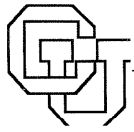
Probabilistic Models of Computer Deadlock *

Clarence A. Ellis

CU-CS-041-74



University of Colorado at Boulder
DEPARTMENT OF COMPUTER SCIENCE

*This work was supported by NSF Grant #GJ-660..

PROBABILISTIC MODELS OF COMPUTER DEADLOCK*

by

Clarence A. Ellis
Department of Computer Science
University of Colorado
Boulder, Colorado

Report #CU-CS-041-74 April, 1974

Key Words and Phrases: Deadlock, Deadly Embrace, Probabilistic Automata, Stochastic Processes, Operating Systems

CR Categories: 4.32, 5.22

* This work was supported by NSF Grant #GJ-660

ABSTRACT

As the number of processes and resources increase within a computer system, does the probability of that system's encountering deadlock increase or decrease? The problem of deadlock in computer systems and a model applicable to the investigation of this problem are presented. The model treats sequences of resource activity as potential members of the set of strings accepted by a probabilistic automaton. This paper, after explaining the model and its application, describes a transformation on the automaton which makes it amenable to calculations of the probability of deadlock. These calculations consist of:

1. Derivation of necessary and sufficient conditions for an automaton to be well-behaved -- formally described as accepting a normalized language, and
2. Usage of these conditions to yield closed-form equations of deadlock probability under several definitions thereof.

Although the automaton model used in these calculations is a probabilistic pushdown automaton, it is indicated that the procedures described can also be applied to other types of probabilistic automata modeling other deadlock situations. Results of calculations on actual computer system models are also described, indicating that within the types of systems considered, the probability of deadlock increases.

INTRODUCTION

The author has presented elsewhere[4] preliminary results concerning a model of deadlock in computer systems consisting of a probabilistic finite automaton. That model constituted a Markov process and so was quite tractible for mathematical analysis. One restriction of that model was that if two or more processes were waiting for a resource it would be randomly allocated to one of the processes. This paper presents an extension of the previous model which is a technique for modeling the more realistic allocation schemes of first come first serve and last in our first out service. This automaton is developed and then applied to calculate the probability of deadlock in computer systems. This model also forms an interesting study because it is a non-Markovian process.

THE DEADLOCK PROBLEM

Deadlock is a phenomenon which can occur in a wide variety of settings. For example, if two people hold halves of a treasure map and each refuses to give up his half until he has obtained the other's half, then there is a deadlock situation such that neither can find the treasure (assuming both halves of the map are necessary to locate the treasure). In the context of a large multiprocessor or multiprogramming computer system, deadlock (sometimes called deadly embrace) can occur if two or more processes, i.e., concurrently executing programs, each wait for a resource held by the other. Usually, these processes must remain idle for an indefinite amount of time until this situation is detected and manual intervention by the computer operator corrects this situation. Deadlock is possible only when three conditions are satisfied in a computer system:

1. A process can claim exclusive control over the resource it holds.
2. A resource cannot be preempted from a process.
3. A process can hold a unit of resource while requesting another unit of a (possibly different) resource, i.e., a circular wait is possible.

As a result of the elucidation of these criteria necessary for deadlock, the automatic detection and prevention of deadlock has recently received a lot of attention in the literature[5,6,7,8,9,13]. Our approach in this paper is different from these previous citations because we use probabilistic techniques to investigate the likelihood of deadlock occurring in certain classes of computer systems. If it can be shown that the probability of deadlock increases as the number of processes and resources becomes large, then the deadlock problem will need to be explicitly dealt with (perhaps in hardware or firmware) in multi-multi-processor systems of the future.

THE MODEL

A computer system consisting of a set of processes and varying units of various types of resources can be described at any particular instant of time as being in some state defined by: (1) the number of units of each type of resource available, (2) the number of units being held by each process, and (3) the number of units being requested by each process. A transition or change to another state corresponds to a new request, an allocation, or some other system action causing any of the above three parameters to change. Holt[9,10] shows that this type of system can be graphically represented by a System State Diagram. This directed graph can also be viewed as a finite state automaton provided that the number of states is finite. Furthermore, for each possible transition out of a given state, one can attach a probability to the occurrence of this event. We further require that the sum of the probabilities associated with transitions out of a given state sum to one for each state in the system yielding a probabilistic automaton model of the computer system. A specific example and formal definitions follow.

Consider a system composed of two processes, P_1 and P_2, and two units of a resource. The following actions may occur in this example and correspond to transitions of the probabilistic automaton.

A. Process P_i may request a unit of resource, denoted r_i in which case the process is suspended (he waits) until a unit is allocated where $i = 1$ or 2. Process P_i can only request one unit at a time and never request more if all units (two in this case) have already been allocated to process P_i.

B. If process P_i is in a suspended state waiting for a unit of resource the system may allocate a unit to P_i denoted a_i, provided all units of the resource have not already been allocated.

