Proceedings of the 1989 Winter Simulation Conference
E.A. MacNair, K.J. Musselman, P. Heidelberger (eds.)

A LOGIC FOR SIMULATING DISCONTINUOUS SYSTEMS

Sanjai Narain
Jeff Rothenberg

RAND Corporation
1700 Main Street
Santa Monica, CA 90406

ABSTRACT

This paper presents DMOD, a formalism for simplifying the synthesis and analysis of programs for simulating discontinuous systems. It lacks the concept of explicit state. Its programs are constraints upon event occurrences, based upon a novel view of the causality relation. Constraints can freely refer to the past and future of causing events. Simulation is regarded as inference of event occurrences from constraints. An event is said to occur when an interesting proposition becomes true. The new concept of partially instantiated events is introduced.

DMOD can be regarded as a formalization of the widely used event scheduling view of the discrete-event simulation technique. However, it shows how event occurrences can be computed without devices of scheduling, unscheduling or event queues, which are intrinsic to this view. Due to partially instantiated events, DMOD can also be considered more general.

1.0 INTRODUCTION

Simulation of a system means computation of how its state evolves with time. It is essential for obtaining insight into system behavior when its direct observation is too inconvenient, expensive, dangerous or impossible. A system is said to be discontinuous if it has at least one state parameter whose value over the entire time line cannot be described by a single well behaved function of time. A function of time is said to be well behaved, e.g. a sine or square wave, if its properties are well understood. Examples of discontinuous systems routinely arise in science, medicine, manufacturing, robotics, computer hardware or computer animation.

The reason that systems can be discontinuous is that events occur in them. These cause one well behaved function to stop being a description of a parameter and another such function to start being so. For example, a well behaved function describes position of a billiard ball till an event of its colliding with another object occurs. This causes the function to change to another well behaved one.

This paper presents DMOD, a first-order logic formalism for simulating discontinuous systems. It is based upon the following *fundamental assumption*:

> Once the event occurrences till time T are known the state of the system can be computed at any point of time till T.

Thus, simulation can be regarded as computation of event occurrences. This is a generalization of a similar idea in [Misra 1986] where the only events are message transmissions.

DMOD proposes a method of computing event occurrences based upon a novel view of the causality relation. It exhibits three conceptual differences over most traditional simulation formalisms, particularly those based upon the widely used discrete-event technique (for a modern survey see [Evans 1988]):

(a) Purely declarative basis. Programs in most traditional formalisms frequently contain side-effect causing operations such as event scheduling (insertion) and unscheduling (deletion) upon an event queue. The abstract logic that these are intended to implement is left implicit, but is far from obvious. Thus, questions such as about program correctness can be difficult to answer.

DMOD programs are first-order logic constraints upon event occurrences defining a novel view of the causality relation. (An important feature of constraints is that they can freely refer to the past and future of causing events). Constraints are defined separately from procedures for performing inference upon them. Thus, logical properties of programs can be studied independently of these procedures.

Two useful results are obtained as a result. First, the device of partially instantiated events is identified which yields

new programming possibilities. Second, *we are able to compute event occurrences without any use of event queues, scheduling or unscheduling.* The algorithm is, however, quite inefficient, so another one is developed which restores these devices, but uses them in a fundamentally different way.

(b) Absence of explicit state. Programs in most traditional formalisms manipulate an object called state. The state is a collection of tuples **<P,V,T>** each meaning that the value of state parameter **P** is **V** at time **T**. As simulation time advances new tuples are added to state in a destructive or non-destructive manner. As discussed below, reference to states and events in the past of simulation time is frequently required. Thus, all relevant tuples about the past must be retained, so states can become quite complex and difficult to visualize.

Programs in DMOD manipulate history i.e. the sequence of occurring events. History can be easier to visualize than state as temporal and causal orderings can be imposed upon events in it. By the fundamental assumption all states and events in the past and present of a time can be recovered from history till that time. Thus, while state parameters are still objects of contemplation, *explicit* states are not. History could be included as an extra state parameter in traditional formalisms, but that would be conceptually redundant.

(c) **More general definition of event.** Frequently, events are defined to be state changes. However, events such as message transmissions are awkward to regard as state changes. Sometimes an event is defined to be a message transmission. However, satisfaction of conditions e.g. it beginning to rain at sunset, or the temperature of ocean becoming higher than that of land are awkward to regard as events in this sense. DMOD considers an event to occur when an interesting proposition becomes true. As discussed below, a wide range of happenings can now be regarded as events.

Section 2.0 discusses the causality relation. Section 3.0 defines DMOD. Section 4.0 contains examples of DMOD programs. Section 5.0 describes two simulation algorithms. Section 6.0 makes some final remarks.

2.0 THE CAUSALITY RELATION

An important method of computing event occurrences is via the causality relation between events. Intuitively, **causes(A,B)** means **A** is responsible for, or brings about **B**. More precisely, it can be regarded as an abbreviation for **if A occurs then B occurs**, so that event occurrences can be computed using:

occurs(B) if causes(A,B) & occurs(A).

where **occurs(B)** means the event **B** occurs. However, inference of the causality relation from typical statements about it can be quite difficult. This is because statements can refer to the past as well as the future of causing events. The former happens when modeling agents with memory such as doctors or credit verifiers. The latter happens when delays occur between causes and effects and during these delays, conditions arise which preclude expected effects from occurring. An example of a statement which refers to both the past and the future of the causing event (and its formalized version) are:

An event of an aircraft taking off towards a radar at time T causes an event of detection by that radar at time T+20 provided:

(a) an event of its being instructed to fly low has not occurred before T &
(b) an event of its changing course does not occur between T and T+20 &
(c) the radar's intensity is above threshold for each time between T and T+20.

---

causes(flies_towards(A,R,T),detects(A,R,T+20)) if

(a) not exists(X).
X<T & occurs(told_to_fly_low(A,X)) &
(b) not exists(X).
T<X<T+20 & occurs(changes_course(A,X)) &
(c) not exists(X).
T<X<T+20 & intensity(R,X)=<threshold.

Here **flies_towards(A,R,T)** denotes an event of aircraft **A** taking off towards radar **R** at time **T**. Similarly for **detects(A,R,T)**, **told_to_fly_low(A,T)**, **changes_course(A,T)**. **intensity(R,T)** denotes the intensity of radar **R** at time **T**.

The difficulty of reasoning with such a statement arises from two sources. First, there is mutual recursion between this statement about **causes** and the above definition of **occurs**. The recursion can be quite difficult to control. Suppose the event **E=flies_towards(a,r,0) occurs, where a** is an aircraft and **r** is a radar. In order to infer from the second rule that **causes(E,F)** where **F=detects(a,r,20)**, and hence from the first rule that **occurs(F)**, we have to infer that there is no event **G=changes_course(a,X)** such that **occurs(G)** and **0<X<20**. As time is real-valued we cannot hope, even in principle, to iterate over all points of time.

The second source of difficulty is that rules can refer to

states e.g. **intensity**. As time can be real-valued the number of distinct states, e.g. positions of a moving object, can be non-computably infinite. It is impossible to keep an explicit record of all these. Note that the above rules would strain, if not lie outside the reasoning capability of, most current theorem proving or logic programming systems.

DMOD removes this difficulty by proposing that causality be resolved only in the context of a sequence of events. This allows causality to be defined, at the object level, without reference to occurrence. Occurrence is defined at the metalevel inside the inference procedure. The context enables arbitrary reference to past and future.

3.0 DEFINITION OF DMOD

Let **F** be a condition about the simulated system which takes **m+1** arguments, **m>=0**, and the last argument ranges over *real-valued* time. Let **F** be defined for entities **a1,..,am,t**. Then the proposition **F(a1,..,am,t)** is said to become true if **F(a1,..,am,t)** is true but there exists a finite time interval immediately preceding **t** such that for each time instant **X** within it **F(a1,..,am,X)** is false. Thus, we distinguish between a proposition being true and becoming true. Where **P** is a unary condition, these ideas can be illustrated in the following diagram:

```
1---2---3---4---5---6.... (time line)
<--P false-><-----P true-----....
```

P becomes true at 4 but not at 4.1, or 3.9 Let **F1,F2,..,Fk** be a special set of conditions about the simulated system called event-defining conditions. Each **Fi** takes **m+1** arguments, **m>=0**, and the last argument ranges over real-valued time. Where **Fi** is defined for entities **a1,..,am,t**, the proposition **Fi(a1,..,am,t)** is called an event and **t** is called its time-stamp. If the event **Fi(a1,..,am,t)** becomes true then it is said to occur at **t**. Thus, we distinguish between an event and its occurrence.

This definition of events enables us to regard a wide range of happenings as events. For example, the sending of **token** by **machine** to **printer** at time **t** can be regarded as the event **sends(machine,token,printer,t)**. Where **P** is a parameter, define the condition Q_P**(t)=there exists time x, x<t, such that for all u≠t between x and t, value of P at u is not equal to the value of P at t**. Now, the event of the value of **P** changing at **t0** can be represented as the proposition Q_P**(t0)**. The event of the temperature of water becoming higher than that of land at time **t0** can be represented as **R(water,land,t0)** where **R(x,y,t)=temperature(x,t)>temperature(y,t)** and **temperature(x,t)** is the temperature of **x** at time **t**.

Now, the set of event-defining conditions must be chosen in such a way that the fundamental assumption of DMOD is satisfied. Thus, we do not have to regard each of the possible happenings as events, only those which have some bearing upon computation of state. For example, to compute position of the billiard ball we need only consider collisions as events and not changes of position.

Specification of which events occur is done using an alternative view of causality called **causal_connection**. This relation is defined between two events and a context of events. If it holds, the two events are said to be causally connected in this context. Causal connectedness is similar to connectedness between nodes in a network. Two events may be causally connected in one context but not in another. *Note that events in the context do not have to occur*. This is the basis for avoiding the mutual recursion above. For convenience, **causal_connection** is defined as a four-ary relation using Horn clauses, called causality rules, each of the form:

causal_connection(E,HE,F,HEF) if
causality_predicted(E,HE,F) &
prediction_unfalsified(E,HE,F,HEF).

```
^            E                        F
o---o---o---o--o------o-----o---o   (S)
<---HE-->       <-----HEF---->
```

In order for **causal_connection** to be defined **E,HE,F,HEF** must be related as in the figure above. **S** is a sequence of events sorted in increasing order of time stamps ending in **F**. **E** appears before **F**, **HE** is the sequence of all events in **S** up to but not including **E**, and **HEF** is the sequence of all events in **S** between **E** and **F** but not including either. The above rule is to be read as:

> If from the information available till **E** it is predicted that there is a causal connection between **E** and **F**, and this prediction is not falsified by information collected between **E** and **F**, then **E** is causally connected to **F** in **S**.

