Symbolic Simulation: Theory and Application to Protocol Modeling and Validation

Ritu Chadha & Işıl Sebüktekin

Bellcore
Morristown, NJ 07960

Abstract

We present a novel technique called symbolic simulation, and its application to protocol modeling and validation. Whereas simulation produces a trace of the behavior of a system for certain fixed parameter values, symbolic simulation allows one to simulate the behavior of a system for entire ranges of parameter values. This in turn allows verification of certain properties of the system behavior for those ranges of values. This paper lays the theoretical foundation for symbolic simulation, and describes its application to proving safety and liveness properties of protocols. As an example, we have modeled a timer-based connection management protocol and validated one of its safety properties. We conclude that symbolic simulation is a useful means for modeling and validation of complex, real-time communication protocols.

1 Introduction

DMOD (Declarative MODeling) [7, 8] is a discrete-event simulation system that, unlike other existing simulation tools, simplifies modeling temporal concepts such as state transitions, system state, event preemptions, historical reference, and dense time. DMOD is a suitable tool for producing simulations of the behavior of a system when all the timing parameters for the system, i.e. the times at which events occur, are known in advance. However, it often happens that timing values are given as ranges of real numbers rather than fixed values. In order to simulate the behavior of a real-time system for the entire ranges of its parameter values, we extended DMOD to *symbolic DMOD*, using Constraint Logic Programming [5], to perform symbolic simulation, which allows event times and system parameters to lie in ranges rather than being fixed, as explained in Section 2.2. This in turn allows verification of system properties for those ranges of values. In [11], Constraint Logic Programming is used to simulate discrete event processes; however, only discrete time is handled, whereas DMOD can handle continuous time.

We used symbolic DMOD for modeling and validating a safety property of a complex and dynamic communication protocol - a timer-based connection management protocol named CMSC (Connection Management with Synchronized Clocks) [2]. [2] provides a detailed description of CMSC and mathematical proofs of its safety and liveness properties. As an example of the application of symbolic DMOD to protocol modeling and validation, we present a validation of a specific safety property of the protocol, and compare it with its manual proof in [2]. Other safety and liveness properties could be validated similarly. Application of symbolic DMOD to analysis of other protocols, e.g. SONET, can be found in [9, 10].

2 DMOD & Symbolic DMOD

In this section, we establish the theoretical foundations for DMOD and symbolic DMOD, and describe a procedure for performing symbolic simulation.

2.1 DMOD

DMOD is based upon the assumption that the behavior of a system can be modeled by its history, i.e. the sequence of the *discrete* events which occur in it. Thus, simulation of a system can be regarded as computing its history. DMOD defines a formal framework for modeling systems and computing their histories.

Definition 2.1 A *DMOD structure* is a tuple *(Events, causes, init_event)* where:

- *Events* is an enumerably infinite set of ordered pairs *(e,t)*, called events, where e is some object and t is a non-negative real number called the *timestamp* of e, written $time((e,t)) = t$.
- $init\_event \in Events$ and is called the *initial event*.
- *causes(E,HE,F)* is a relation between events E and F, and a finite sequence HE of events satisfying:

0-8186-6685-4/94 $4.00 © 1994 IEEE

(a) $\forall E \forall HE$ the set $\{F \mid causes(E,HE,F)\}$ is finite
(b) $\forall E \forall HE \forall F(causes(E,HE,F) \Rightarrow time(E) \leq time(F))$.
An event is assumed to be discrete and instantaneous. *causes(E,HE,F)* means E causes F given that HE is the sequence of all the events before E.

Let *P=(Events, causes, init_event)* be a DMOD structure and $S=[E_0, E_1, ...]$ be a finite, or enumerably infinite sequence of events in *Events*. We have the following three definitions:

Definition 2.2 S is said to be *temporally ordered* if for each i, $i \geq 0 \Rightarrow time(E_i) \leq time(E_{i+1})$ whenever E_{i+1} exists.

Definition 2.3 S is said to be *causally-sound* if every event in S has a cause in S, i.e. $\forall j(j > 0 \Rightarrow \exists i(i < j \wedge causes(E_i, [E_0, .., E_{i-1}], E_j)))$.

Definition 2.4 S is said to be *causally-complete* if and only if it contains all caused events, i.e. $\forall G \forall i$ $(causes(E_i, [E_0, .., E_{i-1}], G) \Rightarrow \exists j(j > i \wedge E_j = G))$.

Definition 2.5 A *history* for a DMOD structure 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.

Definition 2.6 Let $Seq=[E_0, E_1, ..]$ be a temporally ordered sequence of events. Then *Seq* is said to be *strictly progressive* iff either *Seq* is finite, or *Seq* is infinite and for each real number t there is an i such that $t < time(E_i)$. Thus, a sequence of events is not strictly progressive if its timestamps converge.

Definition 2.7 Let S be a set of events. Then $E \in S$ is an *earliest event* in S if the timestamp of E is less than or equal to that of every other event in S.

If S contains concurrent events (i.e. with the same timestamp), there can be more than one earliest event in S. Also, if S is finite, it always contains an earliest event. We now present a procedure for computing a set of histories associated with a DMOD structure *P=(Events, causes, init_event)*.

Procedure 2.1 Let P be a DMOD structure. Record $E_0 = init\_event$ as the initial event. Suppose a finite initial segment $[E_0, E_1, .., E_m]$ of the history has been computed. We need to compute the next event E_{m+1}. For each $i, 0 \leq i \leq m$, let $Effects_i$ be the set of events F such that $causes(E_i, [E_0, .., E_{i-1}], F)$ and F is not already in $[E_0, E_1, .., E_m]$. Due to the restriction on *causes*, this set is finite. Let $S_m = \cup_{i=0}^{m} Effects_i$. S_m is also finite. If it is empty, halt. Otherwise, it contains an earliest event E_{m+1}; take it as the next event. ■

Intuitively, given a partial history H we determine the set S_m of all the events not in H, which are caused by an event in H. We pick as E_{m+1} an earliest event in S_m. Note that S_m can contain more than one earliest event (i.e. concurrent events) so the procedure is non-deterministic. A different history would be computed for each choice of E_{m+1}. A more efficient procedure has been developed which constructs S_m incrementally from S_{m-1} instead of computing it in its entirety. We can now prove:

Theorem 2.1 Soundness and Completeness of Procedure 2.1 Let P be a DMOD structure. A strictly progressive sequence of events $[E_0, E_1, ..]$ is computed by Procedure 2.1 iff it is a history. ■

The proof is straightforward and is omitted here.

2.2 Symbolic DMOD

As mentioned in Section 1, we extended DMOD in order to be able to handle events which occur in certain time intervals, rather than at fixed times. Instead of requiring the timestamp to be a non-negative real number, we allow it to be a variable, which could optionally have some constraints on its value.

If the time at which an event occurs is no longer fixed, but is a variable, then how do we order events temporally, as described in Procedure 2.1, where we constructed a history by picking earliest events? Clearly, the result of a simulation will no longer be a unique history, but rather a tree of histories, with each node of the tree being a decision point based on relative values of timestamps. The approach resembles that in [6]; however, their model does not address preemption issues, unlike DMOD. Also, their model allows the value of a timestamp to vary during a simulation run, whereas in symbolic DMOD, we assume that the value of a timestamp is some fixed value subject to given constraints. In both [3] and [12], preemption is handled, but there is no concept of allowing events to occur in time intervals. The emphasis of the work on model-checking ([1], [4]) is on verifying a property which is given as a formula of temporal logic by automatically comparing it with a state transition graph representing system behavior, whereas our approach is a discrete-event simulation approach which actually computes histories of events that can themselves be used to gain intuition about system behavior. For example, properties of the system that are to be proved are not always obvious. In such cases, looking at system behavior by examining histories can provide intuition about what properties could be proved for the system. These properties can then be proved using symbolic simulation.

We now formalize symbolic DMOD. We will then show the relationship between DMOD and symbolic DMOD histories; briefly, for every symbolic DMOD

history, there exists a corresponding DMOD history which is an instance of it; and conversely, given a DMOD history, there exists a symbolic DMOD history that it is an instance of.

Definition 2.8 A symbolic DMOD structure is a tuple *(Events, Constraints, Vars, causes, init_event)*, where

- *Events* is an enumerably infinite set of ordered pairs *(e,t)*, called events, where *e* is some object and *t* is either a variable in *Vars* ranging over the non-negative real numbers, or a non-negative real number, called the *timestamp* of *e*,written $time((e,t)) = t$.
- *Constraints* is a satisfiable set of constraints on elements of *Vars* (the set *Constraints* represents the conjunction of all the constraints in the set).
- *Vars* is a set of variables.
- $init\_event \in Events$ and is called the *initial event*.
- *causes(E,HE,F)* is a relation between events E and F, and a finite sequence HE of events satisfying:

(a) $\forall E \forall HE$ the set $\{F \mid causes(E,HE,F)\}$ is finite

(b) $\forall E \forall H E \forall F$ $\{causes(E,HE,F) \wedge Constraints \Rightarrow time(E) \leq time(F)\}$.

Definition 2.9 Let P = *(Events, Constraints, Vars, causes, init_event)* be a symbolic DMOD structure and $S = [E_0, E_1, ...]$ be a finite, or enumerably infinite sequence of events in *Events*. Then S is said to be *temporally ordered* if the set $\{time(E_0) \leq time(E_1), time(E_1) \leq time(E_2), ..., time(E_i) \leq time(E_{i+1}), ...\} \cup$ *Constraints* is satisfiable.

The terms *causally-sound* and *causally-complete* for symbolic DMOD structures are defined as they were for DMOD structures (Def. 2.3 and 2.4).

Definition 2.10 Let P be a symbolic DMOD structure. A *complete history set HS* for P is a set of finite or infinite sequences of events such that:

(a) Each sequence in HS begins with *init_event*

(b) For every $H \in HS$, no event occurs more than once in H

(c) For every $H \in HS$, H is causally-sound

(d) For every $H \in HS$, H is causally-complete

(e) For every $H \in HS$, H is temporally ordered

(f) If there exists a sequence of events H from *Events* such that H begins with *init_event*, no event occurs more than once in H, H is causally-sound, H is causally-complete, and H is temporally ordered, then H must belong to HS.

Definition 2.11 Let P = *(Events, Constraints, Vars, causes, init_event)* be a symbolic DMOD structure. Let S be a finite subset of *Events*. We say that an element e of S is an *earliest event* in S with respect to a set of constraints C if for every f in S, $f \neq e \Rightarrow (C \Rightarrow time(f) < time(e)$ is not valid).

We now present a procedure for computing a complete history set for a symbolic DMOD structure. In what follows, history segments are called $H_m^i = [E_0^i, E_1^i, ..., E_{m-1}^i]$, where i numbers the history segments and m is the number of elements in H_m^i. For a given m, $Effects_j^i$ is the set of effects (to be defined) of the j^{th} element of H_m^i.

Procedure 2.2 Let P be a symbolic DMOD structure. Record $H_1^0=[E_0]$ as the initial segment, where E_0 is the initial event, and let $CONSTRAINTS_1^0$ = *Constraints*. Suppose a set of $n+1$ finite, initial segments H_m^i, $0 \leq i \leq n$, of histories has been computed for which computation has not yet halted, each with m events and each with a set of constraints $CONSTRAINTS_m^i$. For each initial segment H_m^i, we need to compute the next event E_m^i. Suppose $H_m^i = [E_0^i, E_1^i, ..., E_{m-1}^i]$. For each $i, 0 \leq i \leq n$, and for each $j, 0 \leq j \leq m-1$, let $Effects_j^i$ (called the set of *effects* of E_j^i) be the set of events F such that $causes(E_j^i, [E_0^i, .., E_{j-1}^i], F)$ and F is not already in $[E_0^i, E_1^i, .., E_{m-1}^i]$. Due to the restriction on *causes*, this set is finite. Let $S_m^i = \cup_{j=0}^{m-1} Effects_j^i$, for $0 \leq i \leq n$. Again, these $n+1$ sets are finite. For each of these sets S_m^i, if it is empty, halt. Otherwise, S_m^i contains a set of earliest events $EARLY_m^i$, with respect to the constraints in $CONSTRAINTS_m^i$, with k^i elements (say). For each initial history segment H_m^i, we obtain k^i new histories, constructed by appending to the initial history segment H_m^i the k^i different elements of $EARLY_m^i$ as the last element, and k^i new sets of constraints corresponding to these histories, formed by taking the conjunction of $CONSTRAINTS_m^i$, the constraint that the new history is temporally ordered, and the constraint $time(L) \leq time(K)$ for all events K in $EARLY_m^i$, where L is the last element of the history.■

Intuitively, given a partial history H_m^i we determine the set S_m^i of all the events not in H_m^i, which are caused by events in H_m^i. We pick a set $EARLY_m^i$ of all earliest events from S_m^i, and construct k^i new history segments with $m+1$ elements each (where k^i is the cardinality of the set $EARLY_m^i$). In practice we use a much more efficient procedure than the one outlined above. Note that it is necessary to introduce the extra constraints during the computation of the histories to ensure that the histories are temporally ordered, which ensures that HE is a well-defined sequence in the *causes* relation. Also, note that $CONSTRAINTS_m^i$ is still satisfiable for any i, m, because of the definition of earliest event and the *causes* relation.

We now present the relationship between DMOD and symbolic DMOD histories, as mentioned earlier in this section. The following two theorems prove that a

symbolic history is in fact a finite representation of an infinite number of DMOD histories.

Definition 2.12 Let V be a set of variables. Let σ be an assignment of real numbers to the variables in V. Let A be an object containing zero or more variables from V. Then $A(\sigma)$ is defined to be the object A with all occurrences of variables from V replaced by their assignments from σ.

E.g. if V={$t1, t2$}, σ={$t1 \leftarrow 4, t2 \leftarrow 5$}, A={$a(t1), b(t2), c(t2)$}, then $A(\sigma)$={$a(4), b(5), c(5)$}.

Theorem 2.2 Let S = *(Events, Constraints, Vars, causes, init_event)* be a symbolic DMOD structure. Let H be a symbolic history derived from S with constraints set $CONSTRAINTS_H$. Let σ be an assignment of non-negative real values to the variables in *Vars* such that σ satisfies $CONSTRAINTS_H$. Let P = *(Events(σ), causes(σ), init_event(σ))* be a DMOD structure. Then there exists a history H' derived from P such that $H' \subseteq H(\sigma)$, where H' and $H(\sigma)$ are the same except that any duplicate events in $H(\sigma)$ are removed from H'.

Proof: Suppose we start generating a history H' for the DMOD structure P. We show by induction on the length of this history that we can produce H' such that $H' \subseteq H(\sigma)$ and H' and $H(\sigma)$ are the same except that any duplicate events in $H(\sigma)$ are removed from H'. Let $H = [E_0, E_1, ...]$.

Base case: Since the initial event for P is simply the instance of the initial event for S with all variables replaced by ground values as specified by σ, the theorem is trivially true for histories of length 1.

Induction hypothesis: Suppose a segment H'_k of length $k+1$ of the history H' has been computed, $H'_k = [F_0, F_1, ..., F_k]$, where for every i, $0 \leq i \leq k$ there exists j, $0 \leq j \leq m$, for some $m \geq k$ such that $F_i = E_j(\sigma)$, and every event in $H_m(\sigma)$ is a member of H'_k, where $H_m = [E_0, E_1, ..., E_m]$.

Induction step: Let us now derive the next event F_{k+1} for the history H'. Let S'_k be the union of all the effects of events $F_0, F_1, ..., F_k$. Let S_m be the union of all the effects of events $E_0, E_1, ..., E_m$. Using the fact that the causality relation for the DMOD structure P is an instantiation of the causality relation for the symbolic DMOD structure S, and that H'_k and $H_m(\sigma)$ contain the same events, it can be shown that $S'_k \subseteq S_m(\sigma)$, and the only events that could be in $S_m(\sigma)$ and that are *not* in S'_k are events that have already appeared in H'_k.

The next event in the history H is E_{m+1}, therefore using the above result, either $E_{m+1}(\sigma) \in S'_k$, or $E_{m+1}(\sigma) \in H'_k$. If the latter is true, we are done; if the former is true, then since E_{m+1} is an earliest event in S_m (by definition), $E_{m+1}(\sigma)$ must be an earliest event in S'_k (because σ satisfies $CONSTRAINTS_H$). We can therefore pick the next event for the history H' to be $F_{k+1} = E_{m+1}(\sigma)$. Thus the history segment H'_{k+1} of H' with $k+2$ events is $H'_{k+1} = [F_0, F_1, ..., F_k, F_{k+1}]$, where for every i, $0 \leq i \leq k+1$, there exists j, $0 \leq j \leq m+1$, for some $m \geq k$ such that $F_i = E_j(\sigma)$, and every event in $H_{m+1}(\sigma)$ is a member of H'_{k+1}, where $H_{m+1} = [E_0, E_1, ..., E_{m+1}]$. This completes the proof by induction. ■

Theorem 2.3 Let S = *(Events, Constraints, Vars, causes, init_event)* be a symbolic DMOD structure. Let P = *(Events(σ), causes(σ), init_event(σ))* be a DMOD structure, where σ is some assignment of non-negative real values to the variables in *Vars* such that σ satisfies *Constraints*. Let H' be a history derived from the DMOD structure P. Then there exists a symbolic history H derived from the symbolic DMOD structure S such that $H(\sigma) \supseteq H'$, where H' and $H(\sigma)$ are the same except that any duplicate events in $H(\sigma)$ are removed from H'.

Proof: The proof is similar to that for Theorem 2.2.

3 DMOD Modeling of CMSC

As mentioned earlier, DMOD has the capability to model real-time systems, unlike most modeling and validation programs. DMOD simplifies modeling temporal concepts such as state transitions, system state, event preemptions, historical reference, and dense time. DMOD has been implemented in Quintus Prolog. In the following sections, we provide examples from the DMOD model of CMSC.

3.1 Causality Relations

In a DMOD model, each specific event can cause zero or more events to occur based on the causality relation (Definitions 2.1 and 2.8) defined as follows:

```
causes(E, HE, F) :-
  E = CausingEvent,
  F = CausedEvent,
  Conditions.
```

`E` and `F` are events of the form `f(T1,...,TN,T)` where `f` is the event-defining operator, `T1,...,TN` are terms and `T` is the timestamp of event `f`, and `HE` is the history of events up to `E` in reverse order. `Conditions` could be mathematical expressions based on state parameter values or indicate the value of some state parameters or constants. Here is an example from our DMOD model in which variables start with upper-case letters, and fixed parameters with lower-case letters.

```
causes(E, HE, F) :-
  E = send(xmtr, Message, Exptime, T),
  F = rec(rcvr, Message, Exptime, T1),
  rtd(RTD),
  T1 is T + (RTD/2),
  T1 =< Exptime,
  random(100, Z),
  Z > 25.
```

In this example, event E (the transmitter **xmtr** sending a message **Message** with expiration time **Exptime** at time T) causes event F (receipt of the same **Message** by the receiver **rcvr** at time T1 which is the sum of T and half the round trip delay RTD). F will not occur if **Message** has completed its lifetime in the system; i.e., if T1 > **Exptime**. **random(100,Z)** is a rule defined elsewhere in the program that generates a random value Z in the range [0,100]. F will not occur if $Z \leq 25$, modeling message losses with 25 % probability.

3.2 State Parameter Definitions

DMOD programs also contain state parameter definitions which are useful for modeling dynamic systems and system states, and therefore Finite State Machines (FSMs) and combination of FSMs. A typical state parameter definition in DMOD is:

```
parameter(Object, Value, History) :-
  History = [E | OldHistory],
  Conditions.
```

Above, a state parameter named **Object** has a value **Value** given that **History** consists of an event E followed by **OldHistory** (history of events up to E). Here is an example for the sequence number parameter (**seq_no**) in our DMOD model of CMSC.

```
seq_no(xmtr, SN, Hist) :-
  Hist = [newmsg(xmtr,setup,Exptime,T)|_],
  SN is 0.
seq_no(xmtr, SN, Hist) :-
  Hist = [newmsg(xmtr,msg(N),Exptime,T)|_],
  sn(SNspace),
  SN is (N+1) mod SNspace.
```

In the above, the **seq_no** parameter, **SN**, is set to 0 after a **setup** message, and is set to **N+1** modulo **SNspace** after a **msg(N)** with sequence number **N**, no matter what the previous events are. "_" stands for "any OldHistory" (or "any value" when used in place of a variable) and "|" means "followed by".

