*Proceedings of the 1991 Winter Simulation Conference*
*Barry L. Nelson, W. David Kelton, Gordon M. Clark (eds.)*

# AN AXIOMATIC BASIS FOR GENERAL DISCRETE-EVENT MODELING

Sanjai Narain

Bellcore, MRE 2E260
445 South Street
Morristown, NJ 07962

## ABSTRACT

The discrete-event technique (DET) is a powerful technique for modeling and simulating dynamic systems. However, with its previous formulations, two problems arise. First, DET models can be difficult to build. DET encourages one to think in terms of the causality relation between events, but provides only indirect means of expressing it. Second, it can be difficult to reason about them, i.e. to prove temporal properties such as safety, not provable by simulation alone. This is because the logic of event occurrences underlying DET is specified only algorithmically, not explicitly. This paper presents DMOD, a new formulation of DET and proposes solutions to the above problems. It also presents an implementation of DMOD models and of the reasoning techniques, using the logic of definite clauses. This logic has high expressive power and simple deductive properties.

## 1 INTRODUCTION

The discrete-event technique (DET) is a powerful technique for modeling and simulating dynamic systems. Many formulations of it have been proposed, e.g. (Fishman 1973, Zeigler 1984, Evans et al. 1965, Evans 1988, Suri 1987). A number of well known commercial simulation packages e.g. SIMSCRIPT or OPNET are based upon it. DET is based upon the idea that the behavior of a system can be represented by its history, i.e. the sequence of events which occur in it. DET defines a technique for specifying and computing event occurrences.

DET is powerful because it is able to model situations where effects of events depend not just on their past but also on their future. These arise frequently in dynamic systems. For example, in a phone system, an effect of lifting a handset at $T$ is dial tone timing out at $T+30$, provided *in between* $T$ and $T+30$, a digit is not dialed. Such a situation also occurs in, for example, preemptive schedulers.

However, with previous formulations of DET, two problems arise. First, DET models can be difficult to build. This is because DET encourages one to think in terms of the causality relation between events, but allows one to express it only indirectly. Second, it is difficult to reason about DET models. Reasoning is necessary to prove temporal properties, such as safety, which cannot be proved by simulation alone. For example, in the phone system, one may wish to prove that if the time $T1$ taken by the user to dial a one-digit number is less than the dial tone timeout value $T2$, connection is assured. Clearly, one cannot simulate for each of the infinite number of pairs $<T1,T2>$, $T1<T2$, and check. Instead, one has to analyze the logic of event occurrences. However, this logic is not explicitly specified. Event occurrences are specified only by the simulation procedure, which is algorithmic.

This paper proposes DMOD, a new formulation of DET. It enables the user to directly specify the causality relation between events. Thereby, model building is greatly simplified. DMOD also defines event occurrences in a logical manner, and uses it to build a framework for formulating and proving temporal properties. Although not discussed in this paper, DMOD is powerful enough to model hybrid systems, i.e. those which exhibit both discrete and continuous behavior. An implementation of both DMOD models and the reasoning framework is shown using the logic of definite clauses (Kowalski 1979, Lloyd 1984), the basis of the widely used language Prolog. This logic has high expressive power and simple deductive properties.

Section 1.1 outlines DET and discusses the above problems in some detail. Section 2 presents DMOD. Section 3 discusses an implementation of it using the logic of definite clauses.

Section 4 presents a framework for proving temporal properties, as well as an implementation of it using the above logic.

### 1.1 Outline of DET

A typical DET model of a system defines:

- A set of time stamped events
- A set of system states
- The relation *schedules(S,E,F)*, meaning that in state *S*, event *E* possibly causes event *F*.
- The relation *unschedules(S,E,F)*, meaning that in state *S*, event *E* precludes event *F*.
- The relation *updates(E,S,S1)*, meaning that event *E* changes system state *S* to system state *S1*.

A history of the system is computed as follows: three data structures, *state, clock, event queue* are maintained. Where *E* is the event in *queue* with the least time stamp, *E* is removed from *queue*, and recorded as occurring. *clock* is set to be the time stamp of *E*. *state* is set to be *newstate* where *updates(E,state,newstate)*. Each event *F* such that *F* is in the queue and *unschedules(newstate,E,F)*, is deleted from *queue*. Each event *F* such that *schedules(newstate,E,F)*, is inserted into *queue*. This sequence of steps is repeated to yield a sequence of event occurrences, i.e. history. Computation of history is also called simulation.

For example, suppose we wished to model one aspect of the simple phone system: when a hand set is lifted, the network waits for 30 seconds for a digit to be dialed. If it is not dialed within this time, a time out occurs. Dialing occurs 15 seconds after the hand set is lifted.

We can model events of lifting a handset at *T*, of dialing at *T*, and of timing out at *T* using, respectively, the expressions *lhs(T), dial(T), to(T)*. Now, *lhs(T)* schedules *dial(T+15)* and *to(T+30)*. Also, *dial(X)* unschedules any event of the form *to(Y)*. Let *lhs(0)* occur. Then, by the above procedure, only *dial(15)* occurs. However, if *lhs(T)* were to schedule *dial(T+40)*, and *lhs(0)* occurs, then both *to(30)* and *dial(40)* occur.