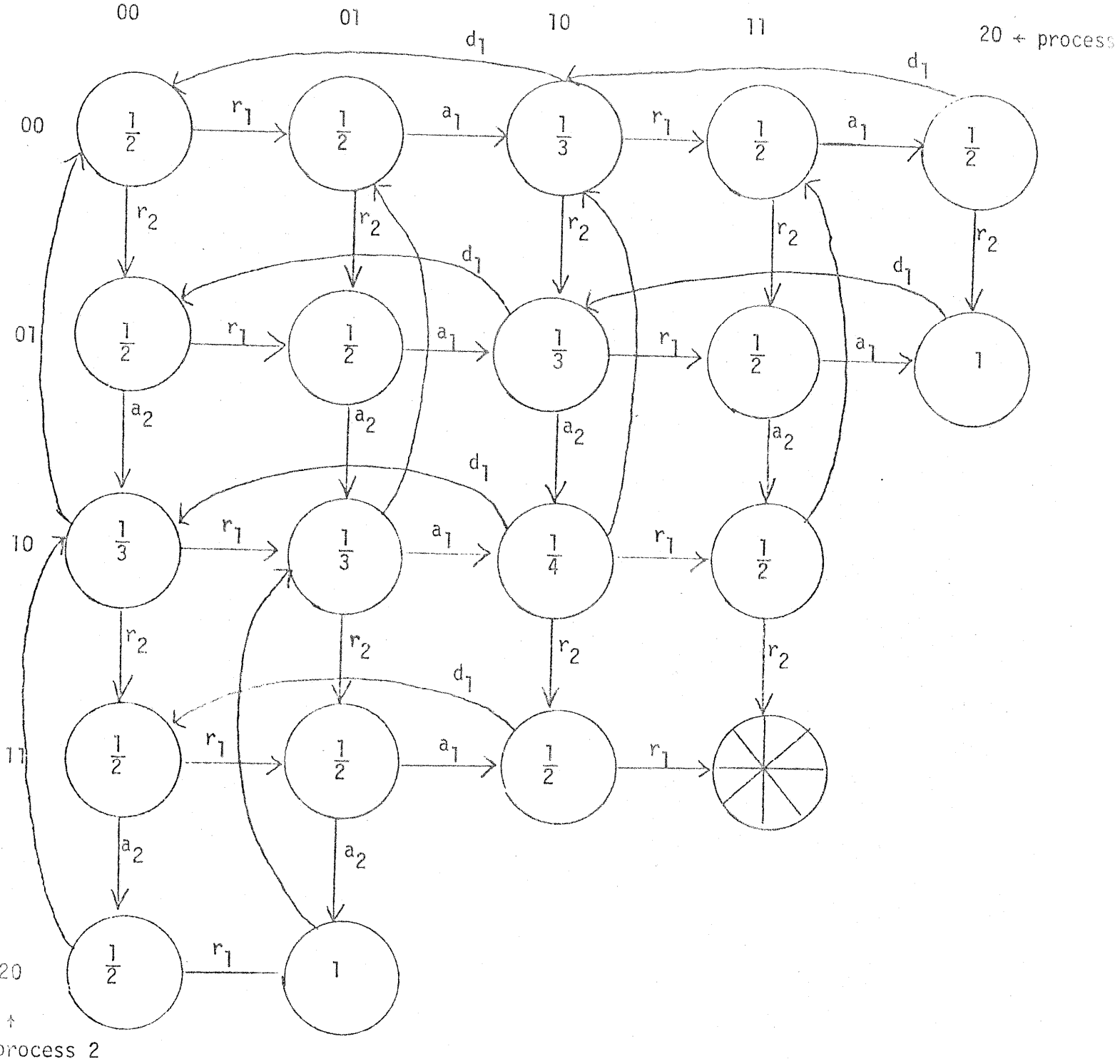


FIGURE 1

A PROBABILISTIC AUTOMATON CORRESPONDING TO A 2 x 2 SYSTEM

C. If process P_i is not in a suspended state, P_i may de-allocate a unit of resource which was previously allocated to him, denoted d_i, implying that the process no longer needs the unit.

Holt[9,10] has considered the System State Diagram for this system. By making some assumptions concerning the probabilities of state changes, a Probabilistic Automaton can be defined from this. Suppose, for example, that without any apriori knowledge of the system we assign equal probabilities to all transitions out of any given state. Then because these equal probabilities must sum to one, we get the automaton represented graphically by Figure 1. Each circle (node) of Figure 1, denoting a possible state of the system, contains the probability of leaving that state. This representation can be used in this case if one makes an equal probability assumption. Thus this number is simply the multiplicative inverse of the number of arcs leaving the node. The upper first node in row 00 and column 00 denotes a state (henceforth designated (00,00)) in which process P_i holds no resources and requests no resources, i = 1, 2. In general, the column labels (row labels) are two digit numbers, the first digit specifying resources held and the second, resources requested by process P_1 (process P_2). The equal probability assumption implicitly imposes a random resource allocation algorithm upon the system because if the system is in a state such that both processes are suspended waiting for a unit of resource, then we randomly choose one or the other. The node in row 01, column 01, of Figure 1 is an example of this. If the system began in the initial state (00,00), (no allocation, no requests), then one way it could get to this state (01,01) is via a request by P_1, $(00,00) \overset{r1}{\rightarrow} (01,00)$ followed by a request by P_2, $(01,00) \overset{r2}{\rightarrow} (01,01)$. Another possibility is that P_2 could first make a request and then P_1, $(00,00) \rightarrow (00,01) \rightarrow (01,01)$.

Thus the arcs are marked to identify the actions of the system (which correspond to transitions of the automaton). The probability of the automaton moving from state (00,00) to (01,01) in two transitions is 1/2, the sum of the probabilities of the paths which allow this to be done:

$$\text{Path 1: } (00,00) \xrightarrow{1/2} (00,01) \xrightarrow{1/2} (01,01), \quad \text{pr(path 1)} = 1/4$$

$$\text{Path 2: } (00,00) \xrightarrow{1/2} (01,00) \xrightarrow{1/2} (01,01), \quad \text{pr(path 2)} = 1/4$$

$$\text{total pr} = 1/2$$

Note that a path is the conjunction of a sequence of system actions, so the path probability is the product of the transition probabilities of the corresponding transitions composing the path. These concepts are formalized in the definitions which follow. Once the system is in the state (01,01), there is a probability of 1/2 of moving to state (10,01) and probability of 1/2 of moving to (01,10) because the node in row (01,01) has arcs to these two states and contains 1/2. This implies that our resource allocation algorithm allocates to process P_1 with probability 1/2 or to process P_2 with probability of 1/2. This random resource allocation model forms a Markov Chain and has been investigated via simulation and via analytic techniques[4]. Two other more realistic algorithms, first-come-first-serve and last-come-first-serve can be modeled by adding an auxiliary storage to our automaton, forming respectively a probabilistic queue automaton and a probabilistic pushdown automaton. In these cases, whenever a request is made by process P_i, the symbol r_i is put into the auxiliary storage, and whenever an allocation is made, the process P_i to whom this allocation is made is determined by the symbol r_i extracted from the storage. This symbol is then discarded.

In the case of a queue, this will be the least recently inserted symbol among the symbols in storage and for a stack, this will be the most recently inserted symbol. The model is formally defined as follows:

Definition: A Probabilistic Language over a vocabulary Σ is a system $\hat{L} = (L,\mu)$ where L is a class of words formed from Σ (we will take this class to be a subset of Σ^*=all finite strings formed from Σ) and μ is a measure on the set L. If μ is a probability measure, then $\hat{L}$ is a Normalized Probabilistic Language.

Definition: A Probabilistic Automaton over a vocabulary Σ is a system $\hat{A} = (K, B, \delta, \alpha, \gamma, b_1)$; K is a finite, nonempty set of states, $s_1, s_2,\ldots,s_n$. B is a finite set of storage tape symbols, $b_1, b_2,\ldots,b_m$. $b_1 \in B$ is the start symbol which initially appears on the storage tape.

$\gamma:s_i \rightarrow R$ is an initial state vector $(\gamma_1, \gamma_2,\ldots,\gamma_n)$ such that $\gamma_i = \gamma(s_i)$ specifies the probability that s_i is the initial state, where R is the set of real numbers.

