))
        (COND((ONEP P)(GO S)))
        (SETQ P (SUB1 P))
        (SETQ VECTOR (PAE VECTOR P (PLUS (GAE VECTOR P) C)))
        (SETQ VECTLIST (CONS VECTOR VECTLIST))
        (SETQ SPROB (SSPROB VECTOR))
        (SETQ T1 (RATAD T1 SPROB))
        (COND((ZEROP(CAR VECTOR))(GO CO)))
        (SETQ T2 (RATAD T2 SPROB))(GO CO)
S       (SETQ C (ADD1 (CAR VECTOR)))
        (SETQ VECTOR (PAE VECTOR 1 0))
        (SETQ CVECT VECTOR)
        (SETQ P 0)
P1      (SETQ P (ADD1 P))
        (SETQ CV (CAR CVECT))
        (SETQ CVECT (CDR CVECT))
        (COND((EQN CV 0)(GO P1)))
VP      (SETQ VECTOR (PAE VECTOR P (SUB1 CV)))
        (SETQ P (SUB1 P))
        (SETQ VECTOR (PAE VECTOR P (ADD1 C)))
        (SETQ SPROB (SSPROB VECTOR))
        (SETQ T1 (RATAD T1 SPROB))
        (COND((NOT(ZEROP(CAR VECTOR)))(SETQ T2 (RATAD T2 SPROB))))
        (SETQ VECTLIST (CONS VECTOR VECTLIST))
        (GO CO)
OUT     (CSETQ RECNORM T1)(CSETQ UNORMA+1 T2)(RETURN VECTLIST) ) ))

↓ NORMSPR GENERATES AND PRINTS ALL THE UNNORMALIZED
STEADY-STATE PROBABILITIES AND RETURNS THEIR SUM. ↓
(NORMSPR(LAMBDA(N L)(PROG(VECTOR P C K   CV CVECT T1 T2 SPROB)
        (TERPRI)
        (SETQ K 0)(SETQ VECTOR (LIST N))(SETQ P L)
        (SETQ CVECT NIL)(SETQ T1 ())(SETQ T2 ())
SK      (SETQ K (ADD1 K))
        (COND((EQN K L)(GO C1)))
        (SETQ VECTOR (CONS 0 VECTOR))
        (GO SK)
C1      (SETQ SPROB (SSPROB VECTOR))
        (SETQ T1 (RATAD T1 SPROB))
        (PRINT BLANK)(PRIN1 BLANK)(PRINT VECTOR)(SETPR SPROB)
        (TERPRI)
CO      (SETQ C 0)
        (COND((EQN N (CAR VECTOR))(GO OUT)))
        (SETQ VECTOR (PAE VECTOR P (SUB1 (GAE VECTOR P))))
        (SETQ C (ADD1 C))
        (COND((ONEP P)(GO S)))
        (SETQ P (SUB1 P))
        (SETQ VECTOR (PAE VECTOR P (PLUS (GAE VECTOR P) C)))
        (SETQ SPROB (SSPROB VECTOR))
        (SETQ T1 (RATAD T1 SPROB))
        (PRINT BLANK)(PRIN1 BLANK)(PRINT VECTOR)(SETPR SPROB)
        (TERPRI)
        (COND((ZEROP(CAR VECTOR))(GO CO)))
        (SETQ T2 (RATAD T2 SPROB))(GO CO)
S       (SETQ C (ADD1 (CAR VECTOR)))
        (SETQ VECTOR (PAE VECTOR 1 Ø))
        (SETQ CVECT VECTOR)
        (SETQ P Ø)
P1      (SETQ P (ADD1 P))
        (SETQ CV (CAR CVECT))
        (SETQ CVECT (CDR CVECT))
        (COND((EQN CV Ø)(GO P1)))
VP      (SETQ VECTOR (PAE VECTOR P (SUB1 CV)))
        (SETQ P (SUB1 P))
        (SETQ VECTOR (PAE VECTOR P (ADD1 C)))
        (SETQ SPROB (SSPROB VECTOR))
        (SETQ T1 (RATAD T1 SPROB))
        (PRINT BLANK)(PRIN1 BLANK)(PRINT VECTOR)(SETPR SPROB)
        (TERPRI)
        (COND((NOT(ZEROP(CAR VECTOR)))(SETQ T2 (RATAD T2 SPROB))))
        (GO CO)
OUT     (CSETQ RECNORM T1)(CSETQ UNORMA+1 T2)(RETURN NIL) ) ))

↓ IDLE COMPUTES THE SUM OF ALL UNNORMALIZED
STEADY-STATE PROBABILITES IN WHICH NO JOBS EXIST
IN NODE ↱NODE↱. L:NUMBER OF DEVICES N:TOTAL NUMBER
OF JOBS          ↓
(IDLE(LAMBDA(NODE N L)(PROG(A)(COND
    ((NULL(SETQ A(PLACE VERTICES NODE)))(GO B)))
    (RETURN(GEN A N (SUB1 L) Ø))
B   (PRQUOTE *(NODE NOT IN NETWORK))(RETURN NIL) )))

↓ GEN CALCULATES ALL UNNOMALIZED STEADY-STATE
PROBABILITIES AND THEIR SUM FOR WHICH THERE ARE NO
JOBS IN NODE SPECIFIED BY INTEGER ↱PL↱.  ↓
(GEN(LAMBDA(PL N L O)(PROG( VECTOR P C K VECTLIST CV CVECT T1
    SPROB)
        (SETQ K Ø)(SETQ VECTOR (LIST N))(SETQ P L)
        (SETQ CVECT NIL)(SETQ T1 ())
SK      (SETQ K (ADD1 K))
        (COND((EQN K L)(GO C1)))
        (SETQ VECTOR (CONS Ø VECTOR))
        (GO SK)
C1      (SETQ T1 (RATAD T1 (SSPROBK VECTOR PL O)))
CO      (SETQ C Ø)
        (COND((EQN N(CAR VECTOR))(RETURN T1)))
        (SETQ VECTOR (PAE VECTOR P (SUB1 (GAE VECTOR P))))
        (SETQ C (ADD1 C))
        (COND((ONEP P)(GO S)))
        (SETQ P (SUB1 P))
        (SETQ VECTOR (PAE VECTOR P (PLUS (GAE VECTOR P) C)))
        (SETQ SPROB (SSPROBK VECTOR PL 0))
        (SETQ T1 (RATAD T1 SPROB))
        (GO CO)
S       (SETQ C (ADD1 (CAR VECTOR)))
        (SETQ VECTOR (PAE VECTOR 1 Ø))
        (SETQ CVECT VECTOR)
        (SETQ P Ø)
P1      (SETQ P (ADD1 P))
        (SETQ CV (CAR CVECT))
        (SETQ CVECT (CDR CVECT))
        (COND((EQN CV Ø)(GO P1)))
VP      (SETQ VECTOR (PAE VECTOR P (SUB1 CV)))
        (SETQ P (SUB1 P))
        (SETQ VECTOR (PAE VECTOR P (ADD1 C)))
        (SETQ SPROB (SSPROBK VECTOR PL 0))
        (SETQ T1 (RATAD T1 SPROB))
        (GO CØ) ) ))
(SSPROBK(LAMBDA(VECTOR PL N)(SSPROB(PUTIN VECTOR PL N )) ))
(PLACE(LAMBDA(L E)(PROG(N)
        (SETQ N 1)
A       (COND((NULL L)(RETURN NIL))
              ((EQ E(CAR L))(RETURN N)))
        (SETQ L (CDR L))
        (SETQ N (ADD1 N))(GO A) ) ))
(PUTIN(LAMBDA(VCTR N NEW)(PROG(Z NCOUNT)
        (SETQ NCOUNT 1)(SETQ Z NIL)
LP      (COND((EQN NCOUNT N)(RETURN (APPEND Z (CONS NEW VCTR)))))
        (SETQ NCOUNT (ADD1 NCOUNT))
        (SETQ Z (APPEND Z (LIST(CAR VCTR))))
        (SETQ VCTR (CDR VCTR))
        (GO LP) ) ))

↓ GAE: GET ARRAY ELEMENT
  PAE: PUT ARRAY ELEMENT  ↓
(GAE(LAMBDA(ARRAY N)(PROG(SUM)
        (SETQ SUM 1)
 LP     (COND((EQN SUM N)(RETURN (CAR ARRAY))))
        (SETQ SUM (ADD1 SUM))(SETQ ARRAY (CDR ARRAY))
        (GO LP) ) ) )
(PAE(LAMBDA(VCTR N NEW)(PROG(Z NCOUNT)
        (SETQ NCOUNT 1)(SETQ Z NIL)
LP      (COND((EQN NCOUNT N)(RETURN (APPEND Z (CONS NEW
              (CDR VCTR))))))
        (SETQ NCOUNT (ADD1 NCOUNT))
        (SETQ Z (APPEND Z (LIST(CAR VCTR))))
        (SETQ VCTR (CDR VCTR))
        (GO LP) ) ))
```

```
↓ AMEAN READS A NEW ⇛MEAN⇛ FROM INPUT.  IF VERTEX
SPECIFIED EXISTS IT REPLACES OLD ⇛MEAN⇛.  IF VERTEX
DOES NOT EXIST IN NETWORK IT CREATES A NEW VERTEX
WITH MEAN SERVICE TIME ⇛MEAN⇛.   ↓
(AMEAN(LAMBDA(LEXLINE)(PROG(LINE I P M L N)
      (SETQ M MLOP)(SETQ L NIL)
      (SETQ I (CAAR LEXLINE))
      (SETQ LEXLINE(CDDR LEXLINE))(SETQ N(LIST I))
  (COND((NOT(MEMBER I VERTICES))(CSETQ VERTICES
        (CONS I VERTICES))))
AA    (SETQ P NIL)
A     (COND((EQUAL (CAR LEXLINE) SLASH)(GO B)))
      (SETQ P (APPEND P (LIST (CAR LEXLINE))))
      (SETQ LEXLINE (CDR LEXLINE))
      (GO A)
B     (SETQ P (EXPRESSION P))
      (SETQ N(APPEND N(LIST P)))
      (COND((NULL (CDR LEXLINE))(GO C)))
      (SETQ LEXLINE (CDR LEXLINE))
      (GO AA)
C     (COND((NULL M)(GO D))
           ((EQ I (CAAR M))(GO CC)))
      (SETQ L (APPEND L (LIST (CAR M))))
      (SETQ M (CDR M))
      (GO C)
CC    (CSETQ MLOP (APPEND L (CONS N (CDR M))))
      (RETURN NIL)
D     (CSETQ MLOP (CONS N MLOP))
      (PRQUOTE(CONS ICOPY(QUOTE(MEANS NOW DEFINED))))
      (PRQUOTE(APPEND(QUOTE(T =))(LIST VERTICES)))
      (RETURN NIL) ) ))

↓ DEDGE DESTROYS THE EDGE (NI P NO) FROM LIST OF
EDGES ⇛E⇛.  ↓
(DEDGE(LAMBDA(L)(PROG(NI NO E P)
      (SETQ E EDGES)
      (SETQ NI (CAAR L))(SETQ NO (CAADDR L))
      (SETQ L NIL)
A     (COND((NULL E)(GO Q))
           ((AND (EQ NI (CAAR E))(EQ NO (CADDAR E)))
             (GO O) ))
      (SETQ L (APPEND L (LIST (CAR E))))
      (SETQ E (CDR E))(GO A)
O     (CSETQ EDGES (APPEND L (CDR E)))
      (CSETQ EDGEFLAG 1)
      (RETURN NIL)
Q     (PRQUOTE(QUOTE(EDGE NOT FOUND)))(RETURN NIL) ) ))

↓ ANEDGE: IDENTICAL LOGICALLY TO ⇛AMEAN⇛   ↓
(ANEDGE(LAMBDA(L)(PROG(NI NO P E)
      (SETQ E EDGES)
      (SETQ NI (CAAR L))
      (SETQ L (CDDR L))
```

```
SKIM   (COND((EQUAL(CAR L)SLASH)(GO ASSMBL)))
       (SETQ P (APPEND P (LIST (CAR L))))
       (SETQ L(CDR L))(GO SKIM)
ASSMBL (SETQ NO (CAADR L))
       (SETQ P (EXPRESSION P))
       (SETQ L NIL)
GET    (COND((NULL E)(GO Q))
             ((AND(EQ NI(CAAR E))(EQ NO(CADDAR E)))
               (GO O)))
       (SETQ L (APPEND L (LIST(CAR E))))
       (SETQ E (CDR E))
       (GO GET)
O      (CSETQ EDGES (APPEND L(CONS(LIST NI P NO)(CDR E))))
             (RETURN NIL)
Q      (CSETQ EDGES(APPEND EDGES(LIST(LIST NI P NO))))
       (CSETQ EDGEFLAG 1)
       (RETURN NIL) ) ))

↓ TPUT: RETURNS UNNORMALIZED THROUGHPUT RATE  ↓
(TPUT(LAMBDA()(PROG(D P Q PL B)
   (SETQ D (SOQ ORIGIN))
A  (COND((NULL D)(RETURN P)))
   (SETQ Q(CAR D))(SETQ D(CDR D))
   (SETQ PL(PLACE VERTICES Q))
   (SETQ B(LENGTHMEANS Q MLOP))
   (SETQ P(CONS(TPUT1 Q PL B)P))(GO A) ) ))

↓ TPUT1: RETURNS UNNORMALIZED THROUGHPUT OF NODE Q ↓
(TPUT1(LAMBDA(Q PL B)(PROG(A SUM MEAN L C)
   (SETQ A DEGMULPROG)(SETQ SUM NIL)(SETQ C NIL)
   (COND((GREATERP B DEGMULPROG)(SETQ B DEGMULPROG)))
   (SETQ L(SUB1(LENGTH VERTICES)))
W  (COND((GREATERP B A)(GO V)))
   (SETQ MEAN(GETMEAN Q B MLOP))
   (SETQ SUM (RATAD(GEN PL (DIFFERENCE DEGMULPROG A) L A)SUM))
   (SETQ A (SUB1 A))  (GO W)
V  (SETQ SUM (LIST SUM MEAN))
   (SETQ C (CONS SUM C))
   (COND((ZEROP A)(RETURN C)))
   (SETQ SUM NIL)(SETQ B A)(GO W) ) ))

↓ PRTPUT: PRINTS THROUGHUT GIVEN THE RESULTS OF *TPUT* ↓
(PRTPUT(LAMBDA(X)(PROG(C D)
A  (COND((NULL X)(RETURN NIL)))
   (SETQ C(CAR X))(SETQ X(CDR X))
B  (COND((NULL C)(GO A)))(PRIN1 BLANK)(PRIN1 PLUSS)
   (SETQ D(CAR C))(SETQ C(CDR C))
   (PRIN1 LBRACK)(SETPR(CAR D))(PRIN1 RBRACK)(PRIN1 SLASH)
   (PRIN1 LBRACK)
   (SETPR(CADR D))(PRIN1 RBRACK)(TERPRI)(GO B) ) ))
(LENGTHMEANS(LAMBDA(Q L)(PROG()
A  (COND((NULL L)(PROG()(PRQUOTE ■(MEANS NOT DEFINED FOR NODE -))
                     (PRIN1 Q)(RETURN NIL)))
```

```
         ((EQ(CAAR L)Q)(RETURN (LENGTH (CDAR L)))))
    (SETQ L(CDR L))  (GO A) ) ))

↓ TEMPLATE DEFINES THE TEMPLATE TO USER SPECIFICATION
IF REQUESTED BY SETTING GLOBAL VARIABLE ⇛VERTICES⇛.↓
(TEMPLATE(LAMBDA()(PROG(D N)
      (SETQ D(PRQUOTE(QUOTE(*DO YOU WISH TO
      SPECIFY THE TEMPLATE ≥: YES/NO;)))))
      (COND((EQUAL(LEXICAL(READER))(QUOTE((YES))))(GO A))
            (T(PROG2 (PRQUOTE(APPEND(QUOTE(T=))(LIST VERTICES)
      ))(RETURN NIL)))))
A     (SETQ D(PRQUOTE(QUOTE(T=))))
      (SETQ D(CDR(LEXICAL(READER))))(CSETQ VERTICES NIL)
B     (SETQ N(CAR D))
      (COND((EQUAL(CADR D)RPAR)(GO C)))
      (SETQ D (CDR D))
      (CSETQ VERTICES(APPEND VERTICES N))(GO B)
C     (CSETQ VERTICES(APPEND VERTICES N))(RETURN NIL) ) ))
```

BIBLIOGRAPHY

[1] K. M. Chandy, "Exponential and Processor-Sharing Queueing Network Models for Computer Systems", unpublished manuscript, (1972).

[2] K. M. Chandy, "The Analysis and Solutions for General Queueing Networks," Proc. Sixth Annual Princeton Conference on Information Sciences and Systems, Princeton Univ., Princeton, N. J. (March 1972).

[3] K. M. Chandy, T. W. Keller and J. C. Browne, "Design Automation and Queueing Networks", Proc. Ninth Annual Design Automation Conference, 9, pp. 357-367 (June 1972).

[4] K. M. Chandy, "Queueing Models in Computer Systems", Proc. First Texas Symposium on Computer Systems, pp. I-4-1-I-4-7, Univ. of Texas at Austin (June 1972).

[5] K. M. Chandy, "Local Balance in Queueing Networks with Several Classes of Customers", SIAM Annual Meeting, Philadelphia (June 1972).

[6] F. Palacios-Gomez, "An Analytic Model of a Multiprogramming System Including a Job Mix", Computer Sciences Dept. Report TR-4, Univ. of Texas at Austin (June, 1972).

[7] F. Baskett and F. Palacios-Gomez, "Processor-Sharing in a Central Server Queueing Model of a Multiprogramming System With Applications", Proc. Sixth Annual Princeton Conference on Information Sciences and Systems, Princeton, N. J. (March 1972).

[8] F. Baskett and R. R. Muntz, "Queueing Network Models with Different Classes of Customers", Proc. COMPCON 72, San Francisco (Sept. 1972).

[9] W. J. Gordon and G. F. Newell, "Closed Queueing Systems with Exponential Servers", Opns. Res., 15, pp. 254-267 (1967).

[10] J. Buzen, Queueing Network Models of Multiprogramming, Ph.D. Thesis, Division of Engineering and Applied Science, Harvard University, Cambridge, Mass. (1971).

[11] W. A. Martin and R. J. Fateman, "The MACSYMA System", Proc. Second Symposium on Symbolic and Algebraic Manipulation, Los Angeles, Calif. pp. 59-75, (1971).

[12] C. Engelman and A. J. Kleinman, "Machine-Made Analytic Solutions to Finite State Markov Processes", MITRE Corp. Report M72-91, (July, 1972).

[13] K. B. Irani and V. L. Wallace, "On Network Linguistics and the Conversational Design of Queueing Networks", JACM 18,4, pp. 616-629, (Oct. 1971).

[14] V. L. Wallace and R. S. Rosenberg, "Markovian Models and Numerical Analysis of Computer System Behavior", Proc. AFIPS SJCC, Vol. 28, pp. 141-148, (1966).

[15] G. E. Collins, "The SAC-1 System: an Introduction and Survey", Proceedings of the Second Symposium on Symbolic and Algebraic Manipulation, Los Angeles, (March, 1971).

[16] J. McCarthy, et. al., LISP 1.5 Programmers Manual, 2nd ed., The MIT Press, Cambridge, Mass. (1969).

[17] E. Parzen, Stochastic Processes, 1st ed., Holden-Day, San Francisco, Calif. (1965).

[18] E. G. Coffman and L. Kleinrock, "Feedback Queueing Models for Time-Shared Systems," JACM 15, 4, pp. 549-576 (Oct. 1968).

[19] J. R. Jackson, "Jobshop-Like Queueing Systems," Man. Sci., 10, pp. 131-142 (1963).

[20] G. Lasseter, A Model of Predictive CPU Schedulers of Known Uncertainty, M.A. Thesis, Dept. of Computer Sciences, Univ. of Texas at Austin, Austin, Texas (1972).

[21] A. C. Hearne, "REDUCE2, A System and Language for Algebraic Manipulation", Proc. of the Second Symposium on Symbolic and Algebraic Manipulation, Los Angeles, (1971).

[22] D. R. Foster, private communication, (1972).

[23] P. M. Morse, Queues, Inventories and Maintenance, John Wiley, New York, (1958).

[24] D.R. Cox and W.L. Smith, Queues, Metheun, London, (1961).

[25] F. Baskett, Mathematical Models of Multiprogrammed Computer Systems, Ph.D. Thesis, Dept. of Computer Sciences, Univ. of Texas at Austin, Austin, Texas, (1970).

[26] T. W. Keller, D. F. Towsley, K. M. Chandy, J. C. Browne, "A Tool for Network Design: The Automatic Analysis of Stochastic Models of Computer Networks", to appear in Proc. COMPCON 73, San Francisco (Feb. 1973).