Palo Alto Research Center

On the Equivalence of Office Models

By Clarence A. Ellis and Gary J. Nutt

XEROX

On the Equivalence of Office Models

BY Clarence A. Ellis AND Gary J. Nutt

December 1979

SSL-79-8

XEROX
PALO ALTO RESEARCH CENTER
3333 Coyote Hill Road / Palo Alto / California 94304

Office Research Group, PARC

ABSTRACT

Office information flow studies have led to a variety of notations, including a class which emphasizes forms flow and another class which emphasizes work station activity. Questions of the relative descriptive power of these notations have arisen. This paper addresses that issue by defining two sets of abstract models which, between them, encompass many of these models. We show that these sets of models are equivalent in their expressive power. However, transformations of models from one type to the other lead to computationally complex (i.e. NP-Complete) algorithms; we also show how reasonable restrictions on the classes of models lead to transformation algorithms having realistic (i.e. polynomial) computational times.

KEY WORDS AND PHRASES

Office modeling, model equivalence, NP-complete algorithms, automata, network flow models, computational complexity

On the Equivalence of Office Models

INTRODUCTION

The perspective of the user of an office model may cause him or her to characterize the office in different ways. Higher level managers often want to see a more abstracted view of the total business than low level managers who may want to see a detailed model of a subset of the total business. A clerk may want to see only a view of his or her portion of the work. In general, an office information system must serve a diverse clientele of customers from the chief executive officer to the secretary, so flexibility is necessary. If one is concerned about whether an order for goods can ever get to the shipping department without having been routed through billing, then a forms-oriented view is preferable; if one is wondering about equitability of workload, then a people-oriented view is most appropriate.

Within offices, one can describe the processing of information by describing a set of "tasks" with precedence constraints. The precedence ordinarily specifies control flow, (e.g. see [ELLI79]); however, it may show data flow, (e.g. see [HAMM77]). The diagrams can be embellished by attaching other information, depending upon the purpose of the model. For example the tasks may be partially interpreted in order to specify service time distributions, (cf. queueing networks [KIEN79]); a supplementary graph may identify domains and ranges for tasks, (cf. precedence graphs in [COFF73]). We refer to this class of models as *precedence-oriented* models.

Another class of models has been oriented toward the specific processing performed at a work station within the office; a model from this class would emphasize the set of tasks that can be performed at the work station at any given time, (e.g. see [BAUM80, NESS77]). These models explicitly assume that only one task from the collection of tasks delegated to a processor can be executed at a time (i.e., the processors -- human or machine -- are thought of as sequential machines). Models within this category are called *state-oriented* models.

We conjecture that if one knows complete information about each work station's information processing functions, then one also knows complete information about the processing of each transaction, and vice versa. In this paper we consider algorithms to convert from a model of one type to a model of the other type. The algorithm to convert precedence to state models, and all others that perform the same task, may result in an exponential explosion of states as a function of the number of tasks in a precedence model. A subclass of the precedence-oriented models is then introduced which can be transformed into equivalent state-oriented models in less than an exponential number of steps.

DEFINITIONS

Precedence-oriented Models

Models which are oriented toward the processing of a form tend to specify the precedence of operations on that form (as opposed to the precedence of operations by the processors of the form). For example if a form must have 4 fields filled in, and the order in which some of them are filled-in is important, then the precedence of fill-in activities can be specified by the model. Thus field a may have to be filled in before fields b and c, and field d cannot be filled until both b and c have been completed. The precedence of the four fill-in activities is $a<b$, $a<c$, $b<d$, and $c<d$.

It is possible to formally define a precedence-oriented model as a directed graph, in order to show formal properties of such models (e.g., consider the "UCLA model" [RAZO79]). In the following definitions, $\mathcal{P}(x)$ denotes the power set of x.

Definition 1: A *precedence model* is a bilogic, directed graph,

(A,P,π,ψ,α)

where

A = a finite, nonempty set of *activity* nodes,

P = a finite, nonempty set of *office operators*,

$\pi = \pi_i \cup \pi_o$

π_i: $A \rightarrow \mathcal{P}(A)$

π_o: $A \rightarrow \mathcal{P}(A)$

ψ: $A \rightarrow \{*,\oplus\}$

α: $A \rightarrow P$ activity assignment to office operators

$\exists! a_0 \ni \pi_i(a_0) = \emptyset$

$\exists! a_n \ni \pi_o(a_n) = \emptyset$

The flow of control in the model is described by A, π, and ψ. If a given activity, x, has AND logic, then $\psi(x) = *$; exclusive OR logic is represented by $\oplus$. Activity x with $\psi(x) = *$ can be initiated iff each activity in $\pi_i(x)$ has terminated; when x terminates, each activity in $\pi_o(x)$ is signalled that x has terminated. Activity x with $\psi(x) = \oplus$ can be initiated if any activity in $\pi_i(x)$ has terminated; when x terminates, exactly one activity in $\pi_o(x)$ is signalled that x has terminated. There exists one entry point and exactly one exit point. The α mapping represents the assignment of activities to office operators, (i.e., workstations or people).

Definition 2: A *transaction* in the model is the set of activity executions which result from the initiation of the entry activity; each transaction begins with the initiation of the entry activity, and ends with the termination of the exit activity. No two activities are allowed to initiate at exactly the same time. Thus for the processing of a transaction, t, there exists an *activity sequence*

$\omega(t) = a_0 a_1 \dots a_u$

listing the order in which activites were initiated in the processing of the transaction.

Figure 1 is a diagram of a precedence model. Activities are represented by circles, precedence by arcs, node logic by a symbol adjacent to the node, and operator assignment by an integer adjacent to a node. For example, activity a_1 has been assigned to operator 1 and it has AND logic; activities a_2 and a_3 are enabled when a_1 terminates. An activity sequence for the model might be

$a_1 a_2 a_6 a_3 a_7 a_8 a_9$

The formal representation of the model is given as:

$A = \{a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9\}$
$P = \{1,2,3\}$

