Deciding Global Partial-Order Properties

Rajeev Alur
Univ of Pennsylvania and Bell Labs
alur@cis.upenn.edu

Ken McMillan
Cadence Berkeley Labs
mcmillan@cadence.com

Doron Peled
Bell Labs
doron@research.bell-labs.com

Abstract

Model checking of asynchronous systems is traditionally based on the interleaving model, where an execution is modeled by a total order between atomic events. Recently, the use of partial order semantics that allows independent events of concurrent processes to be unordered, is becoming popular. Temporal logics that are interpreted over partial orders allow specifications relating global snapshots, and permit reduction algorithms to generate only one representative linearization of every possible partial-order execution during state-space search. This paper considers the model checking problem for temporal logics interpreted over partially ordered sets of global configurations. For such logics, only undecidability results have been proved previously. In this paper, we present a decision procedure for a fragment that contains an eventuality operator and its dual. We also sharpen the undecidability results by showing that a single *until* operator is sufficient for undecidability.

1 Introduction

The *model checking* problem is to decide whether a finite-state description of a reactive system satisfies a temporal-logic specification. The solutions to this problem have been implemented, and the resulting tools are increasingly being used as debugging aids for industrial designs. All of these solutions employ the so-called *interleaving* semantics in which a single execution of the system is considered to be a totally-ordered sequence of events. The (linear) semantics of the system is, then, a set of total-order executions that the system can possibly generate. The specifications are written in the linear temporal logic LTL [13]. The model checker checks if every execution of the system satisfies the LTL-specification[1].

In contrast to the interleaving semantics, the *partial order semantics* considers a single execution as a partially ordered set of events. The partial order semantics does not distinguish among total-order executions that are *equivalent* up to reordering of independent events, thereby, resulting in a more abstract and faithful representation of concurrency, and has attracted researchers in concurrency theory for many years [10, 18]. To understand the difference between the two views, let us consider a distributed system consisting of two processes. Suppose the first process issues either an event a or b, and the second process issues an event c, independently of the first. The interleaving semantics consists of four total-order executions: ac, ca, bc,

[1]In the alternative branching-time paradigm, the semantics of the system is a labeled state-transition graph, and the specification is given as a formula of the *Computation tree logic* (CTL) [3]. Branching-time versions of partial-order semantics are possible, but are not studied in this paper.

and cb. The partial-order semantics consists of two partial orders: the first consists of events a and c with no ordering, and the second consists of events b and c with no ordering[2]. Thus, partial-order semantics, unlike the interleaving semantics, distinguishes between nondeterminism and concurrency. Consequently, specification languages over partial orders can permit a direct representation of properties involving causality and concurrency, for example, serializability.

Apart from the greater expressiveness of partial order specifications, they are interesting due to their compatibility with the so-called partial order reductions. The partial-order equivalence among sequences can be exploited to reduce the state-space explosion problem: the cost of generating at least one representative per equivalence class is typically significantly less than the cost of generating all interleavings [21, 5, 12, 11]. This approach has been shown to be practically useful in the verification tool SPIN [6]. If the specification could distinguish between two sequences of the same equivalence class, as is the case with LTL, the above equivalence cannot be used: the same equivalence class may contain both a sequence that satisfies the specification and a sequence that does not. It is possible to refine the equivalence relation, providing more representatives at the expense of using a bigger state space [12]. The alternative solution is to use a specification logic that is directly interpreted over partial orders. This latter approach demands study of decision problems for partial order logics.

How does one define a temporal logic over partial orders? Two approaches have been proposed to write temporal requirements over a model consisting of a set of partially ordered events, or local states of processes. In *local* partial order logics, the truth of a formula is evaluated at a local state, and the temporal modalities relate causal precedences among local states. Examples of such logics include TRPTL [19] and TLC [2]. In *global* partial order logics, the truth of a formula is evaluated in a *global* state, also called a configuration or a slice, which consists of a consistent set of local states. The temporal modalities of a global logic, such as ISTL [7, 16], relate causal precedences among configurations. Global partial order logics are strictly more general than the local ones. In a partial order, unlike in a total order, there are many ways to proceed from one state to the next, and consequently, the syntax of partial order logics, uses path quantifiers as in CTL. For example, if p is a global state predicate asserting that states of all processes are consistent with one another, then the ISTL-formula $\exists\Diamond p$ asserts that p is a possible global snapshot. A system satisfies $\exists\Diamond p$ if every partial-order execution has some linearization containing a p-state. It should be noted that this property cannot be specified in LTL or CTL or any local partial-order logic[3].

Before we consider the model checking for partial order logics, let us briefly review the solution to the model checking problem for LTL. A system M, possibly with fairness requirements, is viewed as an ω-automaton A_M that generates all possible total-order executions of M. To check whether the system satisfies an LTL-formula φ, the model-checking algorithm first constructs an ω-automaton $A_{\neg\varphi}$ that accepts all the satisfying models of $\neg\varphi$, and tests emptiness of the intersection of the languages of the two automata A_M and $A_{\neg\varphi}$ [22, 8]. For algorithmic verification of partial order logics, the solution is to construct, given a partial order specification φ, an ω-automaton $A_{\neg\varphi}$ that accepts the linearizations of the partial order models of $\neg\varphi$. To check whether the system M satisfies a partial order formula φ, we need to test emptiness of the the intersection of $A_{\neg\varphi}$ and the automaton A_M. Since we know that the automaton $A_{\neg\varphi}$ does not distinguish among the linearizations of the same partial order, the above approach yields a correct result even if A_M generates only one linearization of each partial order execution of M.