The inference procedure arranges that **HE** and **HEF** are bound to appropriate sequences of *occurring* events at run time. Thus, by the fundamental assumption, arbitrarily complex conditions about events *and* states in the past of **E** can be evaluated using **HE** and to those in the future of **E** up to **F** using **HEF**. The history up to time **T** represents all states up to **T** even if they are non-computably infinite. There is no restriction on the forms of definitions of **causality_predicted** or **prediction_unfalsified**.

Rules for computing state parameters, given history, are

called state-computation rules. These can involve complex mathematics, and there is no restriction on their form. A DMOD program consists of a set of causality rules, and a set of state-computation rules. It can be regarded as a model of the simulated system, i.e. a precise description of it for the purpose of simulation.

4.0 EXAMPLES OF DMOD PROGRAMS

4.1 Aircraft and radar

The following rules express the intention of the statement in Section 2.0, where all identifiers beginning with capital letters are variables, while others are constants.

```
causal_connection(E,HE,F,HEF) if
  causality_predicted(E,HE,F) &
  prediction_unfalsified(E,HE,F,HEF).

causality_predicted(E,HE,F,HEF) if
  E=flies_towards(Aircraft,Radar,CT) &
  F=detects(Aircraft,Radar,CT+20) &
  not member(told_to_fly_low(Aircraft,_),HE).

prediction_unfalsified(E,HE,F,HEF) if
  E=flies_towards(Aircraft,Radar,CT) &
  F=detects(Aircraft,Radar,CT+20) &
  not member(changes_course(Aircraft,_),HEF) &
  intensity_above_threshold(Radar,HE,E,HEF).

intensity_above_threshold(R,HE,E,[]) if
  intensity(R,HE*[E])>threshold(R).

intensity_above_threshold(R,HE,E,[Ex|HExF]) if
  intensity(R,HE*[E])>threshold(R) &
  intensity_above_threshold(R,HE*[E],Ex,HExF)
```

Here **flies_towards**, **told_to_fly_low**, **detects** and **changes_course**, are event-defining conditions. [] denotes the empty list, [A|B] the list with head **A** and tail **B**, and an underscore (_) an arbitrary variable. **A*B** denotes the concatenation of lists **A** and **B**. **member(A,L)** is true if **A** is a member of list **L**. **intensity(R,H)** denotes the intensity of radar **R** immediately after the latest event in history **H**. **threshold(R)** denotes the threshold intensity of radar **R**. The definition of **intensity**, not given here, would consist of state-computation rules.

The second rule checks whether an event of the aircraft being told to fly low has occurred in the past of **E**, represented by **HE** and if not, predicts an event of detection. The third rule checks whether an event of aircraft changing course occurs in the future of **E** up to **F**, represented by **HEF**, and also whether in the states represented by **HEF** the radar intensity has ever fallen below threshold. Assuming that intensity can only change when events occur **intensity_above_threshold** simply checks whether the intensity is above threshold after each event in **[E]*HEF.**

Note that intensity could be a continuous parameter so that it would *not* be sufficient to check only at event boundaries, that it remains above threshold. It could fall below threshold in between. However, the definition above could easily be extended. The point is that the entire information till **F** is available and we can write rules for computing anything that can be effectively computed from it.

Where position of an aircraft is a *continuous* state parameter, a state-computation rule for determining it is:

```
position(Aircraft,[Px,Py],H) if
  H=HE*[E] &
  position(Aircraft,[Ax,Ay],HE) &
  velocity(Aircraft,[Vx,Vy],HE) &
  DT is time_stamp(E)-time_stamp(H) &
  [Px,Py]=[Ax,Ay]+[Vx*DT,Vy+DT].
```

Here **position(A,Pos,H)** means that the position of object **A** after the last event in history **Hist** is **Pos**. Similarly for **velocity**. **time_history(H)** denotes the time stamp on the latest event in history **H**. The rule simply states that the position of an aircraft after event E is equal to the position after its predecessor plus its velocity after the predecessor times the time difference. It is assumed that velocity remains constant between two events. To compute the position at any point of time **T** we can write:

```
position(Aircraft,[Px,Py],T,H) if
  history_till(T,H,HT) &
  position(Aircraft,[Ax,Ay],HT) &
  velocity(Aircraft,[Vx,Vy],HT) &
  time_history(HT,TimeH) &
  DT is T-TimeH &
  [Px,Py]=[Ax,Ay]+[Vx*DT,Vy*DT].
```

history_till(T,H,HT) computes all events **HT** in **H** whose time stamp is less than or equal to **T**. The rule simply computes the position after **HT** and adds to it the residual displacement till **T**.

4.2 Discrete-event simulation

The discrete-event technique is widely used for simulating discontinuous systems e.g. [Evans 1967, Kiviat 1967, Fishman 1973, Dahl & Nygaard 1966, Nance 1981, Schruben 1983, Zeigler 1984, Misra 1986, Evans 1988, McArthur 1986]. In this section we first show how the

version of Misra can be expressed in DMOD. (The version by Schruben [1983] has also been expressed but space considerations do not permit its discussion). We then propose that other versions can be similarly expressed.

