Proceedings of the 1980 Winter Simulation Conference
T.I. Ören, C.M. Shub, P.F. Roth (eds.)

# DISCRETE EVENT SIMULATION OF STOCHASTIC AND DETERMINISTIC SEQUENTIAL MACHINE MODELS

Melvin M. Cutler
Strategic Systems Division
Hughes Aircraft Company
Culver City, California 90230*

ABSTRACT: This paper presents and proves the correctness of novel techniques for the mathematical verification of simulation models against sequential machine specifications. The main idea is to say that a discrete event simulation program simulates a given sequential machine specification if and only if there exists a certain type of mapping from it to the specification. It is first shown that discrete event specifications can be realized by using the traditional formalisms of discrete time systems. Then the two main facilitating results are given. A discrete event simulation of a discrete deterministic system is shown to be a formal object whose fidelity to the system can be proved. A discrete event simulation of a discrete stochastic system is shown to be a formal object whose statistical fidelity to the system can be proved.

## 1. INTRODUCTION

When simulation is used in the evaluation of complex systems which are not analytically tractable, the process of verifying that the simulation software does implement the simulated system is a critical one. An intriguing approach is to represent the simulated system as a sequential machine, and to utilize automata-theoretic verification techniques (Zeigler 1976). Sequential machines are widely accepted as specifications for discrete time systems, as are stochastic sequential machines (e.g., Markov chains) for probabilistic discrete time systems. Digital logic circuits and queueing systems, respectively, are easily recognized examples of applications of deterministic and stochastic sequential machine specifications. In trying to apply formal system evaluation techniques, however, two problems arise which impede the practical application of such discrete time specifications. First, many discrete systems (e.g., job shops, airline reservation systems, computer networks) are more naturally described by discrete event specifications -- no central "clock" drives the system to a different state on each "tick". Second, they do not, in general, provide insight into the autonomous behavior of the system -- the internal transitions which carry out the functions dictated by inputs previously received.

This paper presents and proves the correctness of a new approach to these problems. It introduces formalized discrete event simulation languages as aids to the verification of discrete systems. The sequencing of state changes in these simulation languages is event-driven, meaning that the simulation skips dead time between state changes. Also, the languages provide primitives for the simulation of autonomous state changes of arbitrary complexity. And programmers, the implementers of system simulations, are used to dealing with a small but universal set of computational primitives rather than the ad hoc set-defining notation which is the usual description of the state transitions of automata.

The idea that simulation programs could be taken as objects of formal system description is a recent one (Zeigler 1976, Cutler 1979). Verifying that a program simulates the operation of a system has been a significant combinatorial problem for deterministic specifications and an error-prone statistical problem for stochastic specifications. Because stochastic specifications are simulated by the use of a pseudo-random generator which is implemented using a deterministic algorithm, the simulation can only approximate the behavior of a probabilistic system. The large body of a posteriori statistical methods developed over the years to measure this approximation had been grudgingly accepted as necessary for the analysis of simulation results. The promising alternative of a priori verification of stochastic system simulations is a research area of great potential practicality.

* This material is based on work supported in part by the National Science Foundation under Grant MCS78-04725 while the author was a student in the Department of System Science, University of California, Los Angeles.

Taken together, the gaps in simulation and system specification techniques described in this section indicate the following course of action:

1. In order to use simulation programs to their full capabilities, they must be formalized.
2. In order to use automata as system specification tools, they must be adapted to discrete event systems.
3. In order to implement probabilistic specifications with deterministic simulation programs, a concept of deterministic realization and a priori faithfulness of stochastic systems must be developed and defended.
4. In order to link the specification to the simulation, formalized simulation programs must be made verifiable against sequential machine specifications.

The results presented herein form a theoretical basis for these advances in the technology of system verification, and may spawn future research in the area of formal system verification. The formalization of simulation programs is taken from Cutler (1979). Thus we continue the use of a discrete time base, reflecting the practical restrictions of running these programs on digital computers. This facilitates the use of traditional automata-theoretic specifications of systems. Sections 2 and 3 adapt such discrete time automata to discrete event systems, and compare them to the DEVS of Zeigler (1976) and to the stochastic DEVS of Melamed (1976). The ergodic machine model of Aggarwal (1977) is defended as the proper formal system for the deterministic realization of stochastic machines. Section 4 establishes the homomorphism as a necessary and sufficient condition to verify deterministic simulations against deterministic specifications, and Section 5 establishes the "regular" homomorphism as a necessary and sufficient condition to verify stochastic simulations against probabilistic specifications.

## 2. NOTATIONAL PRELIMINARIES

A number of formal systems must be defined in order to cover all possible classes of systems to be simulated or specified. We introduce deterministic discrete event automata and behavior-preserving mappings between them, ergodic machines and behavior-preserving mappings between them, and probabilistic automata and behavior-preserving mappings between them. It is assumed, however, that the reader is familiar with the sequential machine (Mealy or Moore) notation. We use N to represent the set of natural numbers.

The first concept to be introduced is Zeigler's (1976) definition of a sequential machine model of discrete event specifications (DEVS). In this notation, a state set $S_M$ is augmented by a second component which represents the amount of time that the DEVS has been in a state. When this component reaches a value determined by a function $t_M$, the DEVS moves autonomously to another state. This autonomous transition is indicated by the null input symbol $\emptyset$.