[2] Equivalently, partial-order semantics of a systems is a set of equivalence classes of totally-ordered executions. Each equivalence class, called *Mazurkiewicz trace* consists of linearizations of a partially-ordered set of events.

[3] The LTL formula $\Diamond p$ asserts a stronger specification, namely, every linearization of every partial order execution contains a p-state. The CTL formula $\exists\Diamond p$ asserts a weaker specification, namely, some linearization of some partial order execution contains a p-state. A local partial-order logic, like TLC, cannot refer to the global assertion p.

Thus, the verification problem for partial order logics can be solved if a partial order specification can be translated to an ω-automaton accepting the linearizations of its models. Consequently, the main challenge in this approach is to translate the modal operators interpreted over a partial-order to requirements on its linearizations. In other words, by examining one linearization, the automaton needs to infer a property of all equivalent linearizations. This goal has already been met for local partial order logics TrPTL and TLC [19, 2]. Unfortunately, the technique used in these constructions does not lead to similar constructions for global partial order logics. Some simple global specification formalisms were even shown to be undecidable [15, 9].

In this paper, we identify a global partial order logic for which the model checking problem is decidable by presenting a tableau construction. This logic is ISTL$^\diamond$, a subset of the logic ISTL obtained by retaining the modalities $\exists\diamond$ and $\exists\bigcirc$, and their duals, but disallowing the modalities $\exists\Box$ or $\exists\mathcal{U}$. The complexity of verifying that a finite state program satisfies a specification in ISTL$^\diamond$ is linear in the number of global states of the program, and doubly exponential in the size of the specification. We also refine the undecidability result of [15] by establishing that the modality $\exists\mathcal{U}$ is sufficient, by itself, to render undecidability. It is quite surprising that model checking "eventually" is elementary, while model checking "until" is undecidable[4].

The decidability for ISTL$^\diamond$ can also be established by translating its formulas to a first-order language of [4] that has variables ranging over local states, monadic predicates, and a binary partial-order relation. However, this approach leads to a decision procedure of nonelementary complexity. Indeed, the translation from requirements on global cuts to linear sequences of local states is difficult due to various factors: a single local state can participate in many global cuts, there is no monotonicity as incomparable cuts can be maximal for satisfaction of eventuality properties, the logic includes universal quantification over global cuts, and the modalities are nested. We overcome these problems by showing that every ISTL$^\diamond$ formula can be rewritten to a special form such that for each of its eventuality subformulas ψ, there exists a maximal configuration that contains exactly those configurations satisfying ψ. The decision procedure, then, builds on this key insight.

It should be noted that, while the decision procedure for local partial-order logics such as TLC is PSPACE, the decision procedure for the global logic ISTL$^\diamond$ is EXPSPACE. It is open whether EXPSPACE is also a lower bound for ISTL$^\diamond$. Also open is the decidability in presence of the operator $\exists\Box$ that is useful in specifying properties such as serializability[5].

2 A Global Partial-Order Logic

We begin with a model theory for the logic ISTL. Our models can be viewed in one of two ways: either as a partially ordered set of local states (states of individual processes), or as a branching structure (a Kripke model). The Kripke model represents all possible sequences of global states that may be derived from the partial order.

Causal Structures

We will consider first the partially ordered view, which we refer to as a *causal structure*, and then define its relation to a Kripke model. In a causal structure, a state is "local", in the sense that it is occupied by some subset of the processes in the system, which are, in effect, synchronized at that state. All of the states

[4] In CTL both operators have the same cost for model checking, namely, NLOGSPACE; in LTL eventually is in NP, while until is in PSPACE.

[5] If the state predicate q asserts that at most one transaction is active, then $\exists\Box q$ specifies serializability.

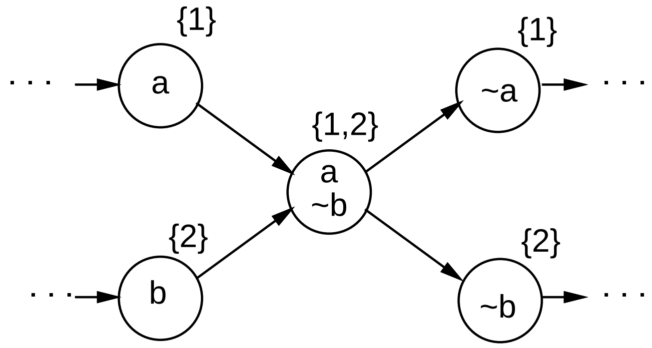


Figure 1: A two-process causal structure, where $P_1 = \{a\}$ and $P_2 = \{b\}$.

occupied by a given process are totally ordered, but no further ordering is imposed on the structure. Thus, the causal structure represents exactly the temporal ordering information that may be inferred from the set of individual histories of the processes. Each atomic proposition of the logic is associated with a particular process, and may only label states that are occupied by that process. As an example, consider the causal structure in Figure 1, which shows two concurrent processes that synchronize at a common state.

To be more precise, let a *concurrent alphabet* be a set P of propositions that is partitioned into disjoint sets, P_x, for $x = 1 \ldots n$. We use N to denote the set $\{1 \ldots n\}$ of processes. A *causal structure* Γ over a concurrent alphabet P consists of the following components:

- *State-space:* For each process x, a countable infinite set S_x of states. These are the states occupied by x. The set $\cup_x S_x$ of all states is denoted S.
- *Causality relation:* For each process x, a total, well founded order $\preceq_x$ on S_x, whose minimal element is a unique state denoted s^I (the global initial state). The relation $(\cup_x \preceq_x)^*$, is denoted $\preceq$, and must be a partial order.
- *State labeling:* For each process x, a mapping v_x that assigns each state $s \in S_x$, a set $v_x(s) \subseteq P_x$ of propositions true in s.