It is reasonable to say that *schedules/unschedules* relations are means to define the *causality* relation between events. Causality defines a sufficient condition for event occurrence: if *E* causes *F* and *E* occurs, then *F* occurs. Thus, "*E* schedules *F* but *G* unschedules *F*" implicitly means "*E* causes *F* provided *G* does not occur in between."

However, it is difficult to specify causality in this indirect manner. This is because it is difficult to associate a precise meaning with *schedules* and *unschedules*. Given that *S* is the state after *E* has occurred, *schedules(S,E,F)* is neither a sufficient nor a necessary condition for occurrence of *F*. *F* can be unscheduled. Also, another event in another state can schedule *F*. Similarly, given that *S* is the state after *E* has occurred, *unschedules(S,E,F)* is neither a sufficient nor a necessary condition for non-occurrence of *F*. *F* can be rescheduled. Also, another event in another state can unschedule *F*.

Furthermore, history is defined only in a procedural manner, not by means of axioms. Thus, it is difficult to show that a property *P* such as safety, is satisfied by all histories. The traditional approach of starting from axioms defining *P* and history, and developing a proof using logical principles, cannot be employed.

DMOD proposes solutions to the above problems. For the first problem, DMOD eliminates the scheduling and unscheduling relations in favor of the single causality relation. *causes(E,HE,F)* means "if *HE* is the history up to (but not including) *E*, then *E* causes *F*". If *E* has occurred, then *causes(E,HE,F) is a sufficient condition for the occurrence of F.*

Situations where effects of events depend on their future can be easily modeled. For example, to model the rule *causes(E,HE,F) if G does not occur in between*, make *E* cause an auxiliary checking event *check(E,F,TF)*, where *TF* is the time stamp of *F*. This event definitely occurs. When it does, it examines the history up to it to see whether *G* occurs after *E*. If so, it causes nothing. Otherwise, it causes *F*. See Section 3.1 for an example.

For the second problem, DMOD defines history by means of axioms. A history is defined to be a sequence of events satisfying two main conditions, *causal soundness* and *causal completeness*. Thus, it becomes possible to understand event occurrences without any reference to event queues. A procedure is also presented for computing histories and, for a wide class of histories, is shown to be equivalent with the axiomatic definition. Event queues are restored in this procedure, but their status as purely efficiency devices is established.

## 2 DMOD: A NEW FORMULATION OF DET

A *DMOD structure* is a tuple *{Events, TimeStamps, time, lt, eq, causes, init_event}* where:

- *Events* is an enumerably infinite set of objects called *events*.

- *TimeStamps* is an enumerably infinite set of objects called *time stamps*.

- *time(E,T)* is a relation between an event $E$ and a time stamp $T$. *time(E,T)* models a function, i.e. is subject to the following existence and uniqueness restriction:

$\forall E \exists T.time(E,T) \wedge \forall S.time(E,S) \supset S=T.$

- *lt(T1,T2)* and *eq(T1,T2)*, *T1,T2* time stamps, are relations defining a total ordering between time stamps, i.e. satisfying:

*(1) Exactly one of {lt(T1,T2), eq(T1,T2), lt(T2,T1)} holds.*
*(2) lt(T1,T2) ∧ lt(T2,T3) ⊃ lt(T1,T3).*
*(3) eq(T1,T1).*
*(4) eq(T1,T2) ⊃ eq(T2,T1).*
*(5) eq(T1,T2) ∧ eq(T2,T3) ⊃ eq(T1,T3).*

- *init_event∈Events* is called the *initial event*.

- *causes(E,HE,F)* is a relation between events $E$ and $F$, and a finite sequence $HE$ of events in *Events* satisfying:

(a) $\forall E \forall HE \forall F$, the set *{F:causes(E,HE,F)}* is finite.
(b) $\forall E \forall HE \forall F$. *causes(E,HE,F)* ∧ *time(E,T1)* ∧ *time(F,T2)* ⊃ *lt(T1,T2)* ∨ *eq(T1,T2)*.

We will use *time(E)* as an abbreviation for *the T such that time(E,T)*. Where *T1,T2* are time stamps, $T1 \leq T2$ is an abbreviation for *lt(T1,T2)* ∨ *eq(T1,T2)*. Similarly for $T1 \geq T2$.

Let $P$ be a DMOD structure and *S=[E0,E1,..]* be a finite, or enumerably infinite sequence of events. Then $S$ is said to be *temporally ordered* if for each $i$, $i \geq 0 \supset time(Ei) \leq time(Ei+1)$ whenever *Ei+1* exists.

We now associate histories with a DMOD structure *P={Events, TimeStamps, time, lt, eq, init_event}*. Let *S=[E0,E1,E2,..]* a finite or enumerably infinite sequence of events. Then $S$ is said to be *causally-sound, cs(S),* if every event in *SR has a cause in S*, i.e.:

$\forall j.j>0 \supset \exists i.i<j \wedge causes(Ei,[E0,..,Ei-1],Ej).$

Note that the singleton sequence *[E0]* is trivially causally sound.

