[22] P. Reiher, F. Wieland, and P. Hontalas. Providing determinism in the time warp operating system- costs, benefits, and implications. In *Workshop on Experimental Distributed Systems*, October 1990.

[23] L. Schruben. Simulation modeling with event graphs. *Communications of the ACM*, 26(11), November 1983.

[24] Som T.K. and Sargent R.G. A formal development of event graphs as an aid to structured and efficient simulation programs. *ORSA Journal on Computing*, 1(2):107–125, Spring 1989.

[25] E.G. Ulrich. Event manipulation for discrete simulations requiring large numbers of events. *Communications of the ACM*, 21(9), September 1978.

[26] S. Umeda. A manufacuring-oriented simulation package to support system planning and its operation. In *Winter Simulation Conference*, December 1992.

[27] H. Vollbrecht and Herdieckerhoff M. Simultaneity handling in sigmus. In *Summer Computer Simulation Conference*, July 1986.

[8] R.M. Fujimoto and D.M. Nicol. State of the art in parallel simulation. In *Winter Simulation Conference*, December 1992.

[9] S. Gai, P.L. Montessoro, and F. Somenzi. Mozart: A concurrent multilevel simulator. *IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems*, 7(9), September 1988.

[10] D. Jefferson. Virtual time. *ACM TOPLAS*, 7(3):404–425, July 1985.

[11] D. Jefferson, B. Beckman, and F. Wieland et al. Distributed simulation and the time warp operating system. In *Symposium on Operating Systems Principles*, Austin, Texas, October 1987.

[12] V. Jha and R. Bagrodia. A unified framework for conservative and optimistic distributed simulation. In *1994 Workshop on Parallel and Distributed Simulation*, Edinburgh, July 1994.

[13] Vikas Jha and Rajive Bagrodia. Transparent implementation of conservative algorithms in parallel simulation languages. In *Winter Simulation Conference*, December 1993.

[14] Leslie Lamport. Time,clock and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–564, July 1978.

[15] B. Lubachevsky and A. Weiss. An analysis of rollback-based simulation. *ACM Transaction on Modeling and Computer Simulation*, 1(2), April 1991.

[16] H. Mehl. A deterministic tie-breaking scheme for sequential and distributed simulation. In *Parallel and Distributed Simulation*, Jan 1992.

[17] J. Misra. Distributed discrete-event simulation. *Computing Surveys*, 18(1), March 1986.

[18] I. Mitrani. *Simulation Techniques for Discrete Event Systems*. Cambridge University Press, 1982.

[19] T. Nakanishi. Modeling problems in the processing of simultaneous events. In *Summer Computer Simulation Conference*, July 1992.

[20] D.L. Parnas. On simulating netoworks of parallel processes in which simultaneous events may occur. *Communications of the ACM*, 12(9), September 1969.

[21] A. Prakash and R. Subramanian. Filter: An algorithm for reducing cascaded rollbacks in optimistic distributed simulations. In *Annual Simulation Symposium*, 1991.

7 Conclusion

In this paper, we formalize the notions of simultaneous events, non-determinism, and lookahead in simulation models. The user may want to specify a particular ordering, or choose a default ordering, among the simultaneous events, either to model the physical system accurately, or simply to ensure determinism. We show that unless we want to burden the user with predicting the runtime causality relations between events, the only way we can ensure that the user defined orderings result in a unique and consistent execution is if there is at least one lookahead LP in every cycle, at every simulation time. This only ensures the *existence* of a unique, consistent execution. Different existing simulation protocols, including the traditional sequential protocol, require additional lookahead guarantees to actually *find* this execution. We prove these requirements for several such protocols. We show that a sequential optimistic protocol does not need any additional guarantees to find the unique, user-consistent execution. We also show that for some simulation protocols, if the user does not care about which order the simultaneous events get processed in, weaker lookahead guarantees are required than what would be required if a specific ordering of simultaneous events was desired.

References

[1] K.M. Chandy and R. Sherman. The conditional event approach to distributed simulation. In *Distributed Simulation Conference*, Miami, 1989.

[2] K.M. Chandy and R. Sherman. Space-time and simulation. In *Distributed Simulation Conference*, Miami, 1989.

[3] Bruce Cota and Robert Sargent. Simultaneous events and distributed simulation. In *Proceedings of the 1990 Western Simulation Conference*, pages 436–440, New Orleans, Louisiana, December 1990.

[4] Mattern F. Virtual time and global states of distributed systems. In *Proceedings of Workshop on Parallel and Distributed Algorithms*, pages 215–226, October 1989.

[5] R. Fujimoto. Parallel discrete event simulation. *Communications of the ACM*, 33(10):30–53, October 1990.

[6] R.M. Fujimoto. Time warp on a shared memory multiprocessor. In *International Conference on Parallel Processing*, 1989.

[7] R.M. Fujimoto. Optimistic approaches to parallel discrete event simulation. *Transaction of The Society of Computer Simulation*, 7(2), June 1990.

that need ordering in the context of event-scheduling based simulation world view. [19] describes modeling examples where wrong order of execution for simultaneous events can cause semantic errors. [20] discusses the issue of simultaneous events in the (sequential) simulation of a network of parallel processes.

In the rest of this section, we focus our attention on the treatment of simultaneous events in the context of parallel/distributed simulation protocols and systems. [17] gives an example where the null message based conservative protocol is able to enforce a particular user defined ordering between simultaneous events at an LP, required for correctness of the model, which a global event list protocol is unable to enforce. [3] deals with the question of providing extra information to the global event list protocol so that it produces the same result as the null message based scheme under the same set of lookahead guarantees. However, their scheme of assigning static priorities is based on the assumption of a statically defined lookahead LP in each cycle. They do talk about dynamic priorities, but only to accommodate newly created LP's. As shown in section 5.1.1, their method can be easily extended to work with a weaker dynamic lookahead guarantee, by computing the priorities dynamically.

[16] describes a sophisticated method to generate tie-breaking priorities among simultaneous events, in order to guarantee deterministic execution. However, these protocol-generated priorities are used as the *secondary* tie-breaker, with the user defined priorities being used as the *primary* tie-breaking mechanism. This still leaves the scope for the user to specify priorities that result in either *no* consistent execution, or *multiple* consistent executions (as shown in figure 3). The protocol generated tie-breaking priorities do not help here because the user has defined ordering for every pair of simultaneous events at an LP. A possible solution would be to use the protocol generated priorities as the primary tie-breaker, and the user defined priorities as the secondary tie-breaker (or ignore user defined priorities if the only objective is to provide determinism). This would correspond to the option 2 mentioned in section 4.3 in which the protocol chooses to enforce only those user defined orderings that are not in conflict with the causality ordering. This would require detecting, at runtime, the causal dependence between events with the same timestamp by associating additional fields with each message([16] proposes one such scheme), and could greatly benefit from the work done in the area of logical time by Lamport[14], Mattern[4], and others. In this paper, however, we choose the approach of enforcing all the user defined orderings, but ensuring that these orderings will produce a unique consistent execution by having the simulation model satisfy certain minimum lookahead guarantees. This eliminates the need for actually detecting causal dependence at the runtime. The lookahead requirement is not a limitation since, as seen in section 5, in order to enforce the user-defined orderings, most parallel protocols, including the optimistic and global event list protocols, require the presence of lookahead anyways.

	Arbitrary	User-consistent
Global Event List(GEL) Protocol	No Guarantee Needed	Each LP is ϵ-lookahead, or a known ϵ-lookahead LP in every cycle (theorem 3)
Null Message Protocol	A known c-lookahead LP in every cycle at every t except when there is an unprocessed message with timestamp t (corollary 1)	A known c-lookahead LP in every cycle at all simulation times(theorem 4)
Conditional Event Protocol	No Guarantee Needed (theorem 5)	Each LP is ϵ-lookahead (theorem 6)
Parallel Time Warp	A c-lookahead in every cycle	
Sequential Optimistic	No Guarantee Needed (theorem 8)	

Table 1: A summary of lookahead guarantees needed by various simulation protocols to enforce the two tie-breaking mechanisms. For the user-consistent mechanism, these guarantees are in addition to the basic guarantee that is needed for all protocols(Guarantee 1).

Theorem 8 *If the simulation model satisfies guarantee 1, the sequential optimistic protocol can enforce the user-consistent tie-breaking mechanism.*

Proof Outline:
We omit the proof of the fact that any committed computation is correct. We will prove that the protocol will make progress i.e. the set of committed events will eventually increase unless there are no more unprocessed events in the model.

Let E_r be the set of committed events at real time r. We will show that there is at least one event, which is not part of E_r, that will eventually be committed. Consider an event e_{ip}, with timestamp t, that is processed at LP i after real time r, and hence does not belong to E_r. If e_{ip} is never rolled back, then it is the event we are looking for. Otherwise, let e_{jq} be the positive event whose execution at LP j directly or indirectly caused the rollback of e_{ip}. Event e_{jq} must have been processed *after* e_{ip} because otherwise, a rollback of e_{ip} would not be necessary. Therefore, e_{jq} cannot be a part of E_r. The timestamp on e_{jq} must be t since it is processed after e_{ip}. In order for e_{jq} to cause a rollback of e_{ip}, there must exist a chain of zero-lookahead LPs from LP j to LP i in *topology(t)*. Therefore, the topological order of LP j must be less than LP i in *influence(t)* (recall the definition of *influence* graph in the proof for the theorem 1).

Now, we claim the event $e_{jq}(t)$ to be the event which will never be rolled back. If it *is* rolled back, then we repeat the above argument. Since every time we apply the argument, we end up with a candidate event at an LP that has a lower topological order in *influence(t)* than that of the LP where our previous candidate was scheduled. Since, there are only a finite number of LP's, eventually we will end up with an event that will never be rolled back. Thus, we show that the set of committed events will eventually increase.
End of Proof.

6 Related Work

Most of the work regarding simultaneous events in discrete event simulation has taken place in the context of sequential (global event list) simulation protocol [27], [26], [25], [23], [19], [9], [20]. [27] discusses the handling of simultaneous events in a (sequential) multiprocessor simulator. Authors note that in case of process based simulation, assigning static priorities to events within the same process, and static priorities to different LP's (to uniquely define the interleaving) may not be able to capture the desired behavior. They define rules, in the context of developing a simulator for a multiprocessor computer, in order to eliminate non-determinism due to simultaneous events.

[26], [9], and [25] discuss impact of simultaneous events on the performance of sequential simulation, and describe data structures that handle large number of simultaneous events efficiently. [23] and [24] discuss detection of simultaneous events

- A processor never blocks while there are unprocessed events in the queue of any of the LP's mapped to that processor. This is sometimes referred to as *extreme optimism*, and excludes protocols which allow for throttling of optimism under certain conditions.

- For incorrect events, cancellation information always *follows* the original event. This excludes protocols like [21] that associate some kind of causality information with the events which, along with the local information stored at an LP, allows them to determine the incorrectness of certain events as soon as they arrive (without requiring an anti-event).

The shared memory implementation of Time Warp based on direct cancellation [6] may reduce the probability of occurrence of the situation described above because of a significantly faster cancellation of incorrect events. However, it still does not eliminate the possibility since cancellation always takes place after the arrival, and possibly after the processing, of the incorrect events.

The above described situation can be avoided by assuming a c-lookahead LP in every cycle of the model, at every simulation time. This lookahead guarantee is similar to the one required for the null event protocol, except that in this case, the lookahead only needs to *exist*, not necessarily be *known* to the simulation engine. With this guarantee, Time Warp can enforce the *user-consistent* tie-breaking mechanism. This guarantee would be required for progress even if user only wanted the *arbitrary* tie-breaking mechanism. We omit a proof that Time Warp will make progress in presence of this guarantee. A proof of the progress property for Time Warp under a stronger guarantee, namely, *every* LP is c-lookahead, is presented in [12]

5.2.2 Sequential Optimistic Protocol

All the simulation protocols that we have considered so far, including the sequential global event list protocol, require some form of lookahead guarantee, in *addition* to guarantee 1 (which is needed to ensure the existence and uniqueness of the user-consistent execution), in order to enforce the user-consistent tie-breaking mechanism. In this section, we describe a protocol that does not need any additional assumptions.

Protocol: Sequential optimistic protocol is basically same as the Time Warp protocol [10] (also see section 5.2.1), with all the LP's mapped to the same processor. We add the following two restrictions:

- Anyone of the LP(s) with the earliest unprocessed input in the entire model is chosen first.

- In case of a rollback, all the incorrect computation in the entire logical system is rolled back before any other events are processed i.e. all the anti-events are processed before another positive event is processed.

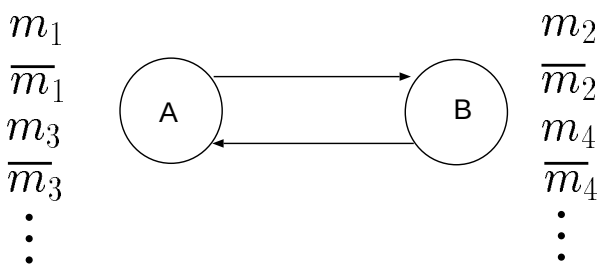


Figure 4: Possibly Infinite Incorrect Event Chain in Optimistic Protocols

equal to the timestamp of the last event processed is a straggler if it is an anti-event *or* an event specified by the user to be executed before an already executed simultaneous event. Upon a rollback, any erroneously sent events are canceled by sending corresponding anti-events. Since there is always buffer space to store newly generated events (because of the assumption made in section 2), a processor is never blocked as long as there is an unprocessed event scheduled at any of the LP's mapped to that processor.

Theorem 7 *If all the LP's are allowed to have a zero, or even ϵ (greater than zero, but arbitrarily small) lookahead, parallel Time Warp cannot be guaranteed to make progress under any tie-breaking mechanism.*

Proof:
Consider 2 LP's, A and B, in the network (figure 4), mapped to two different processors. During the simulation, let A receive an *incorrect* event (an event which was produced due to incorrect optimistic computation and will be rolled back) m_1 with timestamp t. Upon processing it, A sends an event m_2 to B with timestamp $t + \epsilon_1$. Shortly after this, A receives the anti-event $\overline{m_1}$, and sends $\overline{m_2}$ in response to B. But, B has already processed m_2 and sent the output m_3 with timestamp $t + \epsilon_1 + \epsilon_2$ to A, before it receives $\overline{m_2}$. At this time, B sends $\overline{m_3}$ to A, but, A has already processed m_3. This can go on forever with the earliest unprocessed event in the model having a timestamp of $t + \epsilon_1 + \epsilon_2 + ... + \epsilon_n$, where n is increasing. If ϵ_i is allowed to be zero or arbitrarily small (e.g. each $\epsilon_i = \epsilon_{i-1}/2$), this value may never increase beyond a point, thus not allowing a *a correct event with a higher timestamp to be committed.* This problem (for the case where ϵ_i is zero) has been discussed in [15].
End of Proof.

The above situation is possible in both, the aggressive and lazy cancellation[10], versions of Time Warp. Also, it applies to other optimistic protocols e.g. space-time protocol[2]. The only conditions that an optimistic protocol needs to satisfy in order for the above example to apply are:

- LP's A and B may be mapped to two different processors (parallelism).

Progress: Assume the contrary i.e. the protocol deadlocks with some events still remaining unprocessed. Eventually, all events in transit will be delivered. Assuming a new EIT computation is eventually initiated after this, the EIT of every LP will be at least equal to the globally earliest unprocessed event timestamp in the model. At this time, the earliest timestamp event in the model will get processed. Hence, contradiction.
End of Proof Outline.

Theorem 6 *If every LP in the network is an ϵ-lookahead LP, the conditional event protocol can enforce the user-consistent tie-breaking mechanism.*

Proof Outline:
Instead of computing EIT for every LP, we compute EIT+ which is equal to t if the minimum ECOT in the model is t, and all the events in transit have timestamp *greater* than t. Each LP can process an event as long as its timestamp $\leq$ EIT+. This condition, along with the lookahead guarantee made above, ensures that an LP processes an event with timestamp t only when *all* the events with timestamp t have been received by the LP. Therefore, any user defined ordering on simultaneous events can easily be enforced. Progress is guaranteed, since, in case of a deadlock, eventually EIT+ will be equal to the timestamp on the earliest unprocessed event in the model, which will then be processed.
End of Proof Outline.

5.2 Optimistic Protocols

The main distinguishing feature of optimistic simulation protocols is that they allow the possibility of an LP processing an event in an incorrect timestamp order. Upon discovering such an error, the LP is rolled back to a previously saved state, and any output events generated due to the erroneous computation are canceled, typically by sending anti-events[10]. Several optimistic protocols have been proposed in the literature (see [5] for a survey). In this section, we consider the Time Warp [10] protocol as an example of the optimistic protocols.

5.2.1 Time Warp

Protocol: See [10, 11] for a detailed description of the protocol. Here, we only provide a brief summary. Each LP processes the earliest available event in its input queue. State of the LP is periodically checkpointed. Upon receiving a *straggler* event, an event with a timestamp smaller than the latest event processed so far, an LP is rolled back to a timestamp before the timestamp of the straggler event, and is recomputed. If user only desires an *arbitrary* tie-breaking mechanism, an event with timestamp *equal* to the timestamp of the last event processed is a straggler *only if* it is an anti-event. If *user-consistent* tie-breaking mechanism is desired, an event with timestamp

at $t = D$, lookahead of the LP becomes zero. If both A and B have a type L job scheduled to leave at timestamp D, a zero lookahead cycle is created at simulation time D, thus violating the guarantee required by theorem 4. Therefore, if the user wants to specify an ordering between simultaneous events e.g. "process the arrival events before the departure events with the same timestamp and at the same LP", then the null message protocol cannot be used to simulate this model.[4] However, the model does satisfy the guarantee required in corollary 1, since there is a zero lookahead cycle at simulation time D only if there is an unprocessed event with timestamp D (the departure of L job) in the logical system. Therefore, if the user does not care about the ordering among simultaneous events, then the model may be simulated using the null message protocol.

5.1.3 Conditional Event Protocol

It is widely believed that the parallel conservative methods always *require* the existence and the knowledge of lookahead in order to make progress. While this holds for the null message protocol (as discussed in the previous section), we show that it is possible for at least one of the parallel conservative protocol, namely, conditional event protocol[1], to not *require* the presence of any lookahead, as long as the user is satisfied with the *arbitrary* tie-breaking mechanism.

Definition 4 (ECOT) *Earliest Conditional Output Time (ECOT) of an LP, at real time r, is defined to be a lower bound on the timestamp of any output events sent by the LP after real time r, assuming it does not receive any inputs after real time r. It is equal to $T + l$, where T is the timestamp on the earliest* unprocessed *input at the LP at real time r, and l is the lookahead.*

Protocol: EIT for all the LP's, at a given real time r, is computed by finding the minimum of the ECOT's of all the LP's and the timestamp of all the events in transit at real time r (this is just like the GVT calculation in Time Warp [10]). We omit a proof of the fact that this method correctly computes EIT for each LP.

Note that for the sake of efficiency, the conditional event protocol is often used in conjunction with the null message protocol [1, 13]. The lookahead guarantee required by such a combination would be the weaker of the guarantees required by each of the two protocols.

Theorem 5 *Conditional Event Algorithm can enforce the arbitrary tie-breaking mechanism without requiring any lookahead guarantees in the simulation model.*

Proof Outline:
The proof of the fact that each LP processes events in a non-decreasing timestamp order follows from the correctness of the EIT computation.

[4]Some special orderings may be enforceable e.g. "process departure event before any arrival events", but an arbitrary user defined ordering may not be.

of EIT, all the events with timestamp t the LP will ever receive will be in the input queue when the first such event is executed. Hence, all the user defined orderings can be satisfied, provided the protocol makes progress.

We will show the progress property by contradiction. Assume the contrary i.e, that the simulation is deadlocked, and unprocessed event(s) still exist in the model. Consider the smallest EIT, say t, among all LP's in the model. Eventually, all the events with timestamp less than t will be processed, and the earliest unprocessed event in the entire logical system will have a timestamp of t or more. Let c be a constant such that every cycle in *topology(t)* has at least one LP with lookahead c or more. Then, it can be shown that eventually, the EIT of each LP will be updated, using null messages, to $t + c$ or more. If the timestamp of the next unprocessed event in the model is t', we can show, by repeating the above argument, that the minimum EIT in the model will eventually be *greater than* t', at which point all the events with timestamp t' will be processed. Hence, a contradiction. Note that an ϵ lookahead may not be sufficient to guarantee that the minimum EIT in the logical system will eventually advance to t'.
End of Proof Outline.

Corollary 1 *If the guarantee required in theorem 4 is relaxed such that it need not hold for simulation time t, if there is guaranteed to be an event generated in the logical system with timestamp t, the null message protocol can still make progress, but only under the* **arbitrary** *tie-breaking mechanism. Note that this implies that there may be instantaneous zero lookahead cycles.*

Proof Outline:
The protocol is modified to process events with timestamp *less than or equal to* the value of EIT. Therefore in the above proof, we only need to show that the minimum EIT in the model reaches t (and not necessarily *greater than* t). Hence, we can allow the LP's to have zero lookahead at timestamp t, but only as long as there is an unprocessed event with timestamp t. As soon as all the events with timestamp t in the model are processed, it must again satisfy the guarantee required in theorem 4.
End of Proof Outline.

Below we give an example which is used very often in queueing network simulations where it is not possible to satisfy the guarantee required by theorem 4, but, it is possible to satisfy the weaker guarantee required by corollary 1.

Consider a cycle of 2 pre-emptible priority servers A and B. There are two types of jobs in the physical system, type L (Low priority), and type H (High priority). Processing of a type L job may be interrupted by the arrival of a type H job. Suppose the minimum service time for either of the job types is S. Now, let us look at the lookahead characteristics of the LP's. If at simulation time t, an LP is idle i.e. it is not processing any jobs, its lookahead is equal to S. However, if at simulation time t, it is processing a type L job that is tentatively, subject to no H job arriving before, scheduled to depart at time $D > t$, its lookahead at t is equal to $min(S, D - t)$. Thus,

Proof:
In case of the first guarantee, since processing of an event with timestamp t cannot result in another event with timestamp t (or less), all the events with timestamp t that will ever be received by an LP are available when the first such event is processed in GEL. Therefore, the user defined orderings can be easily enforced.

For the second guarantee, the GEL protocol will have to be modified slightly. Recall the proof for theorem 1. We used the *influence* graph, constructed using the lookahead LP's, to show the *existence* of a unique user-consistent execution. Since, the identity of the lookahead LP's and the topology is now *known* to the simulation engine, it can construct *influence(t)* before processing events with timestamp t, and use it to break the ties. This is a straight-forward extension of the method described in [3] which assumes a statically defined topology and lookahead for each LP. Note that the instantaneous topology *topology(t)*, available to the simulation engine, does not have to be exact; it is sufficient if it is a superset of the actual instantaneous topology.
End of Proof.

5.1.2 Null Message Protocol

Definition 3 (EOT) *Earliest Output Time (EOT) of an LP, at real time r, is defined to be a lower bound on the timestamp of any output events sent by the LP, after real time r. It is equal to $min(EIT, T) + l$, where T is the timestamp on the earliest* unprocessed *input at the LP at real time r, and l is the lookahead.*

Protocol: Details of the protocol can be found in [17]. Here, we provide a brief description. Each LP informs its successors, using a null message, of the value of its EOT, every time it changes (this value can be piggybacked with regular events whenever possible). EIT of an LP is equal to the minimum of the largest EOTs received on each of the input channels (recall that the communication channels of the multiprocessor are assumed to be FIFO for the NM Protocol). We omit a proof that the protocol correctly computes the value of EIT for each LP.

In contrast to the GEL protocol, the NM protocol requires a lookahead guarantee even to enforce the arbitrary tie-breaking mechanism. In the next theorem, we first describe a lookahead guarantee that is sufficient for the NM protocol to enforce the user-consistent mechanism, and then as a corollary, we describe a slightly weaker guarantee that is sufficient for it to support the arbitrary mechanism.

Theorem 4 *If, for all t, the simulation engine knows* topology(t) *(or a superset of it), and every cycle in* topology(t) *has a c-lookahead LP, whose identity and lookahead value is known to the simulation engine, the null message protocol can enforce the* user-consistent *tie-breaking mechanism.*

Proof Outline:
Since an LP processes an event only when its timestamp is *less* than the current value

In a conservative protocol, an LP will process only events with timestamp *less than or equal to* the current value of its EIT if the user desires the *arbitrary* tie-breaking mechanism, and only the events with timestamp *less than* the current EIT if the *user-consistent* tie-breaking mechanism is desired. Different conservative protocols only differ in the way they compute the value of EIT for each LP. Since the sequential GEL protocol processes the events in the global timestamp order, it does not need to compute EIT.

First we prove a theorem which applies to *all* conservative protocols

Theorem 2 *Lookahead Guarantee 1 is not sufficient for* **any** *conservative protocol to ensure that it will find the* user-consistent *execution.*

Proof:
Note that guarantee 1 ensures the *existence* of a unique user-consistent execution as we have seen in the previous section. However the following example shows that it is not sufficient to ensure progress in a conservative protocol that tries to find that unique user-consistent execution:

Consider the situation where LP's A and B have events e_1 and e_2 scheduled in their queues, respectively, both with timestamp t. Since the identity of the lookahead LP is *not known* to the simulation engine, A cannot execute e_1 until B executes e_2; since upon executing e_2, B may schedule another event e with timestamp t at A, such that $e\ u_<\ e_1$. B cannot execute e_2 before A executes e_1 for a similar reason. Thus, we have a deadlock situation.
End of Proof.

5.1.1 Global Event List Protocol(GEL)

Protocol: All the pending events are stored in a central global event queue. Events are deleted, and executed, from the event queue in the order of their timestamp, with the new events added to the queue. This is the protocol traditionally used for sequential simulation. GEL protocol does not need any lookahead guarantees to enforce the arbitrary tie-breaking mechanism. However, as shown in theorem 2, guarantee 1 is not sufficient to ensure the progress of GEL protocol, if the user-consistent tie-breaking mechanism is desired. Following theorem describes two possible guarantees each of which is sufficient to allow GEL protocol to enforce the user-consistent tie-breaking mechanism.

Theorem 3 *If any of the following two guarantees are satisfied, GEL can enforce the user-consistent tie-breaking mechanism:*

- *If* **every** *LP in the model is ϵ-lookahead.*
- *If, for all t such that there may be an event scheduled with timestamp t,* topology(t) *is known to the simulation engine, and every cycle in* topology(t) *contains an ϵ-lookahead LP whose identity is known to the simulation engine.*

as a result of processing event sequences with timestamps *less* than t, which are identical in the two executions. Since both executions are user-consistent, the order of execution of the simultaneous events at A is identical in the two executions also. We can inductively prove this for every LP in *state(t)* (induction on the topological order of the LP in *influence(t)*). Thus, the two executions are identical up to and *including* timestamp t, which is a contradiction.
End of Proof Outline.

In summary, we note that if we want a unique user-consistent execution, and we want the user to be free from the burden of analyzing the causality relations between simultaneous events while specifying user defined orderings, the simulation model should satisfy guarantee 1, irrespective of the simulation protocol chosen to execute the model. A particular simulation protocol may require *additional guarantees* in order to ensure that it will find that unique user-consistent execution. We discuss this in the next section.

5 Lookahead Guarantees Required by Simulation Protocols

In this section, we discuss the weakest lookahead guarantees that are required by different simulation protocols in order to ensure progress. Also, we discuss how these guarantees depend on which one of the tie-breaking mechanism user desires. For the *arbitrary* tie-breaking mechanism, no lookahead guarantee is required to ensure the existence of a legal execution. However some protocols may still need lookahead guarantees to ensure their progress e.g. Null Message protocol or Time Warp. For the *user-consistent* tie-breaking mechanism, guarantee 1, which is needed to ensure the existence and uniqueness of user-consistent execution, is common to all protocols. Some protocols may need stronger guarantees to ensure their progress. The main results of this section are summarized in Table 1 at the end of this section.

5.1 Conservative Protocols

A conservative protocol is defined to be one in which each LP processes an event only when it is guaranteed that no earlier event may be received by it. We, therefore, include all the non rollback-based protocols in this class, including the sequential Global Event List Protocol. First, we define the concept of EIT [12] which we use to describe some protocols.

Definition 2 (EIT) *Earliest Input Time (EIT) of an LP, at real time r, is defined as the lower bound on the timestamp of any events that may be delivered to the LP after real time r.*

Option 1 requires the user to be able to predict and understand exact global causality relations between simultaneous events, which in case of complex simulations may be a difficult task. Option 2 is not desirable because at the time of specifying the orderings for the simultaneous events, user has no idea as to which of the specified orderings will actually be enforced. We consider the option 3 in the rest of this paper. In the rest of this section, we describe sufficient lookahead guarantees that will ensure that any LCC user-specified orderings will result in a unique user-consistent execution.

Lookahead Guarantee 1 *For all t, such that there may be an event scheduled at time t during the execution of the simulation model, every cycle in* topology(t) *must have at least one ϵ-lookahead LP. (Note: Recall that the* topology(t) *is the instantaneous topology as defined in section 2. This guarantee is made by the simulation model, and is independent of any protocol.)*

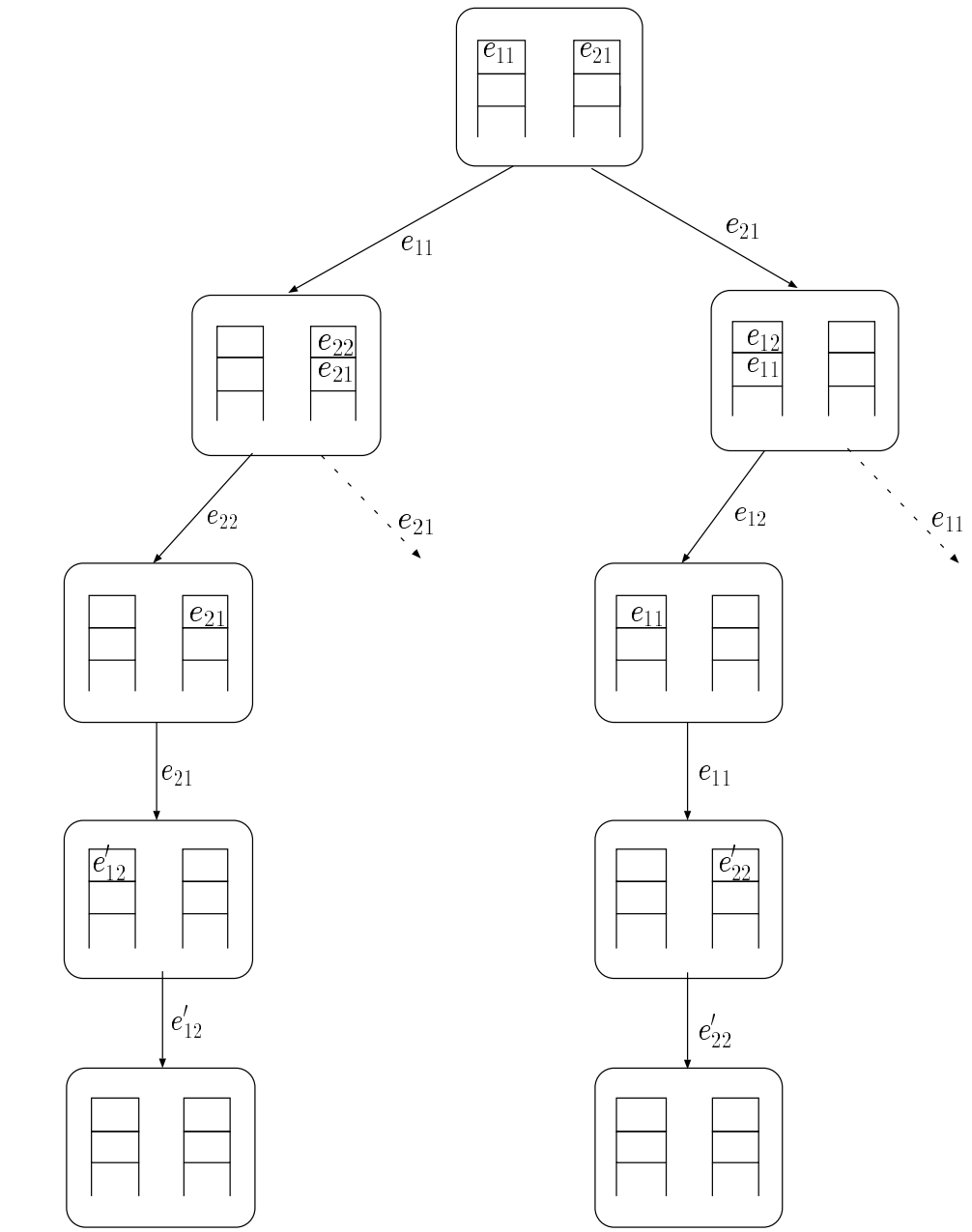
Theorem 1 *If a simulation model satisfies guarantee 1, it will have a unique* user-consistent *execution, for any LCC user defined orderings.*

Proof Outline:
We first show the existence of a user-consistent execution in the execution tree. Then, we will show that all user-consistent executions are *identical*.

For any t such that there may be an event scheduled at timestamp t during the execution of the simulation model, construct, for the purposes of the proof, a directed *influence(t)* graph (as defined in [3]). Each LP in *topology*(t) forms a node in *influence(t)* graph. There is an edge from a node A to node B in *influence(t)* iff there is at least one path from A to B in *topology(t)* that has no ϵ-lookahead LP on it. Because of the lookahead guarantee made above, this graph would be acyclic. Sort *influence(t)* topologically. Choose an execution in the execution tree that satisfies the following condition: no event with timestamp t is processed at an LP until all events with timestamp t scheduled at LP's with lower topological orders in *influence(t)* have been processed. Ties between events on the same LP are broken using the user-specified rules.

Because of the way *influence(t)* is defined, upon processing an event with timestamp t, an LP cannot send timestamp t event to any LP with a lower topological order in *influence(t)*. Therefore, *all* the timestamp t events ever to be received by an LP are available when the first such event is executed. Hence, the execution will be user-consistent.

Now, we will show that any two user-consistent paths in the execution tree are *identical*. Assume the contrary. Let t be the *earliest* timestamp at which the event sequence processed by an LP in one user-consistent path differs from the sequence processed by the same LP in another user-consistent path. Therefore, the *state(t)* and *topology(t)*, and hence *influence(t)* in the two executions are identical. Let A be an LP at the topological order 0 in *influence(t)*. The simultaneous event sets with timestamp t at LP A in the two executions are identical because they are produced



User Defined Orderings

$e_{12} \cup_< e_{11}$ $e_{22} \cup_< e_{21}$

$e_{11} \cup_< e'_{12}$ $e_{21} \cup_< e'_{22}$

Figure 3: Execution Trees 3

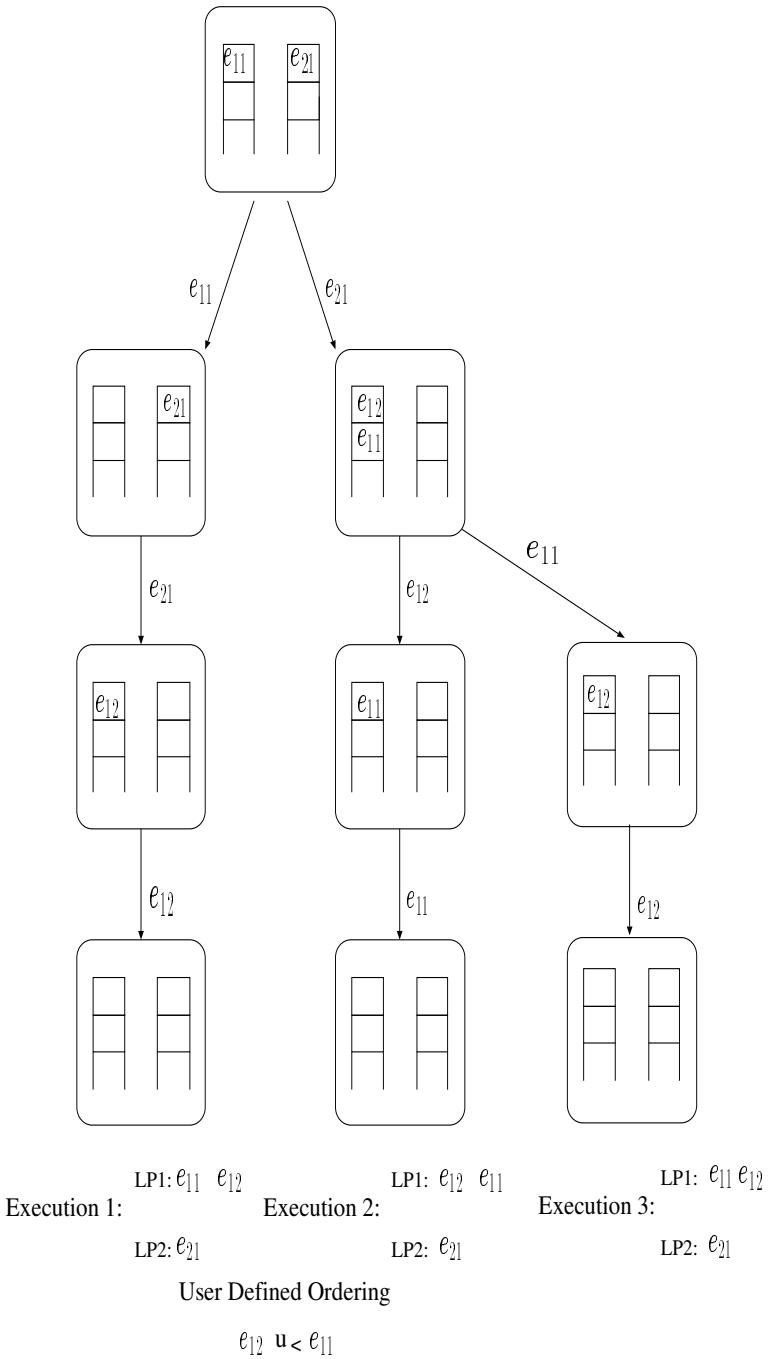


Figure 1: Execution Trees 1

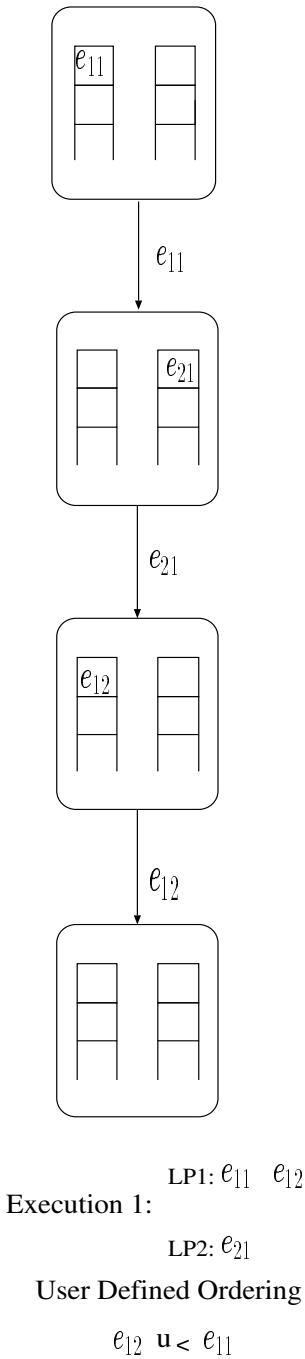


Figure 2: Execution Trees 2

has a timestamp greater than t, it is not shown in the execution tree. As seen in the figure, the execution tree has three executions, two of which are identical. The ordering rules specified by the user require event e_{12} to be executed before event e_{11} at LP 1, independent of the state of the LP. Therefore, only execution 2 is user-consistent, executions 1 and 3 are not.

Given a simulation model, and an LCC ordering rules, can the *existence* and *uniqueness* of a user-consistent execution be guaranteed ? Note that since we are talking about the the executions in the execution tree, and not whether a particular simulation protocol can find it, the discussion is independent of any protocol. First we consider the *existence* of a user-consistent execution. Figure 2 shows an execution tree for a 2 LP simulation model, different from the one whose execution tree was shown in figure 1. As shown, the simulation model has only one execution, which is not user-consistent. The user-specified orderings, however, do satisfy the requirements of being Locally Consistent and Complete. Thus, we conclude that without additional guarantees, LCC orderings cannot ensure the existence of user-consistent execution. We note that in this example, there is no consistent execution because the event e_{12} is indirectly produced by e_{11}, and yet has been specified to be executed before e_{11}.

Probably a little less intuitive is the fact that in some simulation models, LCC orderings may allow the existence of *multiple*, non-identical, user-consistent executions. Consider the execution tree shown in Figure 3. It has several possible executions (the dashed edges in the tree correspond to parts of the tree that we do not show in this figure), out of which two executions, namely, executions 1 & 2 are user-consistent. Therefore, the simulation model has multiple distinct user-consistent executions, despite the fact that the orderings specified by the user are Locally Consistent and Complete.

Both of the above problems arise because we have defined the user-specified orderings in a local fashion, ordering the events on a given LP, and ignoring the causality relation between events *across different LP's*. In order to ensure the existence and uniqueness of the user-consistent executions, we have three options:

1. Require that the user defined ordering rules, in addition to being LCC, also obey certain causality related rules to ensure that there exists a unique user-consistent execution.

2. Ordering rules themselves need only be LCC, however, the simulation protocol transparently detects and ignores the orderings that cause either a lack of user-consistent execution, or a multiplicity of user-consistent executions. One way to accomplish this is to have the protocol append a special field, that captures the causality dependence between simultaneous events, to every event[16].

3. Have the simulation model satisfy certain lookahead guarantees which ensure that *any LCC user defined orderings will result in a unique user-consistent execution.*

protocol-independent, we require that the content of an event generated by an LP be completely determined by the initial state of the LP and the sequence of events processed by the LP so far. For instance, priority may not be dependent on protocol-dependent parameters like "the state of the global queue", "number of rollbacks" etc.

- In place of, or in addition to, the user defined rules, *default* protocol-independent ordering rules may be used. For instance, a lexicographic ordering on the byte representation of the simultaneous events is used in [22]. Since the default rules are protocol-independent, from the point of view of a simulation protocol, they can be considered as being part of the user defined rules.

- In many situations, the ordering of the events is purely based on the contents of the events, and is independent of the current state of the LP. For instance, the orderings "execute events of type A before type B", or "execute events received from LP p before those received from LP q". In such situations, we use $e_i \; u_< \; e_j$ to denote a user specified ordering, where e_i has been specified to be executed before e_j, if they have the same timestamp.

- The above definition assumes that the LP processes one event at a time. It is conceivable that correct modeling of some physical systems may require *simultaneous* processing of two or more simultaneous events. We omit such situations to keep the discussions simple.

We can now formally define a user-consistent execution.

Definition 1 (User-consistent execution) *Let M be any simultaneous event set at LP A in an execution E. Let $< e_0, e_1, ..., e_n >$ be the events in M, in the order in which they have been executed in E. Let, the state of the LP A before processing event e_i be S_i. Then, the execution E is said to be user-consistent with respect to M iff $\forall i, 0 \leq i \leq n$, given the state S_i and the set $\{e_j | i \leq j \leq n\}$, the user defined rules will select event e_i. An execution E is* **user-consistent** *iff it is user-consistent with respect to every simultaneous event set in E.*

Note that the above definition requires that *all* the events with a given timestamp at an LP satisfy the user-defined orderings. In other words, even if a protocol uses the user-defined tie-breaking scheme to choose the next event to execute, among the available unprocessed events, it may not be sufficient to ensure that the resulting execution will be user-consistent. This is because it is possible that all the simultaneous events may not be in the queue when the tie-breaker is used. Figure 1 illustrates *user-consistent executions* using part of an execution tree of a 2 LP simulation model. The tree only shows the events with timestamp t. The two event queues shown inside each node of the tree contain the pending events for each of the two LP's. Each edge is labeled by the next event chosen for execution. Event e_{ij} represents an event scheduled on LP i. If the output event, scheduled as a result of processing an event,

Many sequential simulation systems allow the user to specify order of execution among events with the same timestamp in the *entire logical system*. From the point of view of a parallel simulation, however, it makes more sense to allow a user to order the execution of only the same timestamp events scheduled at a *given LP*. The reasons for this are as follows:

- In the simulation paradigm mostly used for parallel simulation (see section 2), LP's do not share any variables. Therefore, relative ordering of events across different LP's does not affect the outcome of the simulation as long as the simultaneous events on *each* LP are ordered identically. In other words, ordering of simultaneous events at each LP completely determines the outcome of the simulation, irrespective of the ordering across LP's.

- In case of asynchronous parallel execution, where different LP's may be executing on different processors, and have different local simulation clock values at a given real time, enforcing global orderings on simultaneous events may require very expensive global synchronization operations.

Consequently, we define a *simultaneous event set* to be the set of *all* events with the same timestamp, scheduled *at the same* LP. Events belonging to the same simultaneous event set are referred to as simultaneous events. The user can only order the events belonging to the same simultaneous event set. Note that since we want the user specified orderings to be protocol-independent, we use the same definition of simultaneous events for all protocols, including the sequential one, even though it is easy to to implement ordering of simultaneous events across the LP's in case of the sequential protocol.

The second issue is *how* the user specifies these orderings. We assume that the user may define, in addition to the simulation model, a set of rules to order the simultaneous events. Given any set of simultaneous events that may be generated during simulation[3] at an LP, and the current state of the LP, the user defined rules must determine a **unique** next event to be processed, the decision being entirely determined by the current state and the contents and timestamp of the simultaneous events (if two events have exactly the same content and timestamp then it does not matter which one gets executed first). We call such orderings *Locally Consistent and Complete*(**LCC**). All the user defined orderings considered in this paper are assumed to be LCC.

We make a few observations regarding the user-specified orderings:

- Any tie-breaking *priority* field associated with a event is treated as part of the content of the event. Note that since we want the user defined orderings to be

[3]The orderings need only be defined for the simultaneous event sets that may actually be generated during simulation. In case of optimistic simulations, it is possible that the simultaneous event set may contain some erroneous events which will subsequently be canceled. In such a situation, we allow the user defined rules to select *any* event. The actual event selected is unimportant since the computation will be rolled back and canceled subsequently

of the same simulation model may produce different executions. Therefore, any execution in the execution tree of the model is considered a correct execution.

- *User-consistent and deterministic* (called **User-consistent**): The ties are broken according to protocol-independent, user specified (or default) rules. Only the execution(s) in the execution tree that satisfy the user specified ordering rules, called *user-consistent* execution(s), are considered correct executions of the simulation model. We formally define a user-consistent execution in the next section, after we have discussed how the user specifies the orderings. This tie-breaking mechanism also guarantees that a given simulation model will have a unique execution, independent of the simulation protocol.

There are two other related tie-breaking mechanisms we do not consider here. The first is *arbitrary but deterministic* which would allow a simulation protocol to choose any ordering, as long as it guarantees the same ordering in every run. This option could be easily enforced by sequential protocols, without requiring any lookahead guarantees. However, as far as the lookahead implications for the parallel protocols are concerned, this mechanism is equivalent to the second mechanism defined above. Another possibility is a *user-consistent but non-deterministic* execution. This would be applicable when the user defined ordering rules allow several consistent executions. We rule out this possibility by requiring the user specified rules to allow only a unique execution.

In view of the obvious validation and debugging problems that a non-deterministic execution has, a natural question to ask is why a user may prefer the *arbitrary* execution model over the *user-consistent* one. As will see later in this section, and in the next section, the latter requires the simulation model to meet relatively stronger lookahead guarantees. One of the objectives of this paper is to clarify this trade-off for various simulation protocols. Knowing this, the user may choose the *arbitrary* mechanism if the simulation model cannot meet the lookahead guarantees required by the *user-consistent* mechanism.

4.3 User-Specified Orderings

Given that the user may want to order the simultaneous events in a simulation model, the natural question to ask is "How is the user going to specify such orderings ?" In this section, we try to answer this question. More specifically, we try to answer the following three questions:

- Do the user-specified orderings apply to the same timestamp events in the entire logical system, or only those scheduled at the same LP?
- How are the ordering rules specified?
- Under what conditions do these rules guarantee a unique user-consistent execution?

- Children of any node in the tree are defined as follows: Let U be the set of all the *earliest* events among all the pending event queues in the node. There is a child corresponding to each event in U; the state of the child node reflects the change in state and pending event queues as a result of processing that event. There is a *directed* edge from the parent to each child.

- A *leaf* node is one in which all the pending event queues are empty, or the earliest pending event has a timestamp greater than the maximum simulation time.

Any path in the execution tree[2], from the root to a leaf node, is defined to be an **execution** of the simulation model. A few observations about the execution tree:

- An execution is completely characterized by the sequence of events processed at each LP. Two executions in the tree are said to be *identical* if and only if the sequence of events processed by each LP is the same in both of them. Two identical executions will always produce the same simulation output. Two non-identical executions may, or may not, produce the same results, depending on the application. A simulation model, or a tie-breaking mechanism, is said to be *deterministic* if all its executions are *identical*. It is said to be non-deterministic otherwise.

- Ignoring the user defined orderings for simultaneous events, the execution tree captures *all* and *only* the correct executions of the simulation model, **independent of the simulation protocol**. Since we *define* an execution to be a path in the execution tree, this assertion is self-evident. However, we suggest that the readers intuitively convince themselves of the correctness of the assertion with respect to any of the traditional simulation protocols.

- Since we do not restrict the number of events in the simulation model, the execution tree, and hence an execution contained therein, may be infinite. All our results apply to such an execution tree also. In case of infinite executions, even though the protocol will never terminate, it can still said to be making progress if the set of *committed* events increases monotonically.

4.2 Tie-Breaking Mechanisms

Out of the several tie-breaking mechanisms possible, we consider the following two in this paper:

- *Arbitrary and non-deterministic* (called **Arbitrary**): The ties between simultaneous events may be broken in an arbitrary fashion. Further, different runs

[2]execution tree and the paths in the execution tree are not to be confused with the communication topology and cycles in the simulation model itself

in a server, arrival of a job at a server, the initiation of its service, and its departure may be modeled as separate events. In such a case, execution of the departure event for a job at a server LP may schedule the arrival event of the job at next LP, which could be a channel LP or the next server in the network, at the same simulation time.

The order of execution of simultaneous events is immaterial in some simulation applications, except perhaps, for the sake of reproducibility of simulation results. However, in many applications, a particular order of execution among the simultaneous events may be required in order to correctly model the physical system. We consider below some such examples:

- There are many modeling scenarios (e.g. a priority server; also see [17] for an example) where an event X is scheduled by an LP A at timestamp t only if no event of type Y is scheduled for LP A before or *at* timestamp t.
- [23] gives an example of modeling a semi-automatic machine system where executing simultaneous events in the wrong order would leave the system in an error state.

4 Tie-Breaking Mechanisms and User-Specified Orderings

We first introduce some definitions:

4.1 Definitions

For a simulation protocol to be *correct* it must obey the *safety* property – events at each LP must be executed in a non-decreasing order of timestamp, and the *progress* property – the set of *committed* events, in the entire simulation model, must increase with real time, until all the events in the model are processed. A *committed* event in case of conservative protocol simply means a processed event, whereas in case of an optimistic protocol it refers to an event that has been processed and will never be rolled back.

We define an **execution tree** for a given simulation model, *ignoring the user specified orderings*, as follows:

- Each node of the tree represents a state of the logical system. It consists of the state and the pending (unprocessed) event queue for each of the LP's currently in the model. Each queue is ordered by timestamp, with no ordering defined among events with the same timestamp.
- The initial node in the tree consists of the initial state with the queues containing the initial events.

Serve (FCFS) server with a randomly computed service time. We assume that the service time cannot be precomputed before a job actually arrives, since the job may carry the seed for the random function call. Consider the lookahead guarantee that can be made for such an LP at simulation time t. First, the guarantee regarding the *value* of the lookahead: If the server is idle at simulation time t, the lookahead at t is equal to the *minimum* possible service time for the next job to be serviced. If the service time is computed using a random function call, the lower bound on its return value forms the lookahead guarantee for the LP. If the server is busy processing a job at simulation time t, then the lookahead is simply equal to the remaining service time of the job. The second part of the guarantee determines whether the value of the lookahead of the LP is known to, and can be exploited by, the simulation engine.

There are two time scales during the execution of a simulation model - the *simulation time* and the *real time*. We define the properties like lookahead and communication topology to be functions of *simulation time*, since these properties (guarantees) are part of the simulation model. However, while discussing properties of the simulation protocols, and the associated definitions (like EIT and EOT defined in section 5.1), the real time scale is used. For instance, the progress of a simulation protocol has to be shown against the real time.

3 Simultaneous Events

In this section, we discuss some of the sources of simultaneous events and zero-delay (lookahead) LP's in a simulation model. Since there may be many independent parallel activities in the physical system, it is natural for different LP's to schedule events with the same timestamp. Since an LP models an arbitrary portion of the physical system, it may have several events with the same timestamp too, corresponding to many independent parallel activities in that portion of the physical system. Even if the events are not truly parallel in the physical system, they may have the same timestamp in the simulation if the physical time scale is finer than the simulation time scale [27, 26].

Some modeling situations may need zero-delay LP's. For instance, if the time taken to process an event in the physical system is negligible as compared to the other time intervals being modeled, execution of that event may generate other events with the same timestamp[26]. Below, we give some examples of modeling situations where zero-delay LP's may be needed:

- For simulating combinational VLSI circuits, it is often quick and expedient to assume zero-time switching delay [25]. [15] gives an example similar to this: if one end of a pipe full of water is connected to a water faucet, and the faucet is turned on (incoming event), water comes out of the other end (output event) without any delay.

- A common source of zero delay LP's is the way events are defined. For example,

delivery is assumed. For the null message protocol, the delivery is also assumed to be FIFO. We also assume that an unlimited amount of memory is available at each processor to store the incoming events.

Lookahead: An LP in a model is said to have a lookahead $l(\geq 0)$ in a given state, at simulation time t, if the earliest future output that it may generate will have a timestamp greater than or equal to $t + l$. Lookahead of an LP may be a function of simulation time and its state. We distinguish between the *existence* of the lookahead for an LP, and the simulation engine *knowing* its value, and hence being able to exploit it.

A *lookahead guarantee* is a guarantee made by the simulation model regarding the (minimum) *value* of the lookahead of its LPs, and whether these values are *known* to the simulation engine. In the absence of any such guarantee, the lookahead of any LP in the model may be zero. Typical guarantees for the value of an LP may be of two types - ϵ-lookahead, or c-lookahead. The former refers to a greater than zero, but possibly arbitrarily small lookahead. The latter, which is a stronger guarantee, refers to a constant (as opposed to arbitrarily small), greater than zero lookahead. An LP that satisfies either one of these guarantees may be referred to as a *lookahead* LP in the rest of the paper. Lookahead guarantees usually specify either (at least) one LP in every cycle of LP's, or all LP's in the model, to be lookahead LP(s). In the former case, whether the identity of the lookahead LP is known to the simulation engine or not is also part of the guarantee. Similarly, in case of c-lookahead LP's, the lookahead guarantee also determines whether the current value of the lookahead is available to the simulation engine or not. Note that in case of the c-lookahead LP's, different LP's in the model may have different values for the constant, and even the same LP may have different values at different simulation times.

From the point of view of the simulation model, there may not be much difference between an ϵ (arbitrarily small) lookahead, and a c (constant) lookahead i.e. there may not be many simulation models that can satisfy the former, but not the latter. Indeed, due to limited precision of the numbers stored on the computers, the two guarantees are practically equivalent. However, from the point of view of a simulation protocol, the distinction between the two is important. Wherever, an ϵ lookahead guarantee is sufficient to ensure progress in a simulation protocol, the performance (execution time) of the simulation protocol is independent of the actual value of lookahead. However, wherever a c-lookahead is needed for progress, it is an indication that the value of the lookahead also affects the performance of the simulation protocol.

The other distinction we make is between the existence of lookahead LP's, and the simulation engine or protocol knowing their identity and/or the value of their lookahead. This distinction is made in order to distinguish the protocols that need to know the lookahead value e.g. the Null Message protocol, from the ones that do not need to know the actual lookahead value, but still rely on the existence of lookahead in order to guarantee progress e.g. Time Warp.

As an example of a lookahead LP, consider an LP that models a First Come First

event is scheduled at a unique LP, and its execution can affect the state of only that LP, in addition to possibly generating future events for that or any other LP in the simulation model. Once scheduled, an event may not be canceled by the user. There is no restriction on the number of events in the model. Indeed, there may even be infinite events generated with a given timestamp. Any event generated during simulation has two components: *timestamp*, and *content*.

A **simulation model**, or a simulation program, refers to the description of the *behavior* of each LP, the initial states for each LP, an initial set of events, and a maximum simulation time (H). The *behavior* describes how an LP processes an event that it may receive during the course of simulation. We assume that the behavior is described completely i.e. the next state and output event(s) must be defined for all possible combinations of current state and input event for each LP. The set of all possible states and input events must take into account correct as well as possibly incorrect (optimistic) executions. It is typical to define an error state as the next state for all the illegal combinations of current state and input event. The behavior of each LP is assumed to be deterministic i.e. given an initial state and an input event, it will produce a unique next state and a unique set of output events. We assume, for the purposes of this paper, that the user defined ordering rules for simultaneous events are not part of the simulation model, but are defined separately (as discussed in the next section).

Unless specifically noted, we assume a dynamic simulation model i.e. as a result of processing an event, an LP may create a new LP, or destroy itself, at some future simulation time. However, to keep proofs simple, we assume that the execution of an event at simulation time t can only result in creation of new LP's, or its own destruction, at simulation times strictly greater than t. *State*(t), which refers to the state of the logical system at simulation time t, consists of the states of all the entities alive at simulation time t. The state of an LP at simulation time t refers to the state after it has processed all its events with timestamp *less than* t, but no events with timestamp t or greater. The *instantaneous* topology of the logical system at simulation time t, *topology*(t), is a directed graph defined on all LPs alive at time t, such that there is an edge from LP A to LP B if and only if A sends an event with timestamp t to B. A *cycle* in the simulation model refers to a directed cycle in *topology(t)*. Note that the instantaneous topology may, and in most cases does, change from one simulation instant to another. Additionally, in general, the simulation engine may not *know* the instantaneous topology. As we discuss in later sections, some specific protocols may require the knowledge of the topology to make progress. This topology is typically a superset of the instantaneous topologies.

The simulation model may be executed by a sequential or parallel simulation protocol. In the former, all LP's are mapped to the same processor, whereas in the latter, each LP is mapped to a (potentially) different processor. We assume a message passing paradigm is used to implement the parallel simulation protocol, even though the actual hardware may be distributed memory or shared memory. Reliable message

sistent and unambiguous at each simulation object, yet no guarantee may be made regarding the existence of a consistent and unique overall execution. We describe sufficient conditions on the simulation model under which such a guarantee can be made.

Lookahead is the ability of a simulation object to predict its future. It is known to be an important determinant of the performance of parallel simulation protocols, especially the conservative ones. Lookahead is also widely believed to be *required* to ensure progress of parallel conservative protocols, whereas global event list and parallel optimistic protocols are often thought to be free from such a requirement.

We show that for many simulation protocols, the *weakest lookahead guarantee* required, in order to ensure progress, is closely linked to the *tie-breaking mechanism* they are required to enforce. In general, stronger the tie-breaking mechanism, stronger is the lookahead guarantee needed. For example, if user is satisfied with the *arbitrary* tie-breaking mechanism, the simulation model would need to satisfy a weaker lookahead guarantee than the guarantee that would be needed if the *user-consistent* tie-breaking mechanism is desired. We note that one particular parallel conservative protocol, namely, the conditional event protocol, does not need any lookahead guarantee in order to enforce the *arbitrary* tie-breaking mechanism, whereas the conservative null message and optimistic time warp protocols need some form of lookahead guarantees to enforce either of the tie-breaking mechanisms. We also note that the commonly used sequential simulation protocol, the global event list protocol, cannot enforce the *user-consistent* tie-breaking model without assuming additional[1] lookahead guarantees. We show that a sequential optimistic protocol is able to enforce the *user-consistent* tie-breaking mechanism without needing additional lookahead guarantees.

The rest of the paper is organized as follows: Section 2 defines the process-based simulation model that we use in the paper. Section 3 discusses the sources of simultaneous events in simulation models. Section 4 formally discusses the tie-breaking models and user defined orderings. Weakest lookahead guarantees needed by different simulation protocols are discussed in section 5. Section 6 presents related work. Finally, section 7 gives the conclusions.

2 Simulation Model and Assumptions

We adopt a process-oriented, message based simulation paradigm[7] for discrete-event simulation. For the purposes of simulation, the physical system is modeled by a set of *logical processes*(LP's), collectively referred to as the logical system. Each LP models an arbitrary portion of the physical system. Any activity in the physical system is modeled by timestamped event communication among the corresponding LP's. LP's do not share any variables, and interact with each other only through events. Each

[1] beyond what are required to ensure the *existence* of such an execution

Categories and Subject Descriptors: D.1.3 [**Programming Techniques**]: Concurrent Programming- *distributed programming, parallel programming*; I.6.1 [**Simulation and Modeling**]: Simulation Theory- *model classification*; I.6.2 [**Simulation and Modeling**]: Simulation Languages; I.6.5 [**Simulation and Modeling**]: Model Development- *modeling methodologies*; I.6.8 [**Simulation and Modeling**]: Types of Simulation- *discrete event, distributed, parallel*
General Terms: Algorithms, Languages, Theory
Additional Key Words and Phrases: simultaneous events, time ties, lookahead, simulation protocols, optimistic, conservative, null message, conditional event, time warp

1 Introduction

Discrete event simulation is a widely used technique to study the behavior of complex systems and phenomena. The system or phenomenon being studied is modeled by a set of state variables and timestamped events. Execution of an event may change the state of the system and/or schedule future events. There are several different paradigms for discrete event simulation [18]; we follow the process-oriented, message based paradigm in this paper (discussed in more detail in section 2).

A *simulation protocol* refers to a set of rules that are used to correctly execute a simulation. Discrete event simulations have traditionally been carried out using a sequential *global event list* (GEL) *protocol*. It ensures the correctness of the simulation by processing all the events generated in a global timestamp order. The advent of general purpose parallel computers has made parallel simulation, which refers to the execution of discrete event simulation program on parallel computers, a viable approach for many complex simulations. Numerous *parallel simulation protocols*, broadly classified into *conservative* and *optimistic* categories, have been proposed in the literature (see [5] and [8] for a survey). Whereas conservative protocols require that the events be executed in their timestamp order at each simulation object, optimistic protocols allow events to potentially be executed out of order, but guarantee that the final state will reflect the effect of executing the events in their timestamp order at each simulation object.

During a simulation, several events may be generated that have the same simulation timestamp. Such events are referred to as *simultaneous events*. The results of the simulation may vary depending on the order in which these events are executed. Both sequential and parallel simulation protocols have used a variety of *tie-breaking mechanisms*, sometimes ad-hoc and specific to the simulation protocol being used, to order simultaneous events. We define two types of protocol-independent tie-breaking mechanisms. An *arbitrary* tie-breaking mechanism allows *any* non-deterministic ordering to be chosen among the simultaneous events in the simulation. Alternatively, the user may want to define *ordering rules*, or use default ordering rules, to deterministically order the simultaneous events. This is called the *user-consistent* tie-breaking mechanism. We show that in certain situations, the ordering rules may be locally con-

Simultaneous Events and Lookahead in Simulation Protocols

Vikas Jha
Rajive Bagrodia
University of California at Los Angeles

Abstract

A discrete event simulation model may contain several events with the same timestamp, referred to as simultaneous events. The order of execution of such events may affect the outcome of the simulation. Simulation systems and protocols use different, often ad-hoc, tie-breaking mechanisms to order simultaneous events. As a result, it may be impossible to reproduce the results of a simulation model across different simulators. In this paper we define two tie-breaking mechanisms- *arbitrary*, which allows a non-deterministic ordering of simultaneous events, and *user-consistent* which produces a deterministic execution based on simultaneous event orderings specified by the user. The latter requires that the user defined simultaneous event ordering rules be consistent, and if followed, must result in a unique execution. We specify sufficient conditions under which such a guarantee can be made.

We show that for some simulation protocols, the ability to enforce a particular tie-breaking mechanism depends on the lookahead guarantee made by the simulation model; relatively weaker lookahead guarantees may be sufficient for enforcing the *arbitrary* tie-breaking mechanism, whereas stronger guarantees may be needed for the *user-consistent* tie-breaking mechanism. The paper presents a systematic analysis of the minimum lookahead requirements, corresponding to the two tie-breaking mechanisms, for several simulation protocols. The protocols considered include sequential as well as many parallel, both conservative and optimistic protocols.

[1]This work was partially sponsored by NSF PYI Award No. ASC-9157610, and the U.S. Department of Justice/Federal Bureau of Investigation, ARPA/CSTO under contract J-FBI-93-112.

[2]Authors' address: Computer Science Department, University of California at Los Angeles, Los Angeles, CA 90024.