$\delta:K \times B \times \Sigma \rightarrow R(\delta)$ is a mapping called the transition function whose domain is $K \times B \times (\Sigma \cup \lambda)$. The range of this function $R(\delta)$ determines the specific type of probabilistic automaton defined.

$\alpha:K \times B \times (\Sigma \cup \lambda) \times R(\delta) \rightarrow R$ is a mapping called the transition probability function which associates a real value with each possible map of δ, to designate the probability of that transition.

Several different types of normalization of probabilistic automata have been presented in the literature[15,16]. A classification of normalization types is given in Ellis[2]. In this paper, we investigate the probabilistic

pushdown automaton. The results in this paper also hold for probabilistic queue automata. The appropriate transition function and type of normalization for our application are given next.

A Probabilistic Pushdown Automaton (PPA) is defined by a transition function δ mapping from $K \times B \times (\Sigma \cup \lambda)$ to finite subsets of $K \times B^*$. This automaton is normalized if and only if γ is a stochastic vector and $\sum_{b^{s}_{i}} pr\{a:(s_i,b) \vdash (s_j,\beta)\} = 1 \; \forall \; s_i \in K, b \in B$ where $_{b}s_i$ is the set of all transitions out of state s_i with b as the top symbol of the stack. Thus the sum is over all $a \in \Sigma$, $s_j \in K$, and $\beta \in B^*$ such that $(s_j,\beta) \in \delta(s_i,b,a)$ and the $\vdash$ is interpreted to mean that if s_i is the current state of the automaton and b is currently on the top of the stack, then an input of $a \in \Sigma$ can cause the automaton to change to state s_j and replace the symbol b on the stack by the string β. The probability of this event is specified by the function α. We adopt the convention that the leftmost symbol of β is placed highest on the store (first out) and the rightmost symbol lowest on the store. If $\beta = \lambda$, then this denotes the empty string (i.e., popping the stack). The replacement of $a \in \Sigma$ by λ in one of these transitions, i.e., $(s_j,\beta) \in \delta(s_i,b,\lambda)$, is interpreted as a transition without the input of a symbol.

A configuration of a PPA is a pair (s,β), and we say that a PPA $\hat{A}$ is in configuration (s,β) if $\hat{A}$ is in state s with β on the pushdown store. If for $a_1,a_2,\ldots,a_n$ each in $\Sigma \cup \{\lambda\}$, states $s_1,s_2,\ldots,s_{n+1}$ in K and pushdown strings $\beta_1,\beta_2,\ldots,\beta_{n+1}$ in B^* we have:

$$a_i:(s_i,\beta_i) \overset{P_i}{\underset{\hat{A}}{\vdash}} (s_{i+1}, \beta_{i+1}), \; 1 \leq i \leq n$$

then we write

$$a_1,a_2,\ldots,a_n:(s_1,\beta_1) \overset{p}{\underset{*\hat{A}}{\vdash}} (s_{n+1},\beta_{n+1}).$$

Note that in this notation, (s_i,β_i) now represents a configuration rather than a state, stack top pair and the probability P_i of the transition which causes the configuration change is written over the $\vdash$. The subscript $\hat{A}$ will be dropped from $\overset{P_i}{\underset{\hat{A}}{\vdash}}$ whenever the meaning remains clear. The probability p of a sequence of transitions is the product of the probabilities p_i.

The Probability of Acceptance of a string $w\epsilon\Sigma^*$ is

$$\sum_{i=1}^{n} \gamma_i \sum_{\Gamma_i} pr\{w:(s_i,b_o) \overset{}{\vdash}_* (s_k,\lambda)\}$$

where $|K| = n$ and Γ_i is the set of all sequences starting from s_i which input all of w and terminate with an empty pushdown store. Thus s_k can be any state.

The probabilistic language accepted by a PPA $\hat{A}$ is (L,μ) where L is the set of all w such that $w:(s_1,b)\vdash(s_k,\lambda)$ and $\mu(w)$ = probability of acceptance of w.

As an example of these concepts, consider the following PPA which accepts strings of a's of odd length.

$\hat{A}_1 = (\{s_1,s_2,s_f\},\{b_A,b_B\}, \delta, \alpha, (1,0,0), b_A)$ over $\Sigma = \{a\}$
where δ and α are specified by

$\delta(s_1,b_A,a) = \{(s_2,b_Bb_A), (s_f,\lambda)\}$, $\alpha = (p_1,q_1)$
$\delta(s_2,b_B,a) = \{(s_2,b_Bb_B), (s_2,\lambda)\}$, $\alpha = (p_2,q_2)$
$\delta(s_2,b_A,a) = \{(s_f,\lambda)\}$, $\alpha = 1$

We assume $p_i + q_i = 1 (i=1,2)$ so this automaton is normalized. According to the last two parameters of $\hat{A}$, this automaton starts in state s_1 because $\gamma = (1,0,0)$ and initially contains b_A on the stack. The shortest string accepted by $\hat{A}$ is the singleton a via the transition $a:(s_1,b_A) \vdash^{q_1} (s_f,\lambda)$. The probability of acceptance of this string is $\mu(a) = q_1$. The string a^7 = aaaaaaa can be accepted by two different sequences of transitions each with probability $p_1 p_2^2 q_2^3$.

1.

$a:(s_1,b_A) \vdash^{p_1} (s_2,b_B b_A)$

$a:(s_2,b_B b_A) \vdash^{p_2} (s_2,b_B b_B b_A)$

$a:(s_2,b_B b_B b_A) \vdash^{p_2} (s_2,b_B b_B b_B b_A)$

$a:(s_2,b_B b_B b_B b_A) \vdash^{q_2} (s_2,b_B b_B b_A)$

$a:(s_2,b_B b_B b_A) \vdash^{q_2} (s_2,b_B b_A)$

$a:(s_2,b_B b_A) \vdash^{q_2} (s_2,b_A)$

$a:(s_2,b_A) \vdash^{1} (s_f,\lambda)$

2.