Let $P$ be a DMOD structure and *S=[E0,E1,E2,..]* be a finite or enumerably infinite sequence of events. Then $S$ is said to be *causally-complete, cc(S),* iff it contains all caused events, i.e.

$\forall G. \forall i.causes(Ei,[E0,..,Ei-1],G) \supset \exists j.j \geq 0 \wedge Ej=G$

Let $P$ be a DMOD structure. A *history for P* is a finite or infinite sequence $H$ of events such that:

*(a) H begins with init_event*
*(b) H is temporally ordered*
*(c) No event occurs more than once in H*
*(d) H is causally-sound*
*(e) H is causally-complete*

### 2.1 A procedure for computing histories

We now present a procedure for computing a subset of histories associated with a DMOD structure *P={Events, TimeStamps, time, lt, eq, init_event}*. Let *Seq=[E0,E1,..]* be a temporally ordered sequence of events. Then *Seq* is said to be *strictly progressive* iff either *Seq* is finite, or for each time stamp *t∈TimeStamps* there is an $i$ such that *lt(t,time(Ei))*. Thus, *E0,E1,...* is *not* strictly progressive if time stamps of *E0,E1,...* are, respectively:

*1,2,2,2,2,2,2...*

or

*1, 1+1/2, 1+1/2+1/4, 1+1/2+1/4+1/8, ...*

where *1,2,+,/* have their usual meanings. In both cases, time stamps converge to *2*.

Let $S$ be a set of events. Then $E \in S$ is said to be an *earliest event in S* provided for each event $F$ in $S$, *[E,F]* is temporally ordered. Note that if $S$ contains events concurrent with each other there can be more than one earliest event in $S$. Also, if $S$ is finite, it always contains an earliest event.

**Procedure 2.1.1.** Let $P$ be a DMOD structure. Record *E0=init_event* as the initial event. Sup-

pose a partial history *E0,E1,..,Em*, $m \geq 0$, has been computed. We need to compute the next event *Em+1*. For each *i*, $0 \leq i \leq m$, let *Effects_i* be the set of events *F* such that:

*causes(Ei,[E0,..,Ei-1],F), and*
*F is not already in [E0,E1,..,Em], and*
*[Em,F] is temporally ordered.*

Due to the restriction on *causes*, this set is finite. Let *Sm* be the union of *Effects_i*, $0 \leq i \leq m$. Again, this set is finite. If it is empty, halt. Otherwise, it contains an earliest event *Em+1*. Take this to be the next event. □

Intuitively, given a partial history *H* we determine the set *Sm* of all the events *E*, not in *H*, which are caused by an event in *H*. We pick as *Em+1* an earliest event in *Sm*. Note that *Sm* can contain more than one earliest event so the procedure is non-deterministic. A different history would be computed for each choice of *Em+1*.

*Sm* represents the event queue of the DET procedure prevailing after *Em* has been computed. In our case, it is reconstructed in its entirety after every event. Thus, considerable inefficiency can result. To alleviate it, we should construct *Sm+1* incrementally from *Sm*. This is done by deleting *Em+1* from *Sm*, and inserting into it all events *F* such that *causes(Em+1,[E0,..,Em],F)*, *F* is not in *[E0,..,Em+1]*, and *[Em+1,F]* is temporally ordered.

We now prove the correctness of Procedure 2.1.1. The correctness of the more efficient version can be similarly proved.

**Theorem 2.1. Soundness and Completeness of Procedure.** Let *P = {events, timestamps, time, lt, eq, causes, init_event}* be a DMOD structure. Let *E0=init_event*. A strictly progressive sequence of events *[E0,E1,..]* is computed by the above procedure if and only if it is a history.

**Proof:** Simple, but omitted for reasons of space.

## 3 PROGRAMMING DMOD STRUCTURES

Let *P* be a logic program. We let $|\text{-}A$ be an abbreviation for $P|\text{-}A$, i.e. that there is a proof of *A* from *P*. Let *R* an n-ary relation symbol. Let *D1,..,Dn* be subsets of *HU(P)*, the Herbrand Universe of *P*. This is the set of all non-variable terms formed from the function symbols in *P*, and can be taken to be the set of objects that "the logic program is talking about". Then *R* is said to define the relation $\{<t1,..,tn> | t1 \in D1,..,tn \in Dn \wedge |\text{-}R(t1,..,tn)\}$ upon the set *D1 X .. X Dn*, in the presence of *P*.

Let *P* be a logic program and *D* a subset of *HU(P)*. Then *seq(D)* denotes the set of all finite sequences formed from members of *D*.

Let *an_event* and *a_time_stamp* be two relation symbols. Let *events* be the set $\{T | T \in HU(P) \wedge |\text{-}an\_event(T)\}$. *Let timestamps* be the set $\{T | T \in HU(P) \wedge |\text{-a\_time\_stamp(T)}\}$. Note that these two sets can be empty, if there are no clauses defining *an_event* or *a_time_stamp* in *P*.