Causal structures to Kripke models

Now we define the correspondence between causal structures and a restricted class of Kripke models. To do this, we use *configurations* (a.k.a. slices) as our global states. A configuration is a set of states of the causal structure that is left-closed under $\preceq$. This may be thought of as a "partial run" of the structure. Any set of states $\Theta \subseteq S$ can be extended to a configuration $pre(\Theta)$ by simply taking its left closure under $\preceq$. The notion of configuration is illustrated in Figure 2. For a configuration Θ, let fin_Θ be the set of processes x such that $\Theta \cap S_x$ is finite. For any nonempty configuration Θ and any process x in fin_Θ, there is a unique maximal state under the local order $\preceq_x$, we denote this state by Θ^x. If $\Theta \cap S_x$ is infinite, then no such maximal state exists, and let us define $\Theta^x = \perp$ in that case. The tuple $(\Theta^1, \ldots \Theta^n)$ is called the *cut* of the configuration Θ. The notion of a cut is illustrated in Figure 2. We note that any nonempty configuration (finite or infinite) Θ is uniquely defined by its cut.

Configurations are ordered by the subset relation: if $\Theta \subseteq \Delta$ then the configuration Δ is in the future of the configuration Θ. The ordering $\subseteq$ is a partial order over the set of configurations, and is called the *global*

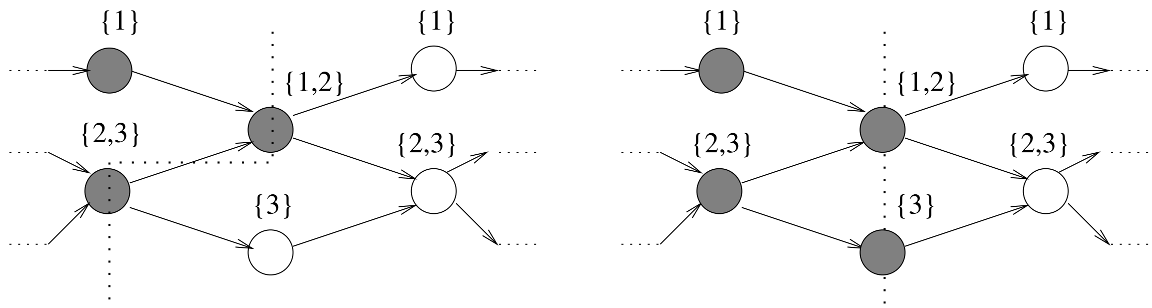
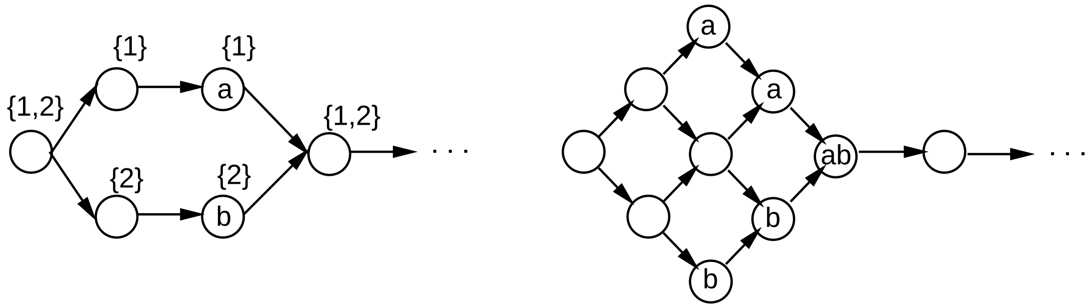


Figure 2: Two configurations. Each one is a left-closed set of states. The associated cut (dashed line) comprises the elements maximal w.r.t. each process.



a) Causal structure

b) Kripke model

Figure 3: Causal structure and associated Kripke model.

partial order. This leads to the definition of a Kripke model of a causal structure. Given a causal structure $\Gamma = (S, \preceq, v)$ over a concurrent alphabet P, let the Kripke model $K_\Gamma = (T, R, L)$, where

- T, the state set, is the set of finite, nonempty, configurations of Γ.
- R, the transition relation, is the transitive reduction of the partial order $(T, \subseteq)$, that is, $\Theta \, R \, \Delta$ if $\Theta \subset \Delta$ and there is no configuration Θ' such that $\Theta \subset \Theta' \subset \Delta$.
- L, the labeling function, maps any configuration $\Theta \in T$ to the set of propositions $\cup_x v_x(\Theta^x)$. That is, a local proposition of a process x is true in Θ iff it is true in the final state Θ^x.

The left and right configurations in Figure 2 are related by R. Figure 3 shows a causal structure and the associated Kripke model.

Syntax and Semantics

The formulas of ISTL are defined inductively by the grammar

$$\varphi \ := \ p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \exists\Diamond\varphi \mid \exists\bigcirc\,\varphi \mid \varphi\,\exists\mathcal{U}\psi \mid \exists\Box\varphi$$

where p is an atomic proposition. Given a causal structure Γ, the formulas of ISTL are interpreted over configurations of Γ. The interpretation of modalities is as in CTL interpreted over the Kripke structure

K_Γ. This is in contrast with the (local) logic TLC [2] whose formulas are interpreted over states of Γ and branching-time modalities are interpreted over Γ itself.