$a:(s_1,b_A) \vdash^{p_1} (s_2,b_B b_A)$

$a:(s_2,n_B b_A) \vdash^{p_2} (s_2,b_B b_B b_A)$

$a:(s_2,b_B b_B b_A) \vdash^{q_2} (s_2,b_B b_A)$

$a:(s_2,b_B b_A) \vdash^{p_2} (s_2,b_B b_B b_A)$

$a:(s_2,b_B b_B b_A) \vdash^{q_2} (s_2,b_B b_A)$

$a:(s_2,b_B b_A) \vdash^{q_2} (s_2,b_A)$

$a:(s_2,b_A) \vdash^{1} (s_f,\lambda)$

Thus (a,q_1) and $(a^7, 2p_1p_2^2q_2^3)$ are members of the probabilistic language generated by this PPA $\hat{A}$. Is this language generated by $\hat{A}$ a normalized probabilistic language? If we set $p_i = q_i = 1/2$ in this example, we find that the probabilistic language generated is <u>not</u> normalized. In the remainder of this paper, we derive a necessary and sufficient condition for a normalized PPA to generate a normalized language, and then using this we calculate an expression for the expected number of transitions of a PPA before it halts. This can be equated to the expected number of actions to deadlock in a computer system. Another calculation will yield the expected number of

occurrences of any $a_i \varepsilon \Sigma$ in the average input string of any PPA $\hat{A}$. This can be equated to the expected number of allocations (or other system actions) to deadlock in a computer system. Before making these calculations, we need to transform a PPA into a one state PPA. An algorithm used in (nonprobabilistic) automata theory[11] will be elucidated and expanded to the probabilistic case in the next section to realize this transformation.

TRANSFORMATIONS ON PROBABILISTIC PUSHDOWN AUTOMATA

Given a PPA $\hat{A} = (K, B, \delta, \alpha, \gamma, b_1)$ we transform it into a one state PPA $\hat{A}' = (\{s\}, B', \delta', \alpha', (1), Z)$ as follows:

Let B' be the set of objects $[s_i\ b_j\ s_k]$ where $s_i, s_k \varepsilon K$, $b_j \varepsilon B$. Also add the symbol Z to the set B' where this new symbol Z is initially on the stack. Let δ' be defined by

1. $\delta'(s, Z, \lambda)$ contains $(s,[s_i\ b_1\ s_j])$ for all $s_i \varepsilon B$, $s_j \varepsilon B$ with its corresponding $\alpha = \gamma_i$. Note that as a model of deadlock, an automaton will usually have a single initial state, that is $\gamma = (100...00)$.
2. Also put the following into δ': $(s,[s_i\ b_j\ s_k],a) \overset{p}{\underset{\hat{A}}{\vdash}} (s,[s_{\ell,1}\ b_{h,1}\ s_{\ell,2}]\ [s_{\ell,2}\ b_{h,2}\ s_{\ell,3}] \ldots [s_{\ell,m}\ b_{h,m}\ s_{\ell,m+1}])$ for each s_i, $s_{\ell,1}$, $s_{\ell,2} \ldots, s_{\ell,m+1}$ in K, where $s_k = s_{\ell,m+1}$, for each a in $\Sigma \cup \{\lambda\}$, and b_j, $b_{h,1} \ldots, b_{h,m}$ in B such that $\delta(s_i, b_j, a)$ contains $(s_k, b_{h,1}\ b_{h,2} \ldots b_{h,m})$ with $\alpha = p$ for this transition.

   If $m = 0$, then $s_k = s_{\ell,1}$, $\delta(s_i, b_j, a)$ contains (s_k, λ) and the transition added to δ' is $(s,\lambda) \varepsilon \delta'(s,[s_i\ b_j\ s_k],a)$. Before justifying this transformation we show an application of these rules to the previous example automaton labeled $\hat{A}_1$. The transformed PPA will be $\hat{A}_1' = (\{s\}, B', \delta', \alpha', (1)$ Z. To construct δ' easily we must realize that some stack symbols may not appear in any valid transition sequence. Thus we can save some effort by starting with the rules for the (s, Z) configuration, then adding rules only

for those stack symbols in B' which are actually generated with positive probability. The rules for (s, Z) are

$$\delta'(s, Z, \lambda) = \{(s,[s_1\ b_A\ s_1]), (s,[s_1\ b_A\ s_2]), (s,[s_1\ b_A\ s_f])\}$$

We did not include the transitions $(s,[s_i\ b_A\ s_j])$ in $\delta'(s, Z, \lambda)$ where $s_i \neq s_1$ because $\gamma_i = 0$ if $i \neq 1$. The probabilities associated with these three transitions are $\alpha = (1,1,1)$. The interpretation of $[s_i\ b_k\ s_j]$ is that we are simulating a multiple state automaton which must somehow make a sequence of transitions from state s_i terminating in state s_j and in the process get rid of b_k which is on the top of the stack. Thus we will find that only the third of the three members of $\delta'(s, Z, \lambda)$ is a viable stack symbol because $\hat{A}_1$ must enter state q_f before it can pop the symbol b_A off of the stack. Using the second rule, we add transitions for $(s,[s_1\ b_A\ s_j])$

$$\delta'(s,[s_1\ b_A\ s_1],a) = \{(s,[s_2\ b_B\ s_1][s_1\ b_A\ s_1]),(s,[s_2\ b_B\ s_2][s_2\ b_A\ s_1]),$$
$$(s,[s_2\ b_B\ s_f][s_f\ b_A\ s_1])\},\ \alpha = (p_1, p_1, p_1)$$
$$\delta'(s,[s_1\ b_A\ s_2],a) = \{(s,[s_2\ b_B\ s_1][s_1\ b_A\ s_2]),(s,[s_2\ b_B\ s_2][s_2\ B_A\ s_2]),$$
$$(s,[s_2\ b_B\ s_f][s_f\ b_A\ s_2])\},\ \alpha = (p_1, p_1, p_1)$$
$$\delta'(s,[s_1\ b_A\ s_f],a) = \{(s,[s_2\ b_B\ s_1][s_1\ b_A\ s_f]),(s,[s_2\ b_B\ s_2][s_2\ b_A\ s_f]),$$
$$(s,[s_2\ b_B\ s_f][s_f\ b_A\ s_f]),\ (s, \lambda)\},\ \alpha = (p_1, p_1, p_1, q_1).$$

Note that some of these stack symbols, i.e., $[s_f\ b_A\ s_i]$, cannot be used further. They are dead-end. Using stack symbols which have thus far occurred we can write:

$\delta'(s,[s_2\ b_B\ s_1],a) = \{(s,[s_2\ b_B\ s_1][s_1\ b_B\ s_1]),(s,[s_2\ b_B\ s_2][s_2\ b_B\ s_1]),$
$(s,[s_2\ b_B\ s_f][s_f\ b_B\ s_1])\}$

$\delta'(s,[s_2\ b_B\ s_2],a) = \{(s,[s_2\ b_B\ s_1][s_1\ b_B\ s_2]),(s,[s_2\ b_B\ s_2][s_2\ b_B\ s_2]),$
$(s,[s_2\ b_B\ s_f][s_f\ b_B\ s_2]),\ (s,\lambda)\}$

$\delta'(s,[s_2\ b_B\ s_f],a) = \{(s,[s_2\ b_B\ s_1][s_1\ b_B\ s_f]),(s,[s_2\ b_B\ s_2][s_2\ b_B\ s_f]),$
$(s,[s_2\ b_B\ s_f][s_f\ b_B\ s_f])\}$

$\delta'(s,[s_2\ b_A\ s_f],a) = \{(s,\lambda)\}$

Since the stack symbols $[s_1\ b_A\ s_1]$,$[s_2\ b_A\ s_1]$,$[s_f\ b_A\ s_1]$,$[s_f\ b_A\ s_2]$, $[s_f\ b_A\ s_f]$, $[s_1\ b_B\ s_1]$,$[s_f\ b_B\ s_1]$,$[s_1\ b_B\ s_2]$,$[s_f\ b_B\ s_2]$,$[s_1\ b_B\ s_f]$, and $[s_f\ b_B\ s_f]$ cannot appear on the right-hand size (they are dead-end) we delete all transitions depending upon them which yields the PPA $\hat{A}_1'$:

$\delta'(s,[s_1\ b_A\ s_f],a) = \{(s,[s_2\ b_B\ s_2][s_2\ b_A\ s_f]),(s,\lambda)\}\ \alpha = (p_1,q_1)$

$\delta'(s,[s_2\ b_B\ s_2],a) = \{(s,[s_2\ b_B\ s_2][s_2\ b_B\ s_2],(s,\lambda)\},\ \alpha = (p_2,q_2)$

$\delta'(s,[s_2\ b_A\ s_f],a) = \{(s,\lambda)\},\ \alpha = (1)$

We next prove that, as can be seen in this example, every string accepted by the original automaton A_1 is also accepted by A_1' with the same probability, so these automata are equivalent in the sense that they accept the same probabilistic language.

Theorem - If $\hat{L}$ is the probabilistic language accepted by some PPA $\hat{A}$, then $\exists\ \hat{A}'$ which is a one state PPA also accepting $\hat{L}$.

Proof - We will show that $w:(s_i,\beta_i) \overset{p}{\underset{*\hat{A}}{\vdash}} (s_k,\lambda)$ iff $w:(s',[s_i\ \ \beta_i\ s_k]) \overset{p}{\underset{*\hat{A}'}{\vdash}} (s',\lambda)$. Even stronger, the probability of acceptance by $\hat{A}'$ of any $w\in\Sigma^*$ is equal to its probability of acceptance by $\hat{A}$.

Using induction on the number of transitions of the automaton, suppose $w:(s_i, \beta_i) \overset{p}{\underset{\hat{A}}{\vdash}} (s_k,\lambda)$ in one transition. This implies w is a string of length one (assuming no λ transitions) and so is the string β of stack symbols. Then our algorithm implies there must be a rule of $\hat{A}'$ of the form $(s', \lambda) \in \delta'(s',[s_i\ \beta_i\ s_k])$, and its probability must be the same under α' as that under α. Thus $w:(s',[s_i\ \beta_i\ s_k]) \overset{p}{\underset{\hat{A}'}{\vdash}} (s',\lambda)$ Now assume the hypothesis is true for any process of up to k-1 transitions. To show it is true for k transition processes, we assert that a first transition (and there may be many) from a given (s_i, β_i) must be of the form $a:(s_i, b) \overset{p_i}{\underset{\hat{A}}{\vdash}} (s_j, b_{\ell 1}b_{\ell 2}\ldots b_{\ell m})$ where a must be the first symbol of $w(a\overline{w}=w)$ and b must be the first symbol of $\beta(bb_1b_2\ldots b_n = \beta$ This must be followed by a set of transition sequences of length k-1 specified by $w:(s_j, b_\ell\ b_{\ell 2}\ldots b_{\ell m}b_1b_2\ldots b_n) \overset{p}{\underset{\hat{A}}{\overset{*}{\vdash}}} (s_k,\lambda)$. By hypothesis, there exists a set of transition sequences in $\hat{A}'$ (one for each way in which $\hat{A}$ accepts $\overline{w}$) whose sum adds to p and each of which can be written as $\overline{w}:(s',[s_j\ b_{\ell 1}\ s_{\ell 1}]$ $[s_{\ell 1}\ b_{\ell 2}\ s_{\ell 2}]\ldots[s_{\ell m-1}\ b_{\ell m}\ s_{\ell m}]\ldots[s_{n-1}\ b_n\ s_n]) \overset{p_i}{\underset{\hat{A}'}{\overset{*}{\vdash}}} (s',\lambda)$. Combining this with the rule $a:(s',[s_ibs_{\ell m}]) \overset{p_i}{\underset{\hat{A}'}{\vdash}} (s',[s_jb_{\ell 1}s_{\ell 1}][s_{\ell 1}b_{\ell 2}s_{\ell 2}]\ldots[s_{\ell m-1}b_{\ell m}s_{\ell m}])$, which is implied by the corresponding rule in $\hat{A}$, we get the desired result. A similar induction argument verifies the only if portion of this proof. Furthermore, the initial probabilities of $\hat{A}$ defined by γ are simulated by the initial transitions from the initial situation (s',Z) of $\hat{A}'$, so the probability of acceptance of a string w is the same within the two automata.

CALCULATIONS ON PROBABILISTIC PUSHDOWN AUTOMATA

Although a pushdown automaton processes its stack symbols from top to bottom (from left to right in our notation) it can be considered to process all of its stack simultaneously within a one state automaton because we know apriori which state to associate with each stack symbol. This parallel processing can be conceptualized as occurring in steps as follows. The zeroth

stack step will correspond to the initial stack symbol Z. The first stack step will correspond to whatever string β_1 of stack symbols is generated by (s',Z). The second step β_2 will correspond to the stack after the application of one transition to (s',b) for every b in β_1. Similarly, the ith stack step will correspond to the string β_i of stack symbols obtained by applying transitions to all stack symbols of β_{i-1}. Figure 2 shows a transition sequence for accepting a^7 expressed in parallel using stack steps of the example one state automaton $\hat{A}_1'$. In this figure we denote stack symbols $[s_1\ Z_1\ s_f]$, $[s_2\ b_2\ s_2]$, and $[s_2\ b_A\ s_f]$ by b_1, b_2, and b_3 respectively. Define the transition generating function for each $b_j \in B$ (where $|B| = k$) as $f_j(x_1, x_2, \ldots, x_k) = \sum_{i=1}^{m_j} \alpha\{a_{i,j}:(s, b_j) \overset{\hat{A}}{\vdash} (s, \beta_{ij})\} x_1^{r_1(\beta_{ij})} \ldots x_k^{r_k(\beta_{ij})}$ where the α function is the probability associated with the transition, m_j denotes the number of distinct transitions possible out of s with b_j on the top of the stack, $\hat{A}$ is a one state PPA, and $r_\ell(\beta_{ij})$ denotes the number of occurrences of b_ℓ within the string β_{ij}. For example, $\hat{A}_1'$ has the following transition generating functions:

$$f_1(x_1, x_2, x_3) = p_1 x_2 x_3 + q_1$$
$$f_2(x_1, x_2, x_3) = p_2 x_2^2 + q_2$$
$$f_3(x_1, x_2, x_3) = 1$$