In the presence of *P*, let:

(a) *time* define *time** upon *events X timestamps*
(b) *lt* define *lt** upon *timestamps X timestamps*
(c) *eq* define *eq** upon *timestamps X timestamps*
(d) *causes* define *causes** upon *events X seq(events) X events*

Let *init_event* be a member of *events*. Then *{events, timestamps, time*, lt*, eq*, causes*, init_event}* is said to be a *structure derived from P*. Now, *P* is said to be a *DMOD program* if there exists a DMOD structure *S* derived from *P*.

### 3.1 Example: Setting Up A Phone Call

We now model the setting up of a call in a simple phone system. When a caller lifts a handset, a dial tone is initiated. The dial tone times out if no number is dialed within a fixed duration. If a number is dialed within this duration, the appropriate phone rings.

We let a time stamp be the term *0* or *s(X)*, *X* a time stamp. Each time stamp denotes a natural number. We use *0,1,2,...* as abbreviations for, respectively, *0,s(0),s(s(0)),....* The definition of *a_time_stamp* is:

*a_time_stamp(X) if natural_number(X).*

*natural_number(0).*
*natural_number(s(X)) if natural_number(X).*

Definitions of *lt* and *eq* are as follows:

*lt(0,s(X)).*
*lt(s(X),s(Y)) if lt(X,Y).*

*eq(X,X).*

The first rule states that *0* is less than every

term of the form *s(X)*. The second states that *s(X)* is less than *s(Y)* if *lt(X,Y)*. The last defines *eq* to be syntactic equality. (Note that *eq* could be defined to hold among non syntactically equal terms, e.g. *2* and *1+1*).

Addition of natural numbers is defined as follows:

```
plus(0,X,X).
plus(s(X),Y,s(Z)) if plus(X,Y,Z).
```

Where *N* is a natural number, *T* a time stamp, and *Caller, Called* arbitrary terms, an event is of one of the following forms, with meanings given in curly braces:

*start(0)* {The initial event with time stamp 0}
*lhs(Caller,Called,T)* {*Caller* lifts handset to call *Called* at *T*}
*dtto(Caller,T)* {Deadline for *Caller* to dial expires at *T*}
*dials(Caller,Called,T)* {*Caller* dials (single digit) number of *Called* at *T*}
*rings(Caller,Called,T)* {*Called* gets a ring from *Caller* at *T*}

The definition of *an_event* is:

```
an_event(start(0)).
an_event(lhs(Caller,Called,T)) if
        a_time_stamp(T).
an_event(dtto(Caller,T)) if
        a_time_stamp(T).
an_event(dials(Caller,Called,T)) if
        a_time_stamp(T).
an_event(rings(Caller,Called,T)) if
        a_time_stamp(T).
```

The time stamp of an event *f(t1,..,tn,t)* is *t*. To define *time*, for each n-ary symbol in *{lhs, dtto, dials, rings, start}* write the definite clause:

```
time(f(T1,..,Tn,T),T).
```

The definition of *causes* is given by the following rules, with informal meanings given in curly braces:

{C1. The initial event causes *user(1)* to lift his hand set to contact emergency at *100.*}

```
causes(E,HE,F) if
        E=start(0),
        F=lhs(user(1),emergency,100).
```

{C2. When *Caller* lifts hand set to contact *Called*, he dials him after delay 4.}

```
causes(E,HE,F) if
        E=lhs(Caller,Called,T),
        F=dials(Caller,Called,FT),
        plus(T,4,FT).
```

{C3. When *Caller* lifts hand set to contact someone at *T*, check after timeout delay 3 whether dial tone times out.}

```
causes(E,HE,F) if
        E=lhs(Caller,Called,T),
        F=check(E,dtto(Caller,FT),FT),
        plus(T,3,FT).
```

{C4. Checking whether dial tone times out at *T* causes dial tone to time out at *T* provided, in the history since the hand set was lifted, there is no event of *Caller* dialing.}

```
causes(E,HE,F) if
        E=check(G,F,T),
        F=dtto(Caller,T),
        G=lhs(Caller,Called,PT),
        HE=_*[G]*HG,
        absent(dials(Caller),HG).
```

{C5. When *Caller* dials *Called* at *T*, *Called* hears a ring after a connection delay 5, provided dial tone has not already timed out at *Caller*.}

```
causes(E,HE,F) if
        E=dials(Caller,Called,T),
        F=rings(Caller,Called,FT),
        plus(T,5,FT),
        absent(dtto(Caller),HE).
```

Let *f* be a function symbol such that for some terms *t1,..,tn*, *f(t1,..,tn)* is an event. Then, where *H* is a sequence of events, *absent(f(t1,..,tk),H)*, $k \leq n$ holds iff there is no event of the form *f(t1,..,tk,Xk+1,..,Xn)* in *H*. *absent* is definable using a logic program.

**Theorem 3.1.1.** *P* is a DMOD program.

**Proof.** We have to show that the structure derived from *P {events, timestamps, time*, lt*, eq*, causes*, start(0)}* is a DMOD structure.

