NOTATION FOR MARKOVIAN SYSTEMS

by

Winfried K. Grassmann
University of Saskatchewan
and Purdue University

Department of Statistics
Division of Mathematical Sciences
Mimeograph Series #456

July 1976

Notation for Markovian Systems.

Winfried K. Grassmann, University of Saskatchewan.

Key words: Queueing, Erlang queues, Queueing systems, integer programming.

Abstract: This paper gives a concise notation to describe Markovian systems. The application is explained for the example of parallel queues and for two cases of Erlang queues. The translation of the notation into a transition matrix is also shown.

The notation has a similarity with integer programming, which can be exploited, both when formulating systems with the notation and when translating it to the transition matrix.

1. Introduction

In queueing theory, there used to be a strong tendency to investigate each model by a tailor-made method. There is a theory for sequential queues, a theory for parallel queues, a theory for queueing networks, etc.

Recently, several authors have tried to overcome this specialization and have proposed models that can be applied in a great number of different queueing situations and possibly other areas.

Specifically, it turns out that a great number of queueing problems are in fact Markov processes. A Markovian queue, e.g. is by definition a Markov process, but so are all Erlang queues. Also, systems of Markovian queues, such as parallel queues, sequential queues etc. are Markov processes. This approach is also useful because it allows to investigate systems which are not queueing systems such as inventory systems. They, too, are often Markov processes.

There is a great number of nice theories about Markov processes. However, before one can apply these theories, a great number of problems have to be overcome.

The first, and major problem, is the curse of dimensionality. Most systems give rise to multidimensional Markov processes, and the number of states tends to increase with the power of the dimension. It is for this reason that Markov processes modelling systems usually have an extremely high number of states, say 1000 and over.

The transition matrix of a Markov process with 1000 states has apparently 1000^2 = one million entries.

To find all these entries by hand is impossible, and programs generating the matrices have to be developed. There are several approaches in this direction [5,10]. The next problem is then to find transient or steady state probabilities, and the last step is to summarize these probabilities by averages, variances and whatever other measures are required.

This paper proposes now a very general notation to describe Markovian systems, which can be used as input for a matrix generator. It also develops an efficient matrix generator. The paper does not contain any information how to derive transient and steady state probabilities since this is done elsewhere [3,6,8,10]. Also, the calculation of performance measures is not discussed.

The notation proposed resembles the one proposed by Irani [7], though it was created independently by us. The matrix generator proposed is a backtracking algorithm which is very efficient for enumerations [2,9].

Though the work was originally used for a general program analysing transient queues (4), the results of this paper should be applicable to a much wider area.

2. Systems and their mathematical description

In this paper, any multidimensional stochastic process is called system. In other words: For each time t, a system can be described by a set of random variables $X_1, X_2, \ldots, X_d$. The d-tupel $X_1, X_2, \ldots, X_d$ will be denoted by $\underline{X}$ and will be called the state of the system. If, at a certain point in time, all X_i assume certain values x_{im}, we say state m is realized. m is thus the d-tupel $x_{1m}, x_{2m}, \ldots, x_{dm}$. In some formulae, both $\underline{X}$ and m will be used in place of a column vector. However, this usage should always be obvious from the context.

For our purposes, it is assumed that all X_i can only assume the values 0,1,2... . The set of all states is called sample space and denoted by D_0.

At this point, it might be useful to introduce an example:

Let there be a system that consists of several interrelated entities, called subsystems. Depending on the application, these subsystems may be queues, inventories, sub-populations, etc. Each subsystem contains X_i elements. The elements may be customers waiting for service, units in inventories, individuals of a certain species, etc.

The states of any system will change. In the system described above, state changes may be caused by arrivals to the system, departures from the system, elements changing the subsystem they are in, etc.

Any change of the system from one state to another is called a transition. The transitions, in turn, are partioned into groups of transitions having similar properties. These sets of transitions are called events. An event k may e.g. be an arrival at subsystem X_j because all such arrivals have the common property of increasing X_j by 1. Other events might be a departure from subsystem i, a switch from subsystem i to subsystem j, etc. There are no specific rules how to partition transitions into events, but the problem on hand usually suggests a specific set of events.

For each event k, there is an interevent time T_k. The rate of event k is defined as:

$$\lambda_k(\underline{X}) = 1/E(T_k).$$

The rate may or may not depend on the state $\underline{X}$. Henceforth, it will be assumed that the system is Markovian. This implies that all T_k follow an exponential distribution.