Let φ be a ISTL-formula, Γ be a causal structure, and Θ be a configuration of Γ. Then, the satisfaction relation $(\Gamma, \Theta) \models \varphi$ is defined inductively below (when the structure is understood, $(\Gamma, \Theta) \models \varphi$ is abbreviated to $\Theta \models \varphi$):

$\Theta \models p$	iff	$p \in L(\Theta)$ for an atomic proposition p;
$\Theta \models \neg\varphi$	iff	$\Theta \not\models \varphi$;
$\Theta \models \varphi \wedge \psi$	iff	$\Theta \models \varphi$ and $\Theta \models \psi$;
$\Theta \models \exists\Diamond\varphi$	iff	$\Theta \subseteq \Delta$ and $\Delta \models \varphi$ for some configuration Δ;
$\Theta \models \exists\bigcirc\varphi$	iff	$\Theta R \Delta$ and $\Delta \models \varphi$ for some configuration Δ;
$\Theta \models \varphi\,\exists\mathcal{U}\psi$	iff	there exists a finite sequence $\Theta_0 \ldots \Theta_n$ of configurations such that (i) $\Theta_0 = \Theta$, (ii) $\Theta_i R\, \Theta_{i+1}$ for $0 \leq i < n$, (iii) $\Theta_n \models \psi$, and (iv) $\Theta_i \models \varphi$ for all $0 \leq i < n$;
$\Theta \models \exists\Box\varphi$	iff	there exists an infinite sequence $\Theta_0\Theta_1 \ldots$ of configurations such that (i) $\Theta_0 = \Theta$, (ii) $\Theta_i R\, \Theta_{i+1}$ for all i, and (iii) $\Theta_i \models \varphi$ for all i.

Denote $\Gamma \models \varphi$ iff $(\Gamma, \{s^I\}) \models \varphi$. As in CTL, we can define a variety of auxiliary logical (e.g. disjunction) and temporal (e.g. $\forall\Box$) operators. We use $\wedge_k\varphi_k$ to denote the conjunction of finitely many formulas. We will consider various fragments of ISTL by restricting the syntax to a subset of the temporal modalities: the fragment ISTL$^\Diamond$ allows only $\exists\Diamond$ modality, the fragment ISTL$^{\Diamond,\bigcirc}$ allows $\exists\Diamond$ and $\exists\bigcirc$, and the fragment ISTL$^{\mathcal{U}}$ allows only $\exists\mathcal{U}$ (note that $\exists\Diamond$ is definable from $\exists\mathcal{U}$). All these fragments are closed under boolean operations.

A typical ISTL$^\Diamond$-formula is $\forall\Box(p \rightarrow \exists\Diamond q)$, where p and q are assertions about global snapshots. It means that no matter how the system evolves, if p is a possible snapshot at some time, then q is a possible snapshot at a later time. No formula of LTL or TLC is equivalent to this formula.

3 Decision Procedure for ISTL$^\Diamond$

We now describe a doubly-exponential procedure for deciding satisfiability of formulas of ISTL$^\Diamond$. The procedure first rewrites a formula φ into a boolean combination of subformulas that use negation in a restricted way, and the second step is to construct an automaton that recognizes the set of all linearizations of models of such special formulas.

Normal Form

We consider formulas of a special type, namely, the ones in which negation is not applied to a conjunction. The set of *normal-form* ISTL$^\Diamond$-formulas is the least set such that:

(1) Every atomic proposition is a normal-form formula.
(2) $\exists\Diamond \wedge_k \varphi_k$ is a normal-from formula whenever all φ_k are.
(3) If φ is a normal-form formula then so is $\neg\varphi$.

For example, the formula $\exists\Diamond(a \wedge \neg\exists\Diamond b)$ is in normal form, whereas the formulas $\exists\Diamond(a \vee \exists\Diamond b)$ and $a \wedge \exists\Diamond b$ are not.

An *eventuality* formula of ISTL$^\Diamond$ is a formula of the form $\exists\Diamond\varphi$. For an eventuality formula $\varphi = \exists\Diamond \wedge_k\varphi_k$, the formulas φ_k are called *conjuncts* of φ. Next, we establish that we do not lose any expressive power by restricting ourselves to normal-form formulas: each ISTL$^\Diamond$ formula is equivalent to a boolean combination of normal-form formulas.

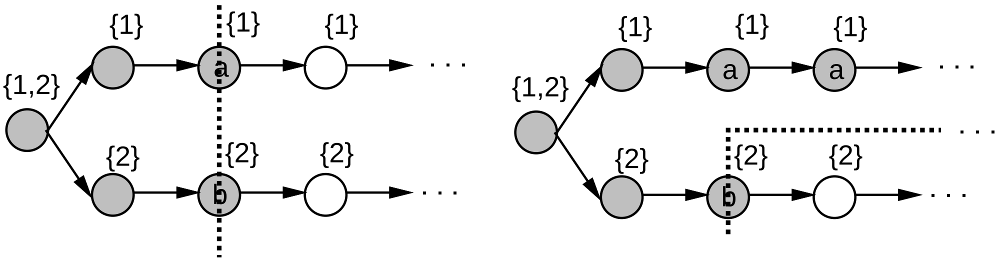


Figure 4: Maximal configuration for the formula $\exists\Diamond(a \wedge b)$ in two different causal structures. Note that in one case, the maximal configuration is infinite for process 1.

