Structural and Behavioral Equivalence of Simulation Models

ENVER YÜCESAN
European Institute of Business Administration
and
LEE SCHRUBEN
Cornell University

It is sometimes desirable to know when two different discrete-event simulation models are, in some sense, interchangeable; that is, whether or not the two models always have the same output when run under identical experimental conditions. This notion of behavioral equivalence, while conceptually simple, is difficult to define in a manner that is both useful and testable. It is difficult or impossible to assert that two simulations are behaviorally equivalent for all possible experiments. In this paper, we present an explicit and sensible definition of behavioral equivalence. Unfortunately, like other definitions in the literature, our definition is not testable in practice.

However, using a general graph-theoretic specification of simulation models, which we call Simulation Graphs, we define a testable property related to behavioral equivalence which we refer to as structural equivalence. We then establish that structural equivalence (a testable property) implies behavioral equivalence (a meaningful property). This permits us to assess when it is safe to substitute one model for another. It then becomes possible to develop algorithms for addressing important problems in simulation model development and verification.

Categories and Subject Descriptors: D.1.7 [**Programming Techniques**]: Visual Programming; I.6.1 [**Simulation and Modeling**]: Simulation Theory—*model classification*; I.6.5 [**Simulation and Modeling**]: Model Development—*modeling methodologies*

General Terms:

Additional Key Words and Phrases: Concurrent programming, discrete-event simulation, event graphs, network structures

INTRODUCTION

The ability to identify equivalent simulation models is desirable from various points of view. For instance, logistical considerations such as ease of implementation on a computer, data requirements, or suitability for the intended

This work was done while Lee Schruben held a National Research Council-Naval Postgraduate School Research Associateship.
Authors' addresses: L. Schruben, School of Operations Research and Industrial Engineering, Cornell University, Ithaca, NY 14853; E. Yücesan, European Institute of Business Administration, INSEAD, Boulevard de Constance, 77305 Fontainebleau Cedex, France.

application may point out that it is more advantageous to use one model over another once their equivalence is established. Another possible scenario is when an existing model is embellished, and it needs to be verified whether or not the underlying structure of parts of both models are the same.

Even though the idea of equivalent simulations is conceptually simple, it is difficult to formalize in an acceptable fashion since any practically useful definition must be *testable* also. That is, it should be possible to identify when two simulation models can be used interchangeably without actually having to run both simulations under all possible experimental conditions and compare their output behavior.

Various definitions of equivalence have appeared in the literature. Overstreet [11] offers three such definitions. *Structural equivalence* is concerned with the objects in a model and their impact on model attributes. For two models to be structurally equivalent, they must have the same objects, and the impact of these objects on model attributes should be the same in both models. *External equivalence* is concerned with the input/output behavior of models. Two models are externally equivalent if they exhibit the same output behavior whenever provided with the same initial conditions. *Derivability* is similar to external equivalence. In this case the outputs need not be similar. However, the output behavior of one model must be derivable from that of the other model for them to be qualified as equivalent.

Schruben [16] offers a definition of "equivalence" based on the behavior of conditioning state variables over time. Sargent's [14] Type I and Type II "equivalences" are based also on the behavior of model variables during execution. We will collectively refer to such definitions as *behavioral.*

Although behavioral-equivalence definitions are intuitively appealing, in general they are not testable. That is, it is highly unlikely that an algorithm can be constructed to determine whether *any* two simulation models are behaviorally equivalent. In this paper, we introduce a definition of equivalence that uses mathematical properties of a simulation model's *structure*. In so doing, we shift the emphasis from model behavior to model structure and produce a testable definition. We also show that structural equivalence (defined explicitly and *differently* from Overstreet) is a sufficient, but not a necessary, condition for behavioral equivalence.

Our development is based on the concept of a *Simulation Graph Model* and graph isomorphism. This is a mathematically explicit formalization of "event graphs" introduced by Schruben [16] to represent the event-scheduling approach in discrete-event simulation. Simulation graphs are introduced next. The rest of the paper is organized as follows: Section 2 defines simulation model equivalence. Sections 3 and 4 illustrate applications. Concluding comments are included in Section 5.

1. SIMULATION GRAPH MODELS

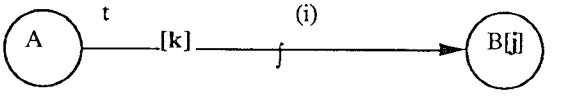
The elements of a discrete-event simulation are *state variables* that describe the state of the system, *events* that alter the values of state variables, and the *logical* and *temporal relationships* between events. An *event graph* [16] is a structure of these elements in a discrete-event system that facilitates the

development of a correct simulation model. Events are represented on the graph as vertices (nodes). Each vertex is associated with a set of changes to state variables. These variables are used to describe system entities. Hence, the emphasis is directly on system events; system entities are represented implicitly.

Relationships between events are represented in an event graph as directed edges (arcs) between pairs of vertices. Each edge is associated with sets of logical and temporal expressions. Two types of edges are distinguished. *Scheduling* edges appear as solid arcs on the graph while *canceling* edges are depicted as dashed arcs. Basically, the edges define under what conditions and after how long of a time delay an event will schedule or cancel another event. There can be multiple edges between any pair of vertices; the edges can point in either direction or may simply point from a vertex to itself.

In an event graph, it is possible also to parameterize the event vertices. Event parameterization is an important modeling convenience but does not increase the generality of these graphs. Parameters keep the graphs from becoming cluttered or possibly infinite [24]. This is achieved through *vertex parameters* and *edge attributes*. A *vertex parameter list* is a vector of state variables associated with a particular vertex. An *edge attribute list*, on the other hand, is a set of expressions associated with a particular edge. These lists are used in scheduling or canceling specific instances of events. For example, in a simulation model depicting the operation of a group of machines, a single *start_processing* event vertex together with parameters might be used to model the start of processing on any of the machines in the group; the value of the vertex parameter designates the particular machine involved. This practice is analogous to passing values to a subroutine using a list of arguments in a high-level programming language. In this paper, we focus solely on parameters that assume a finite or countable number of values.

In summary, the construct



is interpreted as follows: *whenever event A occurs, if condition (i) holds, event B is scheduled to occur in t time units with the parameter vector, j, assuming the current values of the expressions in k.*

The major advantage of this modeling framework is that the network provides a complete and consistent environment for model development [19]. No new icons or constructs need to be defined as the modeling requirements change. In addition, this framework offers practical guidance not only for model specification, but also for model implementation. For instance, the customary notion of an event routine (see, for example, Simscript [2]) in a simulation program may correspond to a subgraph, typically a set of vertices connected by edges with no delays. A simple example is presented next.

Example: *Single-server queueing model.* An event graph for a single-server queueing system will be developed with the objective of estimating the

time customers spend in the system [17]. Suppose (1) that customers arrive every t_a time units and (2) it takes t_s time units to attend to each customer. The state variables used in this model are defined as follows:

—Q represents the number of customers waiting for service
—S denotes the status of the server with 0 = busy and 1 = idle
—id denotes the customer identification number
—in represents the identification number of the customer currently *in* service
—nx is the identification number of the customer who is *next* in line
—$W[i]$ is the total time customer i spends in the system
—CLK represents the current simulation clock.