Definition 1: A Discrete Event System Specification (DEVS) is a system

$$M = \langle X_M, S_M, Y_M, \delta_M, \lambda_M, t_M, T \rangle$$

where

$X_M$ is the set of external events,

$S_M$ is the set of sequential states,

$Y_M$ is the output value set,

T is a time base set,

$\delta_M$: $S_M \times (X_M \cup \{\emptyset\}) \rightarrow S_M$

$\lambda_M$: $S_M \rightarrow Y_M$

$t_M$: $S_M \rightarrow T$

If T is the set of real numbers, then we say that the system is a DEVS with a real time line. If T is isomorphic to a subset of N, then we say that the system is a DEVS with a countable time line. Because

we will simulate these specifications with software which runs on digital computers, the use of a countable time line will be assumed.

A DEVS changes state at specific points in the time domain. We need a more complete characterization of the DEVS in order to describe its operation precisely.

Definition 2: A configuration (u,t,x,y) of M is an element of

$$S_M \times T \times \left\{ X_M \times T \right\}^* \times \left\{ Y_M \times T \right\}^*$$

where u indicates the current state, t indicates the time of the last state transition, x indicates the input which is to be read by M in the future, and y indicates the output which has been produced by M in the past.

Definition 3: A transition of M is defined by the operator "|-" as follows.

If $0 \leq (t'-t) \leq t_M(u)$, then

$$(u,t,(x',t')x,y) \vdash \left\{ \left\{ \delta_M(u,x') \right\}, t', x, y \left\{ \lambda_M \left\{ \delta_M(u,x') \right\} \right\} \right\}$$

If $(t'-t) > t_M(u)$, then

$$(u,t,(x',t')x,y) \vdash \left\{ \left\{ \delta_M(u,\emptyset), 0 \right\}, t+t_M(u), (x',t')x, y \left\{ \lambda_M \left\{ \delta_M(u,\emptyset) \right\} \right\} \right\}$$

If $t' < t$, then |- is undefined (M blocks).

We will identify a transition of M by the operator |- when it is not otherwise clear from context. Let $\vdash^*$ denote the reflexive, transitive closure of $\vdash_M$.

The definition of the transition function restricts the DEVS to change state upon receiving an input x from $X_M$ or when no input appears while the DEVS is in state q for a time $t_M(q)$.

Associated with the DEVS specification of Zeigler (1976) is the idea of a behavior-preserving mapping.

Definition 4: Let $M=\langle X_M, S_M, Y_M, \delta_M, \lambda_M, t_M, T\rangle$ and $M'=\langle X_{M'}, S_{M'}, Y_{M'}, \delta_{M'}, \lambda_{M'}, t_{M'}, T\rangle$ be DEVS. The triple (g,h,k) is a system specification morphism from M to M' if

$$g: X_{M'} \rightarrow X_M$$

$$k: Y_M \rightarrow Y_{M'}$$

and

$$h: S_M \rightarrow S_{M'},$$

such that, for every x in $X_{M'}$, y in $Y_M$, u in $S_M$,

$$h\left\{ \delta_M(u,g(x)) \right\} = \delta_{M'}(h(u),x)$$

$$t_M(u) = t_{M'}(h(u))$$

$$k\left\{ \lambda_M(u) \right\} = \lambda_{M'}(h(u))$$

and

$$h\left\{ \delta_M(u,\emptyset) \right\} = \delta_{M'}(h(u),\emptyset)$$

The mapping h is a DEVS homomorphism if g and k are isomorphisms.

The system specification morphism and the DEVS homomorphism resemble the (ordinary) sequential machine homomorphism, with the additional condition that autonomous transitions are also related, and take place at the same point in time.

The next idea necessary for this formalization is the ergodic machine concept of Aggarwal (1975). This relies on some definitions from measure and ergodic theory.

Definition 5: A probability space is a triple (O,F,P), where O is a set, F is a field of subsets of O, and P is a probability measure on F.

Definition 6: Let (O,F,P) be a probability space, and let T: O $\rightarrow$ O be a transformation on O. T is measure preserving if and only if for any A in F, P({a | T(a)$\in$A}) = P(A).

Definition 7: A set A is T-invariant if and only if {a | T(a)$\in$A} = A.

Definition 8: Let (O,F,P) be a probability space. A transformation T: O $\rightarrow$ O is an ergodic transformation if it is measure-preserving and if the only sets that are T-invariant have measure 0 or 1.

Ergodic transformations preserve the structure of measure spaces (Billingsley 1965). We are concerned only with probability measure spaces, so that in the following discussion (O,F,P) will always refer to a probability space, and T to an ergodic transformation on O.

We are interested in ergodic transformations as a vehicle for the deterministic realization of specifications which utilize the stochastic sequential machine notation. They are of interest for such applications because of the pointwise ergodic theorem (Halmos 1956), repeated below.

Let $I_A$ be the indicator function of the set A; that is $I_A(x)=1$ if x is an element of A and $I_A(x)=0$ otherwise. Then if T is an ergodic transformation on (O,F,P) and w is in O

$$\lim_{n\rightarrow\infty} \frac{1}{n} \sum_{k=0}^{n-1} I_A(T^k(w)) = P(A) \quad \text{a. e.}$$

Thus, if we apply T to an element of O and keep track of the percentage of times it produces an element of a set A in F, then the latter statistic converges to the measure of A, P(A). Clearly, this is a desirable characteristic for deterministic realizations of probabilistic systems, and it is exploited by Aggarwal (1977) in his research into ergodic machines.

Definition 9: An ergodic machine is a system

$$E = \langle X, S, \overline{\Pi}, Y, T, \delta_1, \lambda \rangle \text{ where}$$

X is a set of inputs,

S is a countable set of states,

$\overline{\Pi}$ = ( R, F, P ) is a probability space (we term R the seed space),

Y is a set of outputs,

T is an ergodic transformation on R,

$\delta_1$: S x R x X $\rightarrow$ S is the one-step transition function, and

$\lambda$: S x R $\rightarrow$ Y is the output function.

An ergodic machine is implemented by a deterministic sequential machine, possibly infinite-state, specified by

$$M = \langle X, S \times R, Y, \delta, \lambda \rangle$$

where X, S, R, Y, and $\lambda$ are as in E and, for all s in S, r in R, x in X

$$\delta(s, r, x) = (\delta_1(s, r, x), T(r))$$

Definition 10: Given two ergodic machines $E=\langle X,S,(R,F,P),Y,\delta_1,\lambda \rangle$ and $E'=\langle X',S',(R',F',P),Y',\delta_1',\lambda' \rangle$, E' is a homomorphic image of E if and only if there exists a homomorphism

$$h : S \times R \rightarrow S' \times R'$$

which maps the state set of $M_E = \langle X, S \times R, \delta, \lambda \rangle$ onto the state set of $M_{E'} = \langle X', S' \times R', \delta', \lambda' \rangle$ and homomorphisms g and k

$$g : X' \rightarrow X$$
$$k : Y \rightarrow Y'$$

such that for all x in X', s in S, r in R,

i. $h(\delta(s, r, g(x))) = \delta'(h(s, r), x)$

ii. $k(\lambda(s, r)) = \lambda'(h(s, r))$

E' is a regular homomorphic image of E if there exist homomorphisms

$$h' : S \rightarrow S'$$
$$h'' : R \rightarrow R'$$

such that for all s in S, r in R $(h'(s), h''(r)) = h((s,r))$ This restriction on h is crucial to our formal concept of simulation of stochastic specifications, as we shall see in Section 5.

Definition 10 is analogous to the system specification morphism for Zeigler's iterative system specification and is equivalent to the definition of ergodic machine homomorphism given by Aggarwal (1975) when both the input and output symbol sets of E and E' are equal.

Definition 11: Given an ergodic machine E, for any states s, s' in S and input x in X, the s,s',x transition seed space $A_{s,s'}(x)$ is the set

$$A_{s,s'}(x) = \{r \mid r \in R, \delta_1(s, r, x) = s'\}$$

Since $A_{s,s'}(x)$ is a subset of R, we can require that $A_{s,s'}$ be in F, that is, $A_{s,s'}$ is P-measurable. If E is autonomous (that is, X has exactly one element), it is convenient to represent the transition seed space by $A_{s,s'}$.

The transition seed space concept links the ergodic machine to the stochastic system specification represented by the probabilistic automaton.

Definition 12: A probabilistic automaton is a system $C = \langle S, X, \{Z(x)\}, Y, \lambda \rangle$ where

X is a (finite) set of inputs

S is a set of states of C, $S = \{s_1, \ldots, s_q\}$

$\{Z(x)\}$ is a set of q x q matrices, one for each element of X, with the i,jth entry $Z_{i,j}(x)$ representing the probability of C moving to state $s_j$ under input x given that it is in state $s_i$, subject to the constraint that for all i,

$$\sum_j Z_{i,j}(x) = 1$$

Y is a (finite) set of outputs

$\lambda$: $S \rightarrow Y$ is the output function

C is autonomous if the cardinality of X is 1, in which case we represent $\{Z(x)\}$ as Z. Now the stochastic and deterministic models for probabilistic systems may be linked.

Definition 13: Given an ergodic machine E, the associated automaton $C_E$ is the probabilistic automaton

$$C_E = \langle S, X, \{Z(x)\}, Y, \lambda \rangle$$

where $Z_{ij}(x) = P(A_{i,j}(x))$. We call E a realization of $C_E$.

Because $C_E$ is independent of the ergodic transformation T of E, many ergodic machines have identical associated automata. Thus, there are many ergodic realizations of the same probabilistic automaton. Also, it is easy to show that for every probabilistic automaton C there exists an ergodic machine E such that C is the associated automaton of E (Aggarwal 1977).

Finally, we define the behavior-preserving mappings between probabilistic automata.

Definition 14: A probabilistic automaton $M = \langle X, S, \{Z(x)\}, Y, \lambda \rangle$, $S=\{s_1,\ldots,s_q\}$,

covers a probabilistic automaton $M' = \langle X, S', \{Z'(x)\}, Y, \lambda \rangle$, $S'=\{s'_1,\ldots,s'_q\}$,

if and only if there exists a homomorphism $h: S \rightarrow S'$

such that for every x in X, $s'_i, s'_j$ in S' and $s_k$ in S with $h(s_k)=s'_i$,

$$Z'_{ij}(x) = \sum_{s_m \in h^{-1}(s'_j)} Z_{km}(x)$$

The probabilistic homomorphism, or p-homomorphism, of Aggarwal (1975) is similar to the covering relation in that its structural mapping preserves statistical behavior. Yet its appearance is quite different.

Definition 15: A probabilistic automaton B is a probabilistic homomorphic image of a probabilistic automaton A if there exist ergodic machines E(A) and E(B) such that

E(A) is a realization of A,

E(B) is a realization of B, and

E(B) is a homomorphic image of E(A).

If h is the homomorphism mapping E(A) onto E(B), then h is called a p-homomorphism from A to B.

If E(B) is a regular homomorphic image of E(A), then B is called a regular probabilistic homomorphic image of A. The homomorphism h is called a regular p-homomorphism from A to B.