$\pi_i(a_1) = \emptyset$	$\pi_o(a_1) = \{a_2, a_3\}$	$\psi(a_1) = *$	$\alpha(a_1) = 1$
$\pi_i(a_2) = \{a_1\}$	$\pi_o(a_2) = \{a_4, a_6\}$	$\psi(a_2) = \oplus$	$\alpha(a_1) = 3$
$\pi_i(a_3) = \{a_1\}$	$\pi_o(a_3) = \{a_9\}$	$\psi(a_3) = *$	$\alpha(a_3) = 2$
$\pi_i(a_4) = \{a_2\}$	$\pi_o(a_4) = \{a_5\}$	$\psi(a_2) = *$	$\alpha(a_4) = 3$
$\pi_i(a_5) = \{a_4\}$	$\pi_o(a_5) = \{a_7\}$	$\psi(a_5) = *$	$\alpha(a_5) = 3$
$\pi_i(a_6) = \{a_2\}$	$\pi_o(a_6) = \{a_7\}$	$\psi(a_6) = *$	$\alpha(a_6) = 1$
$\pi_i(a_7) = \{a_5, a_6\}$	$\pi_o(a_7) = \{a_8\}$	$\psi(a_7) = \oplus$	$\alpha(a_7) = 2$
$\pi_i(a_8) = \{a_7\}$	$\pi_o(a_8) = \{a_9\}$	$\psi(a_8) = *$	$\alpha(a_8) = 2$
$\pi_i(a9) = \{a_3, a_8\}$	$\pi_o(a_9) = \emptyset$	$\psi(a_9) = *$	$\alpha(a_9) = 1$

State-oriented Models

The person-oriented view of information processing in the office also incorporates precedence, but the emphasis is on the set of things that an operator can be doing at any given time. An implicit assumption is that a person can do only one nontrivial activity at any given time. Thus, an intuitive model of the processing done by any individual is the finite state machine.

Definition 3: A *state model* is

$\{M_i | 1 \leq i \leq n\}$

where

$M_i = (S_i, s^i_0, I_i, O_i, \delta_i, \gamma_i)$
such that

S_i = a finite, nonempty set of *states*,

s^i_0 = the initial state,

Z = a finite, nonempty *universal alphabet* containing the null symbol λ,

$I_i = Z^n$,

$O_i = Z^n$,

$\delta_i: S_i \times I_i \rightarrow P(S_i)$

$\gamma_i: S_i \times I_i \rightarrow P(O_i)$

Thus, a state model is made up of a collection of n finite state machines, each of which reads its inputs from an n track tape, I_i. Each machine also writes an n track tape, O_i. Machine M_i changes state either by Λ-moves, i.e., the input tape is ignored, $(\lambda,\lambda, ...,\lambda)$, or by the current tuple input. A Λ-move can take place in M_i from state s_j to s_k iff

$\delta_i(s_j,(\lambda,\lambda,...\lambda)) = s_k$.

A move in M_i from state s_r to s_k based on inputs to M_i from M_j is possible iff

$\delta_i(s_r,(\lambda,\lambda,...,u,...\lambda)) = s_k$,

where the "u" is in the jth position of the tuple. M_i can write a "v" output to M_j iff

$\gamma_i(s_r,(...)) = (\lambda,\lambda,...,v,...\lambda)$,

where the "v" is in the jth position of the tuple. It is also apparent that the machine need not necessarily produce output on each atomic move. Thus one track of each automaton's input (output) tape leads from (to) each other automaton, see Figure 2. An output on track j of M_i appears as an input on track i of M_j; this is simply a relabeling of track numbers so that each track is labeled with the automaton that it is connected to. Position i in the input and output tuples represents external inputs and outputs to machine M_i. Any single machine may "process" an external input, provided that its transition function is appropriately defined; similarly, any machine may write external output provided that its output function is appropriately defined. The class of automata is constrained so that exactly one automaton can receive external input, and one automaton can write an external output.

Definition 4: A *transaction* starts with exactly one external input to any machine and ends when any one machine produces an external output; no two transitions occur simultaneously. Thus for any transaction, t, there exists a *state sequence*

$\sigma(t) = s^i_0 s^j_1 \ldots s^k_v$

listing the order in which transitions occur.

Figure 3 shows two state-oriented models; they describe similar tasks, except that the model in 3a presumes that all subtasks are done by the same operator, while that in 3b presumes that three different automata model three different operators. The model shown in Figure 3b is formally

defined by:

$\{M_1,M_2,M_3\}$ where

$S_1 = \{s^1{}_0,a_1,a'_1,a_6,a'_6,a_9\}$

$S_2 = \{s^2{}_0,a_3,a'_3,a''_3,a_7,a'_7,a_8,a'_8,s^2{}_n\}$

$S_3 = \{s^3{}_0,a_2,a_4,a_5,a'_5\}$

$Z = \{\lambda,r,s,u,v,w,x,y,z\}$

$\delta_1(s^1{}_0,(r,\lambda,\lambda)) = \{a_1\}$	$\gamma_1(s^1{}_0,(r,\lambda,\lambda)) = \{\Lambda\}$
$\delta_1(a_1,\Lambda) = \{a'_1\}$	$\gamma_1(a_1,\Lambda) = \{(\lambda,u,v)\}$
$\delta_1(a'_1,(\lambda,\lambda,w)) = \{a_6\}$	$\gamma_1(a'_1,(\lambda,\lambda,w)) = \{\Lambda\}$
$\delta_1(a'_1,(\lambda,xy,\lambda)) = \{a_9\}$	$\gamma_1(a'_1,(\lambda,xy,\lambda)) = \{\Lambda\}$
$\delta_1(a_6,\Lambda) = \{a'_6\}$	$\gamma_1(a_6,\Lambda) = \{(\lambda,z,\lambda)\}$
$\delta_1(a'_6,(\lambda,xy,\lambda)) = \{a_9\}$	$\gamma_1(a'_6,(\lambda,xy,\lambda)) = \{\Lambda\}$
$\delta_2(s^2{}_0,(u,\lambda,\lambda)) = \{a_3\}$	$\gamma_2(s^2{}_0,(u,\lambda,\lambda)) = \{\Lambda\}$
$\delta_2(s^2{}_0,(z,\lambda,\lambda)) = \{a_7\}$	$\gamma_2(s^2{}_0,(z,\lambda,\lambda)) = \{\Lambda\}$
$\delta_2(s^2{}_0,(\lambda,\lambda,s)) = \{a_7\}$	$\gamma_2(s^2{}_0,(\lambda,\lambda,s)) = \{\Lambda\}$
$\delta_2(a_3,(z,\lambda,\lambda)) = \{a'_7\}$	$\gamma_2(a_3,(z,\lambda,\lambda)) = \{(x,\lambda,\lambda)\}$
$\delta_2(a_3,(\lambda,\lambda,s)) = \{a'_7\}$	$\gamma_2(a_3,(\lambda,\lambda,s)) = \{(x,\lambda,\lambda)\}$
$\delta_2(a'_3,\Lambda) = \{a'_8\}$	$\gamma_2(a'_3,\Lambda) = \{(x,\lambda,\lambda)\}$
$\delta_2(a''_3,\Lambda) = \{s^2{}_n\}$	$\gamma_2(a''_3,\Lambda) = \{(x,\lambda,\lambda)\}$
$\delta_2(a_7,\Lambda) = \{a_8\}$	$\gamma_2(a_7,\Lambda) = \{\Lambda\}$
$\delta_2(a_7,(u,\lambda,\lambda)) = \{a'_3\}$	$\gamma_2(a_7,(u,\lambda,\lambda)) = \{\Lambda\}$
$\delta_2(a'_7,\Lambda) = \{a'_8\}$	$\gamma_2(a'_7,\Lambda) = \{\Lambda\}$

