Proceedings of the 1990 Winter Simulation Conference
Osman Balci, Randall P. Sadowski, Richard E. Nance (eds.)

PROVING TEMPORAL PROPERTIES OF HYBRID SYSTEMS

Sanjai Narain
Jeff Rothenberg

RAND Corporation
1700 Main Street
Santa Monica, CA 90406

ABSTRACT

Formal techniques for proving temporal properties are intended either for purely discrete systems, or for purely continuous systems. However, hybrid systems, i.e. those which can exhibit both discrete and continuous behavior frequently arise in the real world. This paper presents a formal approach for modeling such systems and proving their temporal properties. It is based upon utilizing intuitions about the causality relation, and the logic of definite clauses with the SLD-resolution proof procedure. It is illustrated by proving a liveness property about a railroad crossing.

1. INTRODUCTION

Formal techniques for proving temporal properties are intended either for purely discrete systems, or for purely continuous systems. Examples of the former include Temporal Logic e.g. [Pnueli 1977, Owicki & Lamport 1982, Manna & Pnueli 1981, Bernstein & Harter 1981, Jahanian & Mok 1986, Ostroff 1989]. An example of the latter is the calculus of differential equations. However, hybrid systems, i.e. those which can exhibit both discrete and continuous behavior frequently arise in the real world. Examples of these are manufacturing processes, disk drives, robot arms, copying machines or railroad crossings.

This paper presents a formal approach for proving temporal properties of hybrid systems. It is based upon exploiting intuitions about the causality relation, and the logic of definite clauses and SLD-resolution e.g. [Kowalski 1979, Lloyd 1984]. No new temporal logic is introduced. The approach is illustrated by proving a liveness property about a railroad crossing. It can also be used to prove properties of purely discrete systems in new ways.

Contemporary temporal logics frequently make the assumption that the state of a system changes only at discrete points of time. Clearly, when continuous state parameters are involved, this assumption does not hold, at least in any obvious way. For example, the concept of the next position of a moving object, at any given time, is not well defined. (The logic of Owicki & Lamport [1982] does not makes this assumption, however, the author is not aware of its application to reasoning about continuous time and state).

Our approach assumes that only events occur at discrete points of time. This is quite possible, even when continuous parameters are involved. For example, on a pool table, events of collision between balls and between balls and table boundaries, occur only at discrete points of time, even though ball positions change gradually.

Our approach is based upon expressing temporal properties in terms of questions about event occurrences. Many temporal properties are directly questions about event occurrences. For example, does an event ever/never occur? Do at least N events occur before time T? Is there a last event? A question such as "does temperature ever exceed X?" could be expressed using "is an event of turning on the heat always followed by an event of turning off the heat within Y minutes, given that the current temperature is less than X?".

A method of answering questions about event occurrences can be based upon the causality relation between events. Intuitively, the proposition causes(A,B) means event A is responsible for, or brings about event B. As we have good intuitions about it, its definition can be easy to provide. Event occurrences can be computed using:

occurs(B) iff (initial(B) $\vee$ ∃A.occurs(A) $\wedge$ causes(A,B))

This rule states that an event B occurs iff either it is an initial event, or there exists another event A such that A occurs and A causes B. Now, to show that a non-initial event occurs, show that at least one of its causes occurs. Conversely, to show that a non-initial event does not occur, show that none of its causes occurs. Proofs of safety and liveness properties can be developed in this manner. To determine the number of events which occur, repeatedly apply the above rule and count. If each occurring event always causes at least one event, then there is no last event occurrence.

Central to the success of this approach is the simplicity with which the causality relation can be inferred from a suitable model, i.e. formal description, of the system. To facilitate this inference, the model can be a definition of the causality relation itself.

Frequently, however, statements (or rules) defining causality contain guard conditions on the time interval between the causing and caused events. These arise when delays occur between causes and effects and during these delays, conditions arise which preclude expected effects from occurring. For example, we have:

```
causes(
        begins_flight_towards(Aircraft,Radar,T),
        detects(Radar,Aircraft,T+Delay)
    ) if
        intersect_path(Aircraft,Radar,T,Delay) ∧
        ¬∃X. T<X<T+Delay ∧ occurs(changes_course(Aircraft,X)).
```

This rule states that an event of Aircraft beginning flight towards Radar at time T causes an event of Aircraft coming in range of Radar after Delay, where Delay is obtained by intersecting the path of Aircraft at T, with Radar coverage, provided no event of Aircraft's changing course occurs in between T and T+Delay. The detection event is predicted based upon information available at T. If this information changes before detection occurs, then the prediction will not be true. Hence the need for the guard condition at the end.

Another example of such a rule is one for a preemptive server:

```
causes(arrives(Job,T),departs(Job,T+delay)) if
        server_is_idle(T) ∧
        ¬∃X.∃NewJob.
                T<X<T+delay ∧
                occurs(arrives(NewJob,X)) ∧
                pr(NewJob)>pr(Job).
```

This states that an event of Job arriving at the server at T causes an event of Job departing at T+delay, provided server is idle at T, and there is no NewJob which arrives at time X in between T and T+delay, whose priority is higher than that of Job. If such a job does arrive, then Job's processing would have to be interrupted, and it would not depart at T+delay.

Such rules can be quite difficult to reason with, in particular for inferring causality. This is because a recursion arises between definitions of "causes" and "occurs". Effectively, this means that to determine whether an event occurs in the future at T, one has to know about all future event occurrences up to, but not including T. As time is real-valued, the recursion can be quite difficult to control.

A central contribution of this paper is a new view of causality which allows such rules to be expressed using definite clauses. Thereby, the recursion between causes and occurs can be eliminated. Furthermore, a sound and complete proof procedure, called SLD-resolution, can be employed to reason with definite clauses e.g. [Kowalski 1979, Lloyd 1984]. This achieves substantial simplification in the inference of causality. In addition, a wide range of symbolic and qualitative knowledge can be expressed using definite clauses.