Thus, the verification of behavior preservation can be executed in the ergodic machine (deterministic) domain rather than in the stochastic machine (probabilistic) domain. The corresponding definition of Aggarwal (1975) restricts the realizations to the class of generalized ergodic machines whose ergodic transformations must be faithful. We will not explain these terms, but note that no ergodic machine with a finite seed space can be a faithful representative. The stronger condition is used for simplification, not verification, of the simulation of probabilistic systems, and thus is not critical to this research. Though we have modified Aggarwal's definition of p-homomorphism, the relationship it specifies between probabilistic automata has been preserved.

## 3. DISCRETE EVENT VERSIONS OF DISCRETE TIME MODELS

It has been noted that sequential machines, being discrete time models, limit the flexibility of system specifications (Zeigler 1976). However, it is important that the discrete event specifications are capable of describing all the systems of interest. Thus, it is significant that there exist discrete event versions of discrete time models which are equivalent in power. For deterministic systems, it should be clear that the DEVS formalism restricted to countable time domains is sufficient.

For stochastic systems and their deterministic realizations, the existing formalism is shown to "hide" discrete event models:

Definition 16: An SDEVS is a stochastic sequential machine $M = \langle Q, X', \{Z(y|x)\}, Y', \Lambda \rangle$, where

$Q \subseteq S \times N$ is a finite set of pseudo-states, where N is the set of natural numbers and S is a finite set of states

$X' = X \cup \{e\}$ is a set of input symbols

$Y' = Y \cup \{e\}$ is a set of output symbols

$\{Z(x)\}$ is a set of q x q matrices, $q = |Q|$, one for each input symbol, with the i,jth entry of $Z(x)$ representing the probability of M moving to state $q_j$ under input x, given that it is in state $q_i$, subject to the following constraints.

a. For all i,

$$\sum_{j=1}^{q} Z_{i,j}(x) = 1$$

b. For each state $s_i$ in S, there exists a constant $c_i$ depending only on $s_i$ such that

$$\sum_{s_k \in S} Z_{(s_i,j),(s_k,m)}(e) = 1$$

where $m = \begin{cases} 0 & \text{if } j > c_i \\ j+1 & \text{if } j \leq c_i \end{cases}$

$\Lambda$: $S \rightarrow Y'$ is the output function

The beauty of this formalism is that the DEVS mechanism is hidden in the input symbol stream and in the state transition matrices. Thus, it works almost completely within the existing SSM formalism. We can do this because e is now a unit length null symbol rather than the empty symbol, and is included in the input alphabet X. The machine makes a transition from state q when either an input symbol is read or it has been in state q for elapsed time t(q). The SDEVS M is specified so that the Z matrices "simulate" this.

If M "reads" an input symbol x, its transition matrix Z(x) moves M to a state (q,0), with 0 indicating that M has just entered state q. Suppose M reads a null input symbol e in state (p,n). If t(p)>n, then Z(e) deterministically moves M to state (p,n+1); otherwise, the internal transitions represented by Z(e) move M to a distribution of states of the form (q,0). Conditions a and b of the definition make the definition of this behavior consistent with existing SSM formalism.

The discrete time SDEVS specification M pretends to be a discrete event system by chopping up the time line into small but countably many segments. For instance, if the first input (event) $x_1$ is to happen at time $t_1$, then the input to M will begin with $t_1$-1 e's followed by $x_1$.

In order to avoid the trouble of specifying long input strings with many e's, it is reasonable to develop a shorthand input notation for the SDEVS. Suppose we represented the input stream as a string of pairs $(t_i, x(t_i))$, where $x(t_i)$ is to be input to M after $t_i$ time steps. This notation conveys all the information needed for M to operate, although we must make some basic restrictions so that no t is negative, and no two symbols are read at the same time.

Again, a discrete time line is embodied in this SDEVS formalism, diverging from the stochastic DEVS formalism of Melamed (1976), in which a continuous time base was installed. This reflects the practical limitations of digital computer software, and in addition avoids the necessity of providing all the samples of random numbers a priori to control the state sequencing in Melamed's stochastic DEVS.

Since the ergodic machine is also a discrete time model, a construction similar to the SDEVS yields a discrete event specification for the ergodic machine.

Definition 17: A discrete event ergodic machine (DEEM) is an ergodic machine $E=\langle X', Q, \overline{\Pi}, Y', T, \delta_1, \lambda \rangle$ such that

$X' = X \cup \{e\}$ is the finite input set

$Q \subseteq S \times N$ is the pseudo state set

$\overline{\Pi} = (R, F, P)$ is a probability space

$Y' = Y \cup \{e\}$ is the finite output set

T is an ergodic transformation on R

$\delta_1: X' \times Q \times R \rightarrow Q$ is the single step transition function

$\lambda: Q \times R \rightarrow Y$ is the output function

A DEEM is implemented by a deterministic sequential machine, possibly infinite-state, specified by

$$M = \langle X', Q \times R, Y', \delta, \lambda \rangle$$

where X', Q, R, Y', and $\lambda$ are as in E and, for all x in X, s in S, r in R, n in N

$$\delta(x, s, n, r) = (\delta_1(x, s, n, r), 0, T(r))$$

and if t(a)>n

$$\delta(e, s, n, r) = (s, n+1, r)$$

and if $t(a) \leq n$

$$\delta(e, s, n, r) = (\delta_1(e,s,n,r), 0, r)$$

Now that it has been established that discrete event specifications for stochastic and deterministic systems are both feasible and powerful, it is necessary to provide a basis for determining under what conditions a computer program correctly simulates a discrete event specification.

## 4. SIMULATION OF DETERMINISTIC SPECIFICATIONS

For deterministic specifications, this question has been addressed by Zeigler (1976) and Cutler (1979). The approach has been to construct a homomorphism from a "canonical" sequential machine representation

of the program to the specification.

The flowchart model for simulation programs introduced by Cutler (1979) was shown in that paper to be equivalent to the DEVS in the deterministic case. In that proof, and in order to show that it is equivalent to the DEEM in the probabilistic (ergodic) case, the "state machine" is a crucial concept.

This machine is analogous to the implementing sequential machine of the DEEM and ergodic machine formalism; that is, a representation of the behavior of a discrete event simulation program in a state transition sense. The "state" of the program is defined by the values of the variables and the statement about to be executed. The format is specified using the six-tuple system of Cutler (1979), in which the program P is represented by a labeled, directed graph (G,L) and its variables are classified as either ordinary program variables X, random variables R (operated upon by a pseudorandom generator function RAND), list of pending events E, or simulated time T, all of whose possible values are given by the domain assignment function D.

Permissible statements (labels of the nodes of G) are the usual start, assignment, decision, exit (stop) statements which are analogous to those of traditional flowchart programs (Luckham, Park, and Paterson 1970) plus a special event scheduling statement. The interested reader is urged to consult Cutler (1979) for the details we will omit in order to reduce the already unwieldy number of definitions.

In this section, we define canonical sequential machines and DEVS specifications corresponding to the operational definition of the semantics of an arbitrary discrete event simulation program S. These machines M(S) and DEVS(S) define the semantics of S by the interpretive effect of executing the statements on the state of S.

Definition 18: The state machine M(S) of a discrete event simulation program $S=<P,X,R,E,T,D>$, $P=(G,L)$, is the autonomous sequential machine

$$M(S)=< GxD(X)xD(R)xD(E)xN,\ \emptyset,\ D(X)xN,\ \delta_{M(S)},\ \lambda,\ (g,x,r,e,0)>$$

where the states of M(S) are the states of S, the input alphabet is empty since M(S) is autonomous, the output alphabet is a set of ordered pairs, (g,x,r,e,RAND) is an initialization of S, and the output and transition functions are defined as follows.

The function $\delta_{M(S)}$ is defined by cases depending on the label of node g, with M(S) in state (g,x,r,e,t).

i. If g is an assignment node, then $\delta_{M(S)}(g, x, r, e, t) = (g', x', r', e, t)$

where g' is the node to which arc c points, r' = RAND(r,R*), x' = f(x,r',t).

ii. If g is a decision node, then $\delta_{M(S)}(g, x, r, e, t) = (g', x, r', e, t)$

where r'=RAND(r,R*) and g' is either the node to which arc T points if q(x,r',t)=true, or g' is the node to which arc F points if q(x,r',t)=false.

iii. If g is a START node, then $\delta_{M(S)}(g, x, r, e, t) = (g', x, r, e, t)$

where g' is the node to which arc c points.

iv. If g is an EXIT node, then $\delta_{M(S)}(g, x, r, e, t) = (g', x, r, e', t')$

if $e = (p',t')e'$ and g' is the start node of p'. If e is empty, M(S) halts in final state (g, x, r, e, t).

v. If g is a SCHEDULE node, $\delta_{M(S)}(g, x, r, e, t) = (g', x, r', e', t)$

where

g' is the node to which arc c points,

r'=RAND(r,R*),

$e'=FE(e,(P_j, f(x,r',t))$

and the future events update function

$$FE: D(E) \times P \times N \rightarrow D(E)$$

is defined by

$$FE((p_{i_1},n_1)(p_{i_2},n_2)\cdots(p_{i_m},n_m),\ (p,n)) = ((p_{i_1},n_1)\cdots(p_{i_k},n_k)\ (p,n)(p_{i_{k+1}},n_{k+1})\cdots(p_{i_m},n_m)$$

where $n_k \leq n < n_{k+1}$ or $n_k \leq n$ and $k=m$.

The output function of M(S) is simply the projection

$$\lambda((g,x,r,e,t)) = (x,t)$$

Analogous to M(S) is DEVS(S), the DEVS equivalent of S. It resembles M(S) since they share the same state set. Because a DEVS is a discrete event model, its definition omits some of the overhead of that of M(S).

Definition 19: Given a discrete event simulation program S = <P,X,R,E,T,D>, P = (G,L), and the state machine

$$M(S) = \langle S_M, \emptyset, D(X)xN, \delta_M, \lambda_M, s_0 \rangle$$

of S, define the discrete event specification

$$DEVS(S) = \langle \emptyset, S_M, D(X), \delta, \lambda, t_M, N \rangle$$

where there are no external events and therefore

$$\delta: S_M \times \{\emptyset\} \rightarrow S_M$$

with

$$\delta(s,\emptyset) = \delta_M(s)$$

and

$$\lambda(s) = s' \text{ if } \lambda_M(s)=(s',n)$$

for all s in $S_M$, and

$$t_M(g,x,r,e,t) = \begin{cases} 0 & \text{if } L(g) \notin EXIT \\ (t'-t) & \text{if } L(g)=EXIT,\ e=(p',t')e' \\ \infty & \text{if } L(g)=EXIT,\ e \text{ empty} \end{cases}$$

The correspondence is straightforward, since the definition of M(S) already includes a state set and a formal description of the state transition induced by each type of statement in P. We take the same state set for DEVS(S), and synthesize the transition function from the rules of S. Inputs to DEVS(S) are merely the exogenous events specified as $e_0$ in S, and outputs are snapshots of the values of program variables x and simulated time t of S. The time duration function $t_M$ of M is derived using the time until the next event of E.

A crucial concept to the model of discrete event simulation programs is the formal definition of what is meant by "simulation." When we deal with deterministic specifications, the machines M(S) and DEVS(S) are used to define simulation formally.