$$\delta_2(a_8,\Lambda) = \{a''_3\} \qquad \gamma_2(a_8,\Lambda)) = \{(y,\lambda,\lambda)\}$$

$$\delta_2(a'_8,\Lambda) = \{s^2{}_n\} \qquad \gamma_2(a'_8,\Lambda)) = \{(y,\lambda,\lambda)\}$$

$$\delta_3(s^3{}_0,(v,\lambda,\lambda)) = \{a_2\} \qquad \gamma_3(s^3{}_0,(v,\lambda,\lambda)) = \{\Lambda\}$$

$$\delta_3(a_2,\Lambda) = \{a'_5,a_4\} \qquad \gamma_3(a_2,\Lambda) = \{(w,\lambda,\lambda),\Lambda\}$$

$$\delta_3(a_4,\Lambda) = \{a_5\} \qquad \gamma_3(a_4,\Lambda) = \{\Lambda\}$$

$$\delta_3(a_5,\Lambda) = \{a'_5\} \qquad \gamma_3(a_5,\Lambda) = \{(\lambda,s,\lambda)\}$$

Communication Protocol

Any model must either assume implicit communication or explicit communication between office operators. This decision depends upon the purpose of modeling, and the level of modeling detail. (Explicit communication means that there is a separate activity denoting communication whenever synchronization among operators occurs.) A reasonable comparison should consider similar levels of detail in all models; so in this paper we assume implicit communication. The explicit case is subsumed in this analysis, and the explicit model can be obtained by adding another activity for each message transmittal and receipt. Within the precedence oriented model, a message sent from operator 1 during activity a to operator 2 received in activity b is modeled as a precedence requirement that activity a preceed b. Within the state oriented model, communication is modeled as input and output transition functions as specified above. We adopt the conventions that an automaton M_i can simultaneously send to other automata (say j and k) by placing an identifier onto tracks j and k of its output tape. Similarly, automaton M_i can change state upon receipt of input from both j and k by specifying within its δ function entries in positions j and k of its input tape.

Equivalence of Models

There are many ways in which one can consider the equivalence of two models. The two models may represent the same system, but with different perspectives. For example, one model illustrates the steps in processing a transaction, while the other illustrates the tasks of an office worker. This type of equivalence is called *representational equivalence.* The two models may actually represent different systems which accomplish "the same information processing." In this case, the archives and resulting forms from some initial activity may be the same in the two systems, while the intermediate processing of the transaction may be different. This type of equivalence is called *functional equivalence.* Representational equivalence is useful in generating differing views of the same system, while functional equivalence is useful in determining that a variant of a system still performs the same overall function as the original.

Definition 5: Let $X=(A,P,\pi,\psi,\alpha)$ be a precedence model and let $Y=\{m_i \mid 1 \leq i \leq n\}$ be a state model. X and Y are *representationally equivalent* (*R-equivalent*) iff:

i) $|P| = n$

ii) For every activity sequence $\omega(t)= a_0a_1 \ldots a_k$ in X there exists a state sequence $\sigma(t)=s_0s_1 \ldots s_k$ in Y, such that $\exists\ \Theta:A \rightarrow P(S)$ where $\forall a_i \in \omega \Rightarrow \exists s_j \in \sigma$, where $s_j \in \Theta(a_i)$

iii) For every state sequence $\sigma(t)=s_0s_1 \ldots s_k$ in Y there exists an activity sequence $\omega(t)= a_0a_1 \ldots a_k$ in X, such that $\exists\ \Phi:S \rightarrow P(A)$ swhere $\forall s_j \in \sigma \Rightarrow \exists a_i \in \omega$, where $a_i \in \Phi(a_j)$

Informally, X and Y are R-equivalent if

i) there is a correspondence, (the maps Θ and Φ), between the activities of X and the states of Y and,

ii) the sequences of actions as defined by the correspondence is the same.

The models shown in Figure 1 and Figure 3b are representationally equivalent.

REPRESENTATIONALLY EQUIVALENT MODELS

Transforming a State Model to a Precedence Model

Given a state model $Y=\{m_i \mid 1 \leq i \leq n\}$, there exists a precedence model, $X=(A,P,\pi,\psi,\alpha)$, such that Y is representationally equivalent to X.

Algorithm 1:

```
1. Set P and A to $\emptyset$;
2. FOR $1 \leq i \leq |\{M_i\}|$ DO
       Add i to P;
       StatesToTasks[$s^i_0$,i];
   ENDLOOP;
```