The edge conditions for the model are:

(1) (The server is idle): $S == 1$
(2) (Customers are waiting to be served): $Q > 0$.

The event descriptions are presented in Table I while the event graph is depicted in Figure 1.

Several variations of event graphs have appeared in the literature. Pegden [12], Hoover and Perry [7], Law and Kelton [8], and Schruben [17] use these networks to construct discrete-event simulations.

Extensions of event graph analysis have been introduced also. Sargent [14] presents some rules for detection of simultaneously scheduled events and for event reduction. Also, he models a flexible manufacturing system with event graphs. In a more recent paper, Som and Sargent [22] define rules for identifying simultaneously scheduled events and assigning execution priorities. They also introduce a conceptual algorithm for event reduction.

Simulation graphs are a mathematically explicit definition of event graphs. The development is not limited to the event-scheduling world view but targets discrete-event dynamic systems in general. For example, a transformation procedure, the geometric dual, is shown to yield equivalent representations of queueing models under both event-scheduling and activity-scanning world views [21].

Our definition of simulation graphs uses ideas from graph theory. A directed graph, G, is characterized as an ordered triple $(V(G), E(G), \Psi_G)$ consisting of a nonempty set of vertices, $V(G)$, a set $E(G)$, disjoint from $V(G)$, of edges, and an incidence function, Ψ_G, that associates with each edge of G an ordered pair of (not necessarily) distinct vertices of G [1]. A network is defined then as a graph in which additional data are stored in the vertices and edges [9]. A *simulation graph* is defined as an ordered quadruple

$$G = (V(G), E_s(G), E_c(G), \Psi_G)$$

where $V(G)$ is the set of event vertices, $E_s(G)$ is the set of scheduling edges, $E_c(G)$ is the set of canceling edges, and Ψ_G is the incidence function. The data defined on this graph are the following indexed sets:

$\mathcal{F} = \{f_v : \text{STATES} \rightarrow \text{STATES} | v \in V(G)\}$, the set of *state transition functions*

Table I. Event Descriptions

Event Type	Event Description	State Changes
INIT	Initialization	$Q \leftarrow 0$ $S \leftarrow 1$ nx $\leftarrow 1$
ARV	Customer Arrival	$Q \leftarrow Q + 1$ id $\leftarrow$ id $+ 1$ W[id] $\leftarrow$ CLK
BGN	Beginning of Service	$S \leftarrow 0$ $Q \leftarrow Q - 1$ nx $\leftarrow$ nx $+ 1$
END	End of Service	$S \leftarrow 1$ W[in] $\leftarrow$ CLK $- W$[in]

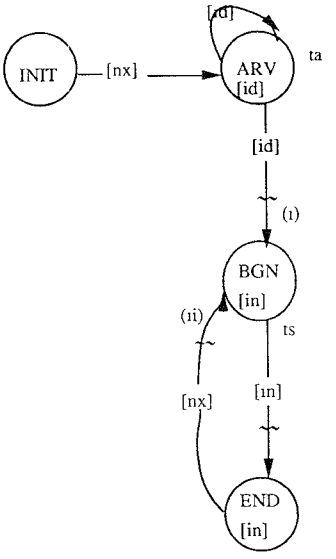


Fig. 1. Single-server queueing model.

associated with vertex v,

$\mathscr{C} = \{C_e : \text{STATES} \rightarrow \{0, 1\} | e \in E_s(G) \cup E_c(G)\}$, the set of *edge conditions*,

$\mathscr{T} = \{t_e : \text{STATES} \rightarrow \mathscr{R}^+ | e \in E_s(G)\}$, the set of *edge delay times*,

$\Gamma = \{\gamma_e : \text{STATES} \rightarrow \mathscr{R}^+ | e \in E_s(G)\}$, the set of *event execution priorities*,

where STATES is defined as in Zeigler [25, p. 62]; $\mathscr{R}^+$ denotes the set of nonnegative real numbers. Note that we define event execution priorities as mappings from the model state to the nonnegative real numbers that depend on the *edge* that schedules an event vertex. This generalization of the usual notion of a priority ranking for tie breaking in simultaneous events can be very useful as illustrated by the example in [18].

A *simulation graph model* (SGM) is defined as:

$$\mathscr{S} = (\mathscr{F}, \mathscr{C}, \mathscr{T}, \Gamma, G).$$

The first four sets in the above five-tuple define the entities in a model. The role played by the simulation graph, G, in the definition of a simulation graph model, $\mathcal{S}$, is analogous to the role of the incidence function, Ψ, in the definition of a directed graph; it organizes these sets of entities into a meaningful simulation model specification. That is, G specifies the relationships between the elements in the sets $\mathcal{F}$, $\mathcal{C}$, $\mathcal{T}$, and Γ.

Example (continued). The entities in the network depicting a single-server queueing system are defined as follows (G is presented in Figure 1):

$\mathcal{F} = \{f_{\text{INIT}}; f_{\text{ARV}}; f_{\text{BGN}}; f_{\text{END}}\} = \{Q \leftarrow 0,\ S \leftarrow 1,\ \text{nx} \leftarrow 1;\ Q \leftarrow Q + 1,\ \text{id} \leftarrow \text{id} + 1,\ W[\text{id}] \leftarrow \text{CLK};\ S \leftarrow 0,\ Q \leftarrow Q - 1,\ \text{nx} \leftarrow \text{nx} + 1;\ S \leftarrow 1,\ W[\text{in}] \leftarrow \text{CLK} - W[\text{in}]\}$.

$\mathcal{C} = \{C_{\text{ARV, BGN}}; C_{\text{END, BGN}}\} = \{\text{S} = 1; Q > 0\}$.

$\mathcal{T} = \{t_{\text{ARV, ARV}}; t_{\text{BGN, END}}\} = \{t_a; t_s\}$.

$\Gamma = \{\gamma_{\text{INIT, ARV}}; \gamma_{\text{ARV, ARV}}; \gamma_{\text{ARV, BGN}}; \gamma_{\text{BGN, END}}; \gamma_{\text{END, BGN}}\} = \{2; 2; 1; 2; 1\}$.

2. EQUIVALENCE OF SIMULATION MODELS

2.1 Preliminary Definitions

In an SGM, an edge condition will be called *simple* if it consists of two arithmetic expressions connected by a relational operator. In other words, a simple edge condition is a relation. The arithmetic expressions may simply be a constant or a variable, whereas the relational operators are "less than," "less than or equal to," "equal to," "not equal to," "greater than," and "greater than or equal to." On the other hand, the edge condition will be called *compound* if it consists of two or more relations joined by Boolean operators AND or OR. For example, ($QSIZE > 0$) is a simple condition while ($QSIZE > 0$) AND ($S = 1$) is a compound edge condition.

Similarly, a vertex will be called *simple* if there is at most one state variable change associated with it. In other words, vertex v is a simple event vertex if its execution alters the value of at most one state variable. Otherwise, the vertex will be called a *compound* vertex. A vertex with no state variable changes will be referred to as the *identity* vertex.

A simulation graph model, $\mathcal{S}$, will be called an *Elementary Simulation Graph Model* (ESGM) and denoted $\mathcal{S}^E$ if it contains only simple event vertices and if all edge conditions are simple. Given a simulation graph model, $\mathcal{S}$, an associated ESGM, $\mathcal{S}^E$, can always be constructed by *expansion*. This is the process of replacing a single event vertex with m ($m \geq 1$) state variable changes by m vertices in series, each with a single state variable change. It is also the process of replacing an edge with a compound condition by a group of identity vertices, each connected by edges with simple conditions. During this process, the logical structure of the original model is always preserved. The procedures for graph expansion are defined in the Appendix. It should also be noted that the application of expansion rules does not generate a unique elementary simulation graph. However, all of the elementary graphs generated in this fashion are isomorphic to one another; hence, they form an equivalence class [1].

Two simulation graph models $\mathcal{S}_1 = (\mathcal{F}_1, \mathcal{E}_1, \mathcal{T}_1, \Gamma_1, G_1)$ and $\mathcal{S}_2 = (\mathcal{F}_2, \mathcal{E}_2, \mathcal{T}_2, \Gamma_2, G_2)$ are called *isomorphic* and denoted $\mathcal{S}_1 \approx \mathcal{S}_2$, if there exist one-to-one and onto mappings (bijections) of the form:

$$\Theta : V(G_1) \rightarrow V(G_2)$$
$$\Phi_s : E_s(G_1) \rightarrow E_s(G_2)$$
$$\Phi_c : E_c(G_1) \rightarrow E_c(G_2)$$
$$\Lambda : \mathcal{F}_1 \rightarrow \mathcal{F}_2$$
$$\Omega : \mathcal{E}_1 \rightarrow \mathcal{E}_2$$
$$\chi : \mathcal{T}_1 \rightarrow \mathcal{T}_2$$
$$\Delta : \Gamma_1 \rightarrow \Gamma_2$$

The mappings $(\Theta, \Phi_s, \Phi_c, \Lambda, \Omega, \chi, \Delta)$ form an *isomorphism* [1]. Note that the first three mappings basically establish a correspondence between the underlying structure of the two models. Simply stated, the mappings (Θ, Φ_s, Φ_c) indicate that names given to entities in a simulation graph are not important since isomorphic graphs form an equivalence class.

The determination of graph isomorphism has drawn considerable attention and has important implications in various fields. For instance, the problem of determining whether two chemical compounds are identical can be reduced to the isomorphism problem where the compounds are depicted as graphs with vertices representing an atom or a group of atoms and edges representing chemical bonds. The isomorphism problem is intractable for general graphs [3]. However, efficient algorithms exist for determining isomorphism of planar graphs [4, 5]. (Efficient planarity testing is also possible [6].) The remaining four mappings are examined in more detail next.

$\Lambda : \mathcal{F}_1 \rightarrow \mathcal{F}_2$. This mapping establishes a correspondence between the state changes associated with different event vertices on the two simulation graphs. For two state transition functions to be declared equivalent, their application to the same state variables must yield identical outcomes. For example, the state transition functions $\{Y \leftarrow 2^*X\}$ and $\{Y \leftarrow X + X\}$ are equivalent. Also, it is required that the state variables have the same range. In cases where these functions have stochastic components, the associated random variables should follow the same probability distribution.

$\Omega : \mathcal{E}_1 \rightarrow \mathcal{E}_2$. This mapping establishes a correspondence between the edge conditions of the two models. The form of these conditions may not necessarily be identical. It is required, however, that they assume the same Boolean value when evaluated with a given state of the model. For instance $(QSIZE > 0)$ and $(Q \geq 1)$ are equivalent edge conditions if Q is always equal to $QSIZE + 1$. If the conditions have any stochastic components, the associated random variables should follow the same probability distribution.

$\chi : \mathcal{T}_1 \rightarrow \mathcal{T}_2$. This mapping establishes a correspondence between the edge delay times of the two models. If the times are deterministic, they are required to have the same value. If they are stochastic, they are required to follow the same probability distribution.

$\Delta : \Gamma_1 \rightarrow \Gamma_2$. This mapping establishes a correspondence between the event execution priorities in the two models. Recall that priorities are nonnegative real numbers. It is required that the same relative order of values be assigned to the respective events for a given state of the model.

It is also possible to define isomorphisms with respect to a subgraph and a subset of the state variables. This provision allows the modeler to focus only on those components of the model that are relevant within the scope of the current simulation study. The following definition of structural equivalence should be understood to hold with respect to a particular subset of state variables and a particular subgraph.

2.2 Structurally Equivalent Simulation Models

The concepts of graph isomorphisms and SGMs are used now to define structural equivalence of simulation models.

Definition. Simulation graph models, $\mathscr{S}_1$ and $\mathscr{S}_2$, are *structurally equivalent* if they have any elementary simulation graph models, $\mathscr{S}_1^E$ and $\mathscr{S}_2^E$, respectively, which are isomorphic.

To determine whether two simulation models are equivalent, the graphs are expanded into ESGMs, and an isomorphism between any pair of the elementary graphs is sought. If such an isomorphism is found, then the two simulation models are declared structurally equivalent. Efficient algorithms exist for establishing graph isomorphism for planar graphs. (The surprising fact that simulation graphs have planar representations is established in [23].) Hence, this structural definition of equivalence is indeed a testable definitions as opposed to definitions based on model behavior since, in the absence of vertex parameters, the number of isomorphic ESGs that can be generated by expanding an SGM is finite [23]. We should note, however, that establishing the mappings Λ, Ω, χ, and Δ can be a difficult—but, nevertheless, possible—task for large models.

Next, we show that structural equivalence defined by graph isomorphism is a sufficient but not a necessary condition for behavioral equivalence. Several preliminary definitions are presented first. An *experimental frame* "specifies a limited set of circumstances under which a system (real system or model) is to be observed or subjected to experimentation" [25]. This is understood to describe the initial conditions of the model implementation including the initial values of the state variables as well as seeds for random number generators and the termination conditions of the model implementation including the stopping time or condition as well as the state variables (and performance measures) to be monitored. The *execution* of an event vertex entails the following: the update of vertex parameters, if any; the update of the values of the associated state variables; the evaluation of all edge conditions; the evaluation of all attributes on those exiting edges where edge conditions are satisfied; the scheduling and/or cancelation of further events. The *occurrence* of an event represents the execution of the corresponding event vertex.

Behavioral equivalence is defined next. In a discrete-event simulation, events occur at discrete points in (simulated) time. Let

$$T(\mathscr{E},\mathscr{S}) = \{t_1, t_2, \ldots, t_n\}$$

be the partially ordered set of points in (simulated) time of event occurrences, for an execution of model $\mathscr{S}$ with experimental frame, $\mathscr{E}$. Analogously, let

$$S(\mathscr{E},\mathscr{S}) = \{S_1, S_2, \ldots, S_n\}$$

be the ordered set of states indexed by event occurrences for an execution of the model, $\mathscr{S}$, with experimental frame, $\mathscr{E}$. The sample path observed during a run can be represented then by the ordered pairs (t_k, S_k) for $k = 0, 1, 2, \ldots, n$.

Definition. Two simulation models $\mathscr{A}$ and $\mathscr{B}$ are behaviorally equivalent with respect to a subset of state variables if, within all experimental frames, $\mathscr{E}$,

$$(1)\ T(\mathscr{E},\mathscr{A}) \approx T(\mathscr{E},\mathscr{B}) \quad \text{and} \quad (2)\ S(\mathscr{E},\mathscr{A}) \approx S(\mathscr{E},\mathscr{B}).$$

The following theorem asserts that structural equivalence implies behavioral equivalence.

THEOREM. *If two SGMs are structurally equivalent then they must be behaviorally equivalent.*

PROOF. (By contraposition). Suppose two SGMs $\mathscr{S}_1$ and $\mathscr{S}_2$ are not behaviorally equivalent. That is, there exists an experimental frame where condition (1) and/or condition (2) of the definition fail to hold. Suppose condition (2) fails, and it is possible to construct all of the mappings, $\Theta, \Phi_s, \Phi_c, \Omega, \chi, \Delta$, except for Λ. That is, while the two ESGs are isomorphic and the edge conditions, event times, and event execution priorities are identical in the two models, the set of states observed under the two models are different. This is possible only if the occurrence of a particular event, say event vertex u, takes the two models into two different states. In turn, this outcome is a direct consequence of the difference between the two state transition functions associated with vertex u, f_u, in the two models. Therefore, when condition (2) fails to hold, it is not possible to construct the mapping Λ, and thus, an isomorphism cannot be established between the two ESGMs.

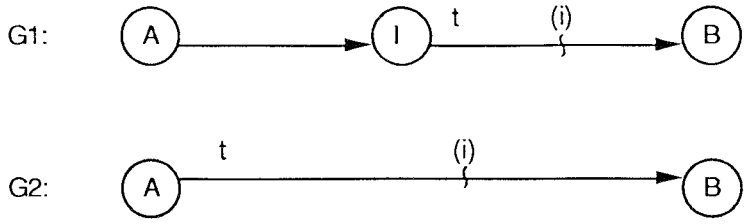
Alternatively, suppose condition (1) fails to hold and suppose it is possible to construct all the mappings, $\Theta, \Phi_s, \Phi_c, \Lambda, \Omega, \Delta$, except for χ. This implies that while the two ESGs are isomorphic and the state transitions, edge conditions, and execution priorities are identical in the two models, the event epochs are different. This discrepancy is attributable directly to the edge delay times in the two models. Recall that the edge delay times are defined as a mapping from the current state to the nonnegative real numbers (t_e: STATES $\rightarrow \mathscr{R}^+$). Thus, there exist two cases in which the event epochs can be different in the two models making it impossible to establish the mapping χ.

Case 1. The edge delay times (the mapping, t_e) are defined differently in the two models. In this case χ clearly fails to hold.

Case 2. The mappings t_e are defined similarly in both models; however, they operate on different states. This is only possible if the occurrence of a particular event takes the two models into two different states. This, in turn, is directly due to the differences between the state transition functions, resulting in the failure of the mapping Λ. Since this outcome contradicts our initial assumption that Λ holds, we conclude that this case is not possible. Therefore, when condition (1) fails to hold, it is not possible to construct the mapping χ, and thus, an isomorphism cannot be established between the two ESGMs.

In each of the above cases, it is not possible to establish the desired isomorphism. Since the ESGMs $\mathscr{S}_1^E$ and $\mathscr{S}_2^E$ are not isomorphic with respect to a particular subset of state variables, then SGMs $\mathscr{S}_1$ and $\mathscr{S}_2$ are not structurally equivalent with respect to that subset of state variables. □

Although it is a testable definition, structural equivalence is only a sufficient condition for behavioral equivalence. A simple example where two models are behaviorally but not structurally equivalent is given by models having the following subgraphs, where vertex I is an identity vertex included in G_1 possibly to make the structure of the model more "transparent." Another example is when the two models have a different number of state variables. In this case, however, there are two possible ways of establishing model equivalence. One possibility is to define it with respect to a subset of state variables (perhaps, with respect to those variables that are relevant within the scope of the study).



Alternatively, we may try first to eliminate unnecessary state variables through the application of Schruben's rule of thumb [16] and subsequently seek an isomorphism between the two models.

Various applications of structural equivalence are illustrated in the following sections. In particular, Section 3 establishes the equivalence between the models of the same single-server queueing system implemented in different world views. Schruben's [16] event reduction rules are revised in Section 4.

3. MODELS IN DIFFERENT WORLD VIEWS

In discrete-event simulation, world views provide alternative approaches to organizing a specification of model behavior. These modeling perspectives are commonly referred to as event scheduling, process interaction, and activity

scanning. Even though each of these world views are supported by one or more simulation programming languages, no universally accepted definitions exist for any of the approaches. This is partly due to the fact that this classification scheme is neither mutually exclusive nor collectively exhaustive. Moreover, the dividing line between these perspectives are fading as simulation programming languages traditionally known to follow a particular world view are adopting constructs of other world views. (Nance [10] clarifies the differences between these perspectives.) Therefore, it is desirable to be able to identify equivalent simulation models constructed under "different" world views. The concept of structural equivalence is useful for such an identification. As an illustration, the equivalence between two single-server queueing models is established. Suppose that the first model is constructed under the process orientation in GPSS [15] while the second model is constructed under the event-scheduling approach [13].

The system operates as follows: customers arrive to the facility every t_a time units to receive service which, in turn, takes t_s time units to complete. A GPSS implementation from [7] is depicted in Table II. Letters in boldface are GPSS keywords.

The process orientation can be described as follows:

> "simulation models include sequences of elements which occur in defined patterns. The logic associated with such a sequence of events can be generalized and defined by a single statement. (. .) A process-oriented language employs such statements to model the flow of entities through a system. These statements define a sequence of events which are automatically executed by the simulation language as the entities move through the process" [13, p. 57].

Figure 2 is a simulation graph depicting the implementation of the GPSS model in Table II. The state variables used in this simulation graph model are defined as follows:

—LINE denotes the number of customers in the queue waiting for service
—SERVER denotes the status of the server with 0 = busy and 1 = idle.

The edge conditions are:

—(Server is idle) SERVER == 1
—(Queue is not empty) LINE > 0.

The event descriptions are presented in Table III.