Section 2 briefly introduces definite clauses and SLD-resolution. Section 3 outlines DMOD, our new approach to modeling dynamic systems. Section 4 presents a DMOD model of a hybrid system, namely, a railroad crossing. Section 5 presents a simulation procedure, i.e. for computing event occurrences from a DMOD program. Section 6 discusses the general framework in which temporal properties can be formulated and proved, and illustrates the ideas by proving a liveness property about a railroad crossing. Section 7 contains a summary.

2. BRIEF INTRODUCTION TO DEFINITE CLAUSES

A definite clause is of the form "A if B1,..,Bk", k≥0, each A, Bi a predication of the form R(t1,..,tn), R an n-ary relation symbol, and each ti a term. A term is either a variable, or of the form f(s1,..,sm), f an m-ary function symbol and each si a term. All variables in a definite clause are universally quantified. The clause "A if B1,..,Bk" is to be read "for all values of variables in the clause, A is true if each of B1,..,Bk is true". A set of definite clauses is called a logic program. An example of a logic program is:

```
anc(X,Y) if p(X,Y).
anc(X,Y) if p(X,Z),anc(Z,Y).

p(elizabeth,charles).
p(charles,william).
p(charles,harry).
```

The first clause (or rule) states that X is an ancestor of Y if X is a parent of Z. The second states that X is an ancestor of Y if X is a parent of Z and Z is an ancestor of Y. The last three clauses state information about the royal family of England.

Given a logic program Q, the SLD-resolution proof procedure is used to prove propositions of the form ∃.P where P is a conjunction of predications P1,..,Pk, k≥0. ∃.P is a shorthand for ∃X1..∃Xm.P where X1,..,Xm are the variables in P. If k=0, the proposition is proved. Such a proposition is denoted by □. Otherwise, a clause "A if B1,..,Bn", in Q is selected, such that Pi and A unify with most general unifier σ, and then ∃.(P1,..,Pi-1,B1,..,Bn,Pi+1,..,Pk)σ is recursively proved. The resulting sequence of propositions is called an SLD-derivation. If □ is derived, the derivation is called successful. Unification is a generalization of pattern matching in which variables in both A and Pi can be bound.

For example, SLD-resolution can be used to prove ∃X.anc(X,harry), in particular, "return" the answers X=charles, X=elizabeth. SLD-resolution is sound and complete. Due to a property called strong-completeness, only a restricted set of SLD-derivations need be constructed. This further simplifies the space of SLD-derivations. For more details see [Lloyd 1984].

3. DMOD: A NEW TECHNIQUE FOR MODELING DYNAMIC SYSTEMS

We assume the usual first-order logic alphabet, consisting of an enumerably infinite list of variables, function symbols of all arities, predicate symbols of all arities, and the logical connectives, e.g. [Lloyd 1984]. A term is defined to be a variable. If X1,..,Xm are terms and f an m-ary function symbol then f(X1,..,Xm) is a term. An expression is said to be ground if it does not contain any variables.

Let f1,f2,... be a fixed subset of the set of function symbols called event-defining function symbols, each of positive arity. An event is defined to be a *ground* term of the form f(a1,..,am,t) where f is an event-defining function symbol of arity m+1, and a1,..,am, t are terms. a1,..,am denote arbitrary objects in the modeled system, but t denotes a non-negative, real-valued time instant, called a time stamp.

An event can denote an action. For example, the event sends(m,s,r,t) denotes the action of agent s sending message m to agent r at time t. An event can also denote a proposition. For example, the event touching(a,b,t) can denote the proposition that the distance between positions of two pool balls a and b, at time t, is 2*r, where r is the radius of a and b. When this proposition becomes true, a collision between a and b can be said to occur at t.

The event-defining functions symbols must be chosen in such a way that the fundamental assumption above is satisfied. For example, suppose one wished only to compute the positions and velocities of a set of pool balls on an infinite pool table. Then, one only needs to regard touching as an event-defining function symbol. Other (interesting) symbols such as equilateral, where equilateral(a,b,c,t) means positions of a,b,c at time t, are on the vertices of an equilateral triangle, need not be regarded as event-defining.

As mentioned in Section 1, a convenient way of computing event occurrences is via the causality relation. As also discussed there, if statements about causality are formalized in the obvious manner, causality can be quite difficult to infer. We now show how causality can be viewed in such a way that it can be defined using definite clauses. We regard causality, not as a binary relation, but as a ternary relation between two events and a context. The context is a temporally ordered sequence of events. We call the new relation causal_connection. If it holds, the two events are said to be causally connected in the 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. For example, consider the following events concerning a ball dropped from a tall building:

```
E1=ball_dropped(0)
E2=ball_caught(5)
E3=ball_thrown_down(6)
E4=ball_casts_shadow_on_window_11(7)
E5=ball_hits_ground(10)
```

Consider the two contexts C1=E1,E5 and C2=E1,E2,E3,E4,E5. E1 can be said to be causally connected to E5 in C1, but not (naturally) in C2. In C2, a causal connection which existed between E1 and E5 is terminated by the appearance of E2. Thus, it is more natural to say that E3 is causally connected to E5 in C2.

Another way to understand causal connection is the following: let C be a temporally ordered sequence of events. Pick out E and F in C such that E appears before F. If all events in C were to occur, then would E be said to be a cause of F? If so, then E is said to be causally connected to F in C.