Each event has a predictable effect. In particular, if $m = x_{1m}, x_{2m},\ldots,x_{dm}$ is the state before event k occurs, the state after event k has occurred equals with probability 1:

$$n = f_k(m) = f_{1k}(m),\ f_{2k}(m),\ldots,f_{dk}(m).$$

State m, the state before event k, will be called originating state, whereas state n is the destinating state.

Any event k can only occur when certain conditions are met. These conditions are of two kinds, namely explicit and implicit:

Any event k can only occur if both originating state and destinating state are part of the sample space D_0. These conditions are the implicit conditions. An example of an implicit condition is e.g. the condition that there can be no departure from subsystem j if $X_j = x_{jm} = 0$, since the X_j of the destinating state would become negative otherwise. All states meeting the implicit conditions form a set D_k^I, named the implicit set of event k.

Some conditions have to be stated explicitly and are therefore called explicit conditions. As an example, suppose arrivals in subsystem j can only occur if $X_j < X_i$, $i=1,2,\ldots,d$, $i \neq j$. Such may e.g. be the case when the subsystems represent parallel queues in front of a counter, and all arrivals join the shortest queue. $X_j < X_i$ would then be an explicit condition of event k.

The set of all d-tupels $\underline{X}$ with integer components satisfying the explicit conditions from the explicit set of event k, called D_k^E. D_k^E may exceed the sample space D_0.

There are not necessarily any explicit conditions. In this case, D_k^E includes all integer d-tupels.

The event set of event k, D_k, is defined as:

$$D_k = D_k^I \cap D_k^E.$$

Clearly, points of this intersection satisfy all explicit and implicit conditions.

Summing up, one can characterize a system with K events by the following expression B:

$$B = [\lambda_k(\underline{X}), f_k(\underline{X}), D_k^E, \underline{X} \varepsilon D_0, k=1,2,\ldots,K].$$

B, together with the present state $\underline{X}$=m allows, at least in principle, to find for any future moment the probability that the system is in state n. As Irani (7) has shown, B actually defines the transition matrix of a continuous Markov process with the state space D_0. The reverse is also true, at least if D_0 is finite, as will be shown later.

For our discussions, B will be restricted as follows.

1) D_0 is finite, i.e. it only contains a finite number of sample points.

2) The conditions defining D_0 are assumed to be linear inequalities, which can be written in matrix form as:

$$C \underline{X} \leq \underline{b} \qquad (1)$$

Here, C is a matrix, and $\underline{b}$ a column vector.

3) The explicit conditions of event k are similarly assumed to be linear inequalities defined by the matrix C_k and a vector $\underline{b}_k$:

$$C_k \underline{X} \leq \underline{b}_k \qquad (2)$$

If there are no explicit conditions, C_k and $\underline{b}_k$ are zero.

4) $f_k(\underline{X})$, the function mapping the originating state into the destinating state, can be expressed as:

$$f_k(\underline{X}) = H_k \underline{X} + g_k. \qquad (3)$$

There, H_k is a d by d matrix, and g_k is a column vector. In almost all cases considered by us, H_k turned out to be the identity matrix.

Under the restrictions stated, B can be completely specified by $\lambda_k(\underline{X})$, H_k, g_k, C_k, $\underline{b}_k$, k=1,2,...,K, C, $\underline{b}$. The next two sections show how to find these vectors and functions in a number of concrete cases. It will turn out that the notation is not only extremely compact, but also extremely flexible.

3. Parallel queues with jockeying

Suppose, the B of the following problem has to be found:

In front of 3 ticket counters, there are 3 different queues. Customers arrive at a rate λ and always join the shortest queue. In case of a tie, they prefer queue 1 over queue 2 and queue 3, and queue 2 over queue 3.

No arrivals occur if (a) one queue exceeds 10 customers or (b) if the number of customers in all three queues together exceeds 15. These restrictions are imposed to keep the sample space finite.

Once in line, customers may switch queues. Specifically, if any queue is 5 or more customers shorter than the queue with the longest line, the last customer of the longest line switches. This insures that the differences between the queues cannot for any length of time exceed 4. In case of a tie between the two longest queues, the customer of the close queue jockeys. It is assumed that queue 1 and 2 are closest, where as queue 1 and 3 are farthest apart.