Proposition 1 *For every eventuality-formula φ of* ISTL$^{\Diamond}$*, there exists a* ISTL$^{\Diamond}$*-formula ψ such that (1) φ and ψ are equivalent, (2) ψ is a disjunction of at most $2^{|\varphi|}$ normal-form eventuality-formulas, and (3) the number of distinct eventuality-subformulas of ψ is at most $2^{|\varphi|+1}$.*

Proof: The proof is by induction on the size of φ. Let $\varphi_1, \ldots \varphi_k$ be the top-level eventuality-subformulas of φ (i.e. each φ_i is a subformula of φ, and there is no eventuality-subformula ψ of φ such that φ_i is a (strict) subformula of ψ). Let m be the length of φ, counting each φ_i to be of size 1, and let m_i be the actual size of φ_i. Then, $|\varphi| \geq m + m_1 + \cdots + m_k$ (assuming that the formula is fully parenthesized). Rewrite φ in disjunctive normal form while treating each φ_i as a proposition. Since $\exists\Diamond$ distributes over disjunction, we have φ equivalent to $\vee_l \psi_l$ with at most 2^m disjuncts ψ_l. Each disjunct ψ_l is of the form $\exists\Diamond \wedge_j \chi_j$ with at most m conjuncts χ_j. Each such conjunct χ_j is either a proposition, a negated proposition, some φ_i, or some negated φ_i. Rewrite each φ_i as a disjunction of normal-form formulas using induction: each φ_i is equivalent to a disjunction of at most 2^{m_i} normal-form formulas. It follows that $\neg\varphi_i$ is also equivalent to a disjunction of at most 2^{m_i} normal-form formulas. Let us revisit $\psi_l = \exists\Diamond \wedge_j \chi_j$, we need to do additional rewriting to get rid of the newly introduced disjunctions. It follows that ψ_l is equivalent to a disjunction of at most $2^{m_1+\cdots+m_k}$ normal-form formulas. Hence, φ is equivalent to at most $2^m 2^{m_1+\cdots+m_k}$ normal-form formulas. Using $|\varphi| \leq m + m_1 + \cdots + m_k$, we have the number of disjuncts bounded by $2^{|\varphi|}$.

To count the number of eventuality-subformulas, observe that the number of eventuality-subformulas after rewriting equals the number of disjuncts at the top-level plus the number eventuality-subformulas generated by rewriting each φ_i. This is bounded by $2^{|\varphi|} + 2^{m_1+1} + \cdots + 2^{m_k+1}$. Using $|\varphi| \geq m + m_1 + \cdots + m_k$, we can bound this number by $2^{|\varphi|+1}$. ∎

Maximal configurations

We now show that every normal form eventuality formula φ has a maximal configuration max_φ, such that a configuration satisfies φ exactly when it is contained in max_φ. Figure 4 shows a formula and its maximal configuration in two different causal structures. We note that the maximal configuration need not be finite (and may also be empty, in the case where the formula is false everywhere). Also, note that such unique maximal configurations may not exist for formulas other than eventuality formulas in normal form. For instance, for the left causal structure of Figure 4, there is no unique maximal configuration satisfying the formula $\exists\Diamond(a \vee b)$.

In the following, we identify every formula φ with the set $\{\Theta \mid \Theta \models \varphi\}$ of configurations satisfying that formula. We also note that a set of configurations is "$\subseteq$-closed" if every subset of a member is also in the set, "$\supseteq$-closed" if every superset of a member is also in the set, and "$\cup$-closed" if the union of any two members is also in the set.

Lemma 1 *If φ and ψ are $\cup$-closed then $\varphi \wedge \psi$ is $\cup$-closed.*

Lemma 2 *Let $\varphi = \exists\Diamond\psi$. φ is $\subseteq$-closed, $\neg\varphi$ is $\supseteq$-closed, and if ψ is $\cup$-closed then so is φ.*

Lemma 3 *If φ is an eventuality formula in normal form then φ is $\cup$-closed.*

Proof: The satisfaction of an atomic proposition depends upon the corresponding local state. Hence, if φ is a proposition or a negated proposition, then φ is $\cup$-closed. If φ is a negated eventuality-formula, then by Lemma 2, φ is $\supseteq$-closed, and hence, $\cup$-closed. Now consider $\varphi = \exists\Diamond \wedge_k \varphi_k$, with each φ_k in normal-form. By induction, φ_k are $\cup$-closed. Then by Lemma 1, so is $\wedge_k\varphi_k$, and hence, by Lemma 2 so is φ. ∎

Proposition 2 *Let φ be an eventuality-formula in normal form. For every causal structure, there exists a configuration max$_\varphi$ such that $\Theta \models \varphi$ iff $\Theta \subseteq$ max$_\varphi$.*

Proof: The set of configurations satisfying the eventuality-formula φ is $\cup$-closed and $\subseteq$-closed. Choose max_φ to be the union of all configurations satisfying φ. ∎

Tableau constraints

We now transform the problem of satisfiability of a normal form eventuality formula φ by augmenting the causal structure with a set of new atomic propositions (each labeling the maximal configuration of some subformula) and formulating a set of "tableau" constraints, such that there exists an augmented structure satisfying the tableau constraints exactly when there exists a structure satisfying φ.

Given a concurrent alphabet P and a normal form formula φ, let P^φ be P augmented by introducing a special proposition c_ψ for every eventuality-subformula ψ. The propositions c_ψ differ from ordinary propositions in that they are allowed to label any subset of the state space S, rather than just the states of one process. In particular, the states labeled with c_ψ are intended to represent max_ψ, the maximal configuration satisfying ψ. We will say that ψ is "correctly labeled" in a given causal structure if c_ψ labels exactly the states in the maximal configuration of ψ. If the maximal configuration max_ψ contains only finitely many states of a process x, then the maximal local state of x that is labeled with c_ψ is denoted c_ψ^x.