3.3 Modeling Event Preemptions

Event preemptions are not unusual in dynamic systems, and the capability to model them is useful. DMOD provides this capability as follows: suppose an event E causes an event F at time $\mathbf{T}_F$, which can be preempted by the occurrence of one or more of the events in a list L. Then, instead of letting E cause F, we let E cause a dummy event "**chk(E,L,F,**$\mathbf{T}_F$**)**", which occurs at time $\mathbf{T}_F$. When **chk(E,L,F,**$\mathbf{T}_F$**)** occurs, it causes F at time $\mathbf{T}_F$ iff none of the events in L occurred between E and **chk(E,L,F,**$\mathbf{T}_F$**)**. This is illustrated in Figure 1, where E_0 is the initial event.

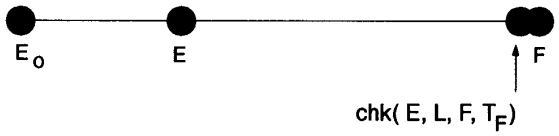

Figure 1: Modeling Preemption

The following causality relation from our DMOD model is an example which indicates that updating (setting) the timer **ts1** at time T will result in its expiration **LIFETIME** time units later, unless **ts1** is updated in between. Thus, timer expiration is preempted by new updates of the timer.

```
causes(E, HE, F) :-
  E=updt_ts1(T),
  F=chk(E,[updt_ts1(_)],exp_ts1(Texp),Texp),
  lt(LIFETIME),
  Texp is T + LIFETIME.
```

4 Validation Using Symbolic DMOD

Despite its strength in modeling and analyzing dynamic systems, DMOD does not have validation capabilities as it is a discrete event simulator rather than a validator. However, symbolic DMOD, implemented using Constraint Logic Programming (CLP($\Re$)), provides validation capabilities for real-time systems as explained in Section 2.2, since it allows full exploration of the system behavior for a whole range of its variables, be they time or other state parameters. We have exercised the simulation and validation aspects of symbolic DMOD for the complex timer-based protocol CMSC. Validation using symbolic simulation means proving protocol properties and theorems about the protocol model. In this section, we present modeling and validation of a specific safety property of CMSC [2]. Modeling and validation of other safety and liveness properties of CMSC could be shown likewise.

4.1 Protocol Validation Exercise

The specific safety property of CMSC we have chosen is the sequence number unambiguity that assures the safe reuse of sequence numbers for messages, so that a sequence number will not be reused until all messages with that number become invalid. CMSC states that the following inequality is satisfied:

```
LIFETIME < SNspace/R
```

LIFETIME represents the maximum lifetime of a message in the network, R the maximum transmission rate of messages, and SNspace the sequence number space. A theorem of CMSC (manually proven in [2]) states that if the above inequality is satisfied for a connection, there will be no two distinct messages with the same sequence number at any time; i.e., there cannot be a new message with sequence number SN in the system while an older message or its retransmissions with the same SN are in transit in the system. The following describes an experiment designed to prove the same theorem using symbolic DMOD.

We chose to vary and constrain only the LIFETIME parameter while we performed symbolic simulation experiments using symbolic DMOD. This was merely due to the fact that CLP($\Re$) interpreter gets overloaded while handling input queries with several variables constrained. Unavoidably, memory restrictions played an important role in designing the validation experiments. We fixed SNspace at 16 and R at 0.5 messages per time unit. We also set the round trip delay, RTD, to 6 time units, which is the time for a message to travel from the transmitter to the remote receiver and for an acknowledgment to come back. Forward and reverse delays are equal in our DMOD model. The inequality LIFETIME < SNspace/R becomes LIFETIME < 32, for the given values. Hence, if we had picked a value of 30 or 31 for LIFETIME, we allow at most 5 transmissions of a message with RTD being 6, i.e., only if a message and all four of its retransmissions are lost during transmission, the transmitter will detect fatal channel failure.

We ran the simulations for 100 time units, allowing the sequence number space to wrap around three times, when R is 0.5 and SNspace is 16. In each of our simulations, we varied the LIFETIME variable as an input constraint, obtained a symbolic history, and the resultant solutions. Ideally, we had two cases to consider: 1) LIFETIME < 32, and 2) LIFETIME $\geq$ 32.

When LIFETIME < 32, we do not expect any two messages with same sequence numbers to coexist in the network, because the constraint imposed on the LIFETIME variable protects against sequence number ambiguity. On the other hand, when LIFETIME $\geq$ 32, we expect the opposite - in fact, multiple ambiguities.

We have split our actual symbolic simulation tests into more than two cases, because as the range of the input constraints increases, there is more to check and process for the CLP($\Re$) interpreter, and the state of the simulation grows exponentially. Also, the LIFETIME $\geq$ 32 case is an unrealistic goal since bounds on the input constraints to the CLP($\Re$) interpreter should be finite. Thus, to better manage our resources, we arranged eight symbolic simulation tests with smaller ranges for the input LIFETIME constraint: 1) 0 $\leq$ LIFETIME < 6, 2) 6 $\leq$ LIFETIME < 12, 3) 12 $\leq$ LIFETIME < 18, 4) 18 $\leq$ LIFETIME < 24, 5) 24 $\leq$ LIFETIME < 28, 6) 28 $\leq$ LIFETIME < 32, 7) 32 $\leq$ LIFETIME < 34, and 8) 34 $\leq$ LIFETIME < 36.

4.2 CLP($\Re$) Input Query

Before discussing test results, we first describe our input query to the CLP($\Re$) interpreter. The following is the query we used for case 7 above. Others are similar except for the bounds on the LIFETIME variable.

```
q(LIFETIME,Msg1,E1,T1,Msg2,E2,T2):-
  LIFETIME < 34, LIFETIME >= 32,
  sim([lt(LIFETIME),rtd(6),r(0.5)],Hist),
  member(newmsg(xmtr,Msg1,E1,T1),Hist),
  member(newmsg(xmtr,Msg2,E2,T2),Hist),
  Msg1 == Msg2,
  T2 > T1,
  T2 <= E1,
  assert(
    sol([LIFETIME,Msg1,E1,T1,Msg2,E2,T2])),
  fail.
```

The q(LIFETIME,Msg1,E1,T1,Msg2,E2,T2) clause is the goal CLP($\Re$) interpreter works on. It is met if all the conditions after the ":-" sign are met, and sol([LIFETIME,Msg1,E1,T1,Msg2,E2,T2]) will store values for these variable terms. By using the assert rule, we guarantee that each sol, for each evaluation of the input query, is saved in a log.

The fail clause forces the goal to fail even if there was a solution that met all the conditions listed above the fail. This then initiates another evaluation of the goal using a new possible set of values for the terms of the query and the simulation continues. The evaluate-fail cycle (each corresponding to a simulation trial) repeats until all possible combinations for the arguments of the goal, are tried and then the simulation halts. We then have a log of all the solutions, if any, as the simulation test exhaustively searched for all possibilities, exploring the whole state space.

The `sim([lt(LIFETIME),rtd(6),r(0.5)],Hist)` statement calls the simulation macro defined in our model of CMSC and is necessary for starting each simulation run. `sim` is called each time the goal is evaluated. It performs a simulation as a function of the input parameters and keeps a log of the history `Hist`. All the simulation information is saved. In our symbolic DMOD model, we have used `LIFETIME`, `RTD`, and `R` as variables. As you can observe, the `RTD` and the `R` variables are set to `6` and `0.5` in `sim`, as we have explained before. Only the `LIFETIME` variable is used in an input constraint which specifies a range the `LIFETIME` variable can vary within. The `LIFETIME < 34, LIFETIME >= 32` statement, just above the `sim` statement specifies the range [32, 34) for our seventh test.

The remaining clauses in the above input query form the core of our experiment as follows. The `member(newmsg(xmtr,Msg1,E1,T1),Hist)` clause is to check if there is any `newmsg(xmtr,Msg1,E1,T1)` event in the `Hist`. In the DMOD model, this event represents a new message with identity `Msg1`, creation time `T1`, and expiration time `E1` (`T1+LIFETIME`) at the transmitter. `Msg1` is either a `setup` message or a `msg(SN)` for `SN=1,2,···,(SNspace-1)`. The `member(newmsg(xmtr,Msg2,E2,T2),Hist)` clause is similar; only `Msg2` is another message with creation time `T2` and expiration time `E2`. We are looking for two messages `Msg1` and `Msg2` with the same identity (`Msg1 == Msg2`), but different creation times (`T2 > T1`), and therefore different expiration times (`E2=T2+LIFETIME > T1+LIFETIME=E1`), such that the latter is created before the former expires (`T2 <= E1`). Thus, we are searching for sequence number ambiguities. We expect `sols` for only `LIFETIME` $\geq$ `32`.

4.3 A Symbolic Simulation Run

Below is a short example from our symbolic simulation tests (first symbolic simulation trial in **Test 1** (`0` $\leq$ `LIFETIME` < `6`)). **Test 1** and all others have many such trials (usually much longer) as they try all possible event histories for the given input constraint.

```
strt([lt(_t1),rtd(6),r(0.5)], 0)
newmsg(xmtr,setup,_t11,0)
send(xmtr,setup,_t11,0)
updt_ts1(0)
updt_ts2(32,0)
chk(updt_ts1(0),[updt_ts1(_h6)],
                        exp_ts1(_t26),_t26)
exp_ts1(_t26)
schdl_newmsg(xmtr,2)
chk(strt(_h0,_h1),[retr(xmtr,_h2,_h3,2)],
                newmsg(xmtr,msg(0),_t31,2),2)
newmsg(xmtr,msg(0),_t31,2)
send(xmtr,msg(0),_t31,2)
updt_ts1(2)
chk(updt_ts1(2),[updt_ts1(_h5)],
                        exp_ts1(_t55),_t55)
exp_ts1(_t55)
chk(schdl_newmsg(xmtr,2),[exp_ts1(_h4)],
                    schdl_newmsg(xmtr,4),4)
exp_ts3(_t29)
exp_ts3(_t58)
chk(updt_ts2(32,0),[updt_ts2(_h7,_h8),
              exp_ts1(_h9)],exp_ts2(32),32)
```

In the above example, the `strt` event causes a `newmsg` event which in turn causes `send`, `updt_ts1`, `updt_ts2`, `chk` (the first one), and `schdl_nwmsg` events, and so on. In this run, as `LIFETIME` < `RTD=6`, the transmitter does not receive an acknowledgment for any of the messages (`set_up` and `msg(0)`) it sends. As a result, the connection timers `ts1`, `ts3`, and eventually `ts2` expire, closing the connection. The simulation then continues with another run.

4.4 Symbolic Simulation Tests

Below, we discuss our simulation tests and their results, which are also summarized in Table 1.

Simulation Results					
	0≤LIFETIME<32			LIFETIME≥32	
range	[0,6)	[6,24)	[24,32)	[32,34)	[34,36)
Test	**1**	**2-4**	**5-6**	**7**	**8**
sols	0	0	0	30	82
case	A	B	B&C	C	C
size	36K	350K	496K	172K	218K

Table 1: Tabular Summary of Simulation Results

For the first six tests, corresponding as a whole to the case `LIFETIME` < `32`, we did not have any solutions (`0 sols`), as we expected. This means that for `LIFETIME` < `SNspace/R`, no two different messages carrying the same sequence number coexisted in the network, showing the desired sequence number unambiguity. However, for `LIFETIME` $\geq$ `SNspace/R`, we observed several `sols` indicating sequence number ambiguity; i.e., there were several distinct messages created at different times carrying the same sequence number. These solutions were recorded, corresponding to 30 `sols` in **Test 7** and 82 `sols` in **Test 8** representing

the general case LIFETIME $\geq$ 32. The sol entries in Table 1 show these results. The **case** and **size** entries provide details about the termination characteristics and simulation durations, as we explain below.

For **Test 1** (0 $\leq$ LIFETIME $<$ 6), simulation trials, one of which is shown above, terminate rather quickly. Since LIFETIME $<$ RTD, the transmitter receives no acknowledgments for the messages it sends, all the connection timers eventually expire, the connection closes, and the simulation continues with a new trial exploring another possibility. The simulation finally ceases after generating and analyzing all possible histories for LIFETIME values in the range [0, 6). This is what we call **case A** in Table 1. The output, which contains the event histories generated by the simulation trials in **Test 1**, is relatively short. The **size** entry in Table 1 indicates 36K characters. We included the **size** entries in the table to give a rough estimate of the simulation length, which is dependent on the protocol operation as well as the input range of the LIFETIME parameter.

For the tests from **Test 2** (6 $\leq$ LIFETIME $<$ 12) to **Test 6** (28 $\leq$ LIFETIME $<$ 32), simulation time kept increasing, as can be observed from the output **sizes** in Table 1 . This is expected as the messages have a greater chance of being delivered as more retransmissions are allowed. If a message is not acknowledged after an RTD time, it is retransmitted, and the retransmission carries the same expiration time as the original message. Therefore, as the difference between the LIFETIME and the RTD values increases, there is a higher number of retransmissions allowed for each message, and the probability that one of the retransmissions will succeed is higher than the probability that the first transmission succeeds. E.g., as we simulate random message losses with a 0.25 probability, the probability that a message and its first retransmission are lost is $0.25^2 = 0.0625$, a message and its first two retransmissions are lost is $0.25^3 = 0.015625$, and so on. However, although most messages are delivered and acknowledged, there are still some messages that don't get acknowledged within their LIFETIME. In other words, there is a finite probability that the message itself and all its consecutive retransmissions are lost, although this probability decreases as the LIFETIME value increases. As in **case A**, such messages cause the connection timers to expire and the connection to close. Then, the simulation continues with a new possible history until there remain no untried possibilities. We call this **case B**, although its only difference from **case A** is that in **case A**, none of the messages gets acknowledged within its lifetime, and the simulation terminates rather quickly. In the simulation trials for **Test 2** to **Test 4**, **case B** was observed as the termination cause.

As the LIFETIME values exceeded 4 RTD, we observed another termination cause which we call **case C**. The termination cause was that the end of the simulation duration was reached (the 100^{th} time unit), meaning that all messages were delivered and acknowledged for that specific simulation trial within the test that includes many more trials. Thus, termination cause for a trial was sometimes **case C** and other times **case B** in **Test 5** and **Test 6**. Given the probability of four consecutive message losses is only $0.25^4 = 0.00390625$, there was actually only one instance of **case B** in **Test 6**. Nevertheless, in all tests for which LIFETIME $<$ 32, there was never a sequence number ambiguity, nor any solutions, as expected.

For **Test 7** (32 $\leq$ LIFETIME $<$ 34), we had 30 unique solutions (sols) as a result of all the trials, each of which lasted 100 time units. Likewise, for **Test 8** (34 $\leq$ LIFETIME $<$ 36), we also had several sols, even more than in **Test 7** (82 unique solutions). These are the tests for which LIFETIME $\geq$ SNspace/R, proving the sequence number ambiguity. Note that the input LIFETIME range we used for these two tests was shorter, yielding relatively shorter output **sizes** than the previous tests. The reason for using shorter LIFETIME ranges is for CLP($\Re$) memory managebility, as there is a lot more activity going on during these tests. Thus, only **case C** was observed as the termination cause in **Test 7** and **Test 8**.

4.5 Sequence Number Ambiguity

For tests in which LIFETIME $\geq$ SNspace/R (**Test 7** and **Test 8**), we had several solutions (sols) indicating sequence number ambiguity. Below are two examples of these sol's defined in the input query as sol([LIFETIME,Msg1,E1,T1,Msg2,E2,T2]).

```
sol([LIFETIME,msg(7),E1,26,msg(7),E2,58]):-
  E1 = LIFETIME + 26,
  E2 = LIFETIME + 58,
  LIFETIME > 32,
  LIFETIME <= 32.5,

sol([LIFETIME,msg(7),E1,26,msg(7),E2,58]):-
  E1 = LIFETIME + 26,
  E2 = LIFETIME + 58,
  LIFETIME <= 35,
  LIFETIME > 34.5,
```

In both of these examples above, there are two different msg(7)'s that may coexist in the network

for a time interval. Both `sol`'s indicate a `msg(7)` transmitted at time 26 and another `msg(7)` transmitted at time 58. In both examples, `msg(7)` expires at time `E1 = 26 + LIFETIME`. This means that `E1` lies in the range `(58, 58.5]` in the first example as $32 < \texttt{LIFETIME} \leq 32.5$, and in the range `(60.5, 61]` in the second example as $34.5 < \texttt{LIFETIME} \leq 35$. As the second `msg(7)` is transmitted at time 58, this makes the time range `[58, E1]` a potentially hazardous duration for sequence number ambiguity. If the transmitter receives an acknowledgment for the first `msg(7)` in the time interval `[58, E1]`, it may perceive it to acknowledgme the second `msg(7)` transmitted at time 58 which in turn would result in successive incorrect protocol operation.

5 Conclusions

Symbolic simulation is a technique that allows modeling and verification of system properties. This paper has established the theoretical foundations for symbolic DMOD. Using symbolic DMOD, we were able to explore the range of `LIFETIME` values of interest to us and confirm the manually derived proof on sequence number unambiguity [2]. The symbolic simulation tests clearly validated the fact that whenever $\texttt{LIFETIME} \geq \texttt{SNspace/R}$, potential hazards for sequence number ambiguity exist. For values of $\texttt{LIFETIME} < \texttt{SNspace/R}$, we were able to investigate the complete real-number range of the `LIFETIME` value, `[0,32)`, not just at specific increments as we would with traditional simulation. Thus, it is safe to say that setting $\texttt{LIFETIME} < \texttt{SNspace/R}$ assures sequence number unambiguity and correct operation for CMSC or similar timer-based protocols.

We have found DMOD and symbolic DMOD useful for protocol modeling and validation, especially for real-time and complex communication protocols. This is due to DMOD's simplicity in modeling real-time and other protocol complexities, and CLP($\Re$)'s symbolic simulation capability to validate and prove safety and liveness properties of a protocol assuring full logical and functional correctness.

Acknowledgments

The authors would like to thank Dave Feldmeier, Sanjai Narain, and Roland Yap for their help during this work. The authors are also grateful to David Cohen, Dave Feldmeier, Yow-Jian Lin, and Sanjai Narain for their useful suggestions in reviewing this paper.

References

[1] R. Alur, C. Courcoubetis, and D. Dill, "Model-Checking for Real-Time Systems", *Proc. 5th Annual IEEE Symp. on Logic in Computer Science*, Philadelphia, PA, pp. 414-425,1990.

[2] E.W. Biersack and D.C. Feldmeier, "A Timer-Based Connection Management Protocol with Synchronized Clocks and its Verification", *Computer Network and ISDN Sys.*, 6(3), June 1993.

[3] E.J. Cameron and Y.J. Lin, "A Real-Time Transition Model for Analyzing Behavioral Compatibility of Telecommunications Services", *SIGSOFT'91 - Software for Critical Sys.*, Dec. 1991.

[4] E.M. Clarke, E.A. Emerson and A.P. Sistla, "Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications", *ACM TOPLAS*, 8(2), 1986.

[5] J. Jaffar and J.-L. Lassez, "Constraint Logic Programming", *Proc. ACM Symp. on Principles of Programming Languages*, Munich, Germany, 1987.

[6] Y.-J. Lin and G. Wuu, "A Constrained Approach for Temporal Intervals in the Analysis of Timed Transitions, *PSTV*, Stockholm, 1991.

[7] S. Narain, "An Axiomatic Basis for General Discrete-Event Modeling", *Proc. of Winter Simulation Conference*, Phoenix, AZ, 1991.

[8] S. Narain, "An Approach to Reasoning about Hybrid Systems", *Int. Jour. of General Systems*, Sept. 1991.

[9] S. Narain, O. Cockings and R. Chadha, "PHOTON: A Software System for SONET Interoperability Analysis", *Globecom'93*, Houston, 1993.

[10] S. Narain, O. Cockings and R. Chadha, "A Formal Model of SONET's Alarm Surveillance Procedures and Their Simulation", *FORTE-93, 6th Int. Conf. on Formal Description Techniques*, 1993.

[11] J.S. Ostroff, "Constraint Logic Programming for Reasoning about Discrete Event Processes", *Jour. Logic Programming*, vol. 11, pp. 243-270, 1991.

[12] R.C. Sekar, Y.J. Lin and S. Narain, "Modeling Hybrid Systems", *IFIP, 12th Int. Symp. on Protocol Specification, Testing. and Verification*, Lake Buena Vista, Florida, 22-25 June, 1992.