To find B for that problem, let X_i be the number of customers in queue i. Clearly, $X_i \geq 0$, in accordance with the model. The state space D_0 is further defined by the following conditions:

1) Each queue must be less than or equal to 10, i.e:

$$X_i \leq 10 \qquad i=1,2,3 \tag{4}$$

2) The total number of customers in all three queues is less than or equal to 15, i.e.:

$$X_1 + X_2 + X_3 \leq 15 \tag{5}$$

3) The difference between two queue is less than or equal to 4, i.e:

$$X_i - X_j \leq 4, \qquad i,j = 1,2,3,\ i \neq j \tag{6}$$

The inequalities (4) - (6) taken together form system (1).

All possible transitions can be partitioned into K = 12 events, which are as follows:

Let event 1 be the arrival at queue 1. Clearly:

$$\lambda_1\ (\underline{X}) = \lambda$$

$$f_1\ (X_1, X_2, X_3) = X_1 + 1,\ X_2,\ X_3$$

$$X_1 \leq X_2$$

$$X_1 \leq X_3$$

The last two inequalities represent the explicit conditions. They indicate that customers join queue 1, unless it exceeds the other queues. To bring these conditions into the form given by (2), they must be written:

$$X_1 - X_2 \leq 0$$

$$X_1 - X_3 \leq 0$$

The implicit conditions, such as $X_1 + X_2 + X_3 \nleq 15$ need not be mentioned because they can be derived from $f_1(\underline{X})$ and D_0. (When $X_1 + X_2 + X_3 = 15$, an arrival would bring the total to 16 and is thus impossible).

Event number 2 is an arrival in the second queue. One has:

$$\lambda_2\ (\underline{X}) = \lambda$$

$$f_2\ (X_1, X_2, X_3) = X_1, X_2 + 1, X_3$$

$$X_2 < X_1$$

$$X_2 \leq X_3.$$

The last two inequalities, (they indicate that arrivals join queue 2, if queue 2 is shorter than queue 1, and if it is not longer than queue 3), can be written as:

$$-X_1 + X_2 \leq -1$$

$$X_2 - X_3 \leq 0$$

Event number 3 is an arrival in the third queue. One has:

$$\lambda_3\ (\underline{X}) = \lambda$$

$$f_3\ (X_1,X_2,X_3) = X_1,X_2,X_3 + 1$$

$$X_3 < X_1 \text{ or } -X_1 + X_3 \leq -1$$

$$X_3 < X_2 \text{ or } -X_2 + X_3 \leq -1$$

Event number 4 is a departure from queue 1. One finds:

$$\lambda_4\ (\underline{X}) = \mu_1$$

$$f_4\ (X_1,X_2,X_3) = X_1-1,X_2,X_3 \tag{7}$$

Event 4 has no explicit conditions. $X_1 \geq 1$ is implied by equation (6) and $X_1 \geq 0$. Furthermore, relation (6) and (7) imply:

$$(X_1-1)-X_j \leq 4. \quad j = 2,3.$$

However, these implicit conditions need not be mentioned.

Event 4 and 5 represent departures from queue 2 and queue 3, respectively. These events are identical with event 3, except that $\lambda_k(m) = \mu_{k-3}$, $k = 4,5$, and X_1 in equation (7) has to be replaced by X_2, respectively, X_3.

Events 6,7,8,9,10,11 and 12 are events describing jockeying. The problem as formulated above implies that jockeying can occur only following a departure. For this reason, one combines the effect of the departure with the effect of the switch of the queue. This means that the queue having the departure loses one customer because his service is completed, and gains one customer because of jockeying, giving a net change of zero. The longest queue, however, has a net loss of 1. These remarks should enable the reader to derive the following relations for event 6, which describes a departure from server 1, followed by a switch from queue 2:

$$\lambda_6(\underline{X}) = \mu_1$$

$$f_6(X_1,X_2,X_3) = X_1,X_2-1,X_3$$

$$X_2 = X_1 + 4$$

There, the explicit condition indicates that following a departure, a switch from queue 2 to queue 1 occurs if X_2 is four elements longer than queue 1 before the departure. To bring the explicit condition into the form required by equation (2), it is written:

$$\begin{aligned} -X_1 + X_2 &\le 4 \\ X_1 - X_2 &\le 4 \end{aligned}$$

The remaining events will no longer be discussed in detail. Rather, we describe B by the following table:

Table 1: B for parallel queues

Event	g_k^T	rate	D_k^E
1	1,0,0	λ	$X_1-X_2 \le 0$, $X_1-X_3 \le 0$
2	0,1,0	λ	$-X_1 + X_2 \le -1$, $X_2-X_3 \le 0$
3	0,0,1	λ	$-X_1 + X_3 \le -1$, $-X_2 + X_3 \le -1$
4	-1,0,0	μ_1	
5	0,-1,0	μ_2	
6	0,0,-1	μ_3	
7	0,-1,0	μ_1	$-X_1 + X_2 \le 4$, $X_1-X_2 \le -4$
8	0,0,-1	μ_1	$-X_1 + X_3 \le 4$, $X_1-X_3 \le -4$, $X_2-X_3 \le -1$
9	-1,0,0	μ_2	$X_1-X_2 \le 4$, $-X_1 + X_2 \le -4$
10	0,0,-1	μ_2	$-X_2 + X_3 \le 4$, $X_2-X_3 \le -4$, $X_1 - X_3 \le -1$
11	0,-1,0	μ_3	$+ X_2-X_3 \le 4$ $-X_2 + X_3 \le -4$
12	-1,0,0	μ_3	$+X_1-X_3 \le 4$ $-X_1 + X_3 \le -4$ $-X_1+X_2 \le 1$
State space (D_0)			$X_1 \le 10$, $X_2 \le 10$, $X_3 \le 10$, $X_1 + X_2 + X_3 \le 15$ $X_1-X_2 \le 4$, $X_1-X_3 \le 4$, $X_2-X_1 \le 4$, $X_2-X_3 \le 4$ $X_3-X_1 \le 4$, $X_3-X_2 \le 4$

This table contains g_k^T, the amount by which $\underline{X}$ increases. Thus, for event 1, $g_k^T = 1,0,0$, because X_1 increases by 1, X_2 by 0 and X_3 by zero. For the other events, g_k^T has to be interpreted in a similar way.

For event 1 to 7, the table only summarizes what was explained before. Event 8 gives a switch from queue 3 to queue 1. This can only happen if $X_3 = X_1 + 4$ and $X_2 \neq X_1 + 4$, or, what is the same, $X_2 < X_3$. Hence the condition $X_2 - X_3 \leq -1$. With these remarks, events 9, 10, 11 and 12 should become clear as well. At the bottom of the table, there are, finally, all inequalities defining D_0.

It is clear that all the information given by table 1 can easily be converted to form computer input. We also note that the mathematical description of the system as given by table 1 is considerably shorter than the English text describing the same problem.

4. Erlang queues

The same ideas used in the previous section should allow the reader to formulate a variety of other queueing problems, including queues with feedback, sequential queues and even queueing networks. However, some minor difficulties arise when analysing queues with Erlang service times. By formulating two different Erlang queues, namely the $M/E_r/S$ queue with homogeneous servers and the $M/E_r/2$ queue with heterogeneous servers, it is shown how to resolve these difficulties. Incidentally, the same techniques can be applied to formulate priority queues.

The state space of the homogeneous $M/E_r/S$ queue is characterized by the random variables $X_0,X_1,X_2,\ldots,X_{r+1}$. X_{r+1} is the queue length, excluding the elements being served, and $X_0,X_1,X_2,\ldots,X_r$ are the number of servers having still $0,1,2,\ldots,r$ phases to do until the service of their customer is complete.

The inequalities determining D_0 are now:

1) There are a total of s servers. Therefore:

$$X_0 + X_1 + \ldots + X_r \leq s \tag{7}$$

2) In order to keep D_0 finite, let X_{r+1} be restricted to 20, or

$$X_{r+1} \leq 20.$$

The events are, essentially, arrivals, completion of a phase, and departures. However, it will turn out that some arrivals and departures are special cases, forming their own events.