In Misra's version the only events are message transmissions. The method of computing event occurrences is rigorously justified. A variable called **clock** and a data structure called event queue are used. When an event occurs **clock** is set to its time stamp. An event occurrence can schedule events into the queue. An event **F** in the queue occurs provided the sender of **F** receives no message at any **t, clock=<t<t1** where **t1** is the time stamp of **F**. If it does then **F** is unscheduled from the queue. The next occurring event is taken to be the earliest event in the queue. These constraints can be expressed using the rule:

Event E causes event F provided E schedules F and no event G occurs between the time stamp on E and that on F, such that the receiver of G is the sender of F.

It can be expressed in DMOD as:

```
causal_connection(E,HE,F,HEF) if
  causality_predicted(E,HE,F) &
  prediction_unfalsified(E,HE,F,HEF).

prediction_unfalsified(_,_,F,HEF) if
  not exists(G).G∈HEF & receiver(G)=sender(F).
```

causality_predicted(E,HE,F) is true whenever **E** schedules **F**.

Other discrete-event formalisms are similar except that events, and conditions for their non-occurrence can be arbitrary. We propose that these are attempts to implement the causality relation. In particular, **causality_predicted** and **prediction_unfalsified** express, in a *declarative* manner, the intention of scheduling and unscheduling. However, unlike in other formalisms, events can be computed in DMOD without use of an event queue. Thus DMOD can be viewed as their formalization as well as simplification. As the next section shows, it is also more general.

4.3 Partially instantiated events

It is possible that given a DMOD program, the query **causality_predicted(E,HE,F)** succeed, in the sense of logic programmming, for ground **E** and **HE** but non-ground **F**. Let the resulting answer substitution be τ. (Logically speaking, this means **causality_predicted** holds for **E,HE** and all possible *ground* instantiations of **F**τ). The resulting partially instantiated event **F**τ can be fully instantiated when **prediction_unfalsified** is checked.

The advantage of this feature is that we do not have to determine the predicted event completely. It illustrates how **causality_predicted** is fundamentally different from the scheduling operation of the discrete-event technique, which is not intended to insert a non-ground event into the event queue.

The feature can be employed to obtain a simple implementation of activity scanning or demons, which requires considerable extra programming in conventional formalisms. A condition is to be monitored and whenever it becomes true, an event, e.g. of operator notification, is to occur. A condition becomes true after an event if it switches from false to true when the event occurs. Assuming that the condition is only to be checked after event occurrences, these requirements can be expressed by the rule

Event scan(Cond,T) causes event notify_operator(Cond,T1) provided there is an event E in between T and T1 such that Cond becomes true after E and T1 is the time stamp of E.

Note that when **scan(Cond,T)** occurs **T1** is not known. The rule can be expressed in DMOD as:

```
causal_connection(E,HE,F,HEF) if
  causality_predicted(E,HE,F) &
  prediction_unfalsified(E,HE,F,HEF)

causality_predicted(
  scan(Cond,Time),_,inform_operator(Cond,FutureTime)).

prediction_unfalsified(E,HE,F,HEF) if
  E=scan(Cond,Time) &
  F=inform_operator(Cond,FutureTime) &
  HE*[E]*HEF=A*[B]*C &
  B∈HEF &
  holds(Cond,A*[B]) &
  not holds(Cond,A) &
  FutureTime=time_stamp(B).
```

causality_predicted predicts an event of informing the operator but at an as yet unknown **FutureTime**. **HE*[E]*HEF** is the sequence of events till the time stamp on **F**. Let it be equal to **A*[B]*C**. If **Cond** holds after **A*[B]** but not after **A** then it becomes true after **B**, so **FutureTime** can be taken to be the time stamp of **B**. Now the single event **scan(λH.load_average(H)>10,0)** causes, in general, multiple events of operator notification corresponding to times when load average exceeds **10.** Of course, **B** must occur after **scan(Cond,Time)**, i.e in **HEF**.

As another example of the use of partially instantiated events consider a boxing match. We know that once an event **starts_match(A,B,T)** occurs where **A,B** are the boxers, an event **wins_match(X,T+120)** will occur, where **X** is one of **A,B** and **120** seconds is the duration of the match. However, **X** is not determined at **T** but only at **T+120** when all the blows have been laid. In DMOD we can say:

causality_predicted(
starts_match(A,B,T),_,wins_match(X,T+120)) if
member(X,[A,B]).

prediction_unfalsified(E,HE,F,HEF) if
E=starts_match(A,B,T) &
F=wins_match(X,T+120) &
winner(HE*[E]*HEF,X).

The rule for **causal_connection** is the same as in the previous example. **winner(H,X)** is true if **X** is the winning boxer given that the sequence of events in the match is **H**.

4.4 Hamming's problem revisited

This example shows the application of DMOD to a non-simulation domain. The problem is to generate in ascending order, all numbers which are divisible by no primes other than 2,3 or 5. An equivalent formulation, due to Dijkstra [1976], is to generate in ascending order, all numbers defined by the following axioms:

(a) 1 is in the sequence
(b) If x is in the sequence, then so are 2*x, 3*x and 5*x.
(c) The sequence contains no values except those on account of (a) and (b).

The generation of a number **N** is represented by the occurrence of event **e(N)** where **e** is a unary event-defining condition. The above axioms can be restated using causality as:

(a) e(1) occurs.
(b) e(X) causes each of e(2*X), e(3*X), e(5*X).
(c) No other events occur except those on account of (a) and (b).

These can be expressed in DMOD as:

causal_connection(E,HE,F,HEF) if
causality_predicted(E,HE,F) &
prediction_unfalsified(E,HE,F,HEF).

causality_predicted(start(0),_,e(1)).
causality_predicted(e(X),_,e(2*X)).
causality_predicted(e(X),_,e(3*X)).
causality_predicted(e(X),_,e(5*X)).

prediction_unfalsified(_,_,_,_).

The last rule says that predictions are never unfalsified. Now, the sequence of occurring events is **start(0),e(1),e(2),e(3),e(4), e(5),e(6),e(8),.....** Note that the DMOD version almost exactly mirrors the orginial specification.

4.5 Prime numbers

We show how to generate all prime numbers between **2** and **N**. Let **start(N,0)** be a special initial event. Then, generation of primes can be expressed by the following causality rule:

causes(start(N,0),e(X)) if
X is an integral time in between 2 and N &
no event e(T) properly occurs in between
start(0,N) and e(X) such that T divides X.

This can be expressed in DMOD as:

causal_connection(E,HE,F,HEF) if
causality_predicted(E,HE,F) &
prediction_unfalsified(E,HE,F,HEF).

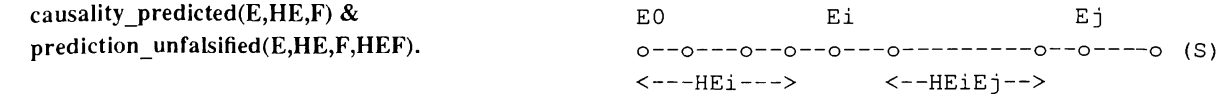
causality_predicted(start(N,0),_,e(X)) if
integer_between(2,N,X).

prediction_unfalsified(start(N,0),_,e(X),HEF) if
not exists(U).U∈ HEF,divides(U,X).

Now, given that **start(10,0)** occurs, the other occurring events are **e(2),e(3),e(5),e(7).**

5.0 SIMULATION IN DMOD

We now show how, given that the initial event has occurred, we can compute which other events occur, i.e. the entire history. We proceed roughly in a bottom-up manner: compute what events the initial event causes in the history, then compute what events these cause in the history, and so on. As causality itself requires history to be evaluable, an apparent circularity arises. To resolve it we first define the history precisely, taking the aid of the following diagram:



Ea,..,Eb, a>b denotes the empty sequence. Informally,

HAB means the sequence of events between **A** and **B** but not including either.

Let **S=E0,E1,E2,...** be a sequence of events sorted in increasing order of time-stamps. Then **S** is said to satisfy causal-soundness if for each **j, j≠0,** there exists **i, i<j** such that **causal_connection(Ei,HEi,Ej,HEiEj)** holds, where **HEi** is the sequence **E0,..,Ei-1** and **HEiEj** is the sequence **Ei+1,..,Ej-1**. Note that the initial event **E0** is exempt from requiring a cause. Intuitively, a sequence is causally-sound if every event in it, except the first one, has a cause in it. Of course, a causally-sound sequence may not contain all the events which should intuitively occur, e.g. the sequence **E0** is trivially causally sound. To ensure that it does, it needs to satisfy another property.

Let **S=E0,E1,E2,...** be a sequence of events sorted in increasing order of time-stamps. Then **S** is said to satisfy causal-completeness if it satisfies both of the following:

(a) for each **i, j, i≠j**, if there is an event **G** such that **causal_connection(Ei,HEi,G,HEiEj)** then the time stamp of **G** is not less than that of **Ej** where **HEi** is the sequence **E0,..,Ei-1** and **HEiEj** is the sequence **Ei+1,..,Ej-1**.

(b) Let **S** possess a last event **Ek**. Then there is no event **G** for which there is an **i** such that **causal_connection(Ei,HEi,G,HEiG)** where **HEi** is **E0,E1,..,Ei-1** and **HEiG** is **Ei+1,..,Ek**.

Informally, condition (a) says it must not be possible for an event to occur after **Ej-1** but strictly before **Ej**. Condition (b) says that **S** must not be extendable at the end i.e. it must already contain all events. Note that the sequence **E0** always trivially satisfies (a) but not necessarily (b).

Let **E0** be a special initial event for the simulated system. Assume that **E0** has occurred. A history of the system is defined to be a sequence of events starting at **E0** which is both causally-sound and causally-complete. Intuitively, it contains all of the events whose occurrence is required by the occurrencé of the initial event and the causality rules, and only these events.

More than one sequence can be a history. This happens when concurrent events occur. For example with **causal_connection(p(1),[],q(2),[])** and **causal_connection(p(1),[],r(2),[])** we have two distinct histories **p(1),q(2)** and **p(1),r(2)**, **p(1)** the inital event. After **p(1)** both **q(2)** and **r(2)** can occur.

5.1 A simulation algorithm

Let the initial event **E0** occur. Suppose the history **E0,E1,..,Em** till a certain point of time has been computed. We need to compute the next event **Em+1**. Let **Sm={F1,F2,...}** be the set of events where for each **Fi**, there exists an **Ei** such that **causal_connection(Ei,HEi,Fi,HEiFi)** holds, where **HEi** is the sequence **E0,E1,..,Ei-1** and **HEiFi** is the sequence **Ei+1,..,Em**. Take the next event, **Em+1** to be the event in **Sm** with the least time-stamp. If **Sm** is empty, the algorithm halts. Note that **Sm** need not be finite.