The sets *timestamps* and *events* are enumerable as *HU(P)* is enumerable. As a relation *R** in the given structure is defined by the relation symbol *R* in *P*, to prove a proposition *Q* involving *R*(t1,..,tn)*, prove *Q* with *R*(t1,..,tn)* replaced by *|-R(t1,..,tn)*. For example, to show that:

$\forall E \forall HE.\{F|causes^*(E,HE,F)\}$ is finite

show:

$\forall E \forall HE.\{F|P|\text{-}causes(E,HE,F)\}$ *is finite*

The latter can be shown by a case analysis on the form of *E*. For example, let *E* be an event *lhs(C,N,T)*. Then, *causes(E,HE,F)* succeeds once and only once using Rule C2, as *plus* always succeeds. Similarly for other forms. Similarly, for other propositions. **QED.**

### 3.2 Computing history for the phone system

We now apply Procedure 2.1.1 to the above DMOD structure. The initial event *E0=start(0)*. By (Rule) *C1*:

*S0={lhs(user(1),emergency,100)}, so*
*E1=lhs(user(1),emergency,100)*

By *E1,C2* and *E1,C3*:

*S1={check(lhs(user(1),100),dtto(user(1),103)),*
*dials(user(1),emergency,104)}, so*
*E2=check(lhs(user(1),100),dtto(user(1),103)).*

By *E1,C2* and *E2,C4*:

*S2={dtto(user(1),103), dials(user(1),emergency,104)},*
*so E3=dtto(user(1),103).*

By *E1,C2*:

*S3={dials(user(1),emergency,104)}, so*
*E4=dials(user(1),emergency,104).*

Note that *E4* does not cause an event of ringing at *emergency* as the occurrence of *E3* falsifies the last condition in Rule C5. Thus the history is *[E0,E1,E2,E3,E4]*. It is easily verified that this is the only history.

## 4 REASONING ABOUT DMOD STRUCTURES

We now present a framework for expressing and proving properties about DMOD structures. A natural class of such properties is about their histories. Let *P* be a DMOD structure. Let *history(X)* be true whenever *X* is a history for *P*. Let *X=[E0,E1,.....* Let *prop* be a condition upon, possibly infinite, sequences of events. Typically, one proves that *prop* holds for *all* histories, i.e.:

$\forall X.history(X) \supset prop(X).$ *(A)*

A proposition of the form *(A)* is said to be a *liveness property* iff *prop* is of the form:

$\exists k.cond([E0,..,Ek]).$

where *cond* is a condition upon *finite* sequences of events. Thus, a liveness property states that in every history, there exists some event *Ek* after which *cond* is satisfied.

A proposition of the form *(A)* is said to be a *safety property* iff *prop* is of the form:

$\neg\exists k.cond([E0,..,Ek]).$

where *cond* is a condition upon finite sequences of events. Thus, a safety property states that in every history, there is never any event *Ek* after which *cond* is satisfied.

We now show how safety properties can be expressed in terms of those involving only finite structures. Thereby, one can contemplate programming any proof procedures one may develop.

Let *P={events, timestamps, time, lt, eq, causes, init_event}* be a DMOD structure. Let *S=[E0,E1,..,Ek]* be a finite sequence of events. Then *S* is said to be *internally causally incomplete, ici(S)* iff there exist *i,G* such that *causes(Ei,[E0,..,Ei-1],G)*, but *G* does not occur in *S* and *lt(time(G),time(Ek))*.

It is obvious that if *S* is a finite, initial segment of a history *H*, then it is *not* internally causally incomplete. If it is, then as *H* is temporally ordered, *G* cannot appear after *Ek* in *H*. Thus, *H* is not causally complete.

Let *fto(S)* be true whenever *S=[E0,E1,..,Ek]*, *E0=init_event*, is a finite, temporally ordered sequence of events in which no event occurs more than once. Suppose we wish to prove a safety property, i.e. that some condition *cond* does not

hold for any finite initial segment of any history. Then it is sufficient to show:

$\forall S.fto(S) \wedge cs(S) \wedge cond(S) \supset ici(S) \quad (B)$

Now, let $Q$ be a finite initial segment of a history such that $cond(Q)$. Then, $fto(Q) \wedge cs(Q)$. Hence, by $(B)$, $ici(Q)$. Contradiction.

A brute-force approach for proving $B$ is generating each sequence of events satisfying the left hand side of $B$, $(lhs(B))$ and showing that it also satisfies its right hand side. However, except for the most trivial DMOD programs, the set of all sequences satisfying $lhs(B)$ will be infinite. Thus, we cannot hope to *explicitly* generate each such sequence. If $B$ were true, our approach would not terminate. We can, however, hope to represent an infinite set of sequences by a finite constraint $C$, and work directly with it. Then, we could still expect termination in finite time.

Should we not be able to show from $C$ that each member of the set represented by it satisfies $ici$, we could derive from it a finite set of constraints $C1,..,Ck$ such that the union of the sets represented by these is equal to that represented by $C$. We could then attempt, recursively, to show that each member of $Ci$ satisfies $ici$.