1) Event number 1 is an arrival. Arrivals apparently increase X_{r+1}, hence:

$$f_1(\underline{X}) = X_0,X_1,\ldots,X_r,X_{r+1} + 1.$$

If the arrival rate is λ, one has more over.

$$\lambda_1(\underline{X}) = \lambda.$$

If $X_0 > 0$, the arrival will not increase X_{r+1}, but X_r. This situation is best dealt with by introducing a separate even. For this reason, we impose an explicit condition:

$$X_0 = 0.$$

2) When $X_0 > 0$ while an arrival occurs, one server becomes occupied, i.e. instead of zero phases, this server starts with r phases again. Hence:

$$f_2(\underline{X}) = X_0-1, X_1,\ldots,X_r+1, X_{r+1}.$$

Since all X_i, including X_0 must be non-negative, there are no explicit conditions connected with this event. The rate of the event is again:

$$\lambda_2\ (\underline{X}) = \lambda.$$

3) Events $1 + i$, $i = 2,\ldots,r$ are completion of phases not affecting the queue length X_{r+1}. Their effect is:

$$f_{1+i}\ (\underline{X}) = X_0,X_1,\ldots,X_{i-2}, X_{i-1} + 1, X_i-1, X_{i+1},\ldots,X_r,X_{r+1}.$$

To calculate the rate of the events $1+i$, $i=2,\ldots,r$, suppose each server has a service rate of μ, or, what is the same, the rate of serving one phase is $r\mu$. Since there are X_i servers with i phases to go, event $1+i$ has a rate:

$$\lambda_{1+i}(\underline{X}) = X_i r\mu,\ i=2,\ldots,r$$

All these events have no explicit condition.

4) The next event is a departure followed by the start of the service from the queue. One has:

$$f_{2+r}(\underline{X}) = X_0, X_1-1, X_2, \ldots, X_r+1, X_{r+1}-1$$

$$\lambda_{2+r}(\underline{X}) = X_1 r\mu$$

This event has no explicit condition.

5) The fifth event is departure while $X_{r+1} = 0$. This event can be described as:

$$f_{3+r}(\underline{X}) = X_0+1, X_1-1, X_2, \ldots, X_r, X_{r+1}$$

$$\lambda_{3+r}(\underline{X}) = X_1 r\mu$$

$$X_{r+1} = 0.$$

To sum up, the $M/E_r/S$ queue can be characterized by 3+r events.

As a last example, consider the $M/E_r/2$ queue with the first server having service rate μ_1 and the second, μ_2. Furthermore, it is assumed that (other things being equal) a customer will prefer server 1. For this problem, the state space is described by the 3 random variables X_1, X_2 and X_3. X_3 is the queue length, excluding the elements being served, X_1 is the number of phases server 1 has to do until service is completed, and X_2 is same thing for server 2. The restrictions defining D_0 are now, provided X_3 is limited to 20:

$$X_3 \leq 20$$

$$0 \leq X_1 \leq r$$

$$0 \leq X_2 \leq r$$

$$20\ X_1 \geq X_3 \tag{9}$$

$$20\ X_2 \geq X_3 \tag{10}$$

Relation (9) makes sure that X_1 can only be zero if X_3 is zero and relation (10) gives a similar restriction for X_2. In equalities of type (9) and (10) are frequently used in integer programming, especially in connection with the fixed charge problem. There is thus a connection between integer programming and the notation presented here.

Having defined D_0, the formulation of the events is relatively easy. The events are:

1) An arrival increases X_3. One has:

$$f_1(\underline{X}) = X_1, X_2, X_3+1.$$

If λ is the arrival rate, one finds:

$$\lambda_1(\underline{X}) = \lambda.$$

There are no explicit conditions for this event.

2) An arrival occurs while server 1 is idle ($X_1 = 0$). One has:

$$f_2(\underline{X}) = X_1+r, X_2, X_3$$

$$\lambda_2(\underline{X}) = \lambda.$$

There are no explicit conditions, because $X_1 \leq r$ implies $X_1+r \leq r$ or $X_1 = 0$.

3) An arrival occurs while server 2 is idle, but server 1 is not. Then:

$$f_3(\underline{X}) = X_1,\ X_2+r,\ X_3$$

$$\lambda_3(\underline{X}) = \lambda$$

$$X_1 > 0.$$

The condition $X_1 > 0$ is necessary because if $X_1 = 0$, the arrival prefers server 1.