Intuitively, given a history we determine all the events which are caused by an event in the history with the history as context. Of these we pick the earliest event as the next one.

As there may be more than one event in **Sm** with least time-stamp, the algorithm is non-deterministic. A different sequence would be computed for each choice of **Em+1**, signifying that the system is concurrent. *Note that the algorithm makes no use of event queues.* Also note that it does not make any use of the structure of the definition of **causal_connection.** It is made use of in the next section for developing a more efficient algorithm.

As an example, we show how prime numbers are generated. Let the event **E0=start(10,0)** occur. Clearly, **HE0=[]=HE0Fi.**
causal_connection([],start(10,0),e(X),[]) holds for each **X** where **X** is an integer between **2** and **10**, because there is no event in [] whose time stamp divides **X**. Thus, the set **S0** contains events **{e(2),e(3),e(4),..,e(10)}**. **E1** is taken to be **e(2).**

HE0Fi is now **[e(2)]**. No event of the form **e(X)** causes any event. Now, **causal_connection([],start(10,0),e(X),[e(2)])** holds only for **X=3,5,7,9** as **2** divides the time stamp of any event **e(X)**, **X** even. Thus, **S1={e(3),e(5),e(7),e(9).}**. **E2** is taken to be **e(3).**

HE0Fi is now **[e(2),e(3)]**. It is easily seen that **E3=e(5)**. And so on. The sequence of occurring events is **start(10,0),e(2),e(3),e(5),e(7)**. We now have:

Theorem 1. Correctness of algorithm. A sequence of events **E0,E1,...**, where **E0** is the initial event in the system, is computed by the above algorithm if and only if it is a history.

Proof. If part. Suppose **E0,E1,..,Em. m>=0** has been computed. Assume that the sequence is not causally-sound. As **E0** is causally sound, there exists **k, 0<k** such that

E0,E1,..,Ek-1 is causally-sound but **E0,E1,..,Ek-1,Ek** is not. Then there is no **j, 0=<j=<k-1** such that **causal_connection(Ej,HEj,Ek,HEjEk)** where **HEj=E0,..,Ej-1** and **HEjEk=Ej+1,..,Ek-1**. As **Ek** has been computed as the next event after **Ek-1** it must belong to the set **Sk-1** in the algorithm. But then there must exist an **Ej** as above. Contradiction.

Assume that the sequence is not causally-complete. Suppose condition (a) is violated. Then there exist **i,j, i>=0, j>0, i≠j** and **G** such that **causal_connection(Ei,HEi,G,HEiHEj)** where **HEi=E0,..,Ei-1** and **HEiEj=Ei+1,..,Ej-1** but the time stamp of **G** is less than that of **Ej**. Then **G** must belong to the set **Sj-1** in the algorithm. As **Ej** is computed as the next event after **Ej-1** it must also belong to **Sj-1**. But this contradicts the requirement in the algorithm that the time stamp of **Ej** be less than or equal to that of **G**.

Suppose condition (b) is violated. This contradicts the termination condition of the algorithm.

Only if part. Let **E0,E1,...** be a history. Suppose it is not computed. As **E0** is computed, there exists **k, 0<k** such that **E0,E1,..,Ek-1** is computed but **Ek** is not. As **E0,E1,...** is causally-sound there exists **i, 0=<i=<k-1** such that **causally_connected(Ei,HEi,Ek,HEiEk)** where **HEi=E0,..,Ei-1, HEiEk=Ei+1,..,Ek-1**. Thus, **Ek** belongs to the set **Sk-1** in the algorithm. As **E0,E1,...** is causally-complete there is no **G** whose time stamp is less than that of **Ek**, and for which there exists **j, 0=<j=<k-1** such that **causal_connection(Ej,HEj,G,HEjG)** where **HEj=E0,..,Ej-1, HEjG=Ej+1,..,Ek-1**. Thus, **Ek** is computed as a next event after **Ek-1**. Contradiction. **QED**.

5.2 A more efficient simulation algorithm

The above algorithm can be quite inefficient. To compute the set **Sm** we need to determine for each **Ei** whether there exists **Fi** such that **causality_predicted(Ei,HEi,Fi)** for suitable **HEi**. After **Em+1** has been computed we would repeat this computation for all **i**, except **m+1**, for determining **Sm+1**. In other words, there is no obvious way to incrementally derive **Sm+1** from **Sm**. For example, in the primes example, for each **0=<m=<10** **causality_predicted** repeatedly computes the events **e(2),e(3),..,e(10)**.

We now develop a new algorithm which avoids this. It maintains a queue of items each of which is an event with which a condition is associated. The condition is derived from the **prediction_unfalsified** part of the definition of **causal_connection**. It is evaluated when the history till the time stamp on it has been accumulated. If true, the event is recorded in the history, otherwise discarded. The queue for the next step can be incrementally computed from that for the current step.

The algorithm is similar in spirit to those based upon event queues into which events are scheduled and unscheduled. There is a fundamental difference however. In other algorithms an event is unscheduled as soon as it is determined that it cannot occur. Here, unscheduling occurs only when the history till the time stamp on the event has been accumulated and the associated condition is found to be false. Furthermore, the algorithm highlights the strictly auxiliary nature of event queues. They yield efficiency, but are not inevitably tied up with the logic of event occurrences.

The algorithm makes the assumption that the DMOD program contain the rule:

causal_connection(E,HE,F,HEF) if
causality_predicted(E,HE,F) &
prediction_unfalsified(E,HE,F,HEF).

where each of **E,HE,F,HEF** are variables. The need for it arises because in the algorithm we would like to infer that for *every* **E,HE,F,HEF** if **causality_predicted(E,HE,F)** and **prediction_unfalsified(E,HE,F,HEF)** then **causal_connection(E,HE,F,HEF)**. Without this statement we cannot infer this, even if all rules for **causal_connection** have the above form. For example, let the program consist of a single rule for **causal_connection** with **E,HE,F,HEF** as **e(0),[],f(0),[]** respectively. Now, even when **causality_predicted** and **prediction_unfalsified** hold for **E,HE,F,HEF** as **e(1),[],f(1),[]** respectively, we cannot infer that **causal_connection** holds for these also.

Let the initial event **E0** occur. Let **E0,..,Em** be the history computed till some point of time. Let **Qm** be a finite or infinite set of items each of which is a function of the form **λHEF.prediction_unfalsified(E,HE,F,HEF)** where **E,HE,F** are all ground but **HEF** is a variable. The time stamp of the function is defined to be that of **F**. An item **λHEF.prediction_unfalsified(E,HE,F,HEF)** is in **Qm** if and only if there exists **i, 0=<i=<m** such that:

(a) E=Ei, and
(b) HE=E0,..,Ei-1, and
(c) causality_predicted(E,HE,F) holds, and
(d) time stamp on F is greater than or equal to that on Em.

Let **Rm={F1,F2,..}** be the set of events such that **λHEF.prediction_unfalsified(Ei,HEi,Fi,HEF)** is in **Qm**, and holds for **HEF=Ei+1,..,Em**. Take **Em+1** to be the

event in **Rm** with the least time stamp. If **Rm** is empty, the algorithm halts.

To compute **Qm+1** delete from **Qm** all functions whose time stamp is less than that of **Em+1**. Also add to **Qm** all items **λHEF.prediction_unfalsified(Em+1,HEm+1,Fm+1,HEF)** such that **causality_predicted(Em+1,HEm+1,Fm+1)** holds, where **HEm+1=E0,..,Em**. Thus, **causality_predicted** is only called for **Em+1** as we had hoped. Any functions contributed by **E0,..,Em** are already in **Qm+1** if not deleted above. Clearly, **Qm+1** satisfies the conditions above.

As an example, consider again the generation of prime numbers. **E0=start(10,0)** as in Section 5.1. Each item in **Q0** is **λHEF.prediction_unfalsified(start(10,0),[],e(X),HEF)** where **X=2,3,4,5,..,10** Each function is true for **HEF=[]** so **R0={e(2),e(3),e(4),e(5),..,e(10)}. E1** is taken to be **e(2). λHEF.prediction_unfalsified(start(10,0),[],e(2),HEF)** is deleted from **Q0**. As an event of the form **e(X)** does not cause any events, each item in **Q1** is again **λHEF.prediction_unfalsified(start(0),[],e(X),HEF)** where **X=2,3,4,5,..,10** Thus, **Q1=Q0**.

Now, only functions **λHEF.prediction_unfalsified(start(0),[],e(X),HEF)** where **X=3,5,7,9** are true for **HEF=[e(2)]**. Thus **R2={e(3),e(5),e(7),e(9)}. E2** is taken to be **e(3)**. And so on.

The main difference with the first algorithm is that items are only deleted from the queue. No call is ever made to **causality_predicted** except for constructing **Q0**. In the first algorithm, **causality_predicted** was repeatedly called for constructing each **Sm, 0=<m=<10**. We now have:

Theorem 2. Correctness of improved algorithm. A sequence of events **E0,E1,..** where **E0** is the initial event in the system, is computed by this algorithm if and only if it is a history.

Proof. Suppose **E0,..,Em, m>=0** has been computed. We show that the set **Sm** of the first algorithm is identical to the set **Rm** of the second. An event with the least time-stamp in these is computed as the next event **Em+1**.

Let **Fi** belong to **Rm**. Then there exist **Ei,HEi** and a function **λHEF.prediction_unfalsified(Ei,HEi,Fi,HEF)** in **Qm** such that the function holds for **HEF=Ei+1,..,Em**, where **HEi=E0,..,Ei-1**. The membership of this function in **Qm** implies that **causality_predicted(Ei,HEi,Fi)** also holds. Thus, by the presence of the rule for **causal_connection** above, **causal_connection(Ei,HEi,Fi,HEF)** also holds. Hence **Fi** belongs to **Sm**.

Let **Fi** belong to **Sm**. Then there exist **Ei,HEi** such that **causal_connection(Ei,HEi,Fi,HEF)** holds where **HEi=E0,..,Ei-1** and **HEF=Ei+1,..,Em**. By definition of **causal_connection, causality_predicted(Ei,HEi,Fi)** holds and the time stamp of **Fi** is greater than or equal to that of **Em**. Thus, the function **λH.prediction_unfalsified(Ei,HEi,Fi,H)** is in **Qm**. Again, by definition of **causal_connection**, this function holds for **H=HEF=Ei+1,..,Em**. Thus, **Fi** belongs to **Rm. QED**.