We now wish to formulate a set of constraints that guarantee that a formula is correctly labeled, given that its own subformulas are correctly labeled. Suppose ψ is an arbitrary eventuality-subformula of φ. It must be of the form $\psi = \exists\Diamond(\psi_1 \wedge \cdots \wedge \psi_k)$. We note that c_ψ is a correct labeling exactly when the following three conditions hold:

(1) (the set labeled) c_ψ is a configuration (*i.e.*, left-closed),
(2) every finite configuration contained in c_ψ satisfies ψ (if c_ψ is finite, this is the same as saying that c_ψ satisfies $\psi_1 \wedge \cdots \wedge \psi_k$),
(3) no finite configuration not contained in c_ψ satisfies $\psi_1 \wedge \cdots \wedge \psi_k$ (else c_ψ is not maximal).

Let condition 1 above be denoted C_ψ^1. Condition 2, assuming all of the subformulas of ψ are correctly labeled, is equivalent to the following condition, which we denote C_ψ^2: If c_ψ is nonempty, then

(i) for all $i = 1, \ldots k$, if ψ_i is an eventuality formula, then $c_\psi \subseteq c_{\psi_i}$,
(ii) for all $i = 1, \ldots k$, if ψ_i is a negated eventuality formula, then $c_\psi \not\subseteq c_{\psi_i}$,
(iii) for all $i = 1, \ldots k$, if ψ_i is an atomic (resp. a negated atomic) formula in P_x and $x \in \mathit{fin}_{c_\psi}$ then $\psi_i \in L(c^x_\psi)$ (resp. $\psi_i \notin L(c^x_\psi)$),
(iv) There exists an infinite sequence of configurations $\Theta_1 \subset \Theta_2 \subset \cdots$, such that for every process x if $S_x \subseteq c_\psi$ then $S_x \subseteq \cup_j \Theta_j$ and for every atomic (or negated atomic) formula ψ_i in P_x, $\Theta_j \models \psi_i$ (resp. $\Theta_j \not\models \psi_i$) for all j.

The last condition guarantees that if c_ψ is infinite, then every configuration contained in c_ψ is contained in some finite configuration satisfying $\psi_1 \wedge \cdots \wedge \psi_k$. Finally we deal with condition 3, which guarantees that c_ψ is in fact maximal. Assuming subformulas of ψ are correctly labeled, it is equivalent to the following, which we denote C^3_ψ: There does not exist a finite configuration $\Theta \not\subseteq c_\psi$ such that, for all ψ_i, $i = 1 \ldots k$:

(i) if ψ_i is an eventuality formula, then $\Theta \subseteq c_{\psi_i}$,
(ii) if ψ_i is an atomic (resp. a negated atomic) proposition in P_x, then $\psi_i \in L(\Theta_x)$ (resp. $\psi_i \notin L(\Theta_x)$).

Observe that, in C^3_ψ, there is no requirement that Θ satisfy negated eventuality conjuncts. Now, let C_ψ denote the condition that ψ and all its subformulas are correctly labeled. We then have following by induction on the structure of ψ and the semantics of t he logic:

Lemma 4 *A normal form formula φ is satisfiable exactly when there exists a causal structure over P^φ, satisfying C_φ, where c_φ is true in the initial state.*

In the following, we show how to determine satisfiability of φ algorithmically, by coding causal structures as infinite strings, and constructing an ω-automaton that exactly characterizes C_φ.

Causal structures as ω-words

Let Σ be an alphabet consisting of truth assignments to the propositions $P^\varphi \cup \{at_1, \ldots, at_n\}$. The new propositions at_x will be used to encode the set of processes synchronized at a given state. We say that the causal structure Γ *generates* an ω-word $\overline{a} \in \Sigma^\omega$ iff there exists an enumeration $s_1 s_2 \ldots$ of the state-space S such that (1) for all processes x, $at_x \in \overline{a}_i$ iff $s_i \in S_x$, (2) for all $p \in P$, $p \in \overline{a}_i$ iff $p \in L(s_i)$, and (3) for all $k < l$, it is not the case that $s_l \preceq s_k$. That is, the ω-words generated by Γ are exactly its linearizations, appropriately encoded. Since the partial order $\preceq$ on a causal structure is exactly characterized by the set of total orders on the process state spaces S_x, it follows that:

Lemma 5 *For an ω-word $\overline{a}$ over Σ, there is at most one (up to isomorphism) causal structure $\Gamma_{\overline{a}}$ such that $\Gamma_{\overline{a}}$ generates $\overline{a}$.*

Thus, for a fixed concurrent alphabet P, causal structures can be represented by ω-words over Σ. In this sense, we can say that an ω-automaton A over Σ "recognizes" a set of causal structures, which are exactly those that generate some word accepted by A. By building a finite recognizer for each "tableau constraint" C^i_ψ, and constructing the product of these automata, we obtain a recognizer for C_φ. Testing satisfiability of φ then reduces to testing non-emptiness of this automaton.

Lemma 6 *For any normal form formula φ, there exists a nondeterministic Büchi automaton recognizing C_φ of $O(2^{O(m^2+m2^n)})$ states, where m is the size of φ.*

Proof: For any eventuality subformula ψ, there exists a deterministic Büchi automaton recognizing C^1_ψ of $O(2^n)$ states. This is because left-closure of c_ψ can be tested by a product of n 2-state automata, each of which tests closure w.r.t. $\preceq_x$.