4) Server one completes a phase, but no new service starts, either because the line is empty ($X_3 = 0$), or because there are still more phases to go. Then:

$f_4(\underline{X}) = X_1-1,X_2,X_3$

$\lambda_4(\underline{X}) = r\mu_1.$

5) Server two completes a phase, but no new service starts. Then:

$f_5(\underline{X}) = X_1,X_2-1,X_3$

$\lambda_5(\underline{X}) = r\mu_2.$

6) The next customer enters service at server 1:

$f_6(\underline{X}) = X_1-1 + k,X_2,X_3-1$

$\lambda_6(\underline{X}) = r\mu_1.$

7) The next customer enters service of server 2, i.e:

$f_7(\underline{X}) = X_1,X_2-1 + k,X_3-1$

$\lambda_7(\underline{X}) = r\mu_2.$

The reader may check that all conditions of events 4 to 7 are implied conditions which need not be stated. B for the $M/E_r/2$ queue with hetrogeneous servers is thus fully described.

The limited space prevents us to describe more cases, but we hope that the 3 examples presented give the reader some idea of the power of the notation. The next section gives an algorithm to convert B into a transition matrix, using again ideas developed for solving integer programming problems.

5. The translation

For the purpose of translation, one needs D_k, the set of all points satisfying all implicit and explicit conditions of event k. The implicit conditions are:

$$C\,\underline{X} \leq \underline{b} \tag{11}$$

$$C\,H_k\underline{X} \leq \underline{b}-Cg_k \tag{12}$$

Relation (11) is just relation (1). Relation (12) can be obtained from (1) and (3), using the fact that the $f_k(\underline{X})$ must also satisfy (1). This gives:

$$C\,f_k(\underline{X}) = C[H_k\,\underline{X} + g_k] = C\,H_k\,\underline{X} + Cg_k \leq \underline{b}.$$

Deducting Cg_k from both sides of this inequality gives (12). (11) and (12) together with (2), define D_k. The new system has again the form of relation (2), i.e:

$$C_k'\,\underline{X} \leq \underline{b}'. \tag{13}$$

There, $C_k' = [C,\ C\,H_k, C_k]^T$ and $b' = [\underline{b}, \underline{b}-Cg_k,\ \underline{b}_k]^T$.

In the case that H_k is the identity matrix, (which was true in all examples discussed here), relation (11) and (12) can be combined to give:

$$C\,\underline{X} \leq \underline{b}^*. \tag{14}$$

Here, C is the same matrix as in (11), whereas $\underline{b}^* = [b_v^*]$ is found as:

$$b_v^* = \min\,(b_v,\ b_v - \sum_{i=1}^{d} c_{vi}g_{ki}).$$

Here, b_v is the v<u>th</u> component of $\underline{b}$ and g_{ki} the i<u>th</u> component of g_k. The c_{vi} are similarly the entries of c.

Inequality (14) implies both (11) and (12) for the following reason: If H_k is the identity matrix, the left hand sides of (11) and (12) are $\underline{b}$ and $\underline{b}$-Cg_k, respectively. Now, take the v^{th} inequality of (11) and compare it with the v^{th} inequality of (12). This gives:

$$\sum_{i=1}^{d} c_{vi} X_i \leq b_v$$

$$\sum_{i=1}^{d} c_{vi} X_i \leq b_v - \sum_{i=1}^{d} c_{vi} g_{ki}.$$

It is obviously sufficient to retain only the inequality with the lower right hand side, which is b_v^*. To include this case by relation (13), define C' and b' for H_k=I as:

$$C_k' = [C, C_k]^T$$

$$\underline{b}' = [\underline{b}^*, \underline{b}_k]^T.$$

As was mentioned earlier, events are sets of transitions. Formally transition number h can be expressed as

$$T_h = [m_h, n_h, a_h].$$

There, m_h is the originating state, n_h the destinating state, and a_h the rate. If transition h belongs to event k, one clearly has:

$$T_h = [m_h, f_k(m_h), \lambda_k(m_h)]. \qquad (15)$$

The list of all transitions is called the transition list of the system. The list of the transitions of event k is called the transition list of event k.

The transition list of the system is easily converted into a transition matrix. To see that, define

$$a_{mn} = \sum_{h \epsilon U} a_h \qquad U = [h | m_h = m, n_h = n]$$

and

$$a_{mm} = -\sum_{m \epsilon D_0} a_{mn}.$$

The matrix A = $[a_{mn}]$ is the transition matrix.