For convenience, causal_connection is defined to be a four-ary relation. The condition causal_connection(E,HE,F,HEF) is defined when its arguments are related as follows:

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

S is a *finite* context, i.e. a sequence of events sorted in increasing order of time stamps ending in F. E appears before F in S, 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. If the condition causal_connection(E,HE,F,HEF) holds, E is said to be causally connected to F in context S.

Notation convention. A*B denotes the result of appending sequence A to sequence B.

Let S represent the history of the system till the time stamp on F. Then, by the fundamental assumption above, HE*[E]*HEF is a complete record of all states and events in the past of F. Now, HEF represents a handle around which algorithms for resolving references to the future of E, up to F, can be developed. For example, the first causality rule of Section 1 can be expressed using definite clauses as follows:

```
causal_connection(E,HE,F,HEF) if
        E=begin_flight_towards(A,R,T),
        F=detects(R,A,T+Delay),
        intersect_path(A,R,Delay,HE*[E]),
        non_member(changes_course(A,?),HEF).
non_member(Template,[]).
non_member(Template,[Event|Rest]) if
        non_matches(Event,Template), non_member(Template,Rest).
```

The first rule states that E is causally connected to F in the context HE*[E]*HEF*[F] provided E=begin_flight_towards(A,R,T), F=detects(R,A,T+Delay), Delay is obtained by intersecting the path of A at T, with coverage of R, and no event of the form changes_course(A,?) appears in HEF. Note that by the fundamental assumption, HE*[E] represents the state of the system at T (the time stamp of E), after E has occurred. Thus, intersect_path(A,R,Delay,HE*[E]) replaces the condition intersect_path(A,R,Delay,T) in the original causality rule.

The second rule states that no event of the form Template occurs in the empty list. The third states that no event of the form Template occurs in [Event|Rest], provided Event does not match Template, and no event of the form Template appears in Rest.

A template is a term containing zero or more occurrences of the special 0-ary function symbol ?. ? represents a slot or placeholder. A non-template term P is said to match a template T, provided if occurrences of ? in T are replaced by suitable terms, P is obtained. For example, f(1,2) matches f(?,2), but not f(?,3). A definition of non_matches is easy to write using definite clauses and is not given here.

Note how the condition in the original causality rule:

$\neg\exists X.\ T<X<T+Delay \wedge occurs(changes\_course(Aircraft,X))$.

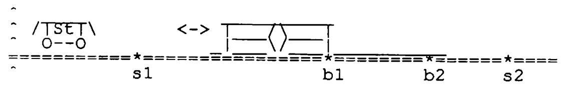
is replaced by:

```
non_member(changes_course(A,?),HEF).
```

The first represents a search over a real-valued interval of time whereas the second a (computationally simpler) iteration over a sequence of discrete events.

A DMOD program is a definition of causal_connection using definite clauses. The full expressive power of definite clauses can be employed while writing these. In particular, qualitative, symbolic and mathematical knowledge can be freely expressed. In the next section we give a full example of a DMOD model.

4. A DMOD MODEL OF A RAILROAD CROSSING



We now model a hybrid system, namely, a railroad crossing. It has been heavily adapted from [Ostroff 1989], in particular, with the introduction of continuous time and position. Its discrete parameters are velocities of barriers, and engines, which change abruptly.

In the above diagram, engines move on a track from left to right with a fixed velocity. The track can be crossed between b1 and b2. A barrier slides back and forth to close or open the crossing. When an engine reaches s1, sensor(1) picks up its position and activates closing of the barrier. When an engine reaches s2, sensor(2) picks up its position and activates opening of the barrier. The barrier opens and closes at a finite speed vb.

An interesting feature of the system is that closing or opening can be interrupted. For example, arrival of an engine at s1 may activate closing, but the engine may move so fast that before the barrier has fully closed, the engine arrives at s2, activating barrier opening.

Notation convention. If X is an event, time(X) represents the time stamp of X, otherwise if it is a history, the time stamp of the last event in X.

Each event is of one of the following forms, with meanings given in curly braces:

```
begin_j(e(X),P,V,T) {engine e(X) begins journey at position P, with
velocity V, at time T}
sensed(e(X),Sensor,T) {e(X) is sensed by Sensor at T}
start(barrier,close,T) {barrier closing starts at T}
end(barrier,close,T)  {barrier closing ends at T}
start(barrier,open,T) {barrier opening starts at T}
end(barrier,open,T) {barrier opening ends at T}
start(0)      {The initial event with time-stamp 0}
```

The following are the causality rules:

{C1. There are k engines e(1),...,e(k). For each i, e(i) begins journey from pe, with velocity ve at ti. }

```
causal_connection(start(0),_,begin_j(e(1),ve,pe,t1),_).
--
causal_connection(start(0),_,begin_j(e(k),ve,pe,tk),_).
```

{C2a. If e(X) begins journey at time T with position P and velocity V, then it is sensed by sensor(1) after time taken to travel (s1-P)/V. Its velocity always remains constant. Similarly for sensor(2). }

```
causal_connection(E,HE,F,HEF) if
        E=begin_j(e(X),V,P,T),
        F=sensed(e(X),sensor(1),T+Delay),
        Delay=(s1-P)/V.
```

{C2b. }

```
causal_connection(E,HE,F,HEF) if
        E=begin_j(e(X),V,P,T),
        F=sensed(e(X),sensor(2),T+Delay),
        Delay=(s2-P)/V.
```

{C3. If e(X) reaches sensor(1) then the barrier starts to close immediately. }

```
causal_connection(E,HE,F,HEF) if
        E=sensed(e(X),sensor(1),T),
        F=start(barrier,close,T).
```

{C4. If e(X) reaches sensor(2) then the barrier starts to open immediately. }

```
causal_connection(E,HE,F,HEF) if
        E=sensed(e(X),sensor(2),T),
        F=start(barrier,open,T).
```

{C5. If barrier starts to close then it ends closing after the time taken to reach fully closed position at its current velocity, provided no engine is sensed by sensor(1) in between. }

```
causal_connection(E,HE,F,HEF) if
        E=start(barrier,close,T),
        F=end(barrier,close,T+Delay),
        position(barrier,P,HE*[E]),
        velocity(barrier,V,HE*[E]),
        Delay=(b2-P)/V,
        non_member_1(sensed(e(?),sensor(2),?),HEF).
```

{C6. If barrier starts to open then it ends opening after the time taken to reach its fully open position at its current velocity, provided no engine is sensed by sensor(1) in between. }

```
causal_connection(E,HE,F,HEF) if
        E=start(barrier,open,T),
        F=end(barrier,open,T+Delay),
        position(barrier,P,HE*[E]),
        velocity(barrier,V,HE*[E]),
        Delay=(P-b1)/-V,
        non_member_1(sensed(e(?),sensor(1),?),HEF).
```

The following are auxiliary rules, e.g. for computing various state parameters such as position and velocity of engines or barriers.

{Position of Object immediately after the last event E in a history HE*[E] is NewPos, provided its position immediately after the last event in HE is OldPos, its velocity then is V, and one adds the incremental displacement.

Note the use of continuous functions +, - and *. More complex functions can similarly be used. }

```
position(Object,NewPos,HE*[E]) if
        position(Object,OldPos,HE),
        velocity(Object,V,HE),
        NewPos=OldPos+V*(time(E)-time(HE)).
```

{Position of Object at T is calculated as above, provided Hist_Till_T is the sequence of all occurring events with time stamps less than or equal to T. }

```
position_at(Object,T,NewPos,Hist_Till_T) if
        position(Object,OldPos,Hist_Till_T),
        velocity(Object,V,Hist_Till_T),
        NewPos=OldPos+V*(T-time(Hist_Till_T)).
```

{Position of e(X) immediately after the last event E in history is P, if E=begin_j(e(X),_,P,_). Similarly, for velocity of e(X). }

```
position(e(X),P,_*[begin_j(e(X),_,P,_)]).
velocity(e(X),V,_*[begin_j(e(X),V,_,_)]).
```

{Velocity of barrier immediately after the last event E in history is 0, if E=end(barrier,_,_). If E=start(barrier,close,_), velocity is +vb. Similarly, when E=start(barrier,open,_). If E is not equal to any of these, then velocity immediately after E is the same as that immediately after the last event in HE. }

```
velocity(barrier,0,_*[end(barrier,_,_)]).
velocity(barrier,+vb,_*[start(barrier,close,_)]).
velocity(barrier,-vb,_*[start(barrier,open,_)]).
velocity(barrier,P,HE*[E]) if
        non_member_2(E, [start(barrier,?,?),end(barrier,?,?)]),
        velocity(barrier,P,HE).
```

{non_member_1 is identical to non_member of Section 2. non_member_2 is similar, except that arguments to non_matches are reversed.}

```
non_member_1(E,[]).
non_member_1(E,[A|B]) if
        non_matches(A,E), non_member_1(E,B).

non_member_2(E,[]).
non_member_2(E,[A|B]) if
        non_matches(E,A), non_member_2(E,B).
```

{Initial conditions. Position and velocity of barrier after the first event start(0) are b1 and 0 respectively.}

```
position(barrier,b1,[start(0)]).
velocity(barrier,0,[start(0)]).
```

5. COMPUTATION OF HISTORY

Given a DMOD program P (a definition of causal_connection) we now show how to use it to compute a sequence of event occurrences, i.e. a history. A history can be computed in a bottom-up manner. Let the initial event occur. Compute the events it causes, then compute the events these cause, and so on. However, as causality itself requires history to be evaluable, the situation is not quite simple. We first define the history declaratively, taking the aid of the following diagram:

```
E0             Ei                  Ej
o--o---o--o--o---o---------o--o-----o (S)
<---HEi--->        <--HEiEj-->
```

Notation conventions. Ea,..,Eb denotes the sequence Ea,Ea+1,..,Eb. If a>b then Ea,..,Eb denotes the empty sequence. Informally, HE denotes the sequence of all events up to but not including E and HAB denotes the sequence of events between A and B but not including either. Thus, HEi denotes E0,E1,..,Ei-1 and HEiEj denotes Ei+1,..,Ej-1.

Let S=E0,E1,E2,... be a finite, or enumerably infinite, sequence of events sorted in increasing order of time-stamps, in which an event appears at most once. Then S is said to be causally-sound if every event in S has a cause in S, i.e.:

```
causally-sound(S) iff ∀j.j>0⊃∃i.i<j∧ causal_connection(Ei,HEi,Ej,HEiEj).
```

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 finite, or enumerably infinite, sequence of events sorted in increasing order of time-stamps, in which an event appears at most once. Then we have:

```
causally-complete(S) iff (a) ∧ (b) where:

(a) is ∀i.∀j.∀G.
        i<j ∧
        ¬G∈ (E0,..,Ej-1) ∧
        causal_connection(Ei,HEi,G,HEiEj)⊃ time(G)≥time(Ej).

(b) is S=E0,E1,..,Ek ⊃ ¬∃G.∃i.causal_connection(Ei,HEi,G,(Ei+1,..,Ek)).
```

Informally, condition (a) says it must not be possible for an event G, not already occurring in E0,..,Ej-1, to occur after Ej-1 but strictly before Ej. Condition (b) says that if S contains a last event Ek, then 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, unique, initial event for the modeled system. Assume that E0 has occurred. A history of the system is defined to be a finite, or enumerably infinite, sequence of events starting at E0 which is both causally-sound and causally-complete, i.e.:

```
history(X) iff
        initial_event(X,E0) ∧
        causally-sound(X) ∧
        causally-complete(X)
```

Intuitively, a history contains all of the events whose occurrence is required by the occurrence of the initial event and the causality rules, and only these events. If there is more than one initial event, a new initial event can be created which causes each of these.

More than one sequence can be a history. This happens when concurrent events occur. A different history results depending on the order in which concurrent events are recorded. 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 initial event. After p(1) both q(2) and r(2) can occur.

5.1 A Simple Procedure to Compute History

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 and Fi is not already in E0,..,Em. Here 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 procedure halts.

Intuitively, given a partial history H we determine all the events Fi which are caused by an event in H with H*[Fi] as context, and not already present in H. Of these we pick the earliest event as the next one.

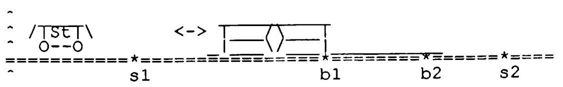
As there may be more than one event in Sm with least time-stamp, the procedure is non-deterministic. A different history would be computed for each choice of Em+1, signifying that the system is concurrent.

Inference of causal_connection can be performed using SLD-resolution. For each i, one can evaluate the query ∃F.causal_connection(Ei,(E0,E1,..,Ei-1),F,(Ei+1,..,Em)) in all possible ways to obtain Sm. We can now prove:

Theorem A. Correctness of simple 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. Simple, [Narain & Rothenberg 1990], but not necessary for this paper.

5.2 Simulating the Railroad Crossing



In the DMOD program in Section 4, let there be just one engine e(1). Let ve=40, vb=10, and s1,s2,b1,b2 be, respectively, 10,20,30,40. Let e(1) begin journey from position 0 at time 0, i.e. start(0) causes just one event as given by:

causal_connection(start(0),_,begin_j(e(1),10,0,0),_).

To illustrate the simulation algorithm, let E0=start(0) occur. Then, by the above rule, S0={begin_j(e(1),40,0,0)}. Clearly, E1=begin_j(e(1),40,0,0).

Now, by C2a and C2b, we obtain S1={sensed(e(1),sensor(1),0.25), sensed(e(1),sensor(2),1.0)}. Even though, by C1, causal_connection(E0,[],E1,[E1]), E1 is not a member of S1 since it has already occurred. Thus, E2=sensed(e(1),sensor(2),0.25). Proceeding this way, we obtain:

E0=start(0)
E1=begin_j(e(1),40,0,0)
E2=sensed(e(1),sensor(1),0.25) (0.25 = (s1-0)/40).
E3=start(barrier,close,0.25)
E4=sensed(e(1),sensor(2),1.0) (1.0 = (s2-0)/40).
E5=start(barrier,open,1.0)
E6=end(barrier,open,1.75).

Note that the engine moves so fast that it reaches s2, before the barrier fully closes. Thus, even though the event G=end(barrier,close,1.25) is potentially caused by E3, its occurrence is "cancelled" by E4=sensed(e(1),sensor(2),1.0). In particular, we do not have causal_connection(E3,[E0,E1,E2],G,[E4]). The presence of E5 makes the last condition in the body of C5 false. Thus G is not in S4 (or S5 or S6). Even though it is in S3, it does not appear in the history because E4 also appears in S3, and has a lower time stamp.

Note that this algorithm makes no use of devices of event queues, scheduling and unscheduling which are the basis of the event-scheduling view of the discrete-event technique e.g. [Evans 1988]. The resulting simplification is of considerable importance in reasoning about programs.

6. REASONING ABOUT MODELS

An especially important aspect of DMOD is that it yields a basis for formally proving temporal properties of modeled systems. Thus, questions can be answered, which cannot be answered by simulating these systems a finite number of times, or for a finite amount of time. Such questions include "what is the maximum value an output parameter can ever achieve?" or "will an event ever occur?".

We first present the general framework in which temporal properties can be formulated and proved. Then, we present a heuristic for guiding the search for proofs. Finally, we illustrate these ideas by proving a liveness property for the railroad crossing system.

Let P be a DMOD program, and E0 a special initial event for P. Then, the definition of a history for P, in Section 5 is:

history(X) iff initial_event(X,E0) $\wedge$ causally-sound(X) $\wedge$ causally-complete(X)

where causally-sound and causally-complete are defined in Section 5. We also need a closed world assumption to express the fact that P embodies complete information about relations defined by it, particularly, causal_connection. Let P |--SLD-- S mean that there is a proof of S from P, via SLD-resolution. We now define:

CWA. SLD-correctness of DMOD programs. Let P be a DMOD program. Let R be an n-ary predicate symbol defined by P, i.e. there is a clause in P of the form "R(X1,..,Xn) if B1,..,Bk", k>=0. Let * be a special symbol not occurring in P. Let A be an expression not containing *. Define A* to be the result of replacing each function or relation symbol F in A by F*. Then:

(A2) $\forall$T1,..,$\forall$Tm.R(T1,..,Tm) iff P* |--SLD-- R*(T1*,..,Tm*)

In particular, P can be regarded as a causality machine. To settle a proposition of the form causal_connection(E,HE,F,HEF), we submit it to this machine and wait for its answer. *For economy of notation, we will not, in general, distinguish between A* and A. Any ambiguity will be resolvable from context.*

Now, to show that each history X satisfies a condition r, we can show that

$\forall$X.history(X) $\supset$ r(X)

is a logical consequence of (A1) and (A2). Let X0,X1,... be the members of a sequence X. Then, an example of r is:

$\exists$k$\exists$T.Xk=end(barrier,close,T)

which denotes the liveness property that the barrier eventually closes. Another example of r is:

$\neg\exists$k.$\exists$E.$\exists$Pe$\exists$Pb.
  position(E,Pe,(X0,..,Xk)) $\wedge$
  b1<Pe<b2 $\wedge$
  position(barrier,Pb,(X0,..,Xk)) $\wedge$
  b1<Pb<b2.

which denotes the safety property that there is no Xk such that both the position of an engine, and that of the barrier, immediately after Xk, are strictly between b1 and b2.

Safety and liveness properties are generally proved for all histories. However, sometimes one may be interested in knowing whether a property r could be satisfied for some, not necessarily all, histories. Then one could show that:

$\exists$X.history(X)$\wedge$r(X)

is a logical consequence of (A1) and (A2) In principle, a theorem prover could accept (A1),(A2) and definitions of r, and prove appropriate properties. However, as there is no restriction on the form of r, proof spaces can be quite complex.

6.1 A Heuristic For Proving Temporal Properties

We now present a heuristic for guiding the search for proofs of temporal properties. It is is based upon expressing temporal properties in terms of questions about event occurrences. For example, to show that the value of a parameter P eventually becomes X, determine which event E can make it so, and show that E occurs. Many temporal properties are directly questions about event occurrences. For example, do at least N events occur before time T, if A occurs before B then does C occur before D, or is there a last event occurrence?

Questions about event occurrences can be answered in a natural manner by utilizing properties of the causality relation between events. For example, to show that a non-initial event occurs, show that at least one of its causes occurs. Conversely, to show that a non-initial event does not occur, show that none of its causes occurs. The important liveness and safety properties can be proved this way. If each occurring event always causes at least one event then there is no last event occurrence.

Central to the success of this approach is the simplicity with which the causality relation can be inferred. As it is expressed using definite clauses, use of the SLD-resolution proof procedure greatly simplifies its inference.

We now make these ideas precise. The following two theorems are used:

Theorem 1. Every non-initial event has a cause. Let P be a DMOD program and e0,e1,... be a history computed from it. Then:

$\forall$k.k>0$\supset\exists$i.i<k $\wedge$ causal_connection(ei,(e0,e1,..,ei-1),ek,(ei+1,..,ek-1)).

Proof: From causal-soundness of history. **QED.**

Theorem 2. If an event does not occur then it must not have a cause. Let P be a DMOD program and H=e0,e1,... a history computed from it. Let F be an event not occurring in H. Let ek be the last event in H whose time stamp is less than or equal to that of F. Then:

$\neg\exists i.0\leq i\leq k \wedge$ causal_connection(ei,(e0,e1,..,ei-1),F,(ei+1,..,ek)).

Proof. From causal-completeness of history. **QED**.

Now, for answering questions about event occurrences, two strategies can be used. First, to show that an event F does not occur in a history, assume that it does. Then, by Theorem 1, there must be an event E prior to F in the history such that E is causally connected to F. By (A2) at least one attempt to prove, by SLD-resolution, that E is causally connected to F succeeds. Show that every such attempt fails, and thereby derive a contradiction. Alternatively, repeat this sequence of steps for E.

Second, to show that an event F occurs in a history, assume it does not. Then, by Theorem 2, no event E in the history such that time(E)≤time(F), is causally connected to F. By (A2), for each such E, every attempt to prove, by SLD-resolution, that E is causally connected to F will fail. Show that at least one such attempt succeeds, and thereby derive a contradiction.

In showing the existence or non existence of proofs, it may be necessary to show that other events occur or do not occur. Then, the above strategies can be recursively employed. The intuitive nature of SLD-resolution greatly simplifies checks for existence or non existence of proofs.

6.2 Proving a Liveness Property

We now illustrate the above ideas by proving a liveness property for the railroad crossing. Let P be the DMOD program in Section 4, with just one engine e(1) beginning journey from position 0, at time 0, with velocity ve. In particular, start(0) causes just one event as specified by:

```
{C1}
causal_connection(start(0),_,begin_j(e(1),ve,0,0),_).
```

Also, let A be the conjunction of:

```
s2>b2>b1>s1,
vb>0,
ve>0,
(b1-s1)/ve > (b2-b1)/vb
```

The last condition states that the time taken for the engine to move from s1 to b1 is greater than that taken by the barrier to move from b1 to b2 (i.e. to fully close).

In the following, let e0,e1,.. be a history H of the DMOD program P, where e0=start(0). "E occurs" means there exists k such that ek=E. We now prove the following liveness property:

There exist k,t such that ek=end(barrier,close,t).

i.e. the barrier eventually ends closing. As we saw in Section 6.2, this is not obvious. The engine may move so fast that before the barrier ends closing, the engine reaches s2, so barrier starts opening.

Plan of proof. We can show that start(barrier,close,s1/ve) occurs. We can also show that the position of the barrier immediately after this event occurs, is b1.

Now, suppose no event of the form end(barrier,close,T) occurs. Then, in particular, end(barrier,close,t) does not occur where t=s1/ve + (b2-b1)/vb. By Theorem 2, it must not possess a cause. Thus, its cause must not be inferrable, by SLD-resolution, from P. Its cause is only inferrable from Rule C5. The first five conditions in the body can be proved. Thus, the sixth condition must not be provable.

This means an event of the form sensed(e(?),sensor(2),?) occurs, whose time stamp is greater than or equal to s1/ve, but less than or equal to t, i.e. the event is between the causing and caused events. But the only event of this form has the time stamp s2/ve. Now, s2/ve = (s1 + (s2-b1) + (b1-s1))/ve = (s1/ve + (s2-b1)/ve + (b1-s1)/ve). By constraint A, this is strictly greater than t which is (s1/ve + (b2-b1)/vb). Contradiction. **QED.**

We now carry out this plan in some detail.

Lemma 1. The event E=begin_j(e(1),ve,0,0) occurs.

Proof. Suppose E does not occur. Let ek be the last event in H whose time stamp is less than or equal to 0. By Theorem 2, there does not exist i, i<k such that causal_connection(ei,(e0,e1,..,ei-1),E,(ei+1,..,ek)). In particular, it is not the case that causal_connection(start(0),[],E,(e2,..,ek)). By (A2), there must not be a successful SLD-derivation starting at this. However, by C1, we have the successful SLD-derivation:

causal_connection(start(0),[],begin_j(e(1),ve,0,0),(e2,..,ek)).
□

Contradiction. **QED.**

Lemma 2. The event E=sensed(e(1),sensor(1),t1) occurs where t1=s1/ve.

Proof. By Lemma 1, begin_j(e(1),ve,0,0) occurs. Proceed as with Lemma 1, using Rule C2a. **QED.**

Lemma 3. The events start(barrier,close,s1/ve) and sensed(e(1),sensor(2),s2/ve) occur.

Proof. Similar to that for Lemma 1, and using Lemma 2.

Lemma 4. If sensed(e(X),sensor(2),T) occurs, then X=1 and T=s2/ve.

Proof Sketch. Only one event of the form start(0) occurs. By C1, only one event begin_j(e(X),V,P,T) occurs where X,V,P,T are, respectively, 1,ve,0,0. By C2b, only one event sensed(e(X),sensor(2),T) occurs, where X=1, T=s2/ve. **QED.**

Lemma 5. Let position(barrier,P,(e0,e1,..,ek)) where ek=start(barrier,close,s1/ve). Then, P=b1.

Proof. The last two rules in P, in Section 4, state that the position and velocity of barrier, immediately after start(0) occurs, are b1 and 0 respectively. We show that there is no event ei, i<k, which makes the velocity of barrier non zero. Then, by rule for computing position, the lemma follows. If such an event occurs then it is of the form start(barrier,?,?). Let it be start(barrier,open,t).

By Theorem 1, it must possess a cause. By C2b, this must be sensed(e(X),sensor(2),t) for some X. By Lemma 4, X=1,t=s2/ve. Contradiction, as s2/ve > s1/ve. Similarly, if ei=start(barrier,close,t). **QED.**

Liveness property. E=end(barrier,close,t1) occurs where t1=s1/ve+(b2-b1)/vb.

Proof. Suppose E does not occur. As b2>b1 and vb>0, t1>s1/ve. Also, by Lemma 3, start(barrier,close,s1/ve) occurs. Let it be ei. By Theorem 2 there is no successful SLD-derivation starting at:

causal_connection(start(barrier,close,s1/ve),he,end(barrier,close,t1),hef)

he=(e0,e1,..,ei-1) and hef=(ei+1,..,ek), where ek is the last event in the history with time stamp less than or equal to t1. Let this goal be G. Clearly, it unifies with the head of C5 with the substitution:

σ={<E,ei>, <HE,he>, <F,end(barrier,close,t1)>, <HEF,hef>}

Then, there is no successful SLD-derivation of the body of C5 to which σ is applied. However, the first five conditions can be proved immediately (using, in particular, Lemma 5). Thus, non_member_1(sensed(e(?),sensor(2),?),hef) cannot be proved. This means that some event sensed(e(X),sensor(2),T) has occurred in between start(barrier,close,s1/ve) and end(barrier,close,t1). Clearly, s1/ve≤T≤t1.

By Lemma 4, X=1 and T=s2/ve. We just inferred s2/ve ≤ s1/ve+(b2-b1)/vb. Thus, (s2-s1)/ve ≤ (b2-b1)/vb. By given constraint A above, (b2-b1)/vb < (b1-s1)/ve. Thus, (s2-s1)/ve < (b1-s1)/ve. Thus, s2<b1. Contradiction with A. **QED.**

7. SUMMARY

Formal techniques to prove temporal properties are generally restricted to either purely discrete systems or to purely continuous systems. It appears far from obvious how they can be extended to reason about hybrid systems which arise frequently in the real world.

In this paper, we presented a formal technique, DMOD, for modeling hybrid systems. It utilizes the tractable proof theory of definite clauses, and our intuitions about causality. We presented an algorithm to simulate with DMOD models. Then, we outlined the general framework in which temporal properties of hybrid systems can be formulated and proved, (given that those systems are modeled using DMOD). Finally, we illustrated these ideas by proving a liveness property about a railroad crossing.

REFERENCES

Apt, K.R., M.H. van Emden (1982), "Contributions to the Theory of Logic Programming,"*Journal of the ACM 29*, 3.

Bernstein, A. and P. Harter (1981), "Proving Real-time Properties of Programs with Temporal Logic," In *Proceedings of ACM Conference on Operating Systems Principles*.

Cameron, E., D. Cohen, B. Gopinath, W. Keese, L. Ness, P. Uppaluru, and J. Vollaro (1988), "The IC* Model of Parallel Computation and Programming Environment," *IEEE Transactions on Software Engineering 14*, 3.

Cammarata, S., B. Gates, J. Rothenberg (1988), "Dependencies, Demons and Graphical Interfaces in the ROSS Language," N-2589-DARPA, RAND Corporation, Santa Monica, CA.

Cleary, J. (1990), "Colliding Pucks Solved Using a Temporal Logic," In *Proceedings of Distributed Simulation Conference*, Society for Computer Simulation, San Diego, CA.

Davis, E. (1988), "A Logical Framework for Commonsense Predictions of Solid Object Behavior," *Artificial Intelligence in Engineering 3*, 3.

Davis, M., S. Rosenschein, and N. Shapiro (1982), "Prospects and Problems for a General Modeling Methodology," N-1801-RC, RAND Corporation, Santa Monica, CA.

de Kleer, J. and J. Brown (1984), "A Qualitative Physics Based upon Confluences," *Artificial Intelligence Journal 24*, 7.

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.

Forbus, K. (1984), "Qualitative Process Theory," *Artificial Intelligence Journal 24*, 85-168.

Galton, A. (1988), *Temporal Logics and Their Applications*, Academic Press, New York.

Harel, D. (1984), "Statecharts: A Visual Approach to Complex Systems," In *Proceedings of Advanced NATO Study Institute on Logics and Models for Verification and Specification of Concurrent Systems, NATO ASI series F 13*, 1-44.

Ho, Y.C. (1987), "Performance Evaluation and Perturbation Analysis of Discrete-event Dynamic Systems," *IEEE Transactions on Automatic Control AC-32*, 7.

Hobbs, J. and R. Moore, Eds. (1985), *Formal Theories of the Commonsense World*, Ablex Publishing Corporation, Norwood, NJ.

IEEE (1989), "Special Issue on Dynamics of Discrete-event Systems," In *Proceedings of the IEEE*.

Iwasaki, Y. and H. Simon (1986), "Causality in Device Behavior," *Artificial Intelligence 29*.

Jahanian, F. and A. Mok (1986), "Safety Analysis of Timing Properties in Real-time Systems," *IEEE Transactions on Software Engineering*.

Jefferson, D. (1985), "Virtual Time," *ACM Transactions on Programming Languages and Systems*.

Kiviat, P.J. (1967), "Digital Computer Simulation: Modeling Concepts," RM-5378-PR, RAND Corporation, Santa Monica, CA.

Kowalski, R. (1979), *Logic for Problem Solving*, Elsevier North Holland, New York.

Kowalski, R. (1986), "Database Updates in the Event Calculus," DoC 86/12, Department of Computing, Imperial College, London.

Kowalski, R. and M. Sergot (1986), "A Logic-based Calculus of Events," *New Generation Computing 4*, Ohmsha Limited and Springer Verlag.

Kuipers, B. (1986), "Qualitative Simulation," *Artificial Intelligence 29*, 289-338.

Lamport, L. (1983), "What Good is Temporal Logic?," In *Proceedings of IFIP*, R.E.A. Mason, Ed. North Holland, Amsterdam.

Lloyd, J. (1984), *Foundations of Logic Programming*, Springer Verlag, New York.

Manna, Z. and A. Pnueli (1981), "Temporal Verification of Concurrent Programs," In *The Correctness Problem in Computer Science*, R.S. Boyer and J.S. Moore, Eds. Academic Press, New York.

McArthur, P. Klahr, and S. Narain (1986), "ROSS: An Object-oriented Language for Constructing Simulations," In *Expert Systems: Techniques, Tools, Applications*, P. Klahr and D. Waterman, Eds. Addison Wesley, Reading, MA.

McCarthy, J. (1987), "Generality in Artificial Intelligence. Turing Award Lecture," *Communications of the ACM*.

McCarthy, J. and P. Hayes (1969), "Some Philosophical Problems from the Standpoint of Artificial Intelligence," In *Machine Intelligence*, B. Meltzer and D. Michie, Eds. Edinburgh University Press, Edinburgh.

McDermott, D. (1982), "A Temporal Logic for Reasoning about Processes and Plans," *Cognitive Science*, 101-155.

Misra, J. (1986), "Distributed Discrete-event Simulation," *Computing Surveys*.

Nance, R. (1981), "The Time and State Relationships in Simulation Modeling," *Communications of the ACM*.

Narain, S. (1990), "A New Simulation Technique and its Implementation in Prolog," To appear in *Prolog and its Applications*, A. Bond, Ed. MIT Press.

Narain, S. and J. Rothenberg (1989), "A Logic for Simulating Dynamic Systems," In *Proceedings of the 1989 Winter Simulation Conference*, E.A. MacNair, K.J. Musselman, and P. Heidelberger, Eds. IEEE, Piscataway, NJ.

Narain, S. and J. Rothenberg (1990), "A New Modeling Technique Based Upon the Causality Relation," WD-4828-DARPA, RAND Corporation, Santa Monica, CA.

Ostroff, J. (1989), "Synthesis of Controllers for Real-time Discrete-event Systems," In *Proceedings of IEEE Conference on Decision Control*.

Owicki, S. and L. Lamport (1982), "Proving Liveness Properties of Concurrent Programs," *ACM TOPLAS 4*, 3, 455-495.

Pnueli, A. (1977), "The Temporal Logic of Programs," In *Proceedings of the 18th Symposium on Foundations of Computer Science*.

Pnueli, A. (1981), "The Temporal Semantics of Concurrent Programs," *Theoretical Computer Science 13*, 45-60.

Peterson, J.L. (1977), "Petri Nets," *ACM Computing Surveys 9*, 3.

Rothenberg, J., S. Narain, R. Steeb, C. Hefley, and N. Shapiro (1988), "Knowledge-based Simulation: An Interim Report," N-2897-DARPA, RAND Corporation, Santa Monica, CA.

Sandewall, E. (1989), "Combining Logic and Differential Equations for Describing Real-world Systems," In *Proceedings of International Conference on Knowledge Representation*, Toronto, Canada.

Schruben, L. (1983), "Simulation Modeling with Event Graphs," *Communications of the ACM*.

Shoham, Y. (1987), "Temporal Logics in AI: Semantical and Ontological Considerations," *Artificial Intelligence Journal 33*, 89-104.

Suri, R. (1987), "Infinitesimal Perturbation Analysis for General Discrete-event Systems," *Journal of the ACM*.

Zeigler, B. (1984), *Multifacetted Modeling and Discrete-event Simulation*, Academic Press, New York.