Now consider verification of C^2_ψ for an eventuality subformula ψ. Testing containment of configurations (for the eventuality subformula cases) requires no information to be remembered about previous states. For each negated eventuality subformula, a two-state automaton with a Büchi acceptance condition is needed to ensure non-containment. For the atomic subformulas, the automaton guesses initially for each process whether c_ψ contains a finite or infinite number of states. For the finite case, it must guess the last state in the configuration. For the infinite case it must guess an infinite number of configurations Θ. However, only one configuration Θ need be "remembered" at any given time. This yields one 4-state machine per process. A Büchi acceptance condition requires that a new configuration Θ be completed infinitely often. Hence, there exists a nondeterministic Büchi automaton recognizing C^2_ψ of $O(4^n 2^k)$ states, where k is the number negated eventuality formulas appearing as conjuncts of ψ.

Now consider the requirement C^3_ψ for an eventuality subformula ψ. This condition states that there *does not exist* a finite configuration satisfying $\psi_1 \ldots \psi_k$, but strictly containing c_ψ (ensuring maximality of c_ψ). The automaton guesses such a configuration. Verifying that the guess is a configuration requires 2^n states. Remaining checks require no additional state. *Complementing* this automaton, by the subset construction, increases the size to $O(2^{2^n})$ states.

The automaton recognizing C_φ is just the product of the above automata for all eventuality subformulas of φ. ∎

Given the automaton for C_φ, we finally need to determine whether it recognizes any causal structures where φ is true in the initial state. This requires an automaton to accept those ω-words which are in fact generated by some causal structure.

Lemma 7 *There is a Büchi automaton G_n accepting exactly those words generated by some causal structure over P^φ, of size $O(1)$.*

Proof: A word is a linearization of some causal structure exactly when every process is included in the initial state, and every process appears infinitely often. The latter can be tested with n Büchi acceptance conditions, where we associate acceptance conditions with transitions. ∎

Theorem 1 *Satisfiability of any* ISTL$^\diamond$ *formula φ can be tested in $O(2^{2^{O(m+n)}})$ time, where m is the size of φ.*

Proof: The formula φ can be translated to boolean combination of normal form formulas containing a total of $O(2^m)$ eventuality-subformulas. The product of the automata for C_ψ for each of these subformulas has $O((2^{O((2^m)^2+2^m 2^n)})) = O(2^{2^{O(m+n)}})$ states. Take the product of this automaton with G_n and a 2-state automaton that checks the boolean combination at the initial state. Emptiness of this automaton can be tested in time proportional to the number of Büchi conditions ($O(n2^m)$) and to the number of transitions of the automaton, which remains $O(2^{2^{O(m+n)}})$. ∎

By a standard argument, it is possible to establish that the decision procedure can be implemented in space exponential in the size of the input, and thus, the problem is in EXPSPACE. It remains open whether EXPSPACE is a lower bound (a lower bound of NP follows trivially from the lower bound for satisfiability of the corresponding LTL fragment).

4 Extensions

Next-time operator

The fragment $\text{ISTL}^{\Diamond,\bigcirc}$ contains the modalities $\exists\Diamond$ and $\exists\bigcirc$. The construction of the previous section for $\text{ISTL}^{\Diamond}$ relies on (1) it suffices to consider only boolean combinations of normal form formulas, and (2) the set of configurations satisfying a normal form formula can be characterized by a maximal configuration. Both these properties continue to hold even after the introduction of the next-time operator, after appropriately modifying the definition of the normal form.

Theorem 2 *Satisfiability of an* $\text{ISTL}^{\Diamond,\bigcirc}$ *formula is decidable is* EXPSPACE.

Undecidability of $\text{ISTL}^{\mathcal{U}}$

We show here that some natural extension to $\text{ISTL}^{\Diamond}$ are undecidable. In [15] it was shown that ISTL is undecidable (a similar proof for related temporal logics appears in [9]). We sharpen the result of [15] by showing that the until operator $\exists\mathcal{U}$ is sufficient to prove undecidability.

Theorem 3 *The logic* $\text{ISTL}^{\mathcal{U}}$ *is undecidable.*

Proof Sketch: It is possible to show that one can encode in $\text{ISTL}^{\mathcal{U}}$ two processes which never interact. Each process represents a sequence of descriptions of Turing machine tapes. Using the $\exists\mathcal{U}$ operator, it is possible to describe two linearizations of the execution: one in which the two processes are interleaved one process at a time, asserting that the two processes have the same sequences of tape descriptions, and that each tape description is locally legal (e.g., has one position for the machine head, one machine state, and that the last tape description is an accepting one). The other linearization allows one process to be exactly one tape description ahead of the other. This allows us to assert that adjacent tape description are related according to the execution of a Turing machine. ∎

Relation to monadic first-order logic of partial order

The decidability of the logic $\text{ISTL}^{\Diamond,\bigcirc}$ can be seen directly by the fact that one can encode a formula of the type $\exists\Diamond\varphi$ and $\exists\bigcirc\ \varphi$ in first order language with the causal order $\preceq$. Specifically, consider the language $\mathcal{L}$ that contains first-order variables, second-order monadic variables (sets or propositions), logical connectives, quantification over first-order variables, and the binary relation $\preceq$. The formulas of $\mathcal{L}$ are interpreted over causal structures. When the relation $\preceq$ is generated by the union of total orders $\preceq_x$, one per each process, the set of linearizations of causal structures satisfying a formula of $\mathcal{L}$ is ω-regular, and can be characterized by a Büchi automaton [4]. The ω-regularity is retained even in presence of second-order quantification.

While the logic $\text{ISTL}^{\Diamond,\bigcirc}$ can be translated to $\mathcal{L}$, the undecidability of the logic $\text{ISTL}^{\mathcal{U}}$ implies that the until operator $\exists\mathcal{U}$ is not definable using second order formulas over partial orders. It is possible to define an until operator $\mathcal{U}$ in the following way:

$\Theta \models \varphi\mathcal{U}\psi$ iff for some $\Theta \subseteq \Delta$, $\Delta \models \psi$, and for each $\Theta \subseteq \Theta' \subset \Delta$, $\Theta' \models \varphi$.

This version of the until operator can be expressed in the first-order language $\mathcal{L}$, and hence, is decidable. A recent result [20] establishes that this until operator, together with next and previous operators, yield a logic that has the same expressiveness as the language $\mathcal{L}$. The only known upper bound for this logic is nonelementary.

Model checking

A system M of concurrently executing processes generates a set of causal structures. The system M satisfies a requirement given as a formula φ of ISTL, if every causal structure generated by A satisfies φ. The model checking problem, then, is to check whether or not M satisfies φ.

The decision procedure outlined in the previous section can be employed to obtain an automata-theoretic solution to the model checking problem of ISTL$^{\diamond}$. From M, we first construct an automaton A_M that accepts linearizations of the causal structures of M (see, for instance, [2] for more details). Then, we construct the automaton $A_{\neg\varphi}$ that accepts the linearizations of the causal structures satisfying $\neg\varphi$. The system M satisfies the specification φ iff the intersection of the languages of the two automata A_M and $A_{\neg\varphi}$ is empty. The complexity of model checking algorithm is linear in the size of the program automaton A_M, and doubly exponential in the size of the specification φ.

Model checking using representatives outlined in [5, 12, 21] can now be used as a heuristic improvement. We know that the language of the automaton $A_{\neg\varphi}$ is *closed*, that is, it does not distinguish among the linearizations of the same causal structure. Hence, the automaton A_M need not generate all linearizations, but at least one linearization of every causal structure. The known on-the-fly depth-first-search algorithms that avoid generating unnecessary interleavings can be combined with the construction of the automaton $A_{\neg\varphi}$ for model checking.

References

[1] R. Alur, K. McMillan and D. Peled. Model-checking of correctness conditions for concurrent objects, *11th Symposium on Logic in Computer Science*, 219–228, 1996.

[2] R. Alur, W. Penczek, and D. Peled. Model-checking of causality properties. *10th Symposium on Logic in Computer Science*, 90–100, 1995.

[3] E.M. Clarke and E.A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. *Workshop on Logic of Programs*, LNCS 131, 52–71, 1981.

[4] W. Ebinger. Logical definability of trace languages. In V. Diekert, G. Rozenberg (Eds.) *The Book of Traces*, World Scientific, 382–390, 1995.

[5] P. Godefroid and P. Wolper. A partial approach to model checking. *Information and Computation* 110 (2), 305–326, 1994.

[6] G. Holzmann and D. Peled. The state of SPIN. *8th Conference on Computer-Aided Verification*, LNCS 1102, 385–389, 1996.

[7] S. Katz and D. Peled. Interleaving set temporal logic. *Theoretical Computer Science* 75, 21–43, 1992.

[8] O. Lichtenstein and A. Pnueli. Checking that finite-state concurrent programs satisfy their linear specification. *11th ACM Symposium on Principles of Programming Languages*, 97–107, 1984.

[9] K. Lodaya, R. Parikh, R. Ramanujam, and P.S. Thiagarajan. A logical study of distributed transitions systems. *Information and Computation* 119, 91–118, 1985.

[10] A. Mazurkiewicz. Trace Theory. In W. Brauer, W. Reisig, G. Rozenberg (eds.), *Advances in Petri Nets 1986*, LNCS 255, 279–324, 1987.

[11] K.L. McMillan. Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits. *Fourth Conference on Computer-Aided Verification*, LNCS 663, 164-177, 1992.

[12] D. Peled. Combining partial order reductions with on-the-fly model checking. *Sixth Conference on Computer Aided Verification*, LNCS 818, 377–390, 1994.

[13] A. Pnueli. The temporal logic of programs. *18th Symposium on Foundations of Computer Science*, 46–77, 1977.

[14] W. Penczek and R. Kuiper. Traces and logic. In V. Diekert, G. Rozenberg (eds.) *The Book of Traces*, World Scientific, 307–390, 1995.

[15] W. Penczek. On undecidability of propositional temporal logics on trace systems. *Information Processing Letters* 43, 147–153, 1992.

[16] D. Peled and A. Pnueli. Proving partial order properties. *Theoretical Computer Science* 126, 143–182, 1994.

[17] S. Pinter and P. Wolper. A temporal logic for reasoning about partially ordered computations. *Third Symposium on Principles of Distributed Computing*, 28–37, 1984.

[18] V.R. Pratt. Modeling concurrency with partial orders. *Intl. J. of Parallel Programming* 15 (1), 33–71, 1986.

[19] P.S. Thiagarajan. A trace based extension of linear time temporal logic. *Ninth Symposium on Logic in Computer Science*, 1994.

[20] P.S. Thiagarajan and I. Walukiewicz. An expressively complete linear time temporal logic for Mazurkiewicz traces. *12th Symposium on Logic in Computer Science*, 1997.

[21] A. Valmari. A Stubborn attack on state explosion. *Proc. 2nd Conference on Computer-Aided Verification*, LNCS 531, 156–165, 1990.

[22] M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. *First Symposium on Logic in Computer Science*, 332–344, 1986.

[23] W. Zielonka. Notes on finite asynchronous automata. *R.A.I.R.O.-Informatique Théorique et Applications* 21, 99–135, 1987.