The transition list of event k can be obtained by enumerating all m_h satisfying (13) and calculating T_h according to (15). The union of the transition lists of all events forms then the transition list of the system, which can then be transformed into a transition matrix. However, as pointed out by Hiller [6], it is preferable to work with the transition list than with the transition matrix.

Incidentally, any transition matrix can trivially be transformed into a transition list, and any transition list is really the B of the system. To see this, take the hth non zero entry of A. Suppose this entry is in row m_h, column n_h, and its value is $a_{m_h n_h} = a_h$. Clearly, this entry defines a transition. Next, each transition h can be thought to form a separate event, with D_h consisting of m_h only, and $f_h(\underline{X}) = n_h, \lambda_h(\underline{X}) = a_h$.

To enumerate all $m_h \in D_k$, it seems appropriate to use a backtrack - algorithm. As shown by Whitehead [9] and Golomb [2], backtrack algorithms are extremely efficient to do such an enumeration.

The essential idea of the algorithm for enumerating all m_n is now as follows:

Suppose, there are upper and lower bounds for each X_i, i.e:

$$x_i^L \leq X_i \leq x_i^u$$

Such bounds can normally be found from inspection of equation (13). Also, Fourier [1] has given a systematic method to find such bounds.

As the next step, one finds a sequence of bounds $\hat{x}_i^L$ and $\hat{x}_i^u$, given X_j assumes the value x_j for j < i:

$$\hat{x}_i^L \leq X_i \leq \hat{x}_i^u, \quad \text{given } X_j = x_j,\ j < i,\ i=1,2,\ldots d.$$

These bounds can be found in a way such that the bound on X_d, $\hat{x}_d^u$ and $\hat{x}_d^u$ are tight in the sense that only points of D_k satisfy these bounds and have $X_j = x_j$, $j < d$ at the same time.

One now can find a first point by setting $X_1 = \hat{x}_1^L$, $X_2 = \hat{x}_2^L, \ldots, X_i = \hat{x}_i^L$, etc. Whenever a new lower bound $\hat{x}_i^L$ is calculated, it is compared with the upper bound $\hat{x}_i^u$. As soon as $\hat{x}_i^L > \hat{x}_i^u$, the points starting with the coordinates $\hat{x}_j^L$, $j < i$ cannot belong to D_k and can be disregarded. As soon as this happens, one or several backtracks are started. By this, we mean that X_{i-1} is replaced by $X_{i-1} + 1$, and new lower and upper bounds are calculated, using the new value for X_{i-1}. In this way, one eventually finds coordinates $X_1 = x_1$, $X_2 = x_2, \ldots, X_{d-1} = x_{d-1}$, and upper and lower bounds for X_d, namely $\hat{x}_d^u$ and $\hat{x}_d^L$. Since these last bounds are tight, one finds $\hat{x}_d^u - \hat{x}_d^L + 1$ points of D_k as follows:

$$X_1 = x_1,\ X_2 = x_2, \ldots,\ X_{d-1} = x_{d-1}, \hat{x}_d^L \le X_d \le \hat{x}_d^u. \tag{16}$$

These points are easily enumerated. Once this is done, one backtracks again. One also backtracks if $\hat{x}_D^L > \hat{x}_d^u$, i.e. if there is no point satisfying (16). In this way, one eventually enumerates all points of m_h of D_k, and the transitions can be evaluated readily, using (15).

The $\hat{x}_i^u$ and $\hat{x}_D^L$ can be evaluated as follows: Let c_{vj} be the elements of C_k' and b_v the elements of b'. Then, one can write equation (13) as:

$$\sum_{j=1}^{d} c_{vj} X_j \le b_v.$$

Hence:

$$c_{vi} X_i \le b_v - \sum_{j=1}^{i-1} c_{vj} x_j - \sum_{j=i+1}^{d} c_{vj} X_j.$$

Now, let $c_{vj} = c^+_{vj} - c^-_{vj}$, $c^+_{vj}, c^-_{vj} \geq 0$, and use the fact that X_j has the upper and lower bound x^u_j and x^L_j.

Then:

$$c_{vi}X_i \leq b_v - \sum_{j=1}^{i-1} c_{vj}x_j - \sum_{j=i+1}^{d} (c^+_{vj}\, x^L_j - c^-_{vj}\, x^u_j]$$

If S_{vi} is the right-hand side of this expression, one has:

$$c_{vi}X_i \leq S_{vi}.$$

For $c_{vi} > 0$, it follows:

$$X_i \leq S_{vi}/c_{vi}$$

Hence, one finds as upper bound:

$$\hat{x}^u_i = \min_{c_{vi}>0} (S_{vi}/c_{vi}).$$

Similarly, one finds as lower bound:

$$\hat{x}^L_i = \max_{c_{vi}<0} (S_{vi}/c_{vi}).$$

The S_{vi} can be calculated recursively:

$$S_{vi} = b_i - \sum_{j=1}^{i-1} c_{vj}x_j - \sum_{j=i+1}^{d} [c^+_{vj}x^L_j - c^-_{vj}x^u_j]$$

$$= S_{vi-1} - c_{vi-1}x_{i-1} + c^+_{vi}\, x^L_i - c^-_{vi}\, x^u_i. \qquad (17)$$

Of course, S_{v1} equals:

$$S_{v1} = b_v - \sum_{j=2}^{d} (c^+_{vj}x^L_j - c^-_{vj}x^u_j).$$

Whenever i is increased, all S_{vi} can be updated, using equation (17). The same equation allows to find S_{vi-1} from S_{vi} when backtracking is to be done.

There are two reasons why this algorithm should be extremely efficient: First of all, the S_{vi}, and with them, the $\hat{x}_i^L$ and $\hat{x}_i^u$ can be found very efficiently, and changed easily when backtracking. Secondly, because of (16), one normally obtains an entire group of solutions at once.

As with other backtrack algorithms, the order of the X_i is not irrelevant. In other words: Though the problem as such would not be affected by renumbering the X_i, the efficiency of the algorithm would change. Generally, it is suggested [2] to determine the X_i such that X_1 has the smallest range of possible values, X_2 the second smallest, etc, until X_d has the largest. This procedure can also be supported by equation (16).

6. Conclusions.

The article presented a notation which allows to describe any finite-state Markovian system. The basic entities of the notation are the events, which can be described by an effect-function, a rate, and a set of conditions. The notation is much more concise than the English language. Furthermore, the functions used to describe the system are in most cases linear and can thus be fully specified by vectors and matrices. Because vectors and matrices can be used as computer input, the notation is extremely suitable for computer programs. However, it should also be suitable for theoretical investigations, especially for classifying queueing problems. Finally it might help bridging the gap between practitioner and theorist because the notation gives a language easily mastered by both practitioner and theorist. When the notation is used as input for computer programs, it is essential to convert it to a transition matrix. This translation can be done efficiently by a specially designed backtrack algorithm.

It should be mentioned here that we wrote a program that made use of these concepts presented in this paper. This program was used as a teaching tool in an undergraduate class. It was a great success because it allowed the students to solve non-standard queueing problems they could not have solved otherwise. It has to be admitted, though, that the mathematical education of these students was limited.

References

1. G. B. Dantzig: Linear Programming and Extensions, Princeton University Press 1963, pp. 84-85.

2. S. Golomb, L. Baumert, "Backtrack Programming", Journal of the ACM 12 (1965), pp. 516-524.

3. W. K. Grassmann, Transient solutions in Markovian queueing Systems, Purdue University, Dept. of Statistics, Mimeograph Series #451.

4. W. K. Grassmann, R. T. Churchman, Users Guide for Transient Solutions for Queueing Networks, Working Paper, Dept. of Comp. Science, Univ. of Saskatchewan.

5. G. K. Grace, T. K. Byers: An Analysis of a Network of Finite-Queue, Multiple-Server Facility, Dept. of Computer Science, University of Missouri, Rolla.

6. F. S. Hillier, R. W. Boling. Finite Queues in Series, Operations Research, 15, pp. 286-303, 1971.

7. K. B. Irani, V. L. Wallace; On Network Linguistics and the Conversational Design of Queueing Networks, Journal of the ACM, Vol. 18, No. 4, Oct. 1971, pp. 616-629.

8. V. L. Wallace, R. S. Rosenberg, Markovian Models and Numerical Analysis of Computer System Behaviour, Proc. AFIPS Spring Joint Computer Conf. 28, 1966, pp. 141-148.

10. T. S. Whitlock, Modeling Computer Systems with Time-Varying Markov Chains, University of North Carolina, Chapel Hill, 1973.