Define the ith step transition generating function recursively as:

$$F_0(x_1, x_2, \ldots, x_k) = x_1$$
$$F_1(x_1, x_2, \ldots, x_k) = f_1(x_1, x_2, \ldots, x_k)$$
$$F_i(x_1, x_2, \ldots, x_k) = F_{i-1}(f_1(x_1, x_2, \ldots, x_k), f_2(x_1, x_2, \ldots, x_k), \ldots, f_k(x_1, x_2, \ldots, x_k)) \text{ for } i > 1.$$

Thus in our example,

$$F_0(x_1, x_2, x_3) = x_1$$
$$F_1(x_1, x_2, x_3) = p_1 x_2 x_3 + q_1$$
$$F_2(x_1, x_2, x_3) = p_1 p_2 x_2^2 + p_1 q_2 + q_1$$

FIGURE 2

AUTOMATON $\hat{A}_1'$

$\delta'(s, b_1, a) = \{(s, b_2b_3), (s,\lambda)\}$, $\alpha_1 = (p_1, q_1)$

$\delta'(s, b_2, a) = \{(s, b_2b_2), (s,\lambda)\}$, $\alpha_2 = (p_2, q_2)$

$\delta'(s, b_3, a) = \{(s,\lambda)\}$, $\alpha_3 = (1)$

	Stack Steps	Input Accepted
β_0	b_1	
β_1	b_2b_3	a_
β_2	b_2b_2 λ	aa_a
β_3	λ b_2b_2	aaaa_a
β_4	λ λ	aaaaaaa

An acceptance sequence for a^7

In each of these expressions, the constant term of F_i denotes the total probability of strings being accepted in i or less stack steps; also the coefficient of the term $x_1^{r_1} x_2^{r_2} \ldots x_k^{r_k}$ in F_i denotes the probability of exactly r_1 occurrences of b_1, r_2 occurrences of $b_2, \ldots, r_k$ occurrences of b_k in the β_i string of the ith stack step. In a normalized PPA, this implies that the limit as $i \to \infty$ of the constant portion of F_i must approach one and the r_j exponents must approach zero. The matrix M of values $\left.\frac{\partial f_i}{\partial x_j}\right|_{x_1, x_2, \ldots, x_k = 1}$ exactly represents the expected number of occurrences of the stack symbol b_j in the set of transitions out of (s, b_i), whereas $\frac{\partial F_i}{\partial x_j}$ represents the expected value at the ith stack step. Thus a normalized PPA must satisfy the criterion $\lim_{i \to \infty} M^i = 0$ in order for the language generated to be normalized. The matrix M will satisfy this criterion if the magnitude of all characteristic roots are less than one. Similarly, if any roots have a magnitude greater than one, the limit diverges. Consider our example $\hat{A}_1'$. The matrix of partial derivatives looks like:

$$M = \begin{bmatrix} 0 & p_1 & p_1 \\ 0 & 2p_2 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

The characteristic equation associated with M is:

$$\phi(y) = y^2(y - 2p_2)$$

Thus this PPA will accept a normalized probabilistic language as long as $p_2 < 1/2$.

For any PPA $\hat{A}$, we can define the expected number of transitions assuming no input = λ transitions, to be $E = \sum_{w \in L} \ell(w)\, \mu(w)$ where L is the set of strings accepted with positive probability by $\hat{A}$, $\ell(w)$ is the length of w, i.e., number of symbols $a \in \Sigma$ in w, and $\mu(w)$ is the probability of acceptance of w by $\hat{A}$.

We can show that if $\hat{A}$ is normalized and its matrix M eigenvalues are less than one, then

$$E = (100\ldots00)\ (I - M)^{-1}\ (\vec{1}\ ).$$

This is true because the average length of all strings accepted by $\hat{A}$ with b_j as the initial symbol is

$$n_j = \sum_{u=1}^{m_j} \alpha_{j,u}\ (1 + \sum_{i=1}^{k} n_i h_i(\beta_{j,u}))$$

where h_i is the number of occurrences of b_i in $\beta_{j,u}$, and $\alpha_{j,u}$ is the probability p associated with the uth transition out of (s,b_j), i.e., $a_{uj}:(s, b_j) \vdash^{p} (s, \beta_{j,u})$.

$$n_j = \sum_{i=1}^{k} \sum_{u=1}^{m_j} \alpha_{j,u}\ h_i(\beta_{j,u}))\ n_i + \sum_{u=1}^{m_j} p_{j,u}$$

By normalization criteria, the second term equals one. The term $\sum_{u=1}^{m_j} \alpha_{j,u}\ h_i(\beta_{j,u})$ is the (j,i) term of the matrix M, thus if $(\vec{1})$ denotes a column vector of all ones, we have $N = MN + (\vec{1})$. Solving for N yields $N = (I-M)^{-1}\ (1)$ where $(I-M)^{-1}$ exists because M has eigenvalues all less than unity. Premultiplying by (100...00) removes all terms except n_1 which is the desired value E.

The expected relative number of occurrences of $a_i \varepsilon \Sigma$ in strings of (L,μ) denoted E_i, can also be calculated. Let $t_i(\beta_{j,u})$ denote the number of occurrences of a_i in $\beta_{j,u}$ of $a_{uj}:(s,b_i) \vdash^{p} (s,p_{j,u})$. Then let T_i be the column vector whose jth element is

$$\sum_{u=1}^{m_j} \alpha_{j,u}\ t_i\ (\beta_{j,u})$$

This term is interpreted as the mean number of a_i's accepted by $\hat{A}$ in one transition with b_j on the top of the stack. From the calculation of E we see that

$$(100...00)\ [I-M]^{-1}\ T_i$$

yields the average number of occurrences of a_i. Division by E makes this value relative since E is the average number of input symbols in a string of L. Thus

$$E_i = (100...00)\ [I-M]^{-1}\ T_i/E.$$

SUMMARY AND CONCLUSIONS

With the methods described in this paper, it is possible to investigate and answer questions concerning probability of deadlock in various models of computer systems, where this deadlock was defined as the expected value of 1) the number of system actions to deadlock or 2) the number of resource allocations before deadlock. All techniques used in this paper were applied to a pushdown automaton modeling last-come-first-serve scheduling. It is significant that, because we considered parallel stack reduction, our techniques apply equally well to queue automata modeling first-come-first-serve scheduling algorithms. This is because in our generating functions, the order of reduction of the stack (or queue) symbols is immaterial at each stack step.

