Implementing A Global Termination Condition and Collecting Output Measures in Parallel Simulation

M. Abrams and D. Richardson

TR 90-54

Implementing a Global Termination Condition and Collecting Output Measures in Parallel Simulation

November, 1990

M. Abrams and D. Richardson
Department of Computer Science, Virginia Tech
Blacksburg, VA 24061

abrams@cs.vt.edu
(703) 231-8457

ABSTRACT:

This paper investigates how to implement arbitrary global termination conditions and collect statistics in a parallel simulation. The problem is first discussed using Chandy and Sherman's space-time framework. Then termination conditions are categorized, and termination algorithms are given for several categories. The chief problem is that if one evaluates the termination condition asynchronously with respect to the simulation, when termination is detected the simulator has already modified old attribute values needed to compute output measures. The major conclusion is that minor modification of time warp permits use of any termination condition. In contrast, conservative protocols permit limited termination conditions unless they are modified to incorporate mechanisms present in optimistic protocols.

1. INTRODUCTION

Motivation

Two open problems in discrete-event, parallel simulation are (1) how to detect when a simulation has satisfied an *arbitrary* termination condition and, given that the termination condition is satisfied, (2) how to calculate output measures and then shut down the simulator. To use parallel simulation protocols as general purpose tools, they must accommodate arbitrary termination conditions and output measures. Considering together the variety of termination conditions, parallel simulation protocols, and desired output measures leads to an enormous design space

Examples of Termination Conditions. Perhaps the simplest termination condition uses the single attribute of simulation time: does the simulation clock equal or exceed the value T? However, time is only one attribute upon which a termination condition may be based, as the following examples illustrate.

E1) In a colliding pucks simulation, have *at least N* collisions occurred in the system?
E2) In a colliding pucks simulation, have *exactly N* collisions occurred in the system?
E3) In a mobile telephone simulation, is this the first time that three telephones' radii of communication overlap?
E4) In a mobile telephone simulation, do three telephones' spheres of communication overlap?
E5) In an open queueing network model simulation, have exactly N jobs departed from the network?
E6) Let δ and ε be constants. In an open queueing network simulation, does the average number of jobs in the system at times t and t-δ differ by less than ε or have at least N jobs entered the system?
E7) In an electrical circuit simulation does the voltage at a set of nodes meets some stability condition (evaluating the stability condition requires a costly numerical integration of a function)?
E8) Each logical process (LP) tests "is my local virtual time (LVT) $\geq$ T?"*
E9) Each LP tests "have I received N messages?"

* The paper uses Chandy-Misra's terminology of logical processes (LP's) communicating by messages (Misra 1986) and Jefferson's terminology of local virtual time (LVT) and global virtual time (GVT) (Jefferson 1985).

A termination condition is *local* if each LP decides when to terminate based purely on local data (i.e., data that is not modified by any other LP). Otherwise a simulation uses a *global termination condition.* E1 through E7 exemplify global termination conditions; E8 and E9 exemplify local termination conditions.

Previous Work. All parallel simulation literature to date, with the exception of Jefferson (1985), does not help to solve the two open problems stated above because they either use a local termination condition ("is my LVT≥T") or use wording similar to Misra's: "while simulation completion criteria is not met do ... endwhile" (Misra 1986). Jefferson describes how to shut down a time warp simulator, but leaves open problems of detecting when a termination condition holds and calculating output measures.

Existing termination detection algorithms (Mattern 1987) do not help solve this topic for four reasons. First, existing algorithms presume that all computations will reach a stable state: if y is a stable property and y is true at some point in the computation, then y is true at later points (Chandy and Lamport, 1985). Termination conditions such as E2 through E6 cannot be expressed in terms of a stable state. Second, simulation programs generally require, on termination, the evaluation of output measures using the state of the simulation at some *past* simulation time. Existing termination detection algorithms do not consider this problem. Third, existing algorithms assume that the stable property is the conjunct of a set of predicates, each of which is tested by exactly one process using only local state information. Termination conditions E1 to E7 do not meet this requirement. Fourth, existing algorithms assume that processes become "idle" when a stable condition is reached (e.g. fixed points in UNITY (Chandy and Misra 1989) and all processes being in the outer level with no guard ready in Franchez's treatment of CSP programs (1980)). However, simulation programs are often designed to be nonterminating, with an outside agent detecting when a user-defined condition such as E1 through E9 holds. An example is steady state simulations (Law and Kelton 1982).

Reexecuting global snapshots (Chandy and Lamport 1985) throughout simulation could detect a *stable* condition. Each processor records its local state and all messages in transit on incoming arcs at possibly simultaneous times. This solves the wrong problem. Unanswered is *when* to initiate a snapshot and how to combine local states to evaluate a global termination condition. Finally, the snapshot algorithm does not exploit the fact that the global state at times not exceeding global virtual time (GVT) never includes messages in transit.

Outline of Paper. The paper first defines and categorizes simulation termination conditions in sections 1 and 3. Later sections investigate how existing protocols can incorporate

arbitrary termination conditions based on the categorization. Section 4 considers an optimistic protocol (time warp (Jefferson 1985)), and section 5 considers a conservative protocol (Chandy-Misra (1986)). Remaining sections discuss the inherent complexity of the problem (section 2) and the implications of this work (section 6).

Termination Conditions

Definition and Examples. A *simulation program* is a representation of a simulation model. A *simulation model* represents the change in a set of *attributes*. References to time in this paper, unless stated otherwise, refer to *simulation* (or *virtual*) time.

The value of an attribute a changes at a sequence of discrete times $t_1, t_2, \ldots$. Therefore the value of an attribute remains the same in time interval $[0,t_1)$, then assumes a new value and remains the same in interval $[t_1, t_2)$, and so on. The *value of an attribute at simulation time t* is the value the attribute was assigned at the largest $t_i \leq t$. In a parallel simulation program, each attribute is said to *belong to* or *be local to* exactly one LP.

A *termination condition,* denoted $C(t)$, specifies a relationship among the values of chosen simulation model attributes at time t. Evaluation of a termination condition yields one of the values "true" or "false." A simulation program is said to *terminate at time t* if and only if:

1. the values of all attributes at time t required by the termination condition are known, and
2. $C(t)$ = true.

If there exists no t that makes $C(t)$ true, then the simulation program does not terminate.

The *output of a simulation program* is the value of a set of *output measures* that are calculated using the attribute values at a time t for which $C(t)$ is true.

2. RELATION TO SPACE-TIME FRAMEWORK

Chandy and Sherman's space-time diagrams (1989) illustrate the inherent complexity of terminating parallel simulation. Chandy and Sherman used space to represent the *set of all LP's* in a simulation program, which presumes the program uses a simulation world view based on

processes. To be applicable to any world view (Balci 1988), space in this paper represents the *set of all attributes* in a simulation model.

Figure 1 represents a simulation with the four attributes a_1, a_2, a_3, and a_4. The times at which each attribute changes value are represented by heavy lines. For example, the value of attribute a_3 remains constant in intervals [0,4) and [4,10).

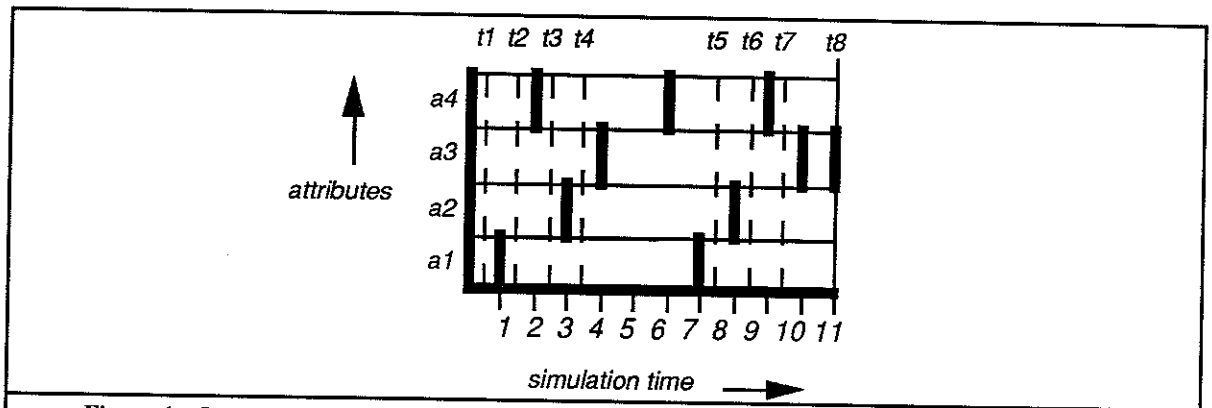

Figure 1: Space-time diagram. Simulation runs for 11 time units. $C(t)=false$ for $t < t_8$ and $C(t_8)=true$.

The values of simulation model attributes at time t are represented by a vertical line with constant time coordinate. To terminate a simulation program, termination condition $C(t)$ is evaluated at a sequence of times $t_1, t_2, ..., t_n$. Figure 1 illustrates one such sequence by dashed vertical lines. A sequential simulator typically evaluates the termination condition after *each* event, object movement, or action occurs, in the event-scheduling, process-interaction, or activity-scan/three-phase-approach, respectively (Balci 1988); parallel simulators may perform fewer evaluations. This has two implications:

Implication 1: Evaluation of output measures requires the values of simulation attributes at a time t for which $C(t)=true$. Conveniently, the *last* value of each attribute calculated must represent the value at time t. An asynchronous parallel simulation inherently violates this.