Where the recursive procedure StatesToTasks is defined by:

```
StatesToTasks: PROCEDURE [s:STATE, i:INTEGER]
  1.  Create activity s in A and delete state s from $S_i$;
      $\alpha(s) \leftarrow i$; $\psi(s) \leftarrow \oplus$;

  2.  IF $\delta_i(s,\beta) = \emptyset$ THEN RETURN;

  3.  FOR EACH transition $\delta_i(s,\beta)$ from s DO;
          Choose s' from $\delta_i(s,\beta)$;
          $\pi_i(s') \leftarrow \pi_i(s') \cup \{s\}$; $\pi_0(s) \leftarrow \pi_0(s) \cup \{s'\}$;
          IF $\beta \neq \Lambda$ THEN
              BEGIN
                  FOR EACH $j \ni \beta_j \neq \lambda$ DO COMMENT: $\beta_j$ is an element of $\beta$
                      FOR ALL $\gamma_j(s'',b'') \ni \beta_j = b''_i$ DO
                          Add a new activity *s' to A; $\psi(*s') \leftarrow *$;
                          $\pi_i(*s') \leftarrow \{s\}$; $\pi_0(s) \leftarrow \{*s'\}$;
                          $\pi_i(s') \leftarrow \{*s'\}$; $\pi_0(*s') \leftarrow \{s'\}$;
                          $\pi_i(*s') \leftarrow \{s''\}$; $\pi_0(s'') \leftarrow \{*s'\}$;
                      ENDLOOP;
                  ENDLOOP;
              END;

          IF $\gamma_i(s,\beta) \neq \Lambda$ THEN
              BEGIN
                  FOR EACH $j \ni \beta_j \in \gamma_i(s,\beta)$ AND $\beta_j \neq \lambda$ DO
                      FOR ALL $\delta_j(s'',b'') \ni \beta_j = b''_i$ DO
                          Add a new activity s* to A; $\psi(s^*) \leftarrow *$;
```

```
                $\pi_i(s^*) \leftarrow \{s\}$; $\pi_0(s) \leftarrow \{s^*\}$;
                $\pi_i(s') \leftarrow \{s^*\}$; $\pi_0(s^*) \leftarrow \{s'\}$;
                $\pi_j(s^*) \leftarrow \delta_j(s'',b'')$; $\pi_0(\delta_j(s'',b'')) \leftarrow \{s^*\}$;
            ENDLOOP;
        ENDLOOP;
    END;
    StatesToTasks[s',i];
    Delete s' from $\delta_i(s,\beta)$;
ENDLOOP;
```

It can be shown that the application of Algorithm 1 to a state machine will produce an R-equivalent precedence model, although the resulting precedence model may contain more than the minimum number of AND nodes. The minimum number could be attained at the expense of complexity; we have chosen to produce precedence models with extra AND nodes in order to simplify the exposition. It is also assumed that these additional AND nodes are distinguishable by an algorithm which transforms precedence models into state models. Algorithm 1 also assumes that "messages" transmitted among the individual automata are unique; without this assumption, it would be necessary to include additional OR nodes to handle cases corresponding to messages arriving on the tape from different sources.

Transforming a Precedence Model to a State Model

Given a precedence model, $X=(A,P,\pi,\psi,\alpha)$, there exists a state model $Y=\{m_i\}$, such that X is representationally equivalent to Y.

Algorithm 2a: Given any activity b, this algorithm finds the set $\Sigma_a(b)$ of all predecessors of b which are performed by the operator, $\alpha(a)$, and which may have been the most recent preceeding activity performed by $\alpha(a)$ (called operant a-predecessors of b). $\Sigma_a(b) \leftarrow \{d | [\alpha(d)=\alpha(a)]$ and $[\exists$ an activity sequence ending in b, $\omega(b) \ni (\forall c \in \omega(b) \ni \alpha(c)=\alpha(a),\ c \leq d)]\}$.

Algorithm 2b: From a precedence graph, this algorithm constructs a set of automata (one for each element of P). In the following, w is an arbitrary activity in a precedence model (A,P,π,ψ,α), immediately preceeded by v and immediately followed by x with $\alpha(w)=i$. The algorithm uses a list for each element of P. Each element, w, of each list consists of a unique name for the instance, w.a, a string of activities to be done, w.b, a string of counters denoting number of predecessors (with same α) of activities, w.c, and a (level,element) pair identifier, w.d.

```
1. FOR $1 \leq j \leq |P|$ DO define $M_j=(\{s^j_0\},s^j_0,A_n,A_n,\emptyset,\emptyset)$ ENDLOOP.

2. FOR $1 \leq i \leq |P|$ DO
   initialize table Li with the entry activities;
```

```
WHILE Li $\neq \emptyset$ DO
    w $\leftarrow$ next element in Li;
    Li $\leftarrow$ Li - w; DONEi $\leftarrow$ DONEi + w;
    FOR EACH $x \in \pi_0(w)$ DO
        IF $\alpha(x)=\alpha(w)$ THEN w.c.x $\leftarrow$ w.c.x -1
    ENDLOOP;
    msg $\leftarrow \lambda$; childNumber $\leftarrow$ 1;

    FOR EACH (y in w.b with w.c.y = 0) AND $(\alpha(y)\neq\alpha(w))$ DO
        IF $\psi(w)\neq\oplus$ THEN
            msg $\leftarrow$ msg + $m_{\alpha(y)}[y]$
        ELSE
            add w'.a to $\delta(w.a,\lambda)$;
            associate with $\delta(w.a,\lambda)$ the output $\gamma(w.a,\lambda)=y$ sent to $\alpha(y)$.
    ENDLOOP;

    FOR EACH (y in w.b with w.c.y = 0) AND $(\alpha(y)=\alpha(w))$ DO
        add element to list Li with a unique name, y.a;
        construct y.b and y.c
            i.e., find all activities which need to be done after y
                  (applying algorithm 2a);
        construct y.d by forming (level+1,childNumber);
        childNumber $\leftarrow$ childNumber+1;
        $S_i \leftarrow S_i \cup \{y.a\}$;
        add y.a to $\delta_i(w.a,\lambda)$;
        define $\gamma_i(w.a,\lambda)$ as msg;
    ENDLOOP;

    IF $\{y \ni \alpha(y)=\alpha(w)\}$ is empty THEN
        $S_i \leftarrow S_i \cup \{w.a^*\}$;
        add w.a* to $\delta_i(w.a,\lambda)$;
        send messages in msg as corresponding $\gamma$ function
ENDLOOP
```

It can be shown that Algorithm 2 transforms a precedence model into an R-equivalent state model. Note that a buffer of messages may accumulate and the receiving automaton must order these messages appropriately.

In algorithm 2, a state machine is generated for each operator in the precedence model. Whenever two activities are independent in the precedence model, yet both executed by the same operator,

then it must be possible for the operator to do only one of those things, *with the order unspecified by the model.* Figure 3a illustrates this observation if one compares it to a precedence similar to the one shown in Figure 1, except that all activities are executed by the same operator. The algorithm handles such cases by allowing the state machine to choose any sequence of executions that could possibly have been chosen by a single operator in the precedence model. The result is that the state machine has to replicate states that correspond to the execution of a given activity. The number of times each such state is replicated is determined by the number of combinations of ways in which the operator could "schedule" the corresponding task.

COMPUTATIONAL COMPLEXITY OF MODEL TRANSFORMATIONS

In this section, we show that any general transformation algorithm from precedence models to state models is NP-Complete. This is tantamount to showing that the transformation is in general intractible [GARE79]. However, this does not rule out the possibility that there are significant subclasses of models for which this transformation is tractable. Later in this paper we elucidate one such subclass.