The calculations described in this paper have been carried out for systems containing small numbers of processes and resources. Results of these calculations appear in the appendices. For example, the 2 x 2 system of Figure 1 described earlier has a mean time to deadlock of 24.72 system actions under first-come-first-serve and last-come-first-serve scheduling, but 25.50 under random allocation.

If the number of units of resource increases while the number of processes remains fixed, one would intuitively expect that the probability of deadlock would decrease. This expectation is substantiated by the data presented in appendix B indicating that as the number of resources increases from 2 to 3,

to 4,..., the expected number of transitions to deadlock increases implying that the system would run for a longer period of time in a deadlock free mode, so the probability of deadlock is smaller. Conversely, increasing the number of processes in a fixed resource system should increase the probability of deadlock because more processes would be competing for the same number of resources. Appendix C confirms this hypothesis for small systems, indicating that an increase in processes decreases the expected number of transitions to deadlock. This phenomenon has, in fact, been observed in actual systems [12]. If one uniformly increases both processes and resources, it is intuitively unclear what happens to the probability of deadlock. Appendix A indicates that as the number of processes (= the number of units of resource) increases, the expected number of transitions to deadlock decreases implying that the probability of deadlock actually increases. This study has been concerned with models of generic process-resource systems; if a specific system was being modeled, one could change the transition probabilities to more realistically reflect the system's behavior. The mean time between relevant system actions could also be obtained in order to translate number of transitions to deadlock into time units.

Areas of further research include extension of this model to systems containing different classes of resources and to consumable resource systems. Also, it could be hypothesized that as the system becomes large, the expected number of transitions to deadlock under the three scheduling algorithms would differ by a negligible amount. Is this true? Can one bound this difference as a function of the number of processes and resources? If not, then perhaps it can be proven that one of the scheduling algorithms always has the lowest probability of deadlock.

REFERENCES

1. Coffman, E. G., Jr., Elphick, M. J., and Shoshani, A., "System Deadlocks," Computing Surveys, Vol. 3, No. 2, Pages 67-78, June, 1971.

2. Ellis, C. A., "Probabilistic Languages and Automata," Ph.D. Dissertation, Computer Science Department, University of Illinois, 1969.

3. Ellis, C. A., "The Halting Problem for Probabilistic Generators," Presented at the Fourth Annual Princeton Conference on Information Sciences and Systems (also in the Journal of the ACM, Vol. 19, No. 3, July, 1972.)

4. Ellis, C. A., "On the Probability of Deadlock in Computer Systems" presented at the Fourth Symposium on Operating Systems Principles October 15-17, 1973.

5. Fontao, R. O., "A Concurrent Algorithm for Avoiding Deadlocks in Multiprocess Multiple Resource Systems," Proceedings of the Third Symposium on Operating Systems Principles, October, 1971.

6. Habermann, A. N., "Prevention of System Deadlocks," Communications of the ACM, Vol. 12, No. 7, Pages 373-377, July, 1969.

7. Havender, J. W., "Avoiding Deadlock in Multitasking Systems," IBM Systems Journal, Vol. 7, No. 2, Pages 74-84, 1968.

8. Hebalkar, P. G., "Deadlock-free Resource Sharing in Asynchronous Systems," Ph.D. Dissertation, Electrical Engineering Department, MIT, Cambridge, Massachusetts, September, 1960.

9. Holt, R. C., "On Deadlock in Computer Systems," Technical Report CSRG-6, University of Toronto, July, 1972 (also available as Ph.D. Dissertation, Department of Computer Science, Cornell University, Ithaca, New York, January, 1971).

10. Holt, R. C., "Some Deadlock Properties of Computer Systems," Computing Surveys, Vol. 4, No. 3, Pages 179-196, September, 1972.

11. Hopcroft, J. E., and Ullman, J. D., Formal Languages and their Relation to Automata, Addison-Wesley, 1969.

12. Hoare, C.A.R. and Gerrott, R. H., "Operating Systems Techniques", Academic Press, 1972.

13. Murphy, J. E., "Resource Allocation with Interlock Detection in a Multi-task System," Proceedings of the AFIPS FJCC, 1968, Vol. 33, Part II, Pages 1169-1176.

14. Nutt, G. J., "Some Applications of Finite State Automata Theory to the Deadlock Problem," Report CU-CS-017-73, University of Colorado, April, 1973.