6.0 FINAL REMARKS

In contrast to most conventional simulations, state parameters in DMOD need only be updated upon demand, that of computing the next event. They need *not* be routinely updated at each event boundary. A caching mechanism can be employed to avoid recomputing parameters. In view of this fact, and the second algorithm above, it is not unreasonable to expect that DMOD would be as efficient as conventional state-oriented simulation formalisms.

A partially instantiated event can lead to an infinite **Qm** in the second algorithm, as it represents an infinite number of ground events. The algorithm is being extended to keep **Qm** finite by allowing insertion of items with partially instantiated events, instead of only their ground instances.

DMOD appears to integrate discrete and continuous simulation methods. Often analytic approaches for computing event occurrences are infeasible. For example, two objects may be moving along complicated trajectories and it may be impractical to compute their collisions by analytically solving simultaneous equations governing their motion. In such cases numerical methods can be freely employed. These can involve incremental advances of time which are the essence of time-stepped simulation. Also, specification of where to employ it can be made within **causality_predicted** and **prediction_unfalsified** rules.

As DMOD's events are more general than message transmissions, it is not clear whether parallel simulation approches of [Misra 1986] or [Jefferson 1985] are directly applicable to DMOD. One possible approach is the following: The set of all causally-sound sequences of events starting at the initial event can be laid out in the form of a tree. The root is the initial event **E0**. If **E0,...,Em** is a branch then **E0,...,Em,Em+1** is another branch provided there exists **i** such that **causal_connection(Ei,HEi,Em+1,HEiEm+1)** where **HEi=E0,..,Ei-1**, and **HEiEm+1=Ei+1,..,Em1. Em+1** is an

immediate descendant of **Em.** Only the causally-complete branches in this tree are histories. This tree can be constructed and searched in parallel.

DMOD serves as a basis for defining in a precise, and conceptually tractable manner, advanced simulation problems such as incremental, backward, or demand-driven simulation. Such problems are currently being investigated by the authors.

ACKNOWLEDGEMENTS

We thank J. Kajiya, L. Miller, N. Shapiro, R. Bagrodia, R. Nance, S. Glicker, P. Fishwick and J. Misra for helpful remarks.

REFERENCES

Carnap, R. [1966]. *Philosophical foundations of Physics.* Basic Books, New York.

Communications of the ACM [1981]. Special issue on simulation and modeling, April.

Dahl, O.-J., Nygaard, K. [1966]. Simula-An Algol-based simulation language. *Communications of the ACM*, vol. 9, no. 9.

Evans, J.B. [1988]. *Structures of discrete-event simulation. An introduction to the engagement strategy.* Ellis Horwood, New York.

Evans, G.W., Wallace, G.F., Sutherland, G.L. [1967]. *Simulation using digital computers.* Prentice-Hall, Englewoods Cliffs, N.J.

Fishman, G. [1973]. *Concepts and methods in discrete-event digital simulation.* John Wiley & Sons, New York.

Galton, A. [1988]. *Temporal logics and their applications.* Academic Press, New York.

Jefferson, D. [1985]. Virtual Time. *ACM transactions on programming languages and systems*, July.

Kiviat, P.J. [1967]. Digital computer simulation: modeling concepts. RM-5378-PR, RAND Corporation, Santa Monica, CA.

McArthur, Klahr, P., Narain, S. [1986]. ROSS: An object-oriented language for constructing simulations. In *Expert Systems: Techniques, Tools, Applications*, (eds.) P. Klahr, D. Waterman, Addison Wesley, 1986.

Misra, J. [1986]. Distributed discrete-event simulation. *Computing Surveys*, March.

Nance, R. [1981]. The time and state relationships in simulation modeling. *Communications of the ACM*, April.

Narain, S. [1989]. DMOD: A logic-based calculus of events for discrete-event simulation. To appear in *Proceedings of AI and Simulation Conference*, Tampa, Society for Computer Simulation.

Oeren, T., Zeigler, B. [1987]. Artificial intelligence in modeling and simulation: directions to explore. *Simulation*, April.

Rothenberg, J. [1986]. Object-oriented simulation: where do we go from here? *Proceedings of the Winter Simulation Conference*, Washington, D.C.

Rothenberg, J., Narain, S., Steeb, R., Hefley, C., Shapiro, N. [1988]. Knowledge-based simulation: An interim report. N-2897-DARPA. RAND Corporation, Santa Monica, CA.

Schruben, L. [1983]. Simulation modeling with event graphs. *Communications of the ACM*, November.

von Wright, G.H. [1968]. *Time, change and contradiction.* Cambridge University Press.

Zeigler, B. [1984]. *Multifacetted modelling and discrete-event simulation.* Academic Press, New York.

Sanjai Narain is a computer scientist at The RAND Corporation where he is developing advanced simulation methodologies. His areas of expertise are logic, programming languages, temporal reasoning and artificial intelligence. He obtained his doctorate in Computer Science from UCLA in 1988. He can be reached at 213-393-0411 or at narain@rand.org.

Mr. Rothenberg is a Senior Computer Scientist at The RAND Corporation who has worked extensively in knowledge-based simulation. He performed his graduate work in AI at the University of Wisconsin in the area of semantic nets for robotic applications. His recent work has been in the development of new formalisms for integrating object-oriented and event-oriented views of discrete-state simulation. He can be reached at 213-393-0411 or at jeff@rand.org.