The class NP is the class of computations that can be performed on a nondeterministic polynomial time bounded Turing machine. This class has received much attention in the literature (see e.g. KARP72, GARE79) and has been shown to include many apparently hard combinatorial problems, such as the "traveling salesman problem". The reader is referred to any of the above cited references for a discussion of the class NP, the reduction of one problem to another, and the matter of encoding problems as Turing machine tapes. Certain problems in NP are *polynomial complete* or *NP-complete*, meaning that if they have polynomial time deterministic algorithms, then so does every problem in NP. Since for no polynomial complete problem has a less than exponential algorithm been found, a proof of NP-completeness for a problem strongly suggests, but does not imply, a proof of intractability.

There are a variety of equivalent abstractions of computation in which to discuss complexity and NP-completeness. Our exposition is carried out in terms of "abstract algorithms" as espoused by Richard Karp [KARP72]:
An *abstract algorithm* Γ is defined as -

(1) a countable set Δ (the domain)
(2) a countable set P (the range)
(3) a finite alphabet A such that $A^* \cap P = \varphi$
(4) an encoding function $E: \Delta \rightarrow A^*$
(5) a transition function $T: A^* \rightarrow A^* \cup P$.

The *computation* of Γ on input $x \in \Delta$ is the sequence y_1, y_2, ... such that $y_1 = E(x)$, $y_{i+1} = T(y_i)$ for all i. If the sequence is always unique, then Γ is a deterministic algorithm, otherwise Γ is a nondeterministic algorithm. If the sequence is finite, then its length t(x) is the running time of Γ on x. Γ is terminating if all its computations are finite. A terminating algorithm Γ computes the function $f_\Gamma:\Delta \rightarrow P$ such that $f_\Gamma(x)$ is the last element of the computation of Γ on x. Thus $f_\Gamma(x) = y_k \in P^*$. A computation is within the class NP iff there is an abstract algorithm which computes the correct output for it in polynomial running time. This means that t(x) = p(length(x)) where p denotes some polynomial. If this can be done by a deterministic algorithm, then the problem is said to be within the class P.

One problem proven to be NP-complete is the *job sequencing problem* in which a set of jobs with

known execution times, deadlines, and penalties must be scheduled sequentially to minimize the sum of the penalties of the jobs which missed their deadlines.

JOB SEQUENCING -
Input: "execution time vector" $(T_1, \ldots, T_k)$
"deadline vector" $(D_1, \ldots, D_k)$
"penalty vector" $(V_1, \ldots, V_k)$
Problem: Find schedule with minimal penalty -

$$\text{minimize} \quad \Sigma \quad (\text{if } T_{\pi(1)} + \ldots + T_{\pi(j)} > D_{\pi(j)} \text{ then } V_{\pi(j)} \text{ else } 0)$$

where the above minimization is performed over all permutations π of $(1,2, \ldots, k)$ and the summation ranges over $j = 1$ to k. Since this problem is known to be NP-complete, its solution by a deterministic algorithm in polynomial time would mean solution to all of the other problems in the NP class and resolution of the famous question "P = NP?" . We will next show that transformation of our office models within deterministic polynomial time would imply solution to the job sequencing problem, and is therefore probably impossible.

Theorem: Transformation from precedence oriented models to state oriented models is an NP-complete computation.

Proof:

(a) First we show that the problem is not too difficult (i.e. it is NP).
The transformation can be performed in polynomial time by a nondeterministic Turing machine which traverses the input precedence graph according to algorithm 2b, and outputs an automaton node at each step. It splits into n machines whenever it encounters a node with n immediate successors in the precedence graph. Any particular Turing machine traverses at most one possible sequential execution path through the precedence graph without looping. Thus its time (# of steps) is a linear function of the number of nodes in the original precedence graph.

(b) Second we show that the problem is not too simple (i.e. it is complete).
If the transformation can be performed by a deterministic algorithm in polynomial time, then the sub-problem of processing k activities with no precedence constraints can also be done. So consider the subclass of precedence models with $|A| = k$; $|P| = 1$; $p_i = p_0 = \varphi$. If this case can be processed, then the job sequencing problem can be solved because the transformation in this case constructs a path for each permutation of the original k activities To solve the job sequencing problem, we interpret the k activities as jobs. The only extra processing which needs to be done is for each node to calculate its minimum penalty sum by taking a minimum over its predecessors, test for its own deadline, and add a penalty to the sum. This processing is only carried out when a

node is visited within the algorithm, so it only adds a linear amount of work in order to solve the job sequencing problem. Since the job sequencing problem is NP-complete, so is the transformation.

As an example, consider the precedence model shown in Figure 4a; activities b, c, d, and e are all enabled when activity a terminates. If all five tasks are executed by the same operator, then that operator may do b through e in any order, i.e., the order in which the operator is to execute the four activities is unspecified. The resulting state model is shown in Figure 4b. From this example, it can be seen that precedence models like the one shown in Figure 4a, where an activity has n successors enabled simultaneously, results in a state machine with $o(k^n)$ states and thus takes exponential time to generate by a deterministic algorithm.

PROPERLY CONCURRENT PRECEDENCE MODELS

Informally, one can describe the effect of $\psi(x) = *$ to mean that successive activities to activity x can be done in parallel, or in either order. Precedence models can be restricted such that the *-specification explicitly specifies (logical or potential physical) concurrency. This is accomplished by restricting the α mapping of the concurrent activities.

Definition 6: Suppose that $x \in A$, and $\psi(x)=*$. Define the *concurrency reach* of task x along arc i, denoted R(x,i), to be the set of all tasks reachable from task x via $i \in \pi_0(x)$, but not reachable via arc $j \in \pi_0(x)$ $\forall i \neq j$.

Definition 7: Let $X=(A,P,\pi,\psi,\alpha)$ be a precedence model, and $\forall x \in A$ such that $\psi(x)=*$ the following is true. If $\forall$ $i,j \in \pi_0(x)$, $i \neq j$ for $q \in R(x,i)$, $r \in R(x,j)$ then $\alpha(q) \neq \alpha(r)$. Then the precedence model is *properly concurrent.*

Consider algorithm 2 which converts a precedence model into a state model. In the case of improper concurrence, it was necessary to create all combinations of states which could be executed by a single operator concurrently. When the precedence model is properly concurrent, then there are no tasks which can be done concurrently by an individual operator. If there were such a task, then it would be in R(x,i) and R(x,j) for some $i \neq j$; but then, by definition, the graph is not properly concurrent. The transformation algorithm is not exponential for this subclass of state models, since the state model never needs to show alternative schedules that the operator is free to choose from. The schedule for the automaton is completely specified by the precedence model.

SUMMARY

Studies of the flow of information in offices have led to a variety of different graph models. Such models tend to emphasize the flow of transactions through an office or the functionality of individual work stations in the office. Two formal notations have been defined in this paper which represent the two approaches; using these notations, we have shown that for any model expressed in one view, a corresponding model in the complementary view can be derived, although this derivation is an NP-complete computation. Finally we introduce a sub-class of transaction-oriented models in which the expression of parallel processing represents real potential concurrency. In this case the transformation between models becomes a tractable (polynomial time) computation.

A formal theory of information flow in the office has not yet been proposed, but the work in this paper is a step in that direction. Informal notions of forms-flow and workstation office models are formalized, and then algorithms are shown which transform models of one type to the other. A rigorous framework for constructing proofs have been provided and a formal proof of NP-completeness has been presented within this framework.

REFERENCES

[BAUM80] Baumann, L. S. and R. D. Coop "Automated Workflow Control: A Key to Office Productivity", *AFIPS Digest of the Office Automation Conference*, March, 1980.

[COFF73] Coffman, E. G. Jr. and P. J. Denning, *Operating Systems Theory*, Prentice-Hall, Inc. Englewood Cliffs, NJ, 1973.

[ELLI79] Ellis, C. A., "Information Control Nets: A Mathematical Model of Information Flow", *ACM Proceedings of the Conference on Simulation, Modeling and Measurement of Computer Systems*, August, 1979, pp. 225-240.

[GARE79] Garey, M. R., and S. Johnson, *Computers and Intractability*, (1979).

[HAMM77] Hammer,M., W. G. Howe, V. J. Kruskal and I. Wladawsky, "A Very High Level Programming Language for Data Processing Applications", *Communications of the ACM*, Vol. 20, No. 11, (November, 1977), pp. 832-840.

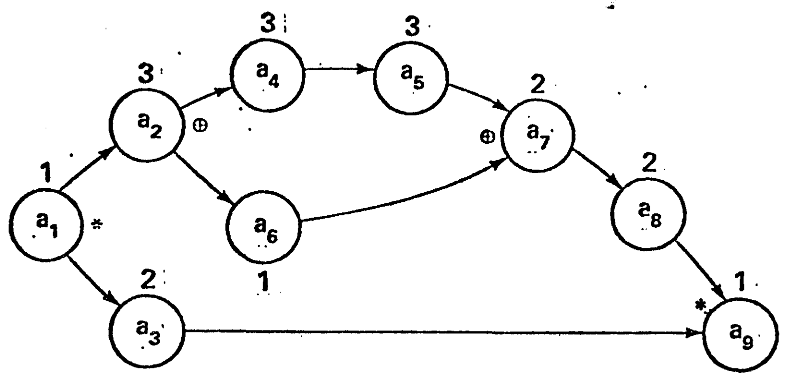
[KARP72] Karp, R. M., "Reducibility Among Combinatorial Problems", in *Complexity of Computer Computations*, Miller, R. E., et.al. editors, (1972), pp. 85-103.

[KIEN79] Kienzle, M. G. and K. C. Sevcik, "Survey of Analytic Queueing Network Models of Computer Systems", *ACM Proceedings of the Conference on Simulation, Modeling and Measurement of Computer Systems*, August, 1979, pp. 113-129.

[NESS77] Ness, D., "Office Automation Project: Non-Deterministic Procedures in Office Processes", Working Paper 77-02-02, Dept. of Decision Sciences, Wharton School, University of Pennsylvania, 1977.

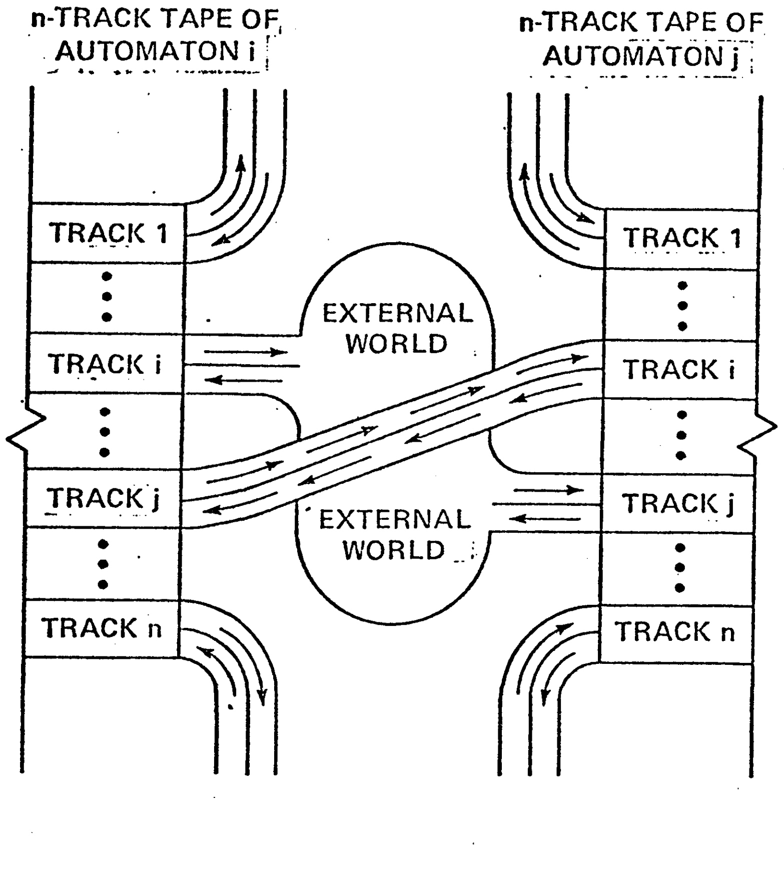
[RAZO79] Razouk, R. R., M. Vernon and G. Estrin, "Evaluation Methods in SARA -- The Graph Model Simulator", *ACM Proceedings of the Conference on Simulation, Modeling and Measurement of Computer Systems*, August, 1979, pp. 189-206.

[ZISM77] Zisman, M. D., *Representation, Specification and Automation of Office Procedures*, Ph.D. dissertation, Dept. of Decision Sciences, Wharton School, University of Pennsylvania, 1977.