It is particularly easy to implement this scheme if each constraint is a set $\{A1,..,Ak\}$, each $Ai$ a condition of the form $Ri(t1,..,tn)$ and $Ri$ a relation defined by a logic program. If $A1,..,Ak$ is treated as a query, and if it is transformed in one SLD-step into the queries $Q1,..,Qm$, then the set of objects represented by $\{A1,..,Ak\}$ is the union of the sets represented by $Q1,..,Qm$. Thus, to show that each member of the set represented by $\{A1,..,Am\}$ satisfies some property, we can construct an SLD-search tree rooted at $\{A1,..,Am\}$. At each node in the tree, a constraint of this form would appear. The deeper the node is, the more "information" it would contain about the objects in $\{A1,..,Am\}$.

It now remains to show that $cs$, $ici$, $cond$ and $fto$ are definable using logic programs. For $cs$ we have:

*cs(Hist) if cs_1(Hist,[]).*

*cs_1(Hist,Done) if same_events(Hist,Done).*
*cs_1(Hist,Done) if*
*Hist=HE*[E]*H,*
*absent(E,Done),*
*HE=HF*[F]*HFE,*
*causes(F,HF,E),*
*cs_1(Hist,[E|Done]).*

In *cs_1(Hist,Done)*, *Done* is a subset of *Hist*. It is true if all events in *Done* have a cause in *Hist* and all remaining events, except the first one in *Hist* also have a cause in *Hist*. The second rule states that *cs_1(Hist,Done) is true if Hist* and *Done* have the same events. Thes third states that *cs_1(Hist,Done)* is true if there is an event $E$ in *Hist* absent in *Done* which possesses a cause in *Hist* and *cs_1(Hist,[E|Done])* is true. Now, *Hist* is causally sound if *cs_1(Hist,[])* is true.

For *ici* we have:

*ici(S) if*
*S=HE*[E]*H*[F],*
*causes(E,HE,G),*
*absent(G,S),*
*time(G,TG),*
*time(F,TF),*
*lt(TG,TF).*

Similarly, *fto* can easily be defined using logic programs. It only remains to choose *cond* in such a way that it is definable using logic programs. But a quite wide range of conditions can be defined in this way.

Let $P$ be a DMOD program extended by definitions of *cs, ici, fto* and *cond*. Then the proposition:

$\forall S.fto(S) \wedge cs(S) \wedge cond(S) \supset ici(S)$

reduces to:

$\forall S.P|\text{-}[fto(S) \wedge cs(S) \wedge cond(S)] \supset P|\text{-}ici(S)$

Propositions of this form also arise in questions of logic program equivalence. Hence proof techniques e.g. of (Clark & Tarnlund 1977, Kanamori & Seki 1986, Baudinet 1988) can be utilized.

### 4.1 Proving properties about the phone system

We now show how our above scheme can be utilized for proving temporal properties about the phone system of Section 3.1. It is easily proved by simulation that the event *rings(user(1),emergency,109)* does not occur. However, suppose, that there is a nuisance *user(2)* who only keeps plugging and unplugging his phone into an outlet at regular time intervals.

His behavior can be expressed using the rules:

*causes(E,HE,F) if*
*E=start(0),*
*F=plugs(user(2),0).*

*causes(E,HE,F) if*
*E=plugs(User,T),*
*F=unplugs(User,T+1).*

*causes(E,HE,F) if*
*E=unplugs(User,T),*
*F=plugs(User,T+1).*

The first rule initiates the infinite loop, whereas the second and third rules sustain it.

Now, the actions of *user(2)* have nothing to do with the non occurrence of *rings(user(1),emergency,109)*. Still, to prove by simulation that this event does not occur we would still generate about 100 events for *user(2)*. It is desirable to avoid such inefficiency. Furthermore, suppose that we wanted to prove that no event of the form *rings(user(1),emergency,X)* ever occurs. This proposition cannot be proved by simulation, as the history is infinite. We could continue forever to generate plugging and unplugging events without realizing that the ringing event cannot occur.

We now show how our proof technique can be used to avoid both the inefficiency and insufficiency of the proof-by-simulation approach. The structure of the proof is as follows: we first prove that if an event *dials(user(1),Place,T)* occurs then *T=104*. We do this by assuming that *dials(user(1),Place,T)* occurs with *T≠104*. It must possess a cause. Its only cause is obtained from C2, and is *lhs(user(1),Place,T-4)*. Its only cause is obtained from C1, and is *start(0)*, so *T-4=100*. Thus, *T=104*. This contradicts *T≠104*.

Now, to prove that *rings(user(1),emergency,109)* does not occur, we assume it does. Then, by C5, so must its cause *dials(user(1),emergency,104)*, subject to the constraint that no time out event occurs before it. However, by C4 we have a contradiction, since by it a time out event does occur at *103*. This is because, by C2, the cause of the dialing event at *104* is one of lifting the hand set at *100*. In turn, by C3, this causes an event of checking whether time out occurs at *103*. By C4, this causes time out at 103, provided between *100* and *103*, no dialing event has occurred. We already established that the only dialing event occurs at *104*. We now carry out this proof in detail.