Implication 2: A sequential simulator automatically finds the earliest time τ at which the termination condition holds (t_8 in Figure 1); an asynchronous parallel simulation can only avoid violating this at a price of efficiency.

Violation of Implication 1: Figure 2 contains space-time illustrations of how Implication 1 may be violated. In a sequential simulation, Figure 2a illustrates that all attribute values are known up to the time at which the last event, object move, or action has occurred. This time is the largest time at which $C(t)$ may be evaluated using known attribute values (denoted T_k). Conservative and optimistic protocols always violate Implication 1, but for

different reasons. In the conservative case (Figure 2b), the largest time at which each attribute's value has been calculated is generally different for each attribute. In the optimistic case (Figure 2c), one distinguishes between *known* and *estimated* attribute values. The latest values of an attribute are generally *estimates* and hence represent different times. (Algorithms in section 4 for optimistic simulation overcome violation of Implication 1 because saved states include the latest *known* attribute value.)

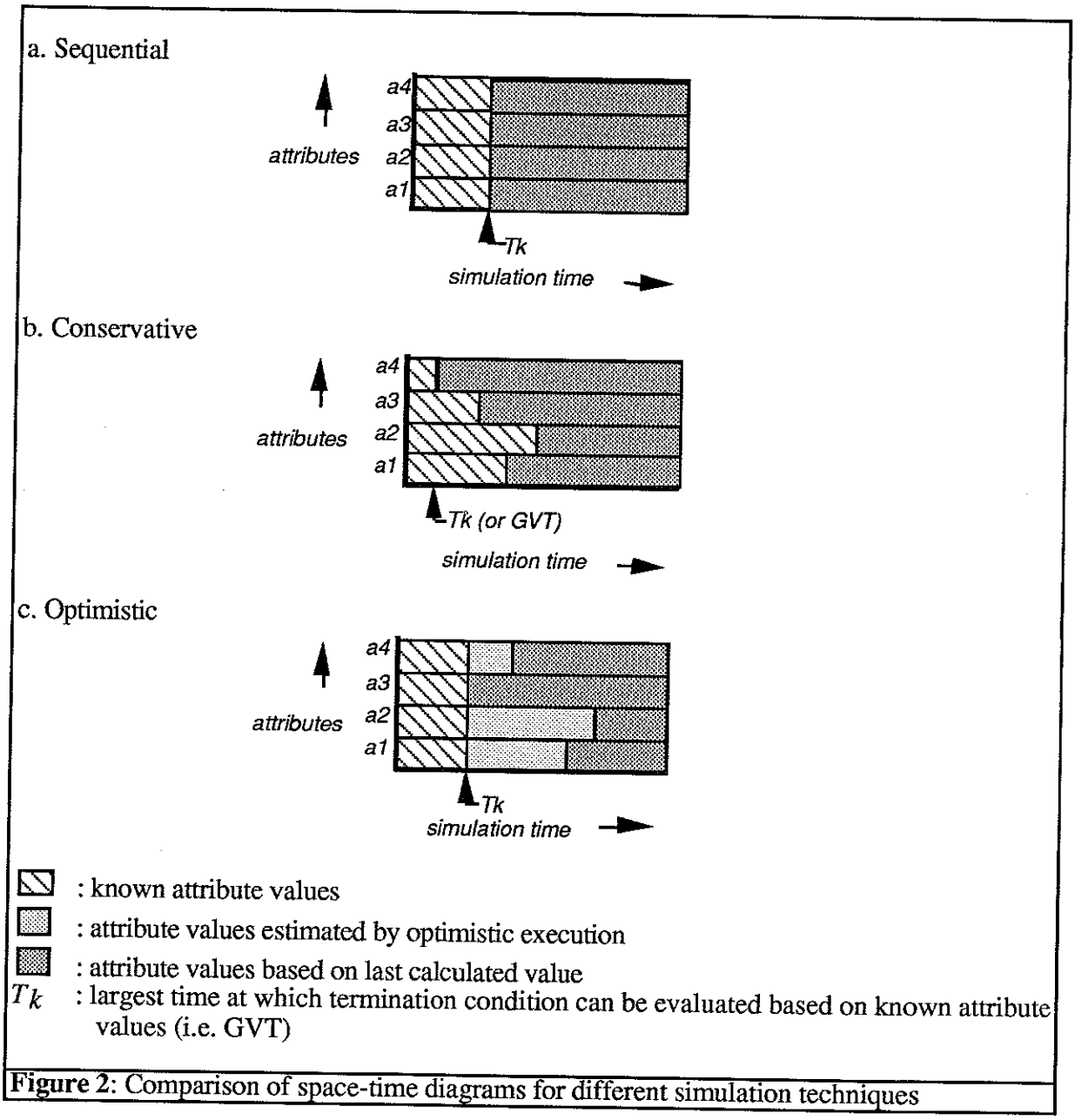

Figure 2: Comparison of space-time diagrams for different simulation techniques

Violation of Implication 2: $C(t)$ may be evaluated (1) conservatively -- using known attribute values -- or (2) optimistically -- using attribute estimates.* Cases (1) and (2) correspond to $t=T_k$ (where T_k is GVT) and $t \in (T_k,\infty]$, respectively. (Note that the attribute value last calculated serves as an estimate of its value up to time ∞, because the value may or may not change in the future. This information is not known until T_k advances.) Implication 2 *may* be violated in case (1) and *will* be violated in case (2). For efficiency one may not wish to evaluate $C(t)$ after each change in known attribute value, thereby violating Implication 2. Another way that Implication 2 would be violated is if GVT advances and fossil collection occurs for attributes older than GVT; this point is addressed later in the algorithms.

One can guarantee that Implications 1 and 2 hold in a parallel simulation by using a *trivial termination condition.* A termination condition is *trivial* if

Trivial 1: *every* LP must evaluate the condition, and
Trivial 2: an LP uses *only* attributes local to it.

Termination conditions E8 and E9 are trivial, while E1 to E7 are non-trivial. Non-trivial termination conditions violate either Trivial 1 or Trivial 2, or both of them. Violation of Trivial 1 or Trivial 2 also implies violation of Implication 1. Consider the implications of violating each part.

Violation of Trivial 1 (See E1 to E4, and E6): The LPs that evaluate the termination condition must tell the remaining LP's to terminate. Therefore there is *feedback* in the simulation due to the termination condition.

Violation of Trivial 2 (See E1 to E7): The attributes of several LPs at time t must be combined. This requires sharing data among LPs, which introduces a variety of issues. Is one of the LPs responsible for reading the attributes of the others? Should all LPs with attributes transmit their values at time t to a separate process responsible for evaluating the termination condition?

Viewed another way, violation of Trivial 1 or Trivial 2 produces a pipelining effect in a parallel simulator, where LPs simulate actions which change attributes that are later used to evaluate the termination condition. This can only be avoided at the expense of parallelism if LPs do not simulate past the largest time at which the termination condition was evaluated.

---

* Note that the choice of whether to evaluate C(t) optimistically is independent of whether the simulator is optimistic. A conservative simulator may evaluate C(t) optimistically.

3. CATEGORIZATION

Table 1 categorizes non-trivial termination conditions. The categorization of a particular termination condition is described by a three letter mnemonic. The convention used here is to write the letter in upper case if it fits the category; otherwise the letter is written in lower case (i.e., $s = \neg S$). A termination condition in category *SUT* admits the simplest and fastest termination algorithm possible, while category *sut* requires the most complex and slowest algorithm.

C(t) fits category...	*...if*
S	*C(t) is stable*
U	*the set of times t that may satisfy C(t) is unconstrained*
T	*the time needed to evaluate C(t) is tiny compared to total sim time*

Table 1: Mnemonics categorizing termination conditions

S: For example, E1 is stable because the number of collisions that have occurred is a nondecreasing function. In contrast, E2 is of the form $f(...)=a$, and is not stable. Similarly, E5 is not stable: if the N-th departure occurs at time t and the N+1-th departure occurs at time t', then E5 is true only in the interval [t,t').

U: E4 is unconstrained, because *any* time at which three telephones' spheres of communication overlap satisfy *C(t)*. In contrast, E3 is constrained because the *smallest* value of t satisfying *C(t)* is sought, representing the first time at which three telephones' spheres of communication overlap satisfies *C(t)*.

T: A simulation using E7 as its termination condition may require several numerical integrations, each of which requires the same execution time as several hundred simulation events.

The implications of categories *S* and *U* viewed in terms of space-time are summarized in Figure 3.

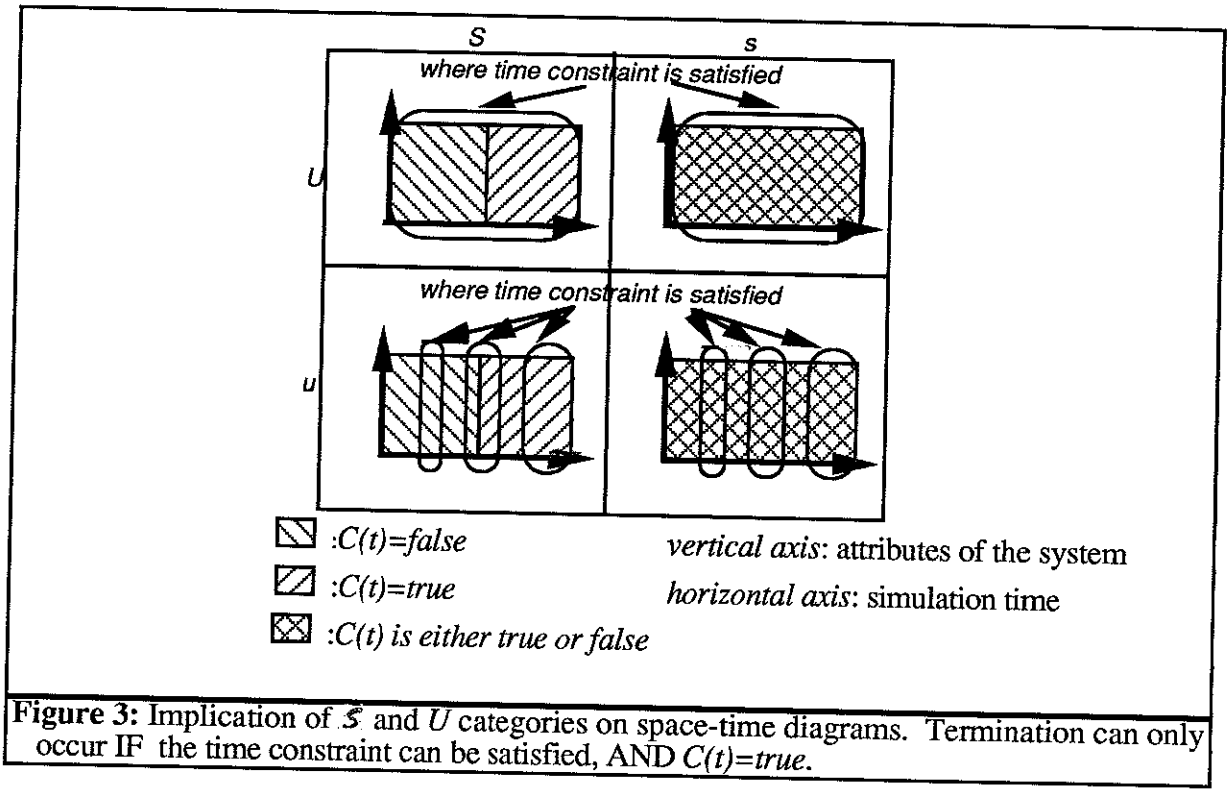

Figure 3: Implication of S and U categories on space-time diagrams. Termination can only occur IF the time constraint can be satisfied, AND $C(t)=true$.

Table 2 categorizes termination conditions E1 through E7.

Termination condition	*Category*
E1	SUT
E2	sUT
E3	suT
E4	sUT
E5	sUT
E6	sUT
E7	sUt or SUt

Table 2: Categorization of non-trivial conditions

4. TERMINATING TIME WARP

Original Termination Mechanism

The termination mechanism provided by the original optimistic protocol, time warp (Jefferson 1985), is elegant but incomplete. Each logical process (LP) sets its local virtual time (LVT) to infinity when its input queue is empty. Simulation ends when the global virtual time (GVT) advances to infinity, implying that all LPs in the simulation have set their LVTs to infinity and no unprocessed messages exist.

One cannot prove that a time warp simulation terminates in finite time, based on Jefferson's original framework, unless he includes the assumption that *a time warp simulation generates a finite number of messages*. Consider termination condition E8, in which each LP tests "is my LVT$\geq T$?" If each LP is structured so that it sends no messages after it detects this termination condition to be true, then the assumption is met, thereby guaranteeing termination.

The termination mechanism in time warp places a burden on the simulation programmer to guarantee that each LP in the system will send a finite number of messages and then stop. It is possible for a simulation programmer to design a termination condition that may deadlock time warp. An example is termination condition E9 applied to a queueing network containing a forking node, where each message represents one customer. At least one of the nodes connected to the fork output will never satisfy E9.

Modified Termination Mechanism

Simulation Program Organization. We propose organizing a simulation program into three parts:

1) A *time warp simulator* (TWS), consisting of a set of LP's running a time warp simulation as described by Jefferson (1985). Each LP tests the local termination condition "is LVT$\geq T_S$"; if this condition holds, then the LP sends no more messages to other LPs. Initially, $T_S=\infty$, implying that the simulation never terminates.
2) A *termination detector* (TD), which evaluates $C(t)$ for all t in a certain set S and selects a *termination time*, T_T, which satisfies the following condition:
   Condition 1: $C(T_T)=true \wedge T_T \leq$ GVT,
   where GVT represents the last global virtual time calculated by TWS.

3) One or more *output measure reporters* (OMR), which each calculate and report one output measure at time T_T.

In order for TD to evaluate $C(t)$ for arbitrary t, it must have access to the values of all required attributes at time t. A sufficient condition for TD to do this is that it has knowledge of the following portion of the space-time diagram: time in interval $[0,t]$ and space coordinates equal to the attributes required to evaluate $C(t)$. Therefore TD knows the time at which each required attribute changes value as well as the new value. A sufficient condition for TD to construct the required space-time diagram is for an LP to automatically send or be queried to send an *update message* to TD whenever the LP modifies an attribute that TD uses. The text and send time of an update message identify the attribute and its new attribute value and identify the LVT at which the LP modified the attribute, respectively. The receive time is not used. An update message means that the attribute has the value given in the message text over time interval $[t_1,t_2)$, where t_1 is the message send time, and t_2 is the send time of the next update message for this attribute. An LP may also send anti-update messages.

OMR requires the value T_T from TD and attribute values at time T_T to calculate output measures. Therefore TD sends OMR the value T_T and TWS must satisfy the following condition:

Condition 2: An LP sends OMR the value of any local attribute at time T_T required for calculation of the output measure.

Three alternative mechanisms that satisfy this condition are given below. *Dissociative termination* requires limited coupling between TD and TWS, while the remaining two mechanisms require TD to set TWS parameter T_S. In *retrospective termination*, TD sets T_T and T_S to a value in TWS's past, while in *prospective termination* TD sets T_T and T_S to a value in TWS's future.

Dissociative termination: A sufficient condition for OMR to know the value of each required attribute at time T_T is for it to know the portion of the space time diagram up to time T_T representing required attributes. In this case each LP that owns a required attribute sends an update message (and possibly anti-update messages) identical in meaning to those discussed for TD. Finally, when TD selects time T_T, TD interrupts and kills TWS. By Condition 1, $T_T \leq$GVT, therefore OMR will not receive an anti-update message with send time less than or equal to T_T; thus killing TWS will not affect OMR. Using dissociative termination, the OMR will have all information needed to calculate C(t) at every time throughout the simulation.

Retrospective termination: Following Tinker and Agre (1990), two GVT calculations are made. The first is the GVT of TWS, and the second is the GVT of TWS and TD together, called World Virtual Time (WVT). Fossil collection is not done when GVT advances, but instead is done when $C(t)=false$ and WVT advances. This permits TD to distinguish known from estimated attribute values on the basis of GVT, and allows roll back to times before GVT but not before WVT. The expression $\forall \tau \in [0,WVT]$, $C(t)=false$ will always hold true.

After TD selects T_T, it sends to each LP in TWS a *terminate* message. The send and receive times of a terminate message are both T_T. Upon receiving a terminate message, an LP will roll back to LVT$\leq T_T$. The TWS is modified to set T_S to the terminate message send time (i.e., T_T). By the local termination condition for an LP ("is LVT$\geq T_S$"), TWS will terminate. Finally, when each LP satisfies the local termination condition, it sends OMR the value of any local attributes that OMR requires.

Prospective termination: When TD finds a value of t for which $C(t)=true$, it then runs an algorithm to freeze the state of a time warp simulator (for example by preempting all running LP's). TWS then runs an algorithm to calculate an upper bound on its current time horizon, denoted T_H. T_H at wall clock time r is the *maximum* of (1) all virtual times in all virtual clocks at time r, and (2) of the virtual send times of all messages that have been sent but have not yet been processed at time r.

Based on the value of t, TD selects a time $T_T > T_H$, such that $C(T_T)$ is guaranteed to be true. Finally, TD broadcasts T_T to all LPs, each LP sets its copy of T_S to T_T, and time warp is unfrozen. As in retrospective termination, by the local termination condition for an LP ("is LVT$\geq T_S$"), TWS will terminate. When each LP satisfies the local termination condition, it sends OMR a single output measure message with the attribute value at time T_T.

This method cannot be applied to every simulation model because one may not always be able to guess a $T_H > T_T$ such that $C(t)=true$. However, it is always possible to make such a guess for termination conditions fitting category S, by the definition of stability: any $t > T_T$ will work.

Comparing Dissociative, Retrospective, and Prospective Termination. An obvious question is which termination mechanism gives the best performance. The number of update messages sent to OMR by dissociative termination grows with the simulation run time, while the number required by retrospective and prospective termination is constant. On this

basis, dissociative termination is less desirable. As previously noted, prospective termination cannot always be used. Hence retrospective simulation appears to be the method of choice.

Terminating Category SUT

A problem to be solved by a termination algorithm is determining what set of *t's* should be used to evaluate *C(t)*. In this respect, category SUT has an interesting property. As Figure 3 illustrates, its space-time diagram is partitioned into two regions, corresponding to the two values of *C(t)*. Hence *C(t)* need only be evaluated periodically (compared to category Su). When a *t* for which *C(t)=true* is found, *any* value of time exceeding *t* can be used as T_T. Therefore prospective termination can be implemented for any simulation with a termination condition of category SU.

The two policies referred to in the algorithm below are chosen to minimize simulation program running time. Any policies can be used that satisfy the following property:

Property 0: Each time algorithm SUT reaches the bottom of its do loop, either *C(t)=true* or future execution of algorithm SUT will evaluate *C(t')* for some *t'>t*.

```
Algorithm SUT
  Gold:=0;
  do
     wait for TWS to advance GVT;
     Gnew:=new value of GVT;
     if C(t) should be evaluated, according to some policy
     then using some policy, select set S={t|t∈(Gold,Gnew]};
        do
          remove an element from S and assign it to t
        until S=∅ ∨C(t);
     endif
     Gold:=Gnew;
until C(t);
invoke dissociative, retrospective, or prospective termination;
/* In dissociative and retrospective termination, TT=t */
```

Terminating Condition suT

Algorithm SUT uses a policy to identify a set of times (set S) at which to evaluate $C(t)$. Referring to the space-time diagram of category *su* in Figure 3, to guarantee that a t is found that makes $C(t)$ true, set S must represent exactly the times in interval $(G_{old},G_{new}]$ described below. Let $C(t)$ require n attributes. Consider an n-tuple in which each element represents the current value of one of the attributes. As a simulation model moves from time G_{old} to G_{new}, it passes through a sequence of n-tuples. Therefore algorithm suT differs from algorithm SUT in that set S contains a set of times such that each time corresponds to a unique n-tuple. For example, Figure 1 illustrates a simulation in which $C(t)$ must be evaluated for exactly the set $\{t_1,t_2, ..., t_8\}$ to implement a termination condition of category *su*, where $C(t)$ might only need to be evaluated at t_8 to implement category SU.

Terminating Category SUt

For termination conditions in category *s*, the following methods may reduce the wall clock time required to run a simulation:

1. Each time that $C(t)$ is evaluated, perform the evaluation in parallel.
2. Evaluate $C(t)$ at different values of t simultaneously.
3. Evaluate $C(t)$ at times greater than T_k using estimated attribute values; this is *optimistic evaluation.*

The first two methods apply parallelism. All three methods may be used together. As is illustrated in the space-time diagrams of Figure 2, TD has estimates of all required attributes for any $t \in (T_k,\infty]$. To process an update or anti-update message, TD identifies whether the message changes an attribute estimate used in any in-progress or completed evaluation of $C(t)$ and invalidates the affected $C(t)$'s.

Given below is an algorithm implementing category SUt termination conditions. Whereas algorithm SUT was sequential, algorithm SUt consists of a set of processes. One or more processes implement the algorithm below. The remaining processes evaluate $C(t)$ at one or more values of t and process update and anti-update messages. A policy is defined to determine (1) the set of values of t at which $C(t)$ is to be evaluated and (2) when to initiate these evaluations. The policy must guarantee Property 0, with "SUt" substituted for "SUT."

```
Algorithm  SUt
  Gold:=0;
  do
     wait for TWS to advance GVT;
     Gnew:=new value of GVT;
     wait for all received but unprocessed update and anti-
       update messages with send time not exceeding GVTnew to
       be processed;
     if ∃t'∈(Gold,Gnew] such that C(t')=true
     then t:=any t'∈(Gold,Gnew] such that C(t')=true;
     Gold:=Gnew;
  until C(t);
  invoke dissociative, retrospective, or prospective
    termination;
/* In dissociative and retrospective termination, TT=t */
```

Terminating Category sut

Algorithm SUt can be modified to handle category sut by guaranteeing that *C(t)* is evaluated for all values *t* that algorithm suT would use.

5. CATEGORY SUT TERMINATION WITH BRYANT-CHANDY-MISRA

This section considers category SUT termination conditions with a conservative protocol: Bryant-Chandy-Misra (BCM) (Misra 1986). The organization of a time-warp based simulation program described in the previous section will be used with the modification that the time warp simulator is replaced by a BCM simulator (BCMS). Furthermore, BCMS runs time warp's GVT algorithm for the purpose of distinguishing known and estimated attribute values.

The dissociative and prospective termination algorithms along with the category SUT algorithm of section 4 described for time warp may be applied unaltered to BCM. However, as noted in the previous section, retrospective termination is universally applicable and requires fewer update messages to OMR. Hence it is of interest to modify retrospective termination for BCM. This is done next.

Retrospective termination for BCM: Recall that TD sends each LP in BCMS a terminate message with send and receive times set to T_T. Retrospective termination with BCM must guarantee Condition 2 from section 4. A sufficient condition to guarantee that Condition 2 can be met is that each LP maintains the following portion of the space-time diagram: time in interval [GVT,LVT], where LVT is the current LVT of the LP, and space coordinates equal to the attributes local to the LP that OMR requires. This implies that each LP in a BCM simulation must maintain a history of all attribute values and the interval over which they held since the last computed GVT. Finally, calculation of WVT as described earlier must be done if fossil collection of attribute values ineligible for use in output measures is desired.

In summary, the algorithm given here for termination detection and output measure collection with a conservative simulation protocol -- BCM -- must add at least some of the mechanism present in a time warp simulation. GVT calculation must always be added. In addition, if retrospective termination is used, saving certain states, WVT calculation, and fossil collection of olds states must be added.

6. CONCLUSIONS

Addressing the problem of detecting global as well as local termination conditions and calculating output measures in asynchronous, discrete-event parallel simulation leads to some good news and some bad news.

The good news is that if the termination condition is stable and there are no constraints on the set of times that may satisfy the condition (category *SU*), then with minimal modifications a time warp implementation can asynchronously evaluate the termination condition and obtain the attribute values needed at a time satisfying the termination condition. Furthermore, there is a simple reason to prefer optimistic protocols to conservative protocols: one must calculate GVT, save states, and collect fossils to calculate output measures, making optimistic protocols naturally suited. Finally, termination rules that require significant time to calculate (category *t*) may not limit simulator performance, because their evaluation can be done in parallel with the simulation and optimistically.

The bad news is categories *s* and *u*. Termination conditions that are not stable or complex constraints on the set of times that can satisfy the termination condition lead to a space-time diagram that looks like a checker board. For example a user that modifies a termination condition to read "Is this the first time X holds" from "Does X hold" may dramatically increase

the simulation run time. At worst a simulator must evaluate the termination condition after *every* change in value of *every* attribute. Therefore increasing the number of attributes in a model may lead to an exponential growth in the number of times at which the termination condition must be evaluated. However, special cases within *s* and *u* may permit efficient detection; one example is when evaluation of *C(t)* requires only one or two attributes.

Viewing the difficulties with category *u* another way, Law and Kelton (1982) distinguish two types of simulations: terminating and steady state. A terminating simulation generally will not fit category *U*, while one can be quite flexible about the run length of a steady state simulation to make its termination condition fit class *U*.

This work raises a number of broader questions. First, can we settle for some inaccuracy in the calculation of output measures in exchange for more efficient termination mechanisms? This adds a ninth dimension to Reynolds' design space of parallel simulation protocols (Reynolds 1988). Second, do simulation protocols based on some form of barrier synchronization have an inherent advantage for categories *u* and *s*? Third, what fundamental theorems can be proven concerning the time required to execute a parallel simulation whose termination rules fit categories *u* or *s*? (An exponential worst case behavior was conjectured earlier.)

REFERENCES

Balci, O. 1988. "The Implementation of Four Conceptual Frameworks for Simulation Modeling in High-Level Languages." in *Proc. of the 1988 Winter Simulation Conference* (San Diego, Calif., Dec.). IEEE, Piscataway, N.J., 287-295.

Chandy, K.M. and L. Lamport. 1985. "Distributed Snapshots: Determining Global States of Distributed Systems." *ACM Transactions on Computing Systems 3:* 63-75.

Chandy, K.M. and J. Misra. 1989. *Parallel Program Design: A Foundation.* Addison-Wesley, Reading, Mass.

Chandy, K.M. and R. Sherman. 1989. "Space-Time and Simulation." in *Distributed Simulation* (San Diego, Calif., Jan.). Soc. for Comp. Sim., San Diego, Calif, 53-57.

Francez, N. 1980. "Distributed Termination." *ACM Transactions on Programming Languages & Systems 2* (1): 42-55.

Jefferson, D. R. 1985. "Virtual Time." *ACM Transactions on Programming Languages & Systems 7*: 404-425.

Law, A. M., and W. D. Kelton. 1982. *Simulation Modeling and Analysis.* McGraw Hill, New York, N.Y.

Mattern, F. 1987. "Algorithms for Distributed Termination Detection." in *Distributed Computing 2*:161-175.

Misra, J. 1986. "Distributed-Discrete Event Simulation" *ACM Computing Surveys 18* (1): 39-65.

Reynolds, P. F. 1988. "A Spectrum of Options for Parallel Simulation." in *Proceedings of the 1988 Winter Simulation Conference* (San Diego, Calif., Dec.). IEEE, Piscataway, N.J., 325-332.

Tinker, P., and J. Agre. 1990. "Adaptive Model Prediction Using Time Warp." in *Distributed Simulation* (San Diego, Calif., Jan.). Soc. for Comp. Sim., San Diego, Calif, 165-168.