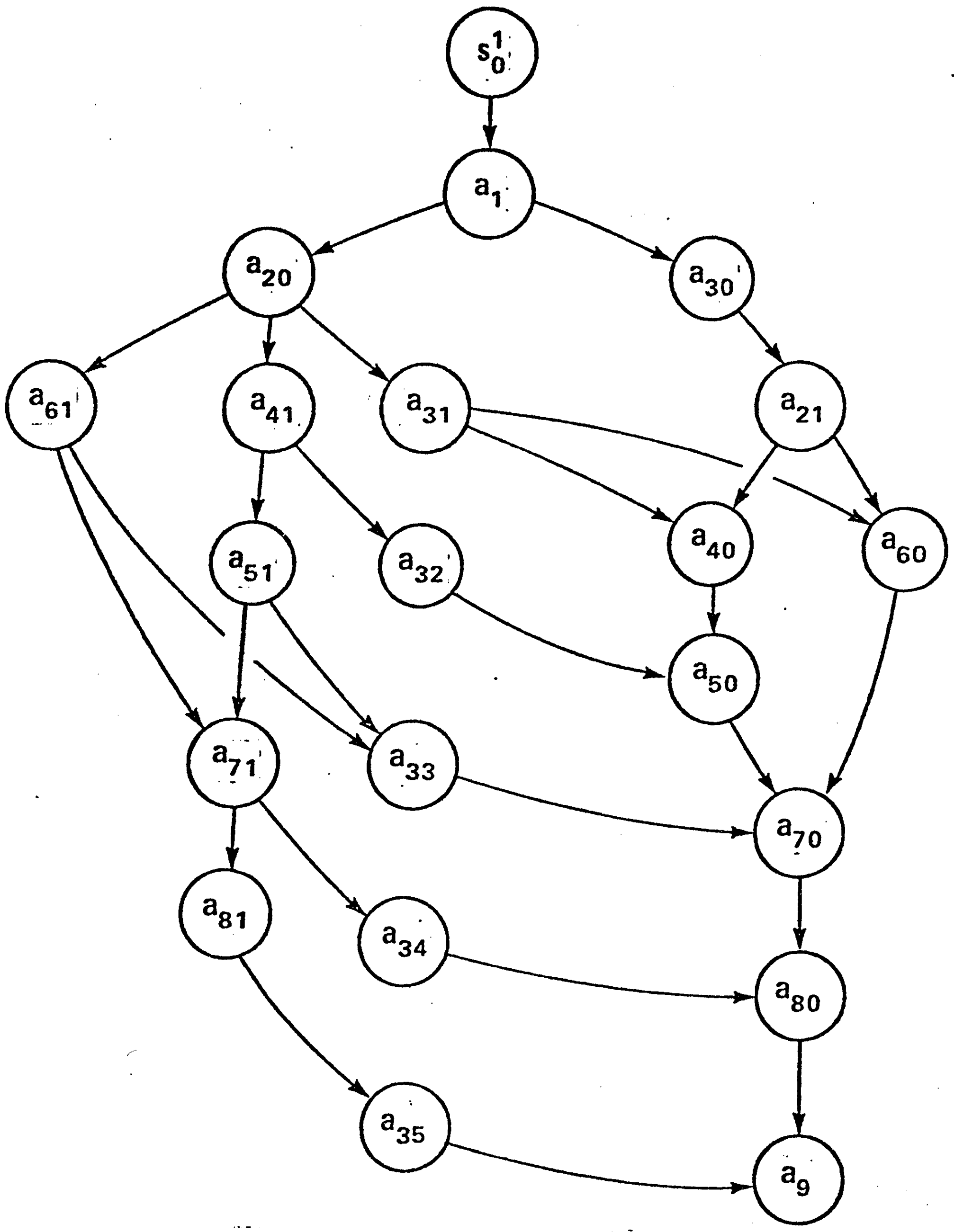
A PRECEDENCE MODEL

FIGURE 1



THE COMMUNICATION NETWORK

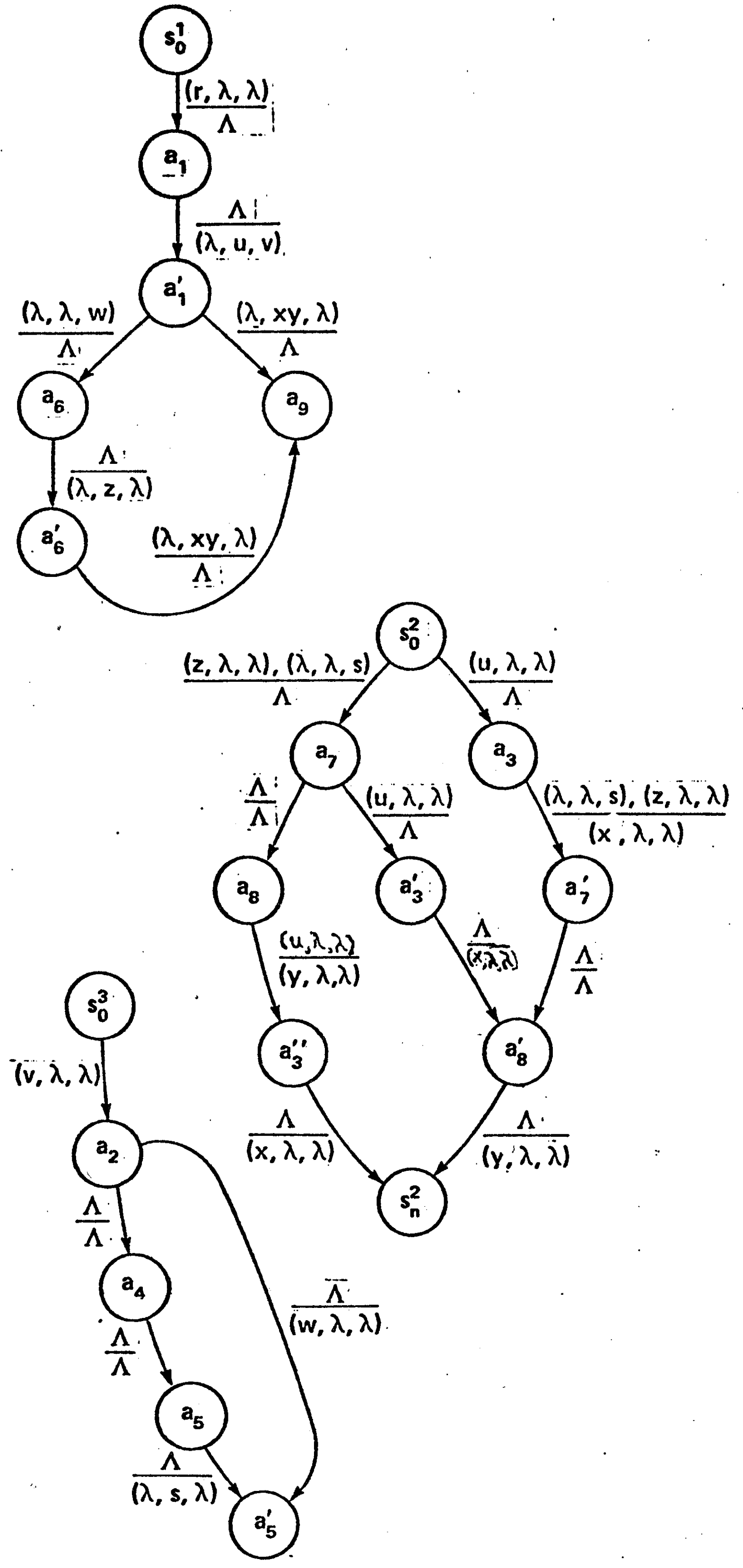
FIGURE 2



EACH ARC IS LABELED Λ/Λ

A STATE MODEL

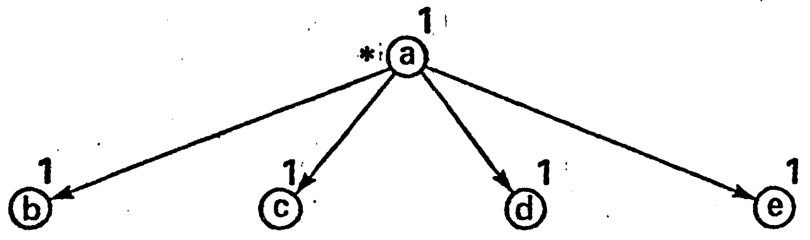
FIGURE 3(a)



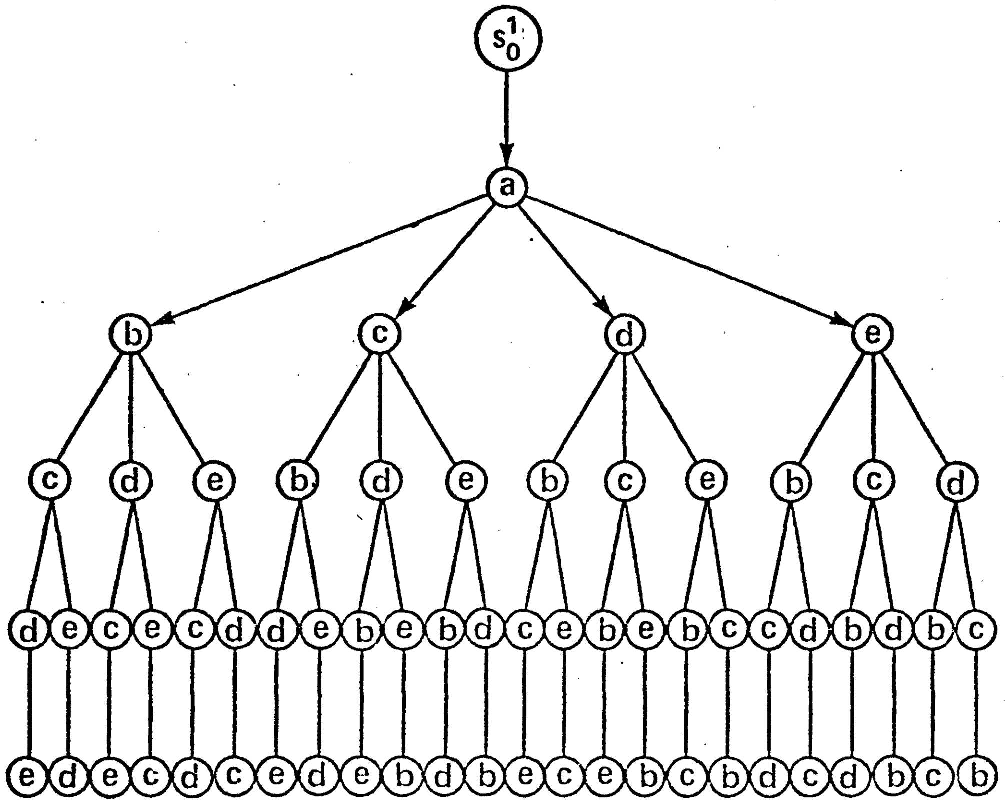
s_0^1
$\frac{(r,\lambda,\lambda)}{\Lambda}$
a_1
$\frac{\Lambda}{(\lambda,u,v)}$
a_1'
$\frac{(\lambda,\lambda,w)}{\Lambda}$
$\frac{(\lambda,xy,\lambda)}{\Lambda}$
a_6
a_9
$\frac{\Lambda}{(\lambda,z,\lambda)}$
a_6'
$\frac{(\lambda,xy,\lambda)}{\Lambda}$
s_0^2
$\frac{(z,\lambda,\lambda),(\lambda,\lambda,s)}{\Lambda}$
$\frac{(u,\lambda,\lambda)}{\Lambda}$
a_7
a_3
$\frac{\Lambda}{\Lambda}$
$\frac{(u,\lambda,\lambda)}{\Lambda}$
$\frac{(\lambda,\lambda,s),(z,\lambda,\lambda)}{(x,\lambda,\lambda)}$
a_8
a_3'
a_7'
$\frac{(u,\lambda,\lambda)}{(y,\lambda,\lambda)}$
$\frac{\Lambda}{(x,\lambda,\lambda)}$
$\frac{\Lambda}{\Lambda}$
a_3''
a_8'
$\frac{\Lambda}{(x,\lambda,\lambda)}$
$\frac{\Lambda}{(y,\lambda,\lambda)}$
s_n^2
s_0^3
(v,λ,λ)
a_2
$\frac{\Lambda}{\Lambda}$
a_4
$\frac{\Lambda}{(w,\lambda,\lambda)}$
$\frac{\Lambda}{\Lambda}$
a_5
$\frac{\Lambda}{(\lambda,s,\lambda)}$
a_5'

A STATE MODEL

FIGURE 3(b)



(a)



(b)

EXPONENTIAL EXPLOSION OF STATES

FIGURE 4

XEROX

XEROX

On the Equivalence of Office Models By Clarence A. Ellis and Gary J. Nutt

SSL-79-8

Xerox Corporation
Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, California 94304