**Lemma 4.1.1.** Let *P* be the DMOD structure modeling the phone system, augmented by the three rules for *user(2)*. Let *S* a finite initial segment of a history. Then, there is no event *dials(user(1),Place,T)* in *S* such that *T≠104*.

**Proof.** We have to show:

*fto(S)* ∧
*cs_1(S,[])* ∧
*S=HE*[E]*H* ∧
*E=dials(user(1),Place,T)* ∧
*T≠104* ⊃
*ici(S).*

Consider the query:

*fto(S),*
*cs_1(S,[]),*
*S=HE*[E]*H,*
*E=dials(user(1),Place,T),*
*T≠104.*

Replacing *cs_1(S,[])* for *E*, we obtain:

*fto(S),*
*cs_1(S,[E]),*
*S=HF*[F]*HFE*[E]*H,*
*causes(F,HF,E),*
*E=dials(user(1),Place,T),*
*T≠104.*

*causes(F,HF,E)* can only be expanded, without immediate failure, using Rule C2, to yield:

*fto(S),*
*cs_1(S,[E]),*
*S=HF*[F]*HFE*[E]*H,*
*F=lhs(user(1),Place,PT),*
*plus(PT,4,T),*
*E=dials(user(1),Place,T),*
*T≠104,*

Expanding *cs_1(S,[E])* for *F* we obtain:

*fto(S),*
*cs_1(S,[F,E]),*
*S=HG*[G]*HGF*[F]*HFE*[E]*H,*
*causes(G,HG,F),*
*F=lhs(user(1),Place,PT),*
*plus(PT,4,T),*
*E=dials(user(1),Place,T),*

$T \neq 104$.

*causes(G,HG,F)* can only be expanded, without immediate failure, using Rule C1 to yield:

*fto(S),*
*cs_1(S,[F,E]),*
*S=HG*[G]*HGF*[F]*HFE*[E]*H,*
*G=start(0),*
*F=lhs(user(1),emergency,100),*
*plus(100,4,T),*
*E=dials(user(1),emergency,T),*
$T \neq 104$,

Expanding *plus(100,4,T)* to completion, we obtain:

*fto(S),*
*cs_1(S,[F,E]),*
*S=HG*[G]*HGF*[F]*HFE*[E]*H,*
*G=start(0),*
*F=lhs(user(1),emergency,100),*
*E=dials(user(1),emergency,104),*
$104 \neq 104$,

But due to the last condition, this query cannot succeed, i.e. it is impossible for $S$ to satisfy the left hand side of the implication at the beginning of the proof. **QED.**

**Theorem 4.1.1.** Let $P$ be the DMOD structure modeling the phone system, augmented by the three rules for *user(2)*. Let $S$ be a finite initial segment of a history. Then, there is no event *rings(user(1),emergency,109)* in $S$.

**Proof.** We have to show:

$fto(S) \wedge$
$cs\_1(S,[]) \wedge$
$S=HE^*[E]^*H \wedge$
$E=rings(user(1),emergency,109) \wedge$
$\supset$
*ici(S).*

Consider the query:

*fto(S),*
*cs_1(S,[]),*
*S=HE*[E]*H,*
*E=rings(user(1),emergency,109).*

Expanding *cs_1(S,[])* repeatedly, starting at $E$ we obtain:

*fto(S),*
*cs_1(S,[B,C,D,E]),*
*B=start(0),*
*C=lhs(user(1),emergency,100),*
*D=dials(user(1),emergency,104),*
*E=rings(user(1),emergency,109),*
*S=HB*[B]*HBC*[C]*HCD*[D]*HDE*[E]*H,*
*absent(dtto(user(1)),HB*[B]*HBC*[C]*HCD).*

Let *N=check(C,dtto(user(1),103),103).*

**Case 1.** $N$ is not in $S$, i.e.:

*fto(S),*
*cs_1(S,[B,C,D,E]),*
*B=start(0),*
*C=lhs(user(1),emergency,100),*
*N=check(C,dtto(user(1),103),103),*
*D=dials(user(1),emergency,104),*
*E=rings(user(1),emergency,109),*
*S=HB*[B]*HBC*[C]*HCD*[D]*HDE*[E]*H,*
*absent(dtto(user(1)),HB*[B]*HBC*[C]*HCD),*
*absent(N,S).*

By Rule C3, *causes(C,HB*[B]*HBC,N)*, so due to the last condition, *ici(S).*

**Case 2.** $N$ is in $S$, i.e.:

*fto(S),*
*cs_1(S,[B,C,D,E]),*
*B=start(0),*
*C=lhs(user(1),emergency,100),*
*N=check(C,dtto(user(1),103),103),*
*D=dials(user(1),emergency,104),*
*E=rings(user(1),emergency,109),*
*S=HB*[B]*HBC*[C]*HCN**
*[N]*HND*[D]*HDE*[E]*H,*
*absent(dtto(user(1)),HB*[B]*HBC*[C]**
*HCN*[N]*HND),*

**Case 2.1.** There is no event of the form *dials(user(1),Place,T)* in *HCN*, i.e.:

*fto(S),*
*cs_1(S,[B,C,D,E]),*
*B=start(0),*
*C=lhs(user(1),emergency,100),*
*N=check(C,dtto(user(1),103),103),*
*D=dials(user(1),emergency,104),*
*E=rings(user(1),emergency,109),*
*S=HB*[B]*HBC*[C]*HCN**

*[N]*HND*[D]*HDE*[E]*H,*
*absent(dtto(user(1)),*
*HB*[B]*HBC*[C]*HCN*[N]*HND)*
*absent(dials(user(1)),HCN).*

Let *HN=HB*[B]*HBC*[C]*HCN*. By Rule C4 *causes(N,HN,dtto(user(1),103))*. Due to the second last condition, *ici(S)*.

**Case 2.2.** There is an event of the form *dials(user(1),Place,T)* in *HCN*. Then $T \leq 103$. By Lemma 1, *S* is not causally sound. **QED.**

Finally, to prove that no event of the form *rings(user(1), emergency,X)* occurs, we show that if such an event occurs, $X=109$. The proof is analogous to that of Lemma 1. Now, use Theorem 1.

## ACKNOWLEDGEMENTS

The initial version of DMOD was developed when the author was at The RAND Corporation. The present, simpler version has been developed at Bellcore. The author is grateful for valuable discussions with Jeff Rothenberg, Norman Shapiro, Louis Miller, Iris Kameny, Michael Mattock, Jim Dewar, Linda Ness, Jane Cameron, Y.-J. Lin, B. Gopinath, P. Varaiya and P. Ramadge.

## REFERENCES

Alur, R., Courcoubetis, C., Dill, D. [1990]. Model Checking For Real-Time Systems. *Proceedings of Fifth IEEE Annual Symposium on Logic in Computer Science*. Philadelphia, PA.

Baudinet, M. [1988]. Proving Termination Properties of Prolog Programs: A Semantic Approach. *Proceedings of IEEE Symposium on Logic in Computer Science.*

Cameron, E., Lin, Y.-J. [1991]. A Real-Time Transition Model for Analyzing Behavior Compatibilities of Telecommunications Services. To appear in *Proceedings of ACM Conference on Software for Critical Systems*, New Orleans, LA.

Clark, K., Tarnlund, S.-A. [1977]. A First-Order Theory of Data and Programs. *Proceedings of IFIP.*

Evans, J.B. [1988]. *Structures of Discrete-Event Simulation. An Introduction to the Engagement Strategy*. Ellis Horwood, New York.

Fishman, G. [1973]. *Concepts and Methods in Discrete-Event Digital Simulation*. John Wiley & Sons, New York.

Glynn, P. [1989]. A GSMP Formalism for Discrete-Event Systems. *Proceedings of the IEEE*, January.

Inan K., Varaiya, P. [1987]. Finitely Recursive Processes. *Discrete Event Systems: Models and Applications*. Lecture Notes in Information Science, 103, Springer Verlag.

Kanamori, T., Seki, H. [1986]. Verification of Prolog Programs Using an Extension of Execution. *Proceedings of Third International Conference on Logic Programming,* London.

Kowalski, R. [1979]. *Logic for Problem Solving*, Elsevier North Holland, New York.

Kowalski, R., Sergot, M. [1986]. A Logic-Based Calculus of Events. *New Generation Computing*, Ohmsha Ltd., & Springer Verlag, 4.

Lloyd, J. [1984]. *Foundations of Logic Programming*. Springer Verlag.

Misra, J. [1986]. Distributed Discrete-Event Simulation. *Computing Surveys*, March.

Narain, S., Rothenberg, J. [1989]. A Logic for Simulating Dynamic Systems. *Proceedings of Winter Simulation Conference*, Washington, D.C.

Ness, L. [1990]. Issues Arising in the Analysis of L.0. *Proceedings of the DIMACS: Computer-Aided Verification Workshop*, June 18-21.

Ostroff, J. [1989]. Sythesis of Controllers for Real-Time Discrete-Event Systems. *IEEE Conference on Decision Control,* Tampa, FL.

Peterson, J.L. [1977]. Petri Nets. *ACM Computing Surveys*, vol. 9, No. 3.

Sandewall, E. [1989]. Combining Logic and Differential Equations for Describing Real-World Systems. In *Proceedings of International Conference on Knowledge Representation,* Toronto, Canada.

Suri, R. [1987]. Infinitesimal Perturbation Analysis for General Discrete-Event Dynamic Systems. *Journal of the ACM*, July 1987.

Zeigler, B. [1984]. *Multifacetted Modeling and Discrete-Event Simulation,* Academic Press, New York.

## AUTHOR BIOGRAPHY

**SANJAI NARAIN** is a researcher at Bellcore, Morristown, NJ. His current interests are in temporal reasoning and real-time programming. The context for his research is the interoperability problem for fiber optics hardware.