Definition 20: Let S = <P,X,R,E,T,D>, P=(G,L), be a discrete event simulation program, with

$$M(S) = \langle S_M, \emptyset, D(X)xN, \delta_M, \lambda_M, s_0 \rangle$$

$$DEVS(S) = \langle \emptyset, S_M, D(X), \delta_{DEVS(S)}, \lambda, t_M, N \rangle$$

The program S is said to simulate a deterministic system M if one of the following conditions is met.

a. $M = \langle Q, \Sigma, Y, \delta, \lambda, q_0 \rangle$ is a deterministic sequential machine and there exists a homomorphism

$$k: \Sigma \rightarrow (P \times T)^*$$

which can be extended to $\Sigma^*$, and a homomorphism

$$h: GxD(X)xD(R)xD(E)xN \rightarrow Q$$

such that to every string s in $\Sigma^*$ and state q in Q there corresponds an initialization $I = (g_0,x_0,r_0,m(s),empty)$ of S, and

$$\delta_{M(S)}(I) = \delta(h(I),s)$$

b. $M = < \Sigma, Q, Y, \delta, \lambda, t, N >$ is a DEVS with a countable time base and there exist

i. a homomorphism

$$m: \Sigma \rightarrow (P \times T)^*$$

which can be extended to $\Sigma^*$,

ii. a system specification morphism (g,h,k) from DEVS(S) to M, and

iii. for every string s in $\Sigma^*$ and state q in Q a corresponding initialization of S,

$$I(s,q) = (g_0,x_0,r_0,m(s),empty)$$

such that if

$$(q,0,s,empty) \overset{*}{\underset{M}{\vdash}} (h(q'),t',s',(k(y_1),t_1)(k(y_2),t_2)\cdots(k(y_n),t_n))$$

then

$$(I(s,q),0,g(s),empty) \overset{*}{\underset{DEVS(S)}{\vdash}} (q',t',g(s'),(y_1,t_1)(y_2,t_2)\cdots(y_n,t_n))$$

In order to simulate a deterministic sequential machine which is not autonomous, the program must initialize the events list correspondingly. The mapping k effects this initialization.

In order to simulate a DEVS, there must be a similar correspondence between the input to the DEVS and the initialization of the program. Also, there must exist a system specification morphism (Zeigler 1976) from DEVS(S) to the DEVS. These two requirements are blended into a single condition which identifies a transition (on null input) of an appropriately initialized DEVS(S) with a transition of the simulated DEVS.

The point of these alternative specifications is that the discrete event and discrete time specifications of a simulation program are not driven by input symbols. Therefore, in order to simulate arbitrary sequential machine specifications, which may have input alphabets, they must encode the input string in their initialization variables. The purest way of effecting this encoding is to utilize the events list as a storage medium for external inputs. The program can then use one procedure for each input in order to simulate the transition caused by that input.

## 5. DETERMINISTIC SIMULATION OF STOCHASTIC SPECIFICATIONS

The use of an analogous homomorphism for stochastic specifications is not acceptable since they are simulated by deterministic programs. This is where the ergodic machine representation earns its keep. The properties of ergodic machines which make them suitable for the deterministic realization of stochastic sequential machines also transfer to a class of discrete event simulation programs.

Definition 21: A discrete event simulation program

$$S=< P, X, R, E, T, D>,$$

$P = (G,L)$, is ergodic if and only if the system

$$E(S) = <PxD(T), Q, (D(R),F,Pr), D(X)xD(T), RAND^*, \delta, \lambda >$$

is an ergodic machine, where

$\delta$ and $\lambda$ are obtained from the definition of DEVS(S),

$Q = GxD(X)xD(E)xD(T)$ is a state set,

F is a field consisting of subsets of D(R),

Pr is a probability measure on F given by

$$Pr(A) = \frac{|A|}{|D(R)|} \text{ for all A in F, and}$$

for some g, x', r', e', (g,x',r',e',RAND*) is an initialization of S.

Thus, a simulation program S is ergodic if DEVS(S) meets certain conditions, the most significant of which is that RAND* is an ergodic transformation on D(R).

It might seriously be asked whether Definition 21 is vacuous, for the use of a finite seed space means that while ergodic transformations on R can exist, their properties are not what we wish (cf. Aggarwal (1975), Proposition 2.8.2). However, the practical nature of our model implies that the property of "faithfulness" upon which Aggarwal relies is inappropriate to the software simulations with which we are concerned.

Finally, we need a formal definition of simulation of probabilistic specifications analogous to Definition 20 for deterministic specifications. It is quite natural to rely on the ergodic machine E(S) for this, as we relied on the state machine M(S) for deterministic specifications.

Definition 22: A discrete event simulation program S = <P,X,R,E,T>, P = (G,L) simulates a stochastic machine M if and only if either M = < Q, X, {Z(x)}, /\ > is a stochastic sequential machine, or M = < X', Y', Q x R, t, $\delta$, \ > is an SDEVS, and

S is ergodic, with associated machine E(S),

$C_{E(S)}$ is the probabilistic automaton associated with E(S), and

M is a regular probabilistic homomorphic image of $C_{E(S)}$ (Definition 15 ).

Next, the results of Aggarwal (1975) are extended to give a necessary and sufficient condition for homomorphisms of ergodic machines to imply a covering homomorphism of their associated automata.

Let M and M' be two ergodic machines such that h is a homomorphism from M to M'.

1. $C_M$ does not necessarily cover $C_{M'}$.
2. Even if $C_M$ covers $C_{M'}$, there may exist no homomorphism from M to M' decomposable into a cross-product of mappings

$$k_1 : S \rightarrow S'$$
$$k_2 : R \rightarrow R'$$

Because this is a negative result, we need only to give counterexamples. It suffices to construct a pair of ergodic machines which are homomorphic images but whose associated probabilistic automata have no covering homomorphism and a pair of ergodic machines which are homomorphic images and whose associated probabilistic automata have a covering homomorphism but which can not be mapped by a decomposable homomorphism.

Example of Condition 1 ---

Consider two ergodic machines:

$$M=<\{0\}, S, \overline{\Pi}, Y, T, d_1, \backslash >$$

$S = \{q_1, q_2, q_3\}$ $\quad R = \{0,1,2,3\}$ $\quad T(r)=(r+1) \bmod 4$

$$M'=<\{0\}, S', \overline{\Pi}', Y', T', d_1', \backslash'>$$

$S' = \{p_1, p_2\}$ $\quad R' = \{0\}$ $\quad T'(0) = 0$

---

$d(q_1,0)=(q_1,1)$ $\qquad$ $d(q_1,2)=(q_3,3)$

$d(q_1,1)=(q_2,2)$ $\qquad$ $d(q_3,3)=(q_3,0)$

$d(q_2,0)=(q_1,1)$ $d(q_2,2)=(q_2,3)$

$d(q_2,1)=(q_2,2)$ $d(q_2,3)=(q_2,0)$

$d(q_3,0)=(q_1,1)$ $d(q_3,2)=(q_3,2)$

$d(q_3,1)=(q_1,2)$ $d(q_3,3)=(q_3,0)$

---

$d'(p_1)=p_2$ $d'(p_2)=p_1$

---

Now consider the homomorphism h.

$h(q_1,0)=p_1$ $h(q_1,1)=p_2$ $h(q_1,2)=p_1$ $h(q_1,3)=p_2$

$h(q_2,0)=p_1$ $h(q_2,1)=p_2$ $h(q_2,2)=p_1$ $h(q_2,3)=p_2$

$h(q_3,0)=p_1$ $h(q_3,1)=p_2$ $h(q_3,2)=p_1$ $h(q_3,3)=p_2$

---

And here are the associated probabilistic automata:

| S | $q_1$ | $q_2$ | $q_3$ |
|---|---|---|---|
| 1 | 0.25 | 0.25 | 0.5 |
| 2 | 0.25 | 0.5 | 0.25 |
| 3 | 0.5 | 0.25 | 0.25 |

| S' | $p_1$ | $p_2$ |
|---|---|---|
| 1 | 0 | 1 |
| 2 | 1 | 0 |

There is clearly no covering homomorphism!

Example of Condition 2 ---

Consider two ergodic machines:

$$M=\langle\{0\}, S, \Pi, Y, T, d_1, \lambda\rangle$$

$S = \{q_1, q_2, q_3\}$ $R = \{0,1,2,3\}$ $T(r)=(r+1) \bmod 4$

$$M'=\langle\{0\}, S', \Pi', Y', T', d_1', \lambda'\rangle$$

$S' = \{p_1, p_2\}$ $R' = \{0,1\}$ $T'(r) = 1-r$

---

$d(q_1,0)=(q_1,1)$ $d(q_1,2)=(q_1,3)$

$d(q_1,1)=(q_1,2)$ $d(q_1,3)=(q_1,0)$

$d(q_2,0)=(q_2,1)$ $d(q_2,2)=(q_2,3)$

$d(q_2,1)=(q_1,2)$ $d(q_2,3)=(q_2,0)$

$d(q_3,0)=(q_3,1)$ $d(q_3,2)=(q_3,3)$

$d(q_3,1)=(q_3,2)$ $d(q_3,3)=(q_3,0)$

---

$d'(p_1,0)=(p_2,1)$ $d'(p_1,1)=(p_1,0)$

$d'(p_2,0)=(p_1,1)$ $d'(p_2,1)=(p_2,0)$

---

Now consider the homomorphism h.

$h(q_1,0)=(p_1,0)$ $h(q_2,0)=(p_1,0)$ $h(q_3,0)=(p_1,0)$

$h(q_1,1)=(p_1,1)$ $h(q_2,1)=(p_1,1)$ $h(q_3,1)=(p_1,1)$

$h(q_1,2)=(p_1,0)$ $h(q_2,2)=(p_1,0)$ $h(q_3,2)=(p_1,0)$

$h(q_1,3)=(p_1,1)$ $h(q_2,3)=(p_1,1)$ $h(q_3,3)=(p_1,1)$

---

And here are the associated probabilistic automata:

| S | $q_1$ | $q_2$ | $q_3$ |
|---|---|---|---|
| $q_1$ | 0.5 | 0.25 | 0.25 |
| $q_2$ | 0.5 | 0.25 | 0.25 |
| $q_3$ | 0.5 | 0.25 | 0.25 |

| S' | $p_1$ | $p_2$ |
|---|---|---|
| $p_1$ | 0.5 | 0.5 |
| $p_2$ | 0.5 | 0.5 |

There is clearly a covering homomorphism between them ($\{q_2, q_3\}$ is one block, and $\{q_1\}$ is the other block). We can, in fact, show that no homomorphism from M to M' is decomposable. How would this hypothetical homomorphism operate on S? It must map exactly two states to one state of S', because otherwise it would either not be onto S' or not be well defined. We can exhaust the possible mappings one by one. If we map $q_1$ and $q_2$ together, then look at $(q_1,0)$ and $(q_2,0)$, which are mapped to the same state in S'. $d(q_1,0)$ and $d(q_2,0)$ must then be mapped to the same state in S', but they can not be since their first components are $q_3$ and $q_2$, respectively. If we map $q_1$ and $q_3$ together, then look at $(q_1,0)$ and $(q_3,0)$, which are mapped to the same state in S'. $d(q_1,0)$ and $d(q_3,0)$ must then be mapped to the same state in S', but they can not be since their first components are $q_3$ and $q_2$, respectively. If we map $q_2$ and $q_3$ together, then look at $(q_2,1)$ and $(q_3,1)$, which are mapped to the same state in S'. $d(q_2,1)$ and $d(q_3,1)$ must then be mapped to the same state in S', but they can not be since their first components are $q_1$ and $q_3$, respectively.

One may begin to wonder if there is any strong relationship between covering and p-homomorphisms. That is, if two ergodic machines have associated probabilistic automata which are homomorphic images, must there be a homomorphism between them? The answer is simply that even this weak condition may not hold. For instance, the seed spaces may be incomparable, with R' much larger than R x S. In that case, no mapping could be a homomorphism since the ergodic property of T' would prevent it.

Because of these results, "simulation" and "covering" must meet the regularity condition in order to be equivalent. The proof of this property is the remaining result. We assume that we have been given an ergodic discrete event simulation program and a probabilistic machine specification. Then consider the probabilistic automaton, $C_E$, associated with the ergodic discrete event simulation program.

Given an SDEVS $C = \langle Q, X', \{Z(x)\}, Y', \Lambda \rangle$ and an ergodic discrete event simulation program $S=\langle P,X,R,V,T,D\rangle$, $P=(G,L)$, whose ergodic machine is E, S simulates C if and only if $C_E$ covers C.

Since S is ergodic (with associated machine E), the result reduces to an equivalence of the covering and regular probabilistic homomorphic relations (from the definition of simulation). That is, $C_E$ covers C if and only if there exists an ergodic machine $E_C$ representing C such that $E_C$ is a regular probabilistic homomorphic image of E.

If part: Follows directly from the definition of simulation and Theorem 2.5.1 of Aggarwal (1975).

Only if part: Suppose $C_E$ covers C. Then a block i of C corresponds to a set of states $C_i=\{i_1,\ldots,i_n\}$ with $i_j=(s_{ij},r_{ij})$.

We represent E as $\langle X, S, (R,F,P), Y, RAND^*, \delta, \lambda \rangle$

and construct $E_C$ as $\langle X, S', (R,F,P), Y, RAND^*, \delta', \lambda' \rangle$

such that $S' = \{C_1,\ldots,C_n\}$

$\delta'(C_i, r_{ij}, x) = (C_k, RAND^*(r_{ij}))$ whenever $\delta(s_{ij}, r_{ij}, x) = (s_{km}, RAND^*(r_{ij}))$

for each input symbol x, $0<i,j\leq q$, and

$$\lambda'(C_i, r_{ij}) = \lambda(s_{ij}, r_{ij}).$$

Thus, we can define the homomorphism

$$h((s_{ij},r_{ij})) = (C_i, r_{ij})$$

which maps states to blocks and is isomorphic on the seed spaces. Then

$$h(\delta(s_{ij}, r_{ij}, x) = h((s_{km}, RAND^*(r_{ij})) = (C_k, RAND^*(r_{ij}))= \delta'(C_i, r_{ij}, x) = \delta'(h((s_{ij}, r_{ij}, x)))$$

and

$$\lambda'(h((s_{ij}, r_{ij})) = \lambda'(C_i, r_{ij}) = \lambda(s_{ij}, r_{ij}).$$

Therefore, $E_C$ is a p-homomorphic image of E. □

## 6. CONCLUSIONS

Existing discrete time sequential machine specifications have been shown to have within them the seeds of discrete event specifications, and a flowchart model for simulation programs has been shown to be equivalent in power and behavior to these specifications. By the use of the ergodic machine formalism, a mechanism has been found by which programs simulating stochastic systems can be verified. A necessary and sufficient condition for this mechanism has been proved. These results constitute a broadly applicable facilitating contribution to techniques for the mathematical verification of simulation models against sequential machine specifications.

## ACKNOWLEDGEMENT

The author would like to give special thanks to Sudhir Aggarwal, who suggested the approach to the simulation of stochastic specifications, and to Sheila Greibach, whose aid in the development of the notation was invaluable. The assistance of Jack Carlyle in placing the ergodic machine model in the proper frame of reference of stochastic specifications was also an important philosophical contribution.

## REFERENCES

Aggarwal, S. (1975), "Ergodic Machines--Probabilistic and Approximate Homomorphic Simplifications," PhD Dissertation, The University of Michigan, 190 pp.

Aggarwal, S. (1977), "Deterministic Representation of Probabilistic Systems by Ergodic Machines," Mathematical Systems Theory 10, pp. 345-361.

Billingsley, P. (1965), Ergodic Theory and Information, Wiley, New York, pp. 1-19.

Cutler, M. M. (1979), "Proving Properties of Simulation Programs for System Verification and Validation," Proceedings Summer Computer Simulation Conference, pp. 610-616.

Halmos, P. R. (1956), Lectures on Ergodic Theory, The Mathematical Society of Japan, Tokyo, 99 p.

Luckham, D. C., Park, D. M., and Paterson, M. S. (1970), "On Formalised Computer Programs," Journal of Computer and System Sciences 4, pp. 220-249.

Melamed, B. (1976), "Analysis and Simplifications of Discrete Event Systems and Jackson Queuing Networks," PhD Dissertation, The University of Michigan, 318 pp.

Zeigler, B. P. (1976), Theory of Modeling and Simulation, John Wiley, New York, 435 p.