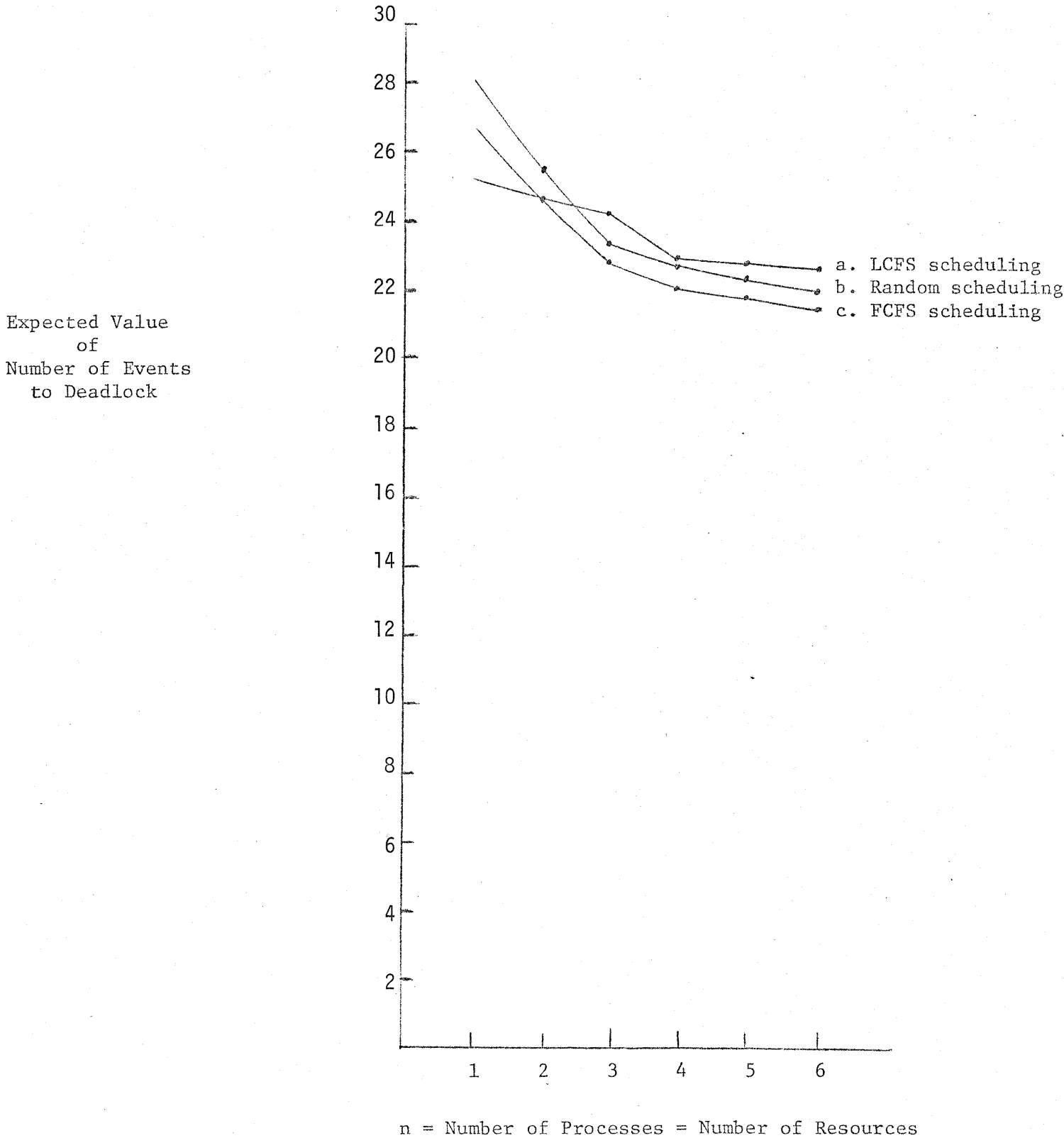
15. Paz, A., Introduction to Probabilistic Automata, Academic Press, 1971.

16. Rabin, M. O., "Probabilistic Automata," Information and Control, Vol. 6, No. 3, Pages 230-245.

CAE:cah

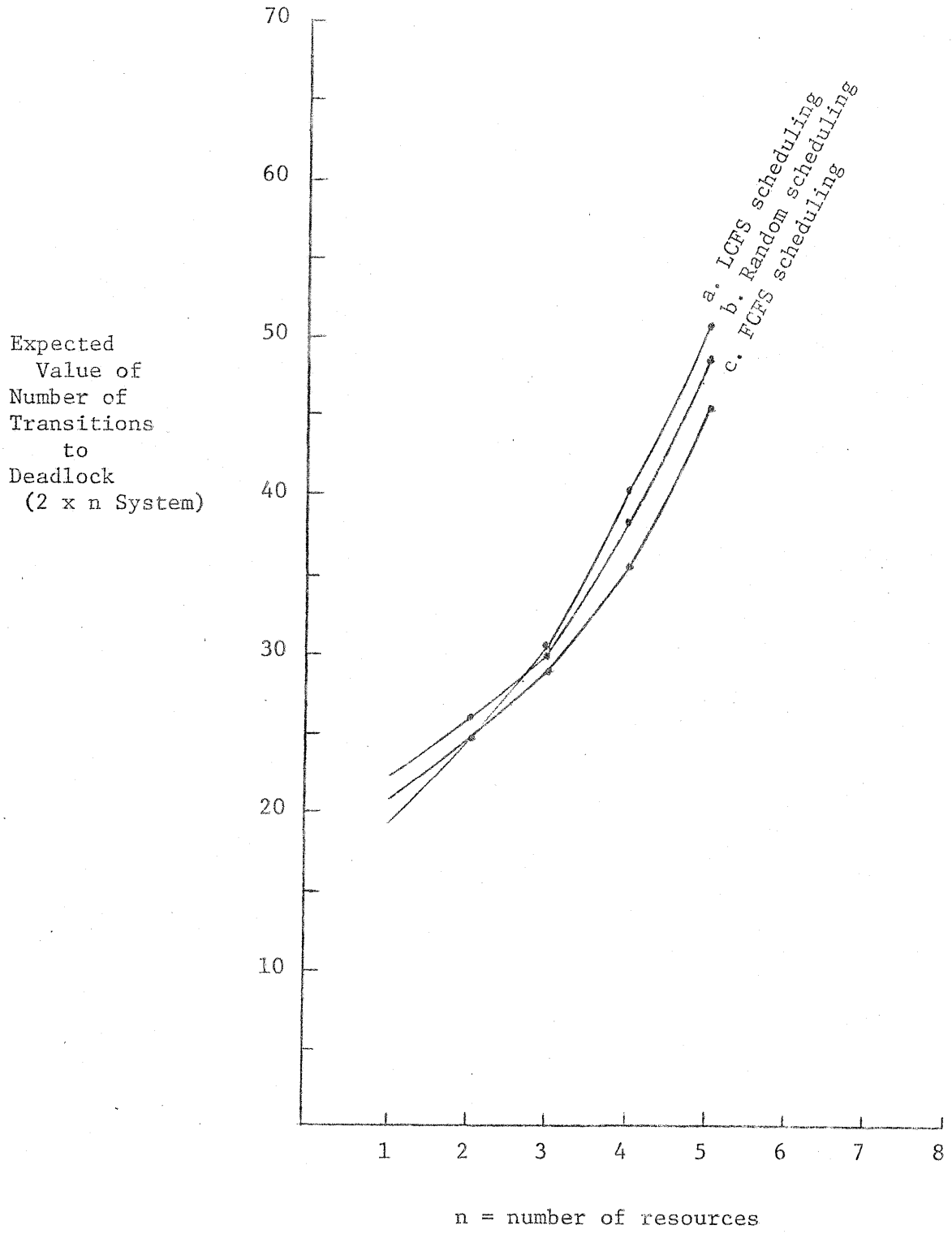
APPENDIX A

MEAN NUMBER OF TRANSITIONS TO DEADLOCK FOR n x n SYSTEM



APPENDIX B

MEAN NUMBER OF TRANSITIONS TO DEADLOCK FOR 2 PROCESS SYSTEMS



APPENDIX C

MEAN NUMBER OF TRANSITIONS TO DEADLOCK FOR 2 RESOURCE SYSTEMS