The event-scheduling representation of the queueing system, on the other hand, requires that the epochs at which the system undergoes state changes be identified and the state changes associated with those epochs be represented as a separate subroutine [13]. The simulation graph model under the event orientation is presented in Figure 3. This SGM is a simplified version of the model presented in the example of Section 1, and it is already expanded into an ESGM. The state variables used in this model are defined as follows:

—Q represents the number of customers waiting for service
—S denotes the status of the server with 0 = busy and 1 = idle.

Table II. Single-Server Queueing Model in GPSS

Simulate		
Generate	t_a	Customers arrive
Queue	Line	Customers queue for service
Seize	Server	Capture server
Depart	Line	Leave queue
Advance	t_s	Conduct service
Release	Server	Free server
Terminate	1	Customer leaves
Start	100	No. of customers to process
End		

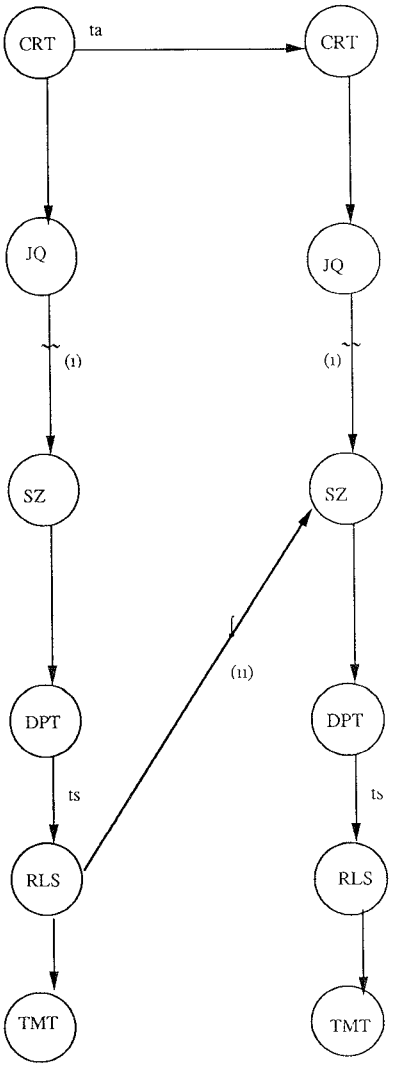


Fig. 2. GPSS model for two customers.

Table III. Event Descriptions for the GPSS Model

Event Type	Event Description	State Changes
GRT	Generate a new customer	
JQ	Join the queue	LINE ← LINE + 1
SZ	Capture server	SERVER ← 0
DPT	Leave the queue	LINE ← LINE − 1
RLS	Free the server	SERVER ← 1
TMT	Customer departs	

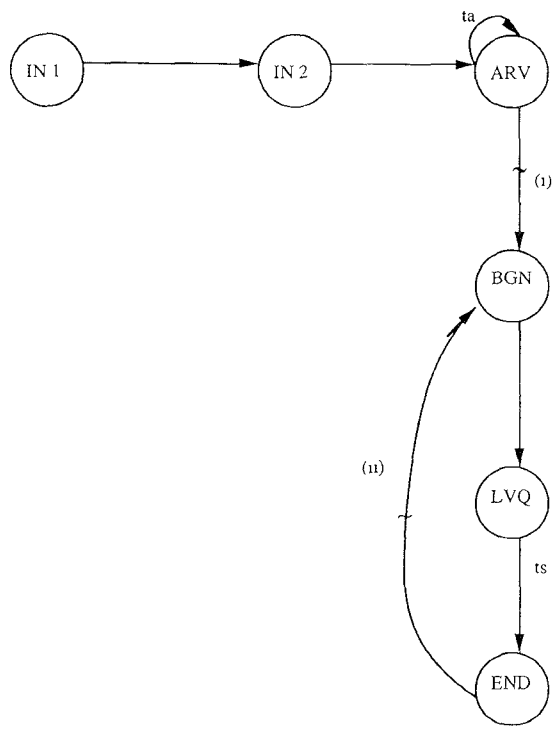


Fig. 3. Elementary simulation graph event-scheduling approach.

The edge conditions for the model are:

—(Server is idle) $S == 1$
—(Customers are waiting to be served) $Q > 0$.

The event descriptions are presented in Table IV.

There are differences in implementation between the two world views. Figure 2 depicts an implementation for *two customers*, where the subgraph containing vertices GRT, JQ, SZ, DPT, RLS, and TMT along with the edges incident to them represent a "customer path." While this approach automati-

Table IV. Event Descriptions for the Elementary Simulation Graph Model

Event Type	Event Description	State Changes
IN1	Initialization	$Q \leftarrow 0$
IN2	Initialization	$S \leftarrow 1$
ARV	Customer arrival	$Q \leftarrow Q + 1$
BGN	Begin service	$S \leftarrow 0$
LVQ	Leave the queue	$Q \leftarrow Q - 1$
END	End of service	$S \leftarrow 1$

cally keeps track of the identity of individual customers, it requires a considerable amount of memory to do so [20]. During the execution of such models, these paths may be thought of as being created and destroyed *dynamically* as transient entities enter and leave the model. Figure 2 should be viewed as *a snapshot of this dynamic environment.*

Within the event-scheduling approach, if each transient entity is to be monitored individually, an ID number for each customer is represented explicitly through vertex parameters and edge attributes as depicted in Figure 1. In fact, a simulation graph is actually an *array* of graphs, one copy for each customer in the system.

Due to these differences in implementation between the two modeling world views, these models are equivalent only *with respect to resident entities*: *the server and the queue.* An isomorphism with respect to the resident entities is sought between the two models. This is given by:

$\Theta : V(G_1) \rightarrow V(G_2)$.
$\Theta(\text{JQ}) = \text{ARV}$.
$\Theta(\text{SZ}) = \text{BGN}$.
$\Theta(\text{DPT}) = \text{LVQ}$.
$\Theta(\text{RLS}) = \text{END}$.

$\Phi_s : E_s(G_1) \rightarrow E_s(G_2)$. See Figures 2 and 3.

$\Phi_c : E_c(G_1) \rightarrow E_c(G_2)$. No canceling edges are used in the models.

$\Lambda : \mathscr{F}_1 \rightarrow \mathscr{F}_2$.

$\Lambda(f_{\text{JQ}}) = f_{\text{ARV}}$ {LINE $\leftarrow$ LINE + 1}.
$\Lambda(f_{\text{SZ}}) = f_{\text{BGN}}$ {SERVER $\leftarrow$ 0}.

$\Lambda(f_{\text{DPT}}) = f_{\text{LVQ}}$ {LINE $\leftarrow$ LINE $-$ 1}.
$\Lambda(f_{\text{RLS}}) = f_{\text{END}}$ {SERVER $\leftarrow$ 1}.

$\Omega : \mathscr{C}_1 \rightarrow \mathscr{C}_2$.
$\Omega(i) = i$.
$\Omega(i) = ii$.

$\chi : \mathscr{T}_1 \rightarrow \mathscr{T}_2$. $\chi(t_a) = t_a$. $\chi(t_s) = t_s$.

$\Delta : \Gamma_1 \rightarrow \Gamma_2$. Although it is not possible to assign event execution priorities in SLAM, it is possible to do so with some other simulation languages (e.g.,

Simscript [2]) in a straightforward fashion. In GPSS, entities created by the GENERATE block can also be assigned transaction priorities.

Since $(\Theta, \Phi_s, \Phi_c, \Lambda, \Omega, \chi, \Delta)$ is the desired isomorphism, we conclude that the two models are equivalent with respect to resident entities (SERVER and LINE) even though they are constructed under different world views. Therefore, either of the implementations can be used to simulate the single-server system monitoring the utilization of the server and the length of the waiting line; that is, the models are interchangeable.

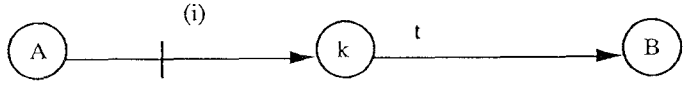
4. RULES FOR EVENT REDUCTION

While illustrating the applications of event graphs to model analysis, Schruben [16] offers several *rules of thumb* to assist the modeler in various tasks. They are called rules of thumb since they are conceptual rules that work "when applied with care." As Som and Sargent [22] point out, "these rules of thumb are not based on a formal framework and thus their validity cannot be determined." Within the context of structural equivalence it is now possible to determine the validity of these "event reduction rules." In particular, these rules are revised and proved as theorems; the impact of event execution priorities and canceling edges is discussed also. Following Schruben's [16] numbering, we define the following rule.

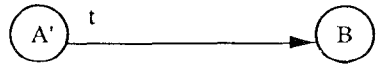
Rule 4(a). Equivalent SGMs are possible with and without event vertex k, if vertex k has no conditional exiting edges and if all edges entering vertex k have zero delay times. If Rule 4(a) applies, *then vertex k may be combined with the originating vertices of the entering edges.*

State variable changes in vertex k are added to changes for these entering event vertices. Delay times on each of the edges exiting vertex k are added to each of the edges entering vertex k.

PROOF. Suppose the following is a subgraph in the simulation graph model, $\mathcal{S}_1$:

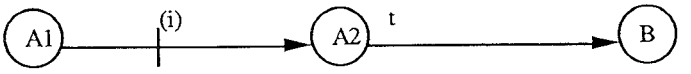


Vertex k satisfies the conditions of Rule 4(a). Define $I(k)$ to be the set of vertices in $V(G)$ with an outgoing edge that is incident to vertex k. In other words, this is the set of predecessors of vertex k. In the above subgraph, $I(k) = \{A\}$. For simplicity, assume that there is exactly one state variable change associated with every event vertex and that all edge conditions are simple. That is, $\mathcal{S}_1$ is assumed to be an ESGM. Applying Rule 4(a), vertex k is merged with the vertices in $I(k)$. This yields the SGM, $\mathcal{S}_2$, with the following subgraph:



where $f_{A'} = f_A \cup f_k$; that is, the state changes associated with vertex k are

simply appended to the state changes associated with vertex A. Note that $\mathscr{S}_2$ is not an ESGM. Therefore, we construct $\mathscr{S}_2^E$ before we seek an isomorphism between the two subgraphs. Application of expansion rules yields the following subgraph:



An isomorphism between the two models, $\mathscr{S}_1$ and $\mathscr{S}_2^E$, is sought next.

Θ: $\Theta(A) = A1$, $\Theta(k) = A2$, and $\Theta(B) = B$.
Φ_s: $\Phi_s((A, k)) = (A1, A2)$, and $\Phi_s((k, B)) = (A2, B)$.
Φ_c: The mapping is vacuous since there are no canceling edges in the subgraph.
Λ: $\Lambda(f_A) = f_{A1}$, $\Lambda(f_k) = f_{A2}$, and $\Lambda(f_B) = f_B$.
Ω: $\Omega(i) = i$.
χ: $\chi(t) = t$.

Δ: Schruben's rules do not consider event execution priorities. With this added dimension, vertex k can be merged with the vertices in $I(k)$ only if vertex k has a *higher* execution priority than all of its predecessors. Under this convention a mapping Δ can always be established.

$(\Theta, \Phi_s, \Phi_c, \Lambda, \Omega, \chi, \Delta)$ is the required isomorphism. Thus, we conclude that with the added convention for event execution priorities, application of Rule 4(a) yields equivalent simulation models. □

Rule 4(c) is "symmetric" to Rule 4(a). We therefore restate it here without proof.

Rule 4(c). Equivalent simulation graph models are possible with and without event vertex k, if vertex k has no conditional exiting edges and if all edges exiting vertex k have zero delay times. If Rule 4(c) applies, then vertex k may be combined with the termination event vertices of exiting edges. *State variable changes in vertex k are added to changes of these succeeding vertices.*

We need to address also the issue of event execution priorities. Under Rule 4(c), vertex k can be allowed to merge with its successor vertices only if it has a lower execution priority than all of them.

Rule 4(b) states that "equivalent event graphs are possible with and without event vertex k, if vertex k has no conditional exiting edges and f_k contains no edge conditioning state-variables." Recall that Schruben's definition of equivalence is based on the behavior of edge-conditioning state variables. It turns out that this is a fairly restrictive definition. For instance, suppose that vertex k is associated with a performance-monitoring state variable, which is updated periodically during the execution of the simulation. Then, even though the conditions for Rule 4(b) hold, eliminating vertex k and appending the associated state change (update of performance

measure) to either the preceding or succeeding vertices may result in the update of the performance-monitoring state variable at the *wrong* instant in simulated time. Therefore, Rule 4(b) is applicable only when there are no state variable changes associated with vertex k, that is, $f_k = \emptyset$. Vertex k is then the identity vertex, which is used solely to make the underlying structure of the model "transparent." Rule 4(b) is revised as follows: equivalent simulation graph models are possible with and without event vertex k, if vertex k has no conditional exiting edges and if there are no state variable changes associated with it. If Rule 4(b) applies, the delay times for each edge exiting vertex k is added to the delay time for each entering edge before vertex k is removed.

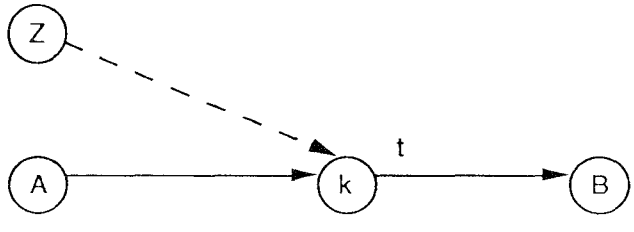
Note that since the application of Rule 4(b) results in deletion of a vertex and, at least, one edge, it is not possible to establish mappings Θ and Φ_s. Hence, this rule cannot be proved using the definition of structural equivalence.

In the light of the above assertion, Rule 5 should also be restated as follows: If $f_k = \emptyset$ for all interior vertices k of an unconditional event tree, then only the leaf vertices on the tree need be included in a simulation graph model. It is important to recognize that the original Rule 4(b) is not a theorem; however, the modification presented here can be applied often with a valid reduction of event vertices.

In the above rules, the discussion of canceling edges is omitted. In other words, the question of whether or not it is possible to remove a vertex with an incoming and/or outgoing canceling edge is not addressed. This is because canceling edges have been shown to be a modeling convenience rather than a necessary modeling tool [23]. Nevertheless, if the user chooses to include them in a model, then the following modifications should be made to the reduction rules:

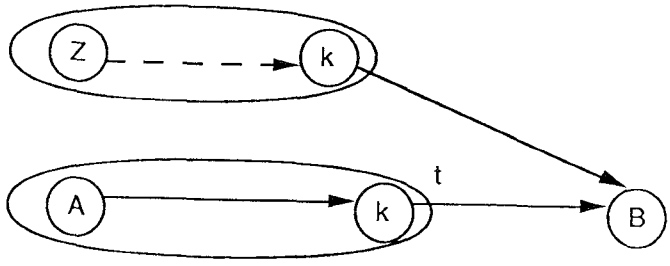
(1) Rule 4(a) does not apply if vertex k has an incoming canceling edge.
(2) Rule 4(c) does not apply if vertex k has an outgoing canceling edge.

To see why the reduction rules do not directly carry over to cases where canceling edges are used, consider the following subgraph, where vertex k satisfies the conditions of Rule 4(a):



The application of the rule yields the following subgraph (drawn so as to demonstrate the implementation of "merged" vertices as a single-event routine). Notice that the (compound) event vertex (consisting of vertices Z and k connected by a canceling edge) is meaningless.

An analogous argument shows that modification (2) to Rule 4(c) is necessary in the presence of canceling edges. The revised "reduction rules" are summarized in Table V.



5. CONCLUDING COMMENTS

In this paper, a testable definition for discrete-event simulation model equivalence is introduced by shifting the emphasis from model behavior to model structure. The new definition (structural equivalence), which is based on establishing isomorphisms between sets of entities making up a discrete-event model, is shown to be a sufficient, but not a necessary, condition for behavioral equivalence. The application of this definition is illustrated then by establishing equivalences between discrete-event systems modeled under different world views. Finally, theorems for event reduction are presented. A logical next step would be to automate this procedure by implementing it on a computer.

APPENDIX: EXPANSION RULES

Given a simulation graph model (SGM), $\mathcal{S}$, an elementary simulation graph model (ESGM), $\mathcal{S}^E$, is generated through the application of expansion rules. Expansion is the process of replacing a single vertex with m state variable changes by m vertices in series, each with a single state variable change. It is also the process of replacing an edge with a compound condition by a series of identity vertices each connected with simple edges. This process preserves the logical structure of the original model.

The expansion rules are presented in this appendix. Proofs of their validity can be found in [23].

EXPANDING COMPOUND EDGE CONDITIONS

Rule 1: *AND* ($\cap$)

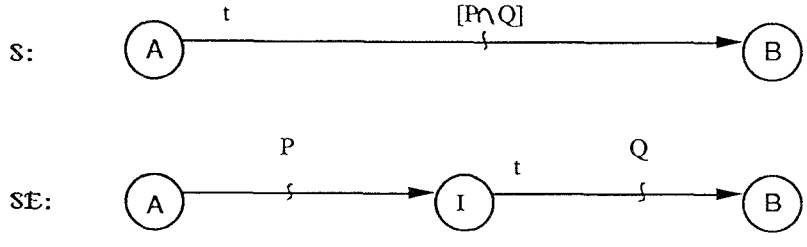
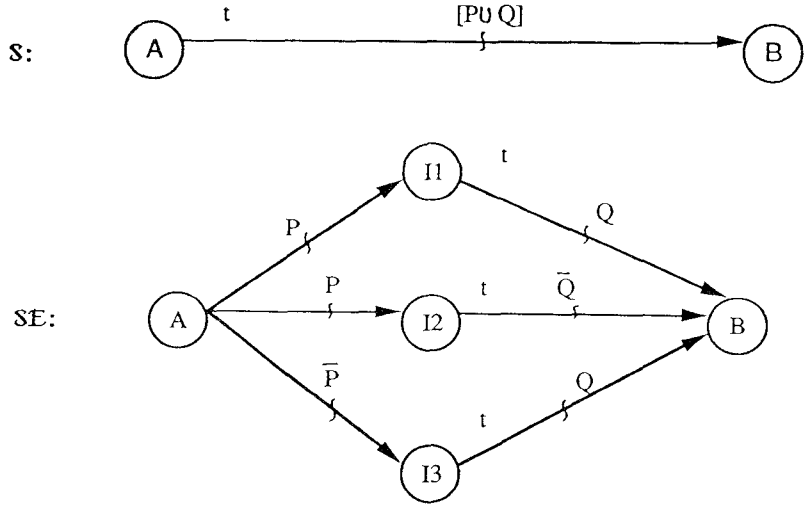


Table V. Event Reduction Rules

Rule 4(a):	Equivalent simulation graph models are possible with and without vertex k if *all* of the following conditions hold: (i) vertex k has no conditional exiting edges (ii) all edges entering vertex k have zero delay time (iii) vertex k has no incoming canceling edges (iv) vertex k has a higher execution priority than all of its predecessors.
Rule 4(b):	Equivalent simulation graph models are possible with and without vertex k if *all* of the following conditions hold: (i) vertex k has no conditional exiting edges (ii) there are no state variable changes associated with vertex k
Rule 4(c):	Equivalent simulation graph models are possible with and without vertex k if *all* of the following conditions hold: (i) vertex k has no conditional exiting edges (ii) all edges exiting vertex k have zero delay times (iii) vertex k has no exiting canceling edges (iv) vertex k has a lower execution priority than all of its successors.
Rule 5:	If $f_k = \varnothing$ for all interior vertices of an unconditional event tree, then only leaf vertices of the tree need be included in the simulation graph model.

Vertex I is the identity vertex. With this rule, it is assigned an execution priority which is *one higher* than that of the origination vertex to ensure the correct execution of the model. Notice that the expansion of a compound edge condition is not unique. However, all possible expansions are isomorphic to one another.

Rule 2: *OR* ($\cup$)

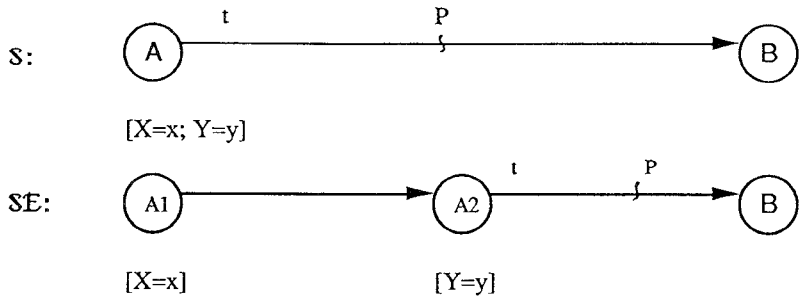


Vertices $I1$, $I2$, and $I3$ are all identity vertices. Their execution priorities should be *one higher* than that of vertex A. $\bar{P}$ means that "condition P does NOT hold." This rule can be generalized by carefully considering all possible combinations of the involved relations, perhaps with the help of a truth table. Once again, notice that rule 2 produces nonunique but isomorphic graphs.

Hybrid cases, where both types of operators are used (e.g., $[(P \cup Q) \cap R]$), can be expanded by applying the above rules sequentially. The order of application does not affect the end result.

EXPANDING COMPOUND EVENT VERTICES

Rule 3:

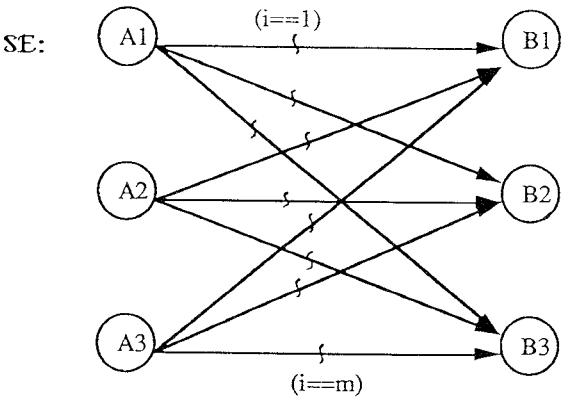


The terms in square brackets denote the state transition functions, f_v, associated with a particular vertex, v. The execution priority of the added vertex should be *one higher* than that of its predecessor. Once again, note that the expansion of a compound event vertex is not unique. However, all expansions are isomorphic. In order to ensure correct execution of the event vertex, care should be exercised in expanding compound vertices. To this end, we can interpret the state transition function as a vector of state variable assignments. In computer implementations, this corresponds to a block of code where the flow of execution is sequential. Adhering to the original sequence of state variable assignments while expanding the vertex practically ensures that the logical structure of the original model is preserved. This rule generalizes to the case of an event vertex with m (> 2) state variable assignments in a straightforward fashion.

EXPANDING PARAMETERIZED VERTICES

Rule 4:

where $j \in \{1, 2, \ldots, m\}$ and $k \in \{1, 2, \ldots, n\}$. Then,



This is the general case illustrating the expansion procedure for parameterized vertices: a subgraph consisting of two parameterized vertices $A[j]$ and $B[k]$ is replaced by a complete bipartite subgraph $K_{m,n}$. The *edge attributes in* $\mathscr{S}$ become *edge conditions in* $\mathscr{S}^E$.

To close this appendix, two results concerning the validity of these rules are reported without proof (see [23] for the proofs).

(1) The simple convention for execution priority assignments used in the expansion process generates an ESGM that preserves the execution order of the original model.

(2) The application of expansion rules yields equivalent ESGMs irrespective of the sequence in which they are applied to expand a given SGM.

Note that the second result is a direct consequence of the fact that isomorphic objects form an equivalence class.

ACKNOWLEDGMENTS

The authors would like to thank the associate editor and two anonymous referees for their valuable suggestions, which led to considerable improvements of an earlier version of the paper.

REFERENCES

1. BONDY, J. A., AND MURTY, U. S. R. *Graph Theory with Applications*. North-Holland, New York, 1976.
2. Consolidated Analysis Centers, Inc. *Simscript II.5 Reference Handbook*. Los Angeles, Calif., 1976.
3. GAREY, M. R., AND JOHNSON, D. S. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Co., New York, 1979.
4. HOPCROFT, J. E., AND TARJAN, R. E. A V^2 algorithm for determining isomorphism of planar graphs. *Inf. Process. Lett.* (1971), 32–34.
5. HOPCROFT, J. E., AND TARJAN, R. E. *Isomorphism of Planar Graphs*. Working paper. 1971.
6. HOPCROFT, J. E., AND TARJAN, R. E. Efficient planarity testing. *J. ACM 21*, 4 (1974), 549–568.

7. HOOVER, S. V., AND PERRY, R. F. *Simulation: A Problem Solving Approach.* Addison-Wesley, Reading, Mass., 1989.
8. LAW, A. M., AND KELTON, W. D. *Simulation Modeling and Analysis.* 2nd ed. McGraw-Hill, New York, 1991.
9. LAWLER, E. *Combinatorial Optimization: Networks and Matroids.* Holt, Rinehart, and Winston, New York, 1976.
10. NANCE, R. E. The time and state relationships in simulation modeling. *Commun. ACM 24*, 4 (Apr. 1991), 173–179.
11. OVERSTREET, C. M. Model specification and analysis for discrete event simulations. Ph.D. dissertation. Virginia Tech., Blacksburg, Virginia, 1982.
12. PEGDEN, C. D. *Introduction to SIMAN.* Systems Modeling Corp., State College, Penn, 1985.
13. PRITSKER, A. A. B. *Introduction to Simulation and SLAM II.* 3rd ed. John Wiley & Sons, New York, 1987.
14. SARGENT, R. G. Event graph modeling for simulation with an application to flexible manufacturing systems. *Manage. Sci. 34*, 10 (1988), 1231–1251.
15. SCHRIBER, T. J. *An Introduction to Simulation Using GPSS / H.* John Wiley, New York, 1990.
16. SCHRUBEN, L. Simulation modeling with event graphs. *Commun. ACM 26*, 11 (Nov. 1983), 957–963.
17. SCHRUBEN, L. *Sigma: A Graphical Simulation System.* The Scientific Press, San Francisco, Calif., 1991.
18. SCHRUBEN, L. Sigma tutorial. In *Proceedings of the 1991 Winter Simulation Conference.* 1991.
19. SCHRUBEN, L., AND YÜCESAN, E. On the generality of simulation graphs. Tech. Rep. #773. Cornell University, Ithaca, N.Y., 1987.
20. SCHRUBEN, L., AND YÜCESAN, E. Transaction tagging in highly congested simulations. *Queueing Syst. 3*, (1988), 257–264.
21. SCHRUBEN, L., AND YÜCESAN, E. Simulation graph duality: A world view transformation for simple queueing models. In *Proceedings of the Winter Simulation Conference.* 1989, pp. 738–745.
22. SOM, T. K., AND SARGENT, R. G. A formal development of event graphs as an aid to structured and efficient simulation programs. *ORSA J. Comput. 1*, 2 (1989), 107–125.
23. YÜCESAN, E. Simulation graphs for the design and analysis of discrete event simulation models. Ph.D. dissertation. Cornell University. Ithaca, N.Y., 1989.
24. YÜCESAN, E. Analysis of Markov chains using simulation graph models. In *Proceedings of the Winter Simulation Conference.* 1990, pp. 468–471.
25. ZEIGLER, B. P. *Theory of Modelling and Simulation.* Reprint Edition. Robert E. Krieger Publishing Company, Inc., Malabar, Fla., 1985.

Received December 1991; accepted June 1992