Undecidability of Partial Order Logics

Rajeev Alur
Dept. of CIS,
Univeristy of Pensylvania,
Philadelphia, PA, 19104, USA

Doron Peled
Bell Laboratories
600 Mountain Ave.
Murray Hill, NJ 07974, USA

Abstract

In this paper we prove the undecidability of certain temporal logics over partial orders. The proof we use is generic and can be applied to characterize various termpoal operators that cannot be used for automatic verification (model checking). We propose that these results partially explain why the partial order model of concurrency has not gained as much popularity as the interleaving model.

1 Introduction

The *causal order* or *partial order* semantics for concurrent systems [8, 13] often challenges the frequently used *interleaving* or *total order* model. The former model has the advantage that it does not assume a fictive order between concurrently executed events. Hence, it might be considered a more intuitive model for concurrency. Yet, to date, most of the work that has been done in concurrency uses the total order model. One reason for this is that the total order model is easier to deal with, with simpler mathematical tools.

Recently, a wealth of temporal logics that are based on the partial order model were suggested [6, 3, 14]. These logics are usually just a simple and natural adaptation of existing temporal logics, such as LTL [12] or CTL [5] to the model of partially ordered execution. A careful analysis reveals that for some specifications written in these logics, the automatic verification of finite state systems is either undecidable, or unreasonably complex.

In this paper we sharpen some undecidability results for partial order logics [9]. We show undecidability of some simple propositional temporal logics even under a simple locality constraint. This constraint imposes that these logics allow only local propositions, each describing properties of a single concurrent agent (process). Several temporal logics for partial order executions where shown to be undecidable without the locality assumption [9]. Yet, the same logics become decidable, as shown in [3, 14] when the locality assumption was imposed. We show that even under this assumption, our logics (and even some minimal fragments of them) remain undecidable.

We regard the proof, rather than the particular theorems and logics we discuss, to be the main contribution of this paper. It allows to assess a generic kind of temporal operators that cannot be used to assert about partially ordered executions. Intuitively, such undecidable operators reason about a linearization generated from a partial order execution. The advantage of using such an operator is that it reflects a single 'convenient' interleaving that is consistent with the partially ordered execution (i.e., consists of the completion of the total order into a partial order). Some phenomena in concurrency theory are naturally described in terms of such interleavings. For example, (commutativity based) serializability [4] asserts that there exists such an interleaving in which database transactions are executed one at a time (although other interleavings consistent with the partial order do exist). Our results show that asserting about *two* such linearizations that are consistent with a single partial order execution results in undecidability.

2 Preliminaries

2.1 A partial order model for concurrency

A partial order structure is a pair $(S, \prec)$, where $\prec$ is a transitive, irreflexive and asymmetric relation. When $s_1 \prec s_2$, we say that s_1 (causaly) *precedes* s_2. When $s_1 \not\prec s_2$ and $s_2 \not\prec s_1$, we say that s_1 and s_2 are *concurrent*. The *history* of an element $s \in S$ is the set of elements preceding s, i.e., $\{s' | s' \prec s\}$, and is denoted by $s \downarrow$.

Partial order structures that model executions of concurrent systems usually satisfy some additional properties.

Finite history For each $s \in S$, $s \downarrow$ is finite.

Finite concurrency There exists some constant c such that there is no set $C \subseteq S$ with c elements such that every pair of elements in C are concurrent.

A *configuration* of $(S, \prec)$ is a finite set of elements $C \subseteq S$ that is history closed. Namely, for each $s \in C$, $s \downarrow \subseteq C$. The set of configurations $\mathcal{C}$ of a partial order $(S, \prec)$ forms a partial order under set inclusion. Notice that $(\mathcal{C}, \subseteq)$ is *directed*, i.e., for each C_1, C_2 in $\mathcal{C}$, there exists some $C \in \mathcal{C}$ such that $C_1 \subseteq C$ and $C_2 \subseteq C$.

When modeling executions of concurrent systems using partial orders, the set S is usually used to model the *local states* (or the local events) of the system. Namely, each state represents some instanteneous description of one of the concurrent agents or processes. The configurations in $\mathcal{C}$ usually model the *global states*. Each configuration represents a consistent description of a system, where a subset of the states in S that are included in the configuration have already occurred. Any finite subset of S that is not a configuration is inconsistent, since it must include some state, but exclude another state that has happened before. Usually the relevant information in a configuration C is its set of maximal (with respect to the order $\prec$) elements $max(C)$.

When reasoning about the execution of concurrent systems, e.g., using temporal logic, we need to define the basic structure, and temporal relation used. With partially ordered executions, one has the choice of directly using the partial order structure $(S, \prec)$, as done e.g., in [3, 11], or using the global states constructed via $(\mathcal{C}, \subseteq)$, as done e.g., in [6, 14]. In this paper, we deal with logics over global states, hence we use the latter structure.

Propositional temporal logics use a finite set of predicates AP, each one of which represents some property of the global state. A labeling function $\mathcal{L} : \mathcal{C} \mapsto 2^{AP}$ assigns to each global state the set of predicates from AP that hold in it.

In this paper we add an important *locality* constraint on the predicate:

> Let L be a local labeling function $L : S \mapsto 2^{AP}$. Then, there is a partition of AP into $AP_1 \uplus AP_2 \uplus \ldots \uplus AP_n$. Each disjoint subset AP_i contains exactly one *process proposition* p_i. (Thus, each process P_i has its own set of predicates AP_i.) The global labeling function $\mathcal{L}$ is formed from the local one by setting
>
> $$\mathcal{L}(C) = \bigcup \{L(s) | s \in max(C)\}$$

Normally, all the local states of a single process (i.e., the states annotated by some common process predicate p_i) are totally ordered.[1]

[1] In some models of concurrency, the sets $AP_1 \ldots AP_n$ are not disjoint. We regard this as a technical issue about how concurrency is modeled. Nevertheless, our results are demonstrated using a simple case in which this distinction is irrelevant.

Define now the successor relation $\triangleright \subseteq \mathcal{C} \times \mathcal{C}$ such that $C_1 \triangleright C_2$ iff $C_2 = C_1 \cup \{s\}$ for some $s \in S$. An *execution sequence* $C_0\, C_1\, C_2 \ldots$ of $(\mathcal{C}, \subseteq)$ satisfies that $C_0 = \emptyset$ and for each $i \geq 0$, $C_i \triangleright C_{i+1}$.

2.2 ISTL

The formulas of ISTL are defined inductively by the grammar

$$\varphi \; := \; p \mid \neg\varphi \mid \varphi \wedge \varphi \mid EF\varphi \mid EX\varphi \mid \varphi EU\psi \mid EG\varphi$$

where $p \in AP$. Given a structure $\mathcal{M} = (\mathcal{C}, \subseteq)$, the formulas of ISTL are interpreted over the configurations in $\mathcal{C}$.

The interpretation of modalities is as in CTL. Let φ be an ISTL-formula, and C be a configuration of $\mathcal{M}$. Then, the satisfaction relation $\mathcal{M}, C \models \varphi$ is defined inductively below. We denote $C \models \varphi$ as a short form for $\mathcal{M}, C \models \varphi$, when $\mathcal{M}$ is clear from the context.

- $C \models p$ iff $p \in \mathcal{L}(C)$ for an atomic proposition p.
- $C \models \neg\varphi$ iff $C \not\models \varphi$.
- $C \models \varphi \wedge \psi$ iff $C \models \varphi$ and $C \models \psi$.
- $C \models \varphi \vee \psi$ iff either $C \models \varphi$ or $C \models \psi$.
- $C \models EX\varphi$ iff $C \triangleright C'$ and $C' \models \varphi$ for some configuration C'.
- $C \models EF\varphi$ iff $C \subseteq C'$ and $C' \models \varphi$ for some configuration C'.
- $C \models \varphi EU\psi$ iff there exists a finite sequence $C_0 \ldots C_n$ of configurations such that (i) $C_0 = C$, (ii) $C_i \triangleright C_{i+1}$ for $0 \leq i < n$, (iii) $C_n \models \psi$, and (iv) $C_i \models \varphi$ for all $0 \leq i < n$.
- $C \models EG\varphi$ iff there exists an infinite sequence $C_0 C_1 \ldots$ of configurations such that (i) $C_0 = C$, (ii) $C_i \triangleright C_{i+1}$ for all i, and (iii) $C_i \models \varphi$ for all i.

We define $\mathcal{M} \models \varphi$ iff $\mathcal{M}, \emptyset \models \varphi$. A concurrent system is modeled by a *set of partial order executions.* Thus, a system $\mathcal{S}$ satisfies an ISTL property φ iff $\mathcal{M} \models \varphi$ for each $\mathcal{M}$ in $\mathcal{S}$.

3 Undecidability Results

In this section we will show some undecidability results for various fragements of ISTL. The proofs will be by a reduction from *Post Correspondence Problem.*

> Let PCP be a finite set of pairs of strings of the form (u_m^1, u_m^2) (we call u_m^1 a *left* word, and u_m^2 a *right word* of PCP) over some alphabet Σ, where $1 \leq m \leq n$ for some n. A *match* is a finite sequence of pairs $(u_{i_1}^1, u_{i_1}^2)$, $(u_{i_2}^1, u_{i_2}^2)$, ..., $(u_{i_k}^1, u_{i_k}^2) \in PCP$ such that for $w^1 = u_{i_1}^1 u_{i_2}^1 \ldots u_{i_k}^1$ and $w^2 = u_{i_1}^2 u_{i_2}^2 \ldots u_{i_k}^2$, we have $w^1 = w^2$. The post correspondence problem is to determine whether such a match exists for PCP.

In the sequel, assume that $\Sigma = \{0, 1\}$. We will call w^1 and w^2 the *matching words.* We will show undecidability for two kinds of problems:

Model checking Decide whether some finite state system, described using some finite state automata, satisfies a given formula φ.

Satisfiability Decide whether a given formula φ has a satisