INTERLEAVING SET TEMPORAL LOGIC

by

S. Katz and D. Peled

Technical Report #456

June 1987
INTERLEAVING SET TEMPORAL LOGIC

by

Shmuel Katz and Doron Peled

Department of Computer Science, Technion, Haifa, Israel.

ABSTRACT.

A new temporal logic and interpretation are suggested which have features from linear temporal logic, branching time temporal logic, and partial order temporal logic. The new logic can describe properties essential to the specification and correctness proofs of distributed algorithms such as those for global snapshots. It is also appropriate for the justification of proof rules and giving temporal semantics to properties such as layering of a program. These properties cannot be described with existing temporal logics. The semantic model of the logic is based on a set of sets of interleaving sequences which reflect partial orders from the underlying semantics of the computational model. For the common partial order derived from sequentiality in execution of each process, the logic will distinguish between nondeterminism due to the parallel execution and nondeterminism due to local nondeterministic choices. The difference in expressive power is thus qualitative, and not merely due to the presence or absence of a particular temporal operator. In the logic, theorems are proven which clarify when it is possible to establish a property $P$ for some of the interleaving computations, and yet conclude the truth of $P$ for every interleaving.
1. Introduction.

Many attempts have been made to design a logic which allows expressing specifications and proving correctness for distributed systems, and the relative expressibility of each has been studied. We deal here with a new kind of temporal logic which can express various new properties that existing logics can not.

Among the various temporal logics that have been proposed are Linear Temporal Logic (LTL) of Manna and Pnueli [MP1,MP2], and Branching Time Temporal Logic (UB) of Ben-Ari, Manna, and Pnueli [BMP] or CTL of Clarke and Emerson [CE]. There are also variants of those logics which differ by extending the temporal modalities, separating the formulas into path formulas and state formulas (CTL* of Emerson and Halpern [EH1]) or using variants of the temporal structure [E]. These logics differ from each other for example in expressive power [EH1], in the transparency of the stuttering property [L2], or in their treatment of fairness [E].

All of the above logics have in common that they are interpreted over the global states of a program, and assume a full interleaving of atomic actions, which generate sequences of global states. This is surprising in that for some years it has been realized [L1, R] that there is an "inherent" partial order among the events in an execution of a distributed program. The order is defined by the sequentiality of local events in a process, and the events of sending and later receiving a particular message. The transitive closure of these sequential relations define a partial order, and all other interleavings of events not related by the order are arbitrary. The above logics, although convenient for expressing global states and proving properties by induction, cannot express properties which follow from the underlying partial order, because that information is lost once the interleaved sequences are generated.

A temporal logic which is connected to distributed programs through an interpretation that does deal directly with partial orders, the Partial Order Temporal Logic (POTL) of Pinter and Wolper [PW], suffers from a complementary difficulty: in a 'pure' partial order approach, there is no concept of a global state. Thus many interesting properties which are global in nature, cannot be expressed at all in the language.

The Interleaving Set Temporal Logic (ISTL) to be defined below will have the benefits of both approaches: the interleaving view of computation will enable reasoning about global states, but the interleavings which come from the same partial order will be grouped together so that each such set can be considered independently when necessary.

This logic will be particularly appropriate for specifying and reasoning about distributed programs and languages which have both independently executing processes, and local nondeterministic choices within the processes. Such nondeterministic constructs are common in most languages for distributed programming, with two well-known examples being the select
construct of Ada, and the guards of CSP [H]. Among other benefits, the explicit nondeterminism allows choosing one of a number of possible communications without imposing an arbitrary ordering in advance. In the partial order that appears frequently in the literature [B, BR, L1, L5, R], it is natural to distinguish between the nondeterminism which arises because of the independent execution of events in different processes, and the nondeterminism due to the explicit choices of the nondeterministic control constructs. However, other partial orders can be considered in the framework of ISTL. For example, a partial order may be defined which takes into account only real causality between events. In such a partial order, two sequentially executed events can be considered as unordered if they do not affect each other (and their relative order is not an interesting factor in the description of the program). This partial order, which can be called the essential partial order, is weaker than the one discussed previously, but still expresses the temporal relations which are required by the underlying semantics.

The modalities of ISTL are based on the branching TL operators (for example of [BMP] or [CE]). A structure of ISTL is however different from that of UB, CTL or CTL* because in those logics the single branching structure of a program includes all the interleaved paths that are associated with any possible execution. It is common to identify a single execution with an interleaving. However, in ISTL, a single execution is identified with a single partial order among the events according to the partial order model. We build the set of all global states which stem from a partial order, and use a relation on the set of global states [R] to form a branching substructure. A structure in ISTL is a set of such branching substructures, each of which is associated with a single execution. For a formula to be satisfied by such a structure, it must be satisfied by each of its substructures taken as a stand-alone structure according to the usual branching semantics.

For each (partial order) execution there can exist several paths which belong to the same branching substructure. The abstraction of a distributed system does not allow us to prove that one path is more correct than another (we do not depend on a global clock). This is a nondeterminism that is due to the execution of unrelated events in different processes. On the other hand, paths that are different because of choosing a different continuation out of several nondeterministic choices inside a process belong to different executions. We can sometimes say that a distributed program is correct under a given specification if for each execution, there exists a path that satisfies the specification. Using the grouping into substructures and the branching modalities, we will show when it is justified to apply this correctness criterion to distributed programs.

This approach, similar to the semantic view suggested in [L1] and [R], has not previously served as the basis of a global temporal logic. Rather, an assertion in linear temporal logic is easiest understood as having a set of (nonbranching) sequences as models, each one being a
structure. This logic expresses correctness properties such as liveness and partial correctness. An LTL formula is a *path formula*. That is, it makes an assertion about an interleaved path. It has the feature of always talking about "all the executions" of a program at the same time. In other words, a formula is valid for a certain program iff it is true in each of the structures (which are interleaved paths) which its possible executions generate.

An assertion in branching temporal logic is modeled by a DAG structure, where we can express the possibility that a program will choose a given alternative. Therefore properties like "there can be an execution that finishes with X=2" may be expressed. A UB formula is a *state formula*, that is, it makes an assertion about all the possible continuations from a global state. A formula is said to be valid for a program iff it is true in every global state of the DAG structure generated by its possible executions.

Partial order temporal logic [PW] allows treating the relative precedence of local events in a single partial order execution. It is meaningless to talk in POTL about global states, since the logic reasons directly on the partial order and does not generate the global states.

ISTL inherits some of the features of each of these logics: From LTL we take the transition between global states and the property that a formula is satisfied by a structure if it is satisfied in any (partial order) execution. The branching structure is clearly from UB or CTL, as is the ability to choose between alternative continuations. From POTL, the view of a single partial order execution as a *stand-alone* (sub)structure is adopted.

By dividing the structure into such substructures and using the branching modalities we are able to express properties that the other temporal logics can not. Among the new properties that we can now consider are: the specification of the *distributed snapshot* algorithm for finding the global states of a distributed system as seen in [CL], the behavior of *communication closed layers* [EF], or giving temporal semantics for distributed languages like CSP.

The rest of this paper is organized as follows: In Section 2 a short description is given of the two models of distributed computation: as a partial order between events and as an interleaving. In Section 3 the informal description of the model of partial order is related to a structure in the sense of logic and set theory, and a formal definition is given for the syntax and semantics of the new logic. In Section 4 a correctness criterion for distributed programs is examined, and the subject of giving a specification and correctness proof for a distributed program in the new logic is discussed. In Section 5 we present some examples which motivate the new TL and in Section 6 the new logic is compared with the existing ones. A proof is shown that there are properties with respect to which this logic is more expressive. In Section 7, an alternative semantic definition of the logic is given and some extensions to the basic definition are discussed. Section 8 deals with deduction in the new framework. Section 9 gives some conclusions.
2. *The two views.*

The underlying semantics of distributed programs can be given with different variants, which clearly define different sets of possible operations. In addition to standard operations such as assignment, testing, and local control, the set of operations will depend on whether there are, e.g., a variable or a fixed number of processes (that is, fork and join operations to spawn or terminate processes are either allowed or disallowed), synchronous or asynchronous communication or communication defined only between two processors or generalized to N-way communication.

2.1. **Definition.** *Events* are single executions of atomic operations. Events are not the same as the syntactic representation of the operations [L3] which are merely patterns of events. Note that each execution of an atomic operation defines a different event. A *process* is an abstract entity which represents a task which can be executed concurrently with other tasks. The division of events to processes can be thought of as an indexing of the events: events indexed by a process identifier are said to occur inside a process (e.g. an assignment), events that are indexed by more than one process identifier occur jointly among several processes (e.g., in a model with synchronous communication when handshaking occurs between two processes).

The logic to be defined in the next section does not depend on the definition of the operations except that two semantic properties are required: The first is that the number of events in the partial order (and therefore the number of processes) is countable. The second is that no event has an infinite number of predecessors. Otherwise, we will not be able to ensure that there are paths which contain each event from a partial order.

There are two ways to look at a distributed computation for the purpose of modeling: The first sees a computation as a partial order between events, and the other as a total order (interleaving) of the events. An excellent description of those two views appears in [L1] and we give here only a short explanation which will be needed in the sequel.

2.2. **Definition.** A (local or global) *snapshot* of a program is an element of the Cartesian product of the values of the (process’ or program’s resp.) variables including program counters. A global snapshot in a model with asynchronous communication includes also for each channel the set of messages sent but not received.

A local event starts from a local snapshot of the process in which it occurs, and ends with a new local snapshot. In a synchronous communication model, a communication event is mutual to two (or more) different processes.

Among events that occur in the same process during a single execution, there exists a natural total order of occurrence. Apart from that, the event of sending a message in one process precedes the event of receiving it in another (or in a synchronous communication model, this is
done within a single joint event). Other natural orderings are defined for events such as join and fork. The transitive closure of this irreflexive relation between events is a partial order that is taken as the model of computation in the partial order approach. As noted before, the partial order that is being described here, although natural and popular among researchers, is only one possibility. The choice of a partial order to represent what we view as a computation is orthogonal to the following definitions and the logic in the next section. This choice will follow from the semantics, and we assume that it at least includes the essential temporal relations mentioned in the introduction.

Although the partial order model is mainly discussed in connection with distributed systems \( [B, L1, L5, R] \), it is also a useful tool for shared memory parallel programs \( [BR] \). One way to introduce shared variables into the execution model is by adding the following constraint: in each partial order execution, each shared memory location imposes a total order among the events which reference it (both read and write). Because of the total order among the events of accessing a shared variable, one can view each of the shared variables as a process by itself. Each of the events of this process is mutual to another process (the one that reads or writes) similarly to a synchronous communication between processes. Therefore, nondeterminism in the access to a shared variable creates alternative events that belong to different executions. Again, this is only one proposed semantics for using a shared variable. A different underlying semantics may allow, for example, concurrent reading.

2.3. Definition. Let \((E, <)\) be a partial order on a set of events \( E \) for a relation \(<\). The term a single execution is taken to be synonymous with "a partial order".

The precedence relation among events represents exactly what our abstraction of distributed systems allows us to infer about the order of events. Knowing that two unrelated (according to \((E, <)\)) events \( e_1 \) and \( e_2 \) were executed, no further conclusions may be reached about their relative order of execution as long as no global clock is given. Two such events are said to be concurrent. Whenever there is an explicit nondeterministic choice in the code of the program, a single (partial order) execution will include only the specific choice made in that execution. Thus in general there will be many partial orders associated with a program.

A global state does not exist directly in the partial order model. In order to incorporate such states, for each possible partial order \((E, <)\) the following terms \([L1]\) may be defined:

2.4. Definition. A slice \( S \) is a finite subset of \( E \) in which the following property holds: for each pair of events \( x \) and \( y \) in \( E \) such that \( x < y \), if \( y \in S \) then \( x \in S \).

2.5. Definition. A global state is identified with a slice. It is characterized by a snapshot which is an element of the Cartesian product of all the local snapshots resulting after the execution of the maximal events in \( S \) according to the relation \(<\). (A maximal event \( x \in S \) is one that for no
y \in S \text{ is it true that } x < y.) In addition, in a nonsynchronous communication model, for every communication channel there is an associated set of messages sent on it by events from S and received by events outside S.

This is a global state in a sense that without a global clock no one can disprove the claim that the program really passed through a synchronous interval of time in which exactly all the events in the slice have already executed and the values of all the variables (and set of messages in transit) agree with the related snapshot. Let $\Delta$ be the set of all slices defined on $(E, \prec)$.

It is possible that two or more different global states of a program have exactly the same snapshots in a single program execution. For example, a loop inside a program may cause the set of variables (propositions, sets of messages) to repeatedly have the same values during a single execution. However, different global states occur because the set of events accumulated into the appropriate slices is extended each time events are executed. Thus different slices may be characterized by the same snapshot.

2.6. Definition. Let $\rho \subseteq (\Delta \times \Delta)$ be a relation such that $s \rho t$ iff $s \subset t$. That is, $s$ has fewer events than $t$ and can be said to precede $t$. Note that $s$ and $t$ must be finite by the definition of $\Delta$.

2.7. Definition. Let $R \subseteq \rho$ be a relation such that $s R t$ iff $s \rho t$ and there is no $r$ such that $s R r$ and $r R t$. That is, two global states $s$ and $t$ are related by $R$ if $t$ is different from $s$ by the execution of a single additional event (by a single process, or by several if it is a synchronous event) $[R]$. If $s R t$, $s$ is said to immediately precede $t$. The connection $\rho = R^*$ holds between $R$ and $\rho$.

The relation $R$ between global states generates a branching structure (DAG) because for a single state, we generally have more than one successor. The alternative possible successors are caused by the occurrence of unrelated (according to the partial order) events in different processes and not by nondeterminism inside a process.

According to the interleaving view, we look at sequences of events. Each sequence is a total order between the (beginning of the) events. Therefore, between each of the events in a sequence, even in different processes, there exists an order. Each event starts from a global state of the program and ends with a new global state.

2.8. Definition. A single maximal sequence of global snapshots is called an **interleaving sequence** or a **path**. (Maximality of a sequence means that it is not a proper prefix of another sequence). A finite and contiguous portion of such a path will be called in the sequel a **finite path**.

The interleaving model can describe an idealization of a reality in which there is a global clock and the events are "timeless", or an implementation of processes using multiprogramming, with no actual parallel execution. Note that if events are not instantaneous and can overlap in time, then none of the interleavings need describe what "actually" occurred. The model is
justified by the assertion that the program will behave "as if" one (or more) of the interleavings represents reality.

It can be easily seen \([L1, R]\) that the set of paths generated by \(R\) is exactly the set of interleaved sequences which are related to a single (partial order) execution. Note that such a relation \(R\) is constructed separately for each partial order execution.

An interesting property of the relation \(P\) that will be used later is:

2.9. Proposition. If \(x P y\) (thus \(x \preceq y\) and \(x\) and \(y\) are finite) then there is a finite path from \(x\) to \(y\).

Proof. consider \(x\) and \(y\) as sets of events, then by acyclicity of the partial order (since it is transitive closed and irreflexive) there is a minimal event \(e\) in \(y - x\). Since all its predecessors are included in \(x\), \(x' = x \cup \{e\}\) is a slice. The rest of the proof follows by a simple induction on \(|y - x|\) which is obviously finite. \(\square\)

3. Towards the new TL.

In this section, a formal definition of the syntax and the semantics of ISTL is given. Since ISTL is aimed towards describing distributed systems with a partial order semantics, it will be introduced in two levels: First, a general structure for ISTL is defined as a set of sets of sequences of states. Each set of sequences contains sequences with some very basic semantic restrictions. The syntax and the semantics of ISTL is defined similarly to the branching semantics of CTL, except that the set of sets of sequences is introduced instead of a single set of sequences. Then, in a second level the set of structures is limited to those that are built from a given set of partial orders. A semantic constraint on the sequences generated from partial orders is defined.

In the semantics of the new logic, a framework is established which would allow introducing a new pair of modal operators: one to express that a formula should be true for all of the sets of sequences and another for the existence of a set of sequences which satisfies a formula. However, in the logic below a simpler "linear" version is given, in which a universal quantification over all of the sets of sequences is always assumed, and therefore need not be explicitly written. An extension which introduces such quantifiers over the set of sets of sequences is defined in Section 7.

For simplicity, we define a propositional ISTL, where the extension to first order is standard.

3.1. Syntax.

Define: \(P\) - A set of atomic propositions.

1. For each proposition \(p \in P\), \(p\) is in ISTL.
2. if Q, W are in ISTL then Q \land W, Q \lor W and \neg Q are in ISTL.
3. if Q is in ISTL then AGQ, AFQ, EGQ, EFQ, AXQ and EXQ are in ISTL.
4. if Q and W are in ISTL then E(QUW), A(QUW) are in ISTL.

We call the following symbols sequence modalities: G='always', F='sometimes', X='next' and $U$='until'.

The sequence quantifiers are: A='on every sequence', E='there exists a sequence'.

Define : first(x) - The first state in a sequence x.

$X^i$ - The suffix of the sequence x starting from its $i^{th}$ state ($x$ begins with $s_0$).

3.2. Definition. A set of sequences is suffix closed [A, E] if every suffix of a sequence which appears in the set is also in the set.

3.3. Definition. A set of sequences is fusion closed if whenever $x_1 s y_1$ and $x_2 s y_2$ are sequences in the set (where $x_i$ is a prefix of a sequence, $s$ is a state and $y_i$ is a suffix of a sequence) then $x_1 s y_2$ is also a sequence in the set.

3.4. Definition. A structure M has the form (S,I,L) where S is a set of states, L is an assignment function $L:S \times P \rightarrow \{\text{true, false}\}$, and I is a set of sets of sequences of states, where every $H \in I$ is a non-empty set of sequences which is both suffix and fusion closed. (A structure which allows the sequence quantifiers to range over a semantically defined set of sequences which is suffix and fusion closed is called an Abrahamson structure [A, CVW].) We will write $S[H]$ for all the states from S that appear in at least one of the sequences in H.

Now the satisfaction relation will be defined:

3.5. Semantics. A structure $M=(S,I,L)$ with a set of sequences $H$ in I and a state $s \in S[H]$ satisfies a formula $f$, (written $M,H,s|= f$) iff:

1. $M,H,s|= p \text{ for } p \in P \text{ iff } L(s,p)=\text{true}.$
2a. $M,H,s|= Q \land W \text{ iff } M,H,s|= Q \text{ and } M,H,s|= W.$
2b. $M,H,s|= Q \lor W \text{ iff } M,H,s|= Q \text{ or } M,H,s|= W.$
2c. $M,H,s|= \neg Q \text{ iff not } M,H,s|= Q.$
3a. $M,H,s|= AGQ \text{ iff for each sequence } x \in H \text{ with } \text{first}(x)=s, \text{ for each } i \geq 0, M,H,\text{first}(x^i)=Q.$
3b. $M,H,s|= AFQ \text{ iff for each sequence } x \in H \text{ with } \text{first}(x)=s, \text{ there exists } i \geq 0 \text{ such that } M,H,\text{first}(x^i)=Q.$
3c. $M,H,s|= EGQ \text{ iff there exists a sequence } x \in H \text{ with } \text{first}(x)=s \text{ and for each } i \geq 0, M,H,\text{first}(x^i)=Q.$
3d. $M,H,s|= EFQ \text{ iff there exists a sequence } x \in H \text{ with } \text{first}(x)=s \text{ and there exists } i \geq 0 \text{ such that } M,H,\text{first}(x^i)=Q.$
3e. \( M, H, s \models \text{EXQ} \) iff there exists a sequence \( x \in H \) with \( \text{first}(x) = s \) and \( M, H, \text{first}(x^{-1}) \models Q \).

3f. \( M, H, s \models \text{AXQ} \) iff for each sequence \( x \in H \) with \( \text{first}(x) = s \), \( M, H, \text{first}(x^{-1}) \models Q \).

4a. \( M, H, s \models \text{E(QUW)} \) iff there exists a sequence \( x \in H \) with \( \text{first}(x) = s \) and there exists \( i \geq 0 \) such that \( M, H, \text{first}(x^{-1}) \models W \) and for each \( 0 \leq j < i \), \( M, H, \text{first}(x^{-1}) \models Q \).

4b. \( M, H, s \models \text{A(QUW)} \) iff for each sequence \( x \in H \) with \( \text{first}(x) = s \) there exists \( i \geq 0 \) such that \( M, H, \text{first}(x^{-1}) \models W \) and for each \( 0 \leq j < i \), \( M, H, \text{first}(x^{-1}) \models Q \).

3.6. Definition. \( M, H \models f \) iff \( M, H, s \models f \) for each \( s \in S[H] \). (That is, a set of sequences \( H \) satisfies a formula, if each of its states satisfies it. Many typical formulas will be trivially true for all but initial states by adding \( \text{at}(\text{START}_i) \) as a conjunct for each process \( i \) to the left of an implication.)

3.7. Definition. \( M \models f \) iff \( M, H \models f \) for each \( H \) in \( I \).

The semantical implication \( \Sigma \models f \), where \( \Sigma \) is a set of ISTL formulas, \( f \) is a single ISTL formula and \( M \) is an ISTL structure, is as follows:

3.8. Definition. \( \Sigma \models f \) iff for every \( M \), whenever \( M \models \Phi \) for each \( \Phi \in \Sigma \), then \( M \models f \).

We do not need all sequence modalities \( G, F, U \) and \( X \) as there is a simple translation which uses semantical equivalences and shows that one needs only \( U \) and \( X \) [CE].

Above, a general framework for dealing with a set of sets of sequences was defined. We now focus our attention on the set of structures which are generated from the partial order model using the previously defined construction of the relation between global states \( R \) (Definition 2.7). \( R \) was constructed from a particular partial order \( (E, <) \) and thus corresponds to a single partial order execution. The sequences we consider will now be paths, and the sets of sequences will be called interleaving sets.

3.9. Definition. \( \text{PATHS}(R) = \{ (s_0, s_1, s_2, \ldots) : \forall i \geq 0 s_j R s_{i+1} \} \).

3.10. Definition. An \textit{acceptable} path \( x \) satisfies the condition that for each event \( e \in E \) (that is, an event that was executed according to the partial order semantics) there exists a global state on \( x \) which (as a slice) contains \( e \). This means that any path includes all the events from the partial order from which it was generated and may not ignore the next event of a process forever.

The criterion of being acceptable (which is called \textit{just} in [R]) differs from the usual fairness definitions [LPS] [Fr] in that the partial orders are already assumed given from the underlying semantics, and these already may or may not be "fair" according to the other definitions.

3.11. Definition. Let \textit{acceptable}(\text{PATHS}(R)) be the \textit{acceptable} paths from \text{PATHS}(R).

The following proposition appears without a proof in [R]:

3.12. Proposition. The set of acceptable paths is not empty.
Proof. We first show that any finite path $p$ can be always extended to include an event $e \in p$. Consider the finite set $S$ of those events which precede $e$ according to the partial order. This set is a slice by Definition 2.4. Consider the union of $S$ with the last slice in the path $p$. A union of two slices is a slice, and by Proposition 2.9 there is a finite path between the last slice in the path $p$ and the union. This path can then be further extended to include $e$.

Now, since the set of events is enumerable by definition, consider the path which is generated by always extending it in the following way: given any finite prefix, choose the next smallest event (according to some enumeration) which is not yet in any slice in the current prefix. Then extend the last slice in the prefix to contain this event. This path will be acceptable.

3.13. Definition. An interleaving set is a set of paths formed from $\text{acceptable} (\text{PATHS}(R))$ where each acceptable finite path (that is, a path in which its last global state has no successor under the relation $R$) is made infinite by repeating its last state.

3.14. Proposition. The set of paths that constitute an interleaving set is suffix closed and fusion closed.

Proof. Follows easily from the fact that if a path is acceptable, then all of its suffixes are also acceptable.

Since we focus our attention on a special interpretation for branching temporal logic, namely "global transitions caused by the partial order model of execution", the specific semantic property of acceptability is built into the logic. A discussion of an alternative definition appears in Section 7.2. Under the restricted interpretation indicated above, a structure will contain a set of interleaving sets, one for each possible partial order of a single program.

The assignment function $L$ assigns values to any propositional variable in any state (which is a slice, since we are restricting our attention to interleaving sets). In the case of first order logic, the assignment function assigns values to variables. It is common to be interested only in the snapshot which characterizes each slice (Definition 2.5). Nevertheless, the definition of $L$ is general and allows a predicate to be dependent upon the entire slice (for example, a predicate which is true when the number of events in a slice is even).

By Definition 3.7, a structure $M$ will satisfy a formula $f$, if each one of the interleaving sets in $M$, taken as an independent branching structure, satisfies $f$. Therefore, a formula here will be considered valid in a structure if it is satisfied by each of the executions. This is a property of "always talking about all the executions", similar to LTL, and opposed to UB which can talk about the existence of an execution. As mentioned above, in Section 7 we discuss a possible extension which makes it possible to express the existence of a partial order execution.

3.15. Example. Let $M$ be a structure of all the interleaving sets obtained from the execution of a distributed program $PR=[P1 || P2]$. Assume that $Q$ is a proposition on global states. The
propositions at(START_1) and at(START_2) have the meaning that control at P1 and P2 respectively is before the beginning of the process. (That is, according to the usual interpretation, the global initial state). Let \( f \) be the formula \((at(START_1) \land at(START_2)) \rightarrow EFQ\).

The meaning of \( M \models f \) is: For every interleaving set \( H \) (that is, for every partial order execution), for every state \( s \) appearing on some path of \( H \), it holds that \( M,H,s \models f \). According to definition 3.5, \( M,H,s \models f \) means: if \( s \) is a state that satisfies \((at(START_1) \land at(START_2))\) then there exists a path in \( H \) starting with \( s \) on which there is a state which satisfies \( Q \). Thus, the meaning of \( M \models f \) is "for every partial order execution, there exists a path from the global initial state that reaches a state which satisfies \( Q \)."

The expressive power of ISTL depends on the set of modalities chosen. Restricting the set of modalities from the previous definition to various subsets generates a hierarchy of logics similar to that in \([EH1]\). We may define ISTL* with the CTL* modalities. The formal definition is given in the Appendix. The difference between the logics is that ISTL forces a path quantifiers A and E to be followed by exactly one of the path modalities F, G, X and U (which are the same as the LTL modalities \( \diamond, \Box, \mathcal{O} \) and Until) while in ISTL* no such restriction is required.


One of the features of the new TL is in expressing properties which make use of the ability to choose one path out of all the paths that correspond to a single execution. Recall that there is no way to externally determine which of these paths, if any, actually occurred, since a global clock is not used. Thus, we are free to choose one that satisfies our specification. A criterion for distributed correctness under a temporal specification might therefore be the existence of a path in every execution which satisfies the desired property. For some properties, for example the total correctness of a distributed program, this criterion is sufficient as will be proven in the sequel. For other purposes, the use of such an approach depends on the way the program is abstracted.

The classical approach to giving a specification to parallel programs is: Because the relative speed of machine instructions is not part of the specification, a program can safely be trusted to satisfy its temporal specification if every interleaving of the concurrent events satisfies the specification. This is natural for LTL assertions which are valid when they are satisfied by all the paths possible under any execution. In the context of ISTL, paths that belong to the same partial order execution are grouped together into a substructure. This suggests investigating when properties of an "existential" kind are natural for distributed programs.

The following theorem can be used in concluding that a property is satisfied by all the (acceptable) paths of an execution of a given program even though the property is proven only for a single path of that execution.
4.1. **Theorem.** The following formula is a semantic implication both in ISTL and in ISTL⁺:

\[ p \rightarrow \text{AG} p \vdash (\text{EF} p \rightarrow \text{AF} p). \]

**Proof.** For each ISTL structure \( M \), assume that for each state in each interleaving set it holds that \( p \rightarrow \text{AG} p \) (Such a property \( p \) is called a *stable* property [CL][B]). Choose an arbitrary interleaving set \( H \) and a state \( s \in S[H] \) such that \( M,H,s \vdash \text{EF} p \) holds. Then, one semantically concludes that \( M,H,s \vdash \text{EF}(\text{AG} p) \). Given that "there is a path (call it \( p_I \)) in \( H \), starting from \( s \), on which there exists a state (call it \( t \)) such that any descendant state of \( t \) satisfies \( p \)", it must be proven that "for each path in \( H \), starting from \( s \), there is a state satisfying \( p \)". Take any path \( p_2 \) in \( H \) which also starts at \( s \). Since the set of paths satisfy the acceptability property from Definition 3.10, it follows that there is a global state \( r \) on \( p_2 \) which (when considered as a slice) contains all the events of \( t \) (in addition it may contain some more events). From Proposition 2.9 it follows that \( r \) is a descendant of \( t \). Since by the stability of \( p \), any descendant of \( t \) satisfies \( p \), then \( r \) satisfies \( p \). That is, \( M,H,s \vdash \text{AF} p \). \( \square \)

Example 5.1 of the next section demonstrates how a temporal deductive system (similar to [AFR]) uses the property proven in Theorem 4.1. A proof rule which is based on proving a total correctness property for one path is used to conclude total correctness for all the paths.

The following definition and proposition are helpful when dealing with properties about a subset of the program variables which are local to a certain process:

4.2. **Definition.** A set \( \text{LPF} \) of *linear path formulas* is a subset of the ISTL⁺ formulas defined as:

1. For each atomic proposition \( p \in P \), \( p \in \text{LPF} \).
2. If \( Q \in \text{LPF} \) then \( \mathbf{G} Q, \mathbf{F} Q, \mathbf{X} Q, \neg Q \in \text{LPF} \).
3. If \( Q, W \in \text{LPF} \) then \( Q \land W, Q \lor W, Q U W \in \text{LPF} \).

It is easy to see that the set of formulas \( \text{LPF} \) corresponds to the set of LTL formulas [EH1] if we change every \( \mathbf{G} \) into \( \square \), \( \mathbf{F} \) into \( \diamond \) and \( \mathbf{X} \) into \( \bigcirc \). The meaning is also preserved under the semantic definition of ISTL⁺ when we interpret such a formula over a single path.

4.3. **Proposition.** For any linear path formula \( L \in \text{LPF} \), if (a) \( L \) does not contain the \( \mathbf{X} \) (NEXT-TIME) modal and (b) all of the events which can affect the atomic propositions (or variables in a first order extension) of \( L \) are totally ordered (for example, events that occur in the same process or events that change the value of a single physical location), then \( \mathbf{EL} \rightarrow \text{AL} \).

**Proof.** With respect to propositions or variables that can be changed only by a subset of the events which are totally ordered, the different paths in the interleaving set are identical except for *stuttering* [L2] (that is, the repeating of identical states a finite number of times). The absence of the NEXTTIME operator in the formula \( L \) prevents forming a path formula that is satisfied by one such path and not by all others. \( \square \)
In order to exploit this property, notice that in every interleaving set those events local to the same process are totally ordered. Therefore, by using the program's text to check that a set of variables is local to a process and making an assertion $L$ only about them, one may conclude $AL$ by proving that $EL$ holds.

The following discussion is less formal than 4.1 and 4.3 because it deals with alternative ways of giving specifications. Let us compare two approaches to correctness: one that says that any property must be tested for all the possible interleaving sets of any execution (this approach is taken in LTL) and the approach that it is sufficient to show that a property is true for at least one path from each execution. Assume that the validity of $EL$ holds in a structure representing a certain program, but $\neg AL$ holds too. That is, there is a different path that does not satisfy $L$ and it is true that $EL \land \neg AL$ (i.e., using elementary equivalence $\neg(EL \rightarrow AL)$). If someone can infer that $\neg(EL \rightarrow AL)$, he must know the relative order between unrelated events. This means that he has in some sense a global clock.

The reason for such a situation is usually due to the under-specification of the system by omitting the description of the observer's events from the system's description. Such an observer can be for example a shared variable, a shared resource, or a human who can watch two printers of two different processes. The observer then imposes a total ordering upon the events that compose the two different paths - the one that satisfies $L$ and the one that does not. When the observer is made a part of the system, two such paths belong to different executions. Then the undesirable one is ruled out by the implied universal quantification over all executions rather than by the local universal quantifier $A$ over paths in a single interleaving set.

The above discussion shows that the specification of a distributed system is very sensitive to the inclusion or absence of external observers from the model. Their inclusion imposes an order between events that were unordered before. We call a system in which all possible observers are included by means of describing their interface events a completely abstracted system. (Lamport [L3, L4] discusses the issue of "interface specification" as a method to force a specification to be correct under each possible interaction with the user.) A criterion for such a distributed system to satisfy a temporal specification which is stated in terms of an interleaved sequence is that for every execution there exists a path that satisfies it.

The notion of "linearization" [D, HW, Pr] plays an important role in reasoning about parallel and distributed algorithms. Linearization means completing the partial order to a total order which contains it. In reasoning about parallel and distributed algorithms it is common to take each of the sequences (which represent computations in the interleaving model) and interchange unrelated events [D, HW, L6]. A similar argument appears when using what is called "logical variables" in a program that has shared variables. Here, an assignment to local logical variables is sometimes said to be attached to the preceding or successive event [HS]. In such situations, a
program may be specified by a (relatively weak) formula Ef for a linear path formula f, even when Af does not follow logically from it.

4.4. Example. Assume two users $P_1$ and $P_2$ are using a shared database, and for consistency, it is required that the transactions they make are non-overlapping. The safety (mutual exclusion) property

$$(\text{at}(\text{START }_1) \land \text{at}(\text{START }_2)) \rightarrow \Box \neg (\text{in}(CS_1) \land \text{in}(CS_2))$$

is equivalent to the ISTL* property

$$A[(\text{at}(\text{START }_1) \land \text{at}(\text{START }_2)) \rightarrow G \neg (\text{in}(CS_1) \land \text{in}(CS_2))]$$

But this enforces a strict requirement which is needed only when the database is not part of the system itself and acts as an outside observer. If we embed the database as a process in the system to which different processes may refer by read and write communication commands, we may use the weaker property

$$E[(\text{at}(\text{START }_1) \land \text{at}(\text{START }_2)) \rightarrow G \neg (\text{in}(CS_1) \land \text{in}(CS_2))]$$

which says that it is sufficient to linearize the events in such a way that they behave as if mutual exclusion occurs in each execution. This may be useful in case some transactions or part of them can overlap in time without both causing concurrent change to the database. If indeed the entire critical section uses the database and the database is abstracted as a sequence of events then EL and AL are equivalent requirements by Proposition 4.3 since the executions of the critical sections are totally ordered.

This does not prevent using mixed assertions with both existential and universal sequence quantifiers on a distributed system if one wishes to, as shown in the next example.

4.5. Example. In a recent paper on properties that a fairness definition must fulfill [AFK], one of the requirements is that if one of the paths in a single partial order computation satisfies the fairness condition then so do the rest of the paths in that execution, and if one does not, then none of the paths satisfies the fairness condition. This is equivalent to the assertion $(Ef \rightarrow Af)$ where $f$ is the fairness constraint. This requirement is reasonable since if a fairness constraint is used in a termination proof to rule out certain executions, it is not possible that a single partial order execution is "partially ruled out".

Comment: An ISTL* formula of the form $E(p \rightarrow q)$ where $p$ is a state formula (contains no path modalities or path quantifiers) and $q$ is any ISTL formula can be easily translated into an ISTL equivalent formula $p \rightarrow Eq$. This applies to the first three examples of the next Section where ISTL is sufficient.
5. Applications of the logic.

In Section 4 we formalized the justification for an approach to correctness which says that "a program is correct under a temporal specification" if there exists at least one path for each execution that satisfies the specification. It will be shown through several examples that such an interpretation is useful and the new logic provides a convenient specification tool:

5.1. Temporal semantics for a distributed language. The Hoare-like axioms and rules of inference for CSP [AFR] have an interesting approach to proving partial correctness. We may give TL axioms and rules which will be similar. (The rule here deals with total correctness instead of partial correctness. There is no problem in defining a partial correctness version for temporal logic). We can see that the rules in [AFR] choose a distinguished path in which the events occur in a convenient order. Consider the rule (which is somewhat simplified, but equivalent with respect to soundness and completeness of the resulting deductive system, as noted in [AFR]):

\[
\{p\} \text{all a'}(t_1), \{t_1\} S_1(t_2), \{t_2\} S_2(q)
\]

\[
\{p\}(a;S_1) || (a';S_2)(q)
\]

in which \(a\) and \(a'\) are a pair of communication commands, and \(S_1\) and \(S_2\) are non-communication program segments. We call \((a;S_1)\) and \((a';S_2)\) bracketed sections because as a consequence of this rule of inference they behave as a single atomic event.

The rule states that we can choose a path in which we first execute the communication event \(a\) then \(S_1\) in the first process, and then \(S_2\) in the second. If the antecedents of the rule can be proven for that situation, the conclusion will hold no matter which interleaving (if any) actually occurred. We can look at it as if we "bend" time according to our needs and use the fact that no one can tell us we are wrong without any evidence (the missing global clock!).

In terms of ISTL the analogous consequence rule depends on the syntax of the verified program which must satisfy the above conditions on \(a\), \(a'\), \(S_1\) and \(S_2\). Section 8 discusses the various parts of deductive systems.

\[
(at(a) \land \text{after}(a' \land t_1)) \rightarrow \text{EF(at}(a') \land \text{after}(a' \land t_1)\}
\]

\[
(* \{p\} \text{all a'}(t_1) */
\]

\[
(at(S_1) \land \text{after}(S_2) \land t_1) \rightarrow \text{EF(at}(S_1) \land \text{after}(S_2) \land t_2),
\]

\[
(* \{t_1\} S_1(t_2) */
\]

\[
(at(S_1) \land \text{after}(S_2) \land t_2) \rightarrow \text{EF(at}(S_1) \land \text{after}(S_2) \land q)
\]

\[
(* \{t_2\} S_2(q) */
\]

\[
(at(a) \land \text{at}(a') \land p) \rightarrow \text{EF(at}(a') \land \text{after}(a' \land t_1)\}
\]

\[
(* \{p\} a' (t_1) */
\]

Suppose that using the rule we have proven a property of the form

\[(at(P_1) \land \ldots \land \text{at}(P_n) \land \Phi) \rightarrow \text{EF(at}(P_1) \land \ldots \land \text{after}(P_n) \land \Psi).\]

Observe that the property
after($P_1$)\&\&...\&\&after($P_n$)\&\&$\Psi$

is stable because a terminating global state is stable. That is,

\[(after(P_1)\&\&...\&\&after(P_n)\&\&$\Psi$) \rightarrow AG(after(P_1)\&\&...\&\&after(P_n)\&\&$\Psi$)\]

Using Theorem 4.1 it follows that

\[(at(P_1)\&\&...\&\&at(P_n)\&\&$\Phi$) \rightarrow AF(after(P_1)\&\&...\&\&after(P_n)\&\&$\Psi$)).\]

5.2. The decomposition of programs into layers [EF]: Instead of decomposing programs into processes, we can decompose programs into communication-closed layers: A single layer is made of several program segments, each one in a different process, which communicate with each other in order to complete a common task. A (distributed) program may be composed of several layers executed sequentially - first the code in each process for the first layer, then the code for the second, etc.

If there are no communications across layer boundaries (i.e., from a segment of one layer in one process to a segment of another in a different process), then the program executes as if there is a synchronization at the time that each layer begins. This is a pleasant property, since a process can begin executing a new layer while the other processes are still in the first layer, and the program will behave as if the processes all began executing the new layer together.

Looking at properties that all the paths satisfy does not seem to give us any insight into how to reason about this phenomenon. However if we look at the partial order model, we can easily prove the following:

5.2.1. Proposition. In any terminating execution of a layered program, there exists a slice $S$ which defines a global state $G$ in which all the processes have finished the first layer, and are about to enter the second one.

Proof. Take the set of events $S$ that are executed according to the code of the first layer. We have to show that these events form a slice. It is sufficient to show that there cannot be two adjacent events $e_1 < e_2$ such that $e_1 \in S$ and $e_2 \in S$. By the choice of $S$, it is evident that such two events cannot belong to the same process, and because there is no communication across layer boundaries, two such events are ruled out. \(\square\)

This means, by the previously mentioned connection between the partial order model and the interleaving model, that for each one of the executions, there exists a path which behaves as if there was a synchronization. Because we cannot say which of the interleavings satisfying the same partial order "really occurred" without a global clock, this distinguished path is as good as any other.

A layered program $PR=\{S_1; Q_1||S_2; Q_2\}$ will satisfy the ISTL formula:

\[(at(S_1) \& at(S_2)) \rightarrow [EF(at(Q_1) \& at(Q_2)) \lor AG(\neg after(S_1)) \lor AG(\neg after(S_2))]\]
There are two layers $S=[S1\|S2]$ and $Q=[Q1\|Q2]$. We start at labels $S1$ and $S2$; that is, at the beginning of the program. Then in each execution which eventually completes the two segments $S1$ and $S2$, there is a corresponding path with a global state in which we are exactly before entering both $Q1$ and $Q2$. Formally, the expression means that for each (partial order) execution, from a state that starts when the processes are at the beginning of the code, there is an interleaved path constructed from the partial order such that the layers do synchronize.

Here again, as in the previous example, one may use Theorem 4.1 to prove total correctness for a layered program by showing the correctness only for those paths with a state that synchronizes the beginnings of the layers.

5.3. The Chandy and Lamport snapshot algorithm [CL]. After the execution of the superimposition of this algorithm on another (basic) algorithm, a global state of the combined superimposed algorithm is recorded. The recorded global state is used for detection of properties which are stable. The global state recorded does not necessarily appear on every interleaved path defined by the program. The important property is that for each (partial order) execution there exists a path that contains the global state which is eventually recorded.

This aspect of the correctness of the snapshot algorithm can be stated:

$$(at(START_1)\land at(START_2)\land ... \land at(START_n) \land EF(finished\land rf)) \rightarrow EFf$$

where $f$ is a formula describing some global property, and $rf$ is a formula that says "property $f$ is recorded". The predicate ‘finished’ is true when the snapshot part has been completed. In terms of ISTL this means: For every (partial order) execution, if there is an interleaving sequence which reaches a state in which the snapshot part is completed and $rf$ is true, then for the same execution there exists an interleaved sequence with a state in which $f$ actually was true.

Other aspects of the correctness of the snapshot algorithm [B] include meta-theorems on the correspondence between the ISTL structures which correspond to the set of executions before and after the superimposition of the snapshot part. Claims about the connections between the set of formulas that the two structures satisfy must also be considered.

From the correctness claim of the snapshot algorithm and the fact that the property that was detected is a stable property (that is, $f \rightarrow AGf$) one might like to deduce that if the property was detected, then on any path in the set that represents the same execution, eventually $f$ will hold forever. From the correctness condition, the stability of $f$ and Theorem 4.1, we may deduce

$$(at(START_1)\land at(START_2)\land ... \land at(START_n) \land EF(finished\land rf)) \rightarrow AFf.$$ 

Again using stability

$$(at(START_1)\land at(START_2)\land ... \land at(START_n) \land EF(finished\land rf)) \rightarrow AF(AGf).$$

5.4. Concurrency. It is convenient that the logic can express the potential concurrency of independent events or operations. These can then be executed on independent processors,
potentially increasing the efficiency of execution. In order to express that the two operations \( e_1 \) and \( e_2 \) which are not communicating operations can run concurrently the following assertion may be used:

\[
(at(START_1) \land at(START_2)) \rightarrow EF[(at(e_1) \land at(e_2)) \land (EX(at(e_1) \land after(e_2)) \land EX(at(e_2) \land after(e_1)))]
\]

In this specification, it is assumed that \( e_1 \) and \( e_2 \) are executed only once during the program, otherwise, the above formula asserts that there is an occurrence of \( e_1 \) and \( e_2 \) which is potentially concurrent. Thus we identify the operations \( e_1 \) and \( e_2 \) with events \( e_1 \) and \( e_2 \). The interpretation of the above formula is explained by the following proposition:

5.4.1. **Proposition.** In terms of slices, \( e_1 \) and \( e_2 \) are concurrent (unrelated by the partial order) iff there exist three slices \( S, S_1 \) and \( S_2 \) such that \( e_1, e_2 \notin S, S_1 = S \cup \{e_1\} \) and \( S_2 = S \cup \{e_2\} \).

**Proof.** (\( \Rightarrow \)) Let \( S_1^* \) be the set of events preceding \( e_1 \) and \( S_2^* \) the set of events preceding \( e_2 \). From Definition 2.4 it follows that \( S_1^* \) and \( S_2^* \) are slices. The union of two slices is a slice. Since \( e_1 \) does not precede \( e_2 \), \( e_1 \notin S_2^* \) and similarly \( e_2 \notin S_1^* \). Therefore, assign

\[
S = S_1^* \cup S_2^*; S_1 = S \cup \{e_1\}; S_2 = S \cup \{e_2\}
\]

(\( \Leftarrow \)) Given \( S, S_1 \) and \( S_2 \) as above, without loss of generality, assume to the contrary of the proposition that \( e_1 \) precedes \( e_2 \). Then according to Definition 2.4, \( S_2 \) (which does not include \( e_1 \)) cannot be a slice. \( \square \)

In the above example we interpret \( e_1 \) as the operation that is executed from the point that \( at(e_1) \) is true until \( after(e_1) \) and similarly for \( e_2 \). \( S \) satisfies \( at(e_1) \land at(e_2) \). \( S_2 \) satisfies \( at(e_1) \land after(e_2) \) and \( S_1 \) satisfies \( at(e_2) \land after(e_1) \).

### 6. Connection with other logics

Some connections between variants of ISTL and other temporal logics are shown in this section. It is clear that grouping the paths into interleaving sets allows us to assert new properties which are based on the underlying partial order. A proof is given which formally shows this.

6.1 **Theorem:** There are properties of programs which can be expressed in ISTL and not in POTL, LTL, UB, CTL or CTL*.

**Proof.** Choose a global property \( Q \) and two programs. In the first program, in each (partial order) execution there is a global state that satisfies \( Q \), while in the second, there are executions in which \( Q \) holds at some state and others where \( Q \) does not hold in any state. For example, let \( Q \) be the property that the value of the variable \( x \) is greater than the value of variable \( y \). Now define the global property \( S \) that says "the program is at the initial global state and the values of
variables $x$ and $y$ are zero". (In first order logic $Q = "x>y\)”, and $S = "\text{at(START)} \land x=0 \land y=0\).) The ISTL formula $P$ defined as $S \rightarrow EFQ$ expresses the property: "In every (partial order) execution of a given program starting with a global state in which $x$ and $y$ are zero, there exists a path that reaches the global state $Q$". It is clear that $P$ distinguishes among structures that belong to programs in which $P$ holds and structures of programs in which $P$ does not hold.

Since POTL is constrained to talk about local states, it is obvious that no formula equivalent to $P$ is expressible in POTL. To prove that there is no formula $F$ that expresses the property $P$ in the other logics, the two programs PR1 and PR2 in Figure 1 are presented.

The two programs are similar in structure (having the same number of processes and the same variables in each process), have the same branching-time lattice structure, but PR1 contains no nondeterminism inside the code of each process, while PR2 is similar to the common translation of a distributed program into a nondeterministic sequential one, having local nondeterminism in one process, but no nondeterminism caused because of the partial order. (The partial order taken here is the usual one, appearing in [B, L1, L5, R].)

The branching structures for PR1 and PR2 can be generated for example by taking all the possible transitions between global states in the operational semantics of Plotkin [Pl]. In this or similar semantics for CSP the local nondeterministic choice is made within a state in which the program control is at the left side of all the arrows [Pl, R]. Therefore, to create an identical structure, an extra skip is needed in PR1. The communication event is considered to be the same as assignment of the value which is sent to the variable in the receiving process [H].

PR1 and PR2 have identical branching (and therefore also linear) structures and identical values of $S$ and $Q$ at each state, as depicted in Figure 2. (For a first order logic, instead of $S$ and $Q$, the structures must be identical with respect to all the subformulas and subterms which appear in $S$ and $Q$.) However, in PR1, $P$ holds, while in PR2, $P$ does not hold. Thus, since $F$ can only express properties of the (identical) structures, no formula $F$ to express $P$ exists in those logics.

---

PR1::skip;[x:=1;yy:=1]

PR2::[P1;true→x:=1;P2!1
   □
   true→P2!1;xx:=1]
   ||P2::P1?y]

Fig. 1 Two programs with similar branching structures.
Fig. 2 The branching structure of PR1 and PR2.
(Which also defines the ISTL single interleaving set for PR1.)

Fig. 3 The two interleaving sets of PR2
The property $Q$ is chosen so that there is an interleaved path for PR1 that reaches a global state in which $Q$ holds. However, there also exists a path in which no global state satisfies $Q$. By the construction of PR1, the two paths for PR1 are in the same interleaving set. On the other hand, every path for PR2 constitutes a different interleaving set as depicted in Figure 3. From this, and by the choice of $Q$, the property $P$ will hold for the ISTL structure of PR1 but not for PR2. □

What we have shown is that with respect to the partial order of sequentiality in execution - the two programs are distinguishable. If however, the "essential partial order" mentioned in the introduction had been used, these two particular programs would have the same ISTL structure, and be indistinguishable. Here we have seen that such a distinction can be made in ISTL but not in the other temporal logics. In sections 4 and 5, examples were given which show when it is desirable to do so.

Another connection between the expressive power of various temporal logics can be seen by reexamining the translation described in Definition 4.2. Recall that there a translation $t$ between an LTL formula $L$ and an ISTL* formula $L'\in \text{LPF}$ is defined such that $\square$ is replaced by $G$, $\diamond$ is replaced by $F$ and $\lozenge$ is replaced by $X$.

To compare LTL with ISTL*, we consider a set of LTL structures $M$ which includes all the paths from the interleaving sets of the ISTL* structure $M^I$ (that is, a suffix and fusion closed set of sequences in which acceptability is enforced). Satisfaction of an LTL formula $L$ over $M$ is defined such that $M \models L$ iff each of the sequences (paths) in $M$, taken as a linear structure, satisfies $L$. It is immediate from the semantic definition of LTL and ISTL* that the following connection holds:

6.2. Proposition. $M \models L \iff M^I \models AL^I$. (A similar proposition referring to CTL* is proven in [EH1]).

For ISTL and ISTL* there are formulas of branching time logic concerning the existence of a path which cannot be expressed. For this reason, an extension for those logics appears in Section 7.1.

7. Alternative definitions and extensions.

7.1. Allowing quantification over the substructures. Theorem 6.1 shows that there are properties which are expressible in ISTL and not expressible in any variant of a linear or branching temporal logic. On the other hand, an existential property over all possible computations, such as "there exists an interleaving computation that reaches a global state in which ..." is not
expressible in ISTL because there is an implied quantifier that says "in every (partial order) execution" in ISTL semantics. There is no real obstacle to adding another level of quantifiers which allows both expressing "for every partial order" and "there exists a partial order". This could be called QISTL (QISTL). The Q stands for "quantified" because this logic allows quantification of ISTL formulas over the set of all partial order executions.

7.1.1. Syntax.
1. If Q is in ISTL, then $\exists Q$ and $\forall Q$ are QISTL.
2. If Q, W are in QISTL then $Q \land W$, $Q \lor W$ and $\neg Q$ are in QISTL.

7.1.2. Semantics.
1a. $M \models \exists Q$ iff there exists an interleaving set H such that $M, H \models Q$.
1b. $M \models \forall Q$ iff for each interleaving set H it holds that $M, H \models Q$. (By the previous semantics, this was written as $M \models Q$.)
2a. $M \models Q \land W$ iff $M \models Q$ and $M \models W$.
2b. $M \models Q \lor W$ iff $M \models Q$ or $M \models W$.
2c. $M \models \neg Q$ iff not $M \models Q$.

Now we can write assertions about the existence of a computation in which a temporal property holds. For example: $\exists (at(START) \rightarrow EFx = y)$ means "there exists an execution in which one of the paths leads to a global state in which $x = y$".

7.2. Alternative semantic definition. The previous definition of a structure M uses a relation R which generates a set of infinite and acceptable paths H which was termed an interleaving set. The path quantifiers A and E are allowed to range only over the set of paths in H [E, EL1, EL2, CVW].

A different approach [E] is to consider all the paths that are generated by the relation R as an interleaving set and embed the acceptability constraint into the formulas. The fact that an interleaving set contains all the paths generated by PATHS(R) is a property called R-generability. As proven in [E], this is equivalent to saying that to the fusion closure and suffix closure properties another property, called limit closure, is added. The limit closure property means, if the paths $x_1 y_1, x_1 x_2 y_2, x_1 x_2 x_3 y_3, ...$ (where $x_i$ is a state and $y_j$ is a path in an interleaving set) then the infinite path $x_1 x_2 x_3 ...$ is also a path in the interleaving set.

Earlier definitions of branching time temporal logics [BMP, CE] give a definition for R-generable structures by specifying only the relation R which generates the paths. On the other hand [E, EH1, CVW] defines a logic over Abrahamson structures where R-generable structures is only a special case.
7.2.1. **Definition.** For a program with \( n \) possible atomic operations, labeled \( l_1 \)– \( l_n \), define `ACCEPTABLE` to be synonymous with

\[
\text{ACCEPTABLE} = \bigwedge_{i=1}^{n} [G((\text{at}(l_i) \land \text{EX}\text{after}(l_i)) \rightarrow \text{F}\text{after}(l_i))]
\]

7.2.2. **Definition.** Define a transformation of formulas \( h \) from a formula \( L \) in ISTL (ISTL\(^*\)) into an ISTL\(^*\) formula \( L^h \) over structures defined this way: each subformula \( A p \) is replaced by \( A(\text{ACCEPTABLE} \rightarrow p) \) and each \( E p \) is replaced by \( E(\text{ACCEPTABLE} \land p) \).

A disadvantage of this approach is that the formalism depends upon fully labeling the program and the interpretation assigned to the predicates `at` and `after`. Another disadvantage is that formulas previously defined in ISTL are now transformed into ISTL\(^*\) formulas.

8. **Deduction in the new framework.**

In order to use the variants of ISTL for proving correctness of programs, one needs a sound (and preferably complete) deductive system. The axioms and consequence rules used for verifying a program in the logic can be divided into several categories [MP3]: (1) Those axioms and consequence rules that stem directly from the definition of the logic. (2) Those that stem from the interest in a restricted set of structures which result from the semantics of the computational model. (3) Those axioms and consequence rules that follow from the syntactical structures of a specific program and the domains of interest. It is convenient to look at each part of the deductive system as an "increment" to the consequence rules and axioms.

First we consider the rules from the basic definition of the logic. The definition of ISTL (ISTL\(^*\)) simply interprets a branching temporal logic formula simultaneously over a set of branching Abrahamson structures. Therefore, any valid CTL formula which is valid over Abrahamson structures is also valid for ISTL and vice versa. The same connection holds between CTL\(^*\) and ISTL\(^*\). In [CVW] it is proven that a CTL\(^*\) formula is satisfiable over the class of Abrahamson structures iff it is satisfiable over probabilistic structures [LS]. Therefore, the sound and complete deductive system in [LS] is applicable to ISTL\(^*\) too.

Most deductive systems [BMP, EH2] are given for an R-generable definition of the structures (see Section 7.2) rather than for Abrahamson structures. One may look at R-generable structures as a special case of Abrahamson structures. Therefore, the set of tautologies for R-generable structures is a superset of those for Abrahamson structures. Extending the set of axioms from [LS] into a complete deductive system for CTL\(^*\) over the R-generable structures is an open problem.

In our context, the additions which follow from the intended family of interpretations must be considered. When restricting the set of structures to be sets of interleaving sets (Definition 3.13), all the valid formulas remain valid, but additional formulas now become valid.
Therefore, one might like to add to the deductive system of [LS] enough axioms and consequence rules in order to get a sound and complete system over the class of interleaving sets. It is clear from 4.1 that the consequence rule $p \rightarrow AGp \vdash EFp \rightarrow AFp$ should be added to the deductive system and that it then remains sound. Finding sufficient consequence rules and axioms to guarantee completeness over our interpretation is an open problem.

Another interesting issue here is that the underlying partial order semantics might further restrict the class of structures (for example, by allowing only a fixed number of processes). For each such restriction it might be interesting to find the corresponding "increment" of the deductive system.

Finally, the third category of rules should be added. Clearly, in verifying programs over a given programming language, it is necessary to have a set of consequence rules and axioms which relate the program syntax with the logic. Part of these rules can be given in a "generic" way, not referring to specific variables or labels of a program. An example is the proof rule given in Example 5.1. Another source for deduction rules comes from a specific program, by giving a formula which specifies some properties of the program (which is called the program part in [MP3]).

Note that yet another source for deduction rules is the domain of interest. When defining a first order version of a logic over a specific domain (the integer numbers for example), one must include (non-temporal) consequence rules and axioms which define the semantics of the relations and functions of this domain. These issues, which are orthogonal to the particular features of ISTL, will not be considered further here.

Another issue for further research is extending the deductive system to deal with the QISTL and QISTL* definitions.


A temporal framework for reasoning about global states which are constructed from partial orders was suggested. The major motivation for the new logic and interpretation is to rigorously express and prove within a uniform formalism properties which were previously explained informally [D, EF].

A correctness criterion which is very natural for dealing with partial orders and very natural to ISTL is linearization: For each partial order, choose a single total order which contains it to represent the computation. Various aspects of this correctness criterion appear in [D, HW, HS] and are defined rigorously in the new framework.

It is evident that the following property, which was formally proven in 4.1 is important to the understanding of many phenomena which are explained using the partial order model: A
stable property which occurs on one interleaving sequence will eventually hold on any interleaving sequence which is a completion of the same partial order.

Even if interleaving sets are not directly needed to express a property (for example, total correctness is traditionally expressed in LTL) it might be convenient to move into ISTL. Properties which stem from the partial order are sometimes not interesting in themselves (like, for example, a possible global state in an execution in examples 5.2, 5.3) but are helpful as an intermediate stage in proving other, much more common, properties such as total correctness. A deductive system which makes use of the ability to reason about linearizations of partial orders can be used here (Example 5.1).

The novel property of ISTL and its extensions is that, whenever convenient, it allows us to use global states in proofs and specifications while thinking in terms of partial orders.

The logic QISTL* is strong enough to embed the syntactic proof systems of [MP2, MP3, AFR] and suggests the addition of new proof rules which take advantage of the underlying partial orders.

Acknowledgements. We appreciate the extremely helpful conversations we have had on this subject with Allen Emerson, Nissim Francez, Janos Makowsky, Shlomit Pinter, Amir Pnueli, Wolfgang Reisig, and Pierre Wolper.

References:


E. A. Emerson, C. L. Lei, Temporal model checking under generalized fairness constraints, Proc. 18th Hawaii international conference on system sciences, 1985, 277-288.


Appendix:

The definition of a CTL * like extension for ISTL termed ISTL * now follows:
Syntax.

1. For each proposition \( p \in P \), \( p \) is a state formula.
2. If \( Q, W \) are state formulas, then \( Q \land W, Q \lor W \) and \( \neg Q \) are state formulas.
3. If \( Q \) is a path formula, then \( AQ \) and \( EQ \) are state formulas.
4. Any state formula \( Q \) is a path formula.
5. If \( Q \) and \( W \) are path formulas, then \( Q \land W, Q \lor W \) and \( \neg Q \) are path formulas.
6. If \( Q \) is a path formula, then \( FQ, GQ \) are path formulas.
7. If \( Q, W \) are path formulas, then \( (QUW) \) is a path formula.
8. If \( Q \) is a path formula, then \( XQ \) is a path formula.

Semantics. (The numbers below correspond to those in the syntactic structure defined above.)

1. \( M,H,s \models p \) for \( p \in P \) iff \( L(s,p) \models true \).
2a. \( M,H,s \models Q \land W \) iff \( M,H,s \models Q \) and \( M,H,s \models W \).
2b. \( M,H,s \models Q \lor W \) iff \( M,H,s \models Q \) or \( M,H,s \models W \).
2c. \( M,H,s \models \neg Q \) iff \( \neg M,H,s \models Q \).
3a. \( M,H,s \models AQ \) iff every path \( x \in H \) with \( first(x)=s \), \( M,H,x \models Q \).
3b. \( M,H,s \models EQ \) iff there exists a path \( x \in H \) with \( first(x)=s \) such that \( M,H,x \models Q \).
4. \( M,H,x \models Q \) iff \( M,H,first(x) \models Q \).
5a. \( M,H,x \models Q \land W \) iff \( M,H,x \models Q \) and \( M,H,x \models W \).
5b. \( M,H,x \models Q \lor W \) iff \( M,H,x \models Q \) or \( M,H,x \models W \).
5c. \( M,H,x \models \neg Q \) iff \( \neg M,H,x \models Q \).
6a. \( M,H,x \models FQ \) iff there exists \( i \geq 0 \) \( M,H,x^i \models Q \)
6b. \( M,H,x \models GQ \) iff for each \( i \geq 0 \) \( M,H,x^i \models Q \)
7. \( M,H,x \models (QUW) \) iff there exists \( i \geq 0 \) such that \( M,H,x^i \models W \) and for each \( 0 \leq j < i \) \( M,H,x^j \models Q \)
8. \( M,H,x \models XQ \) iff \( M,H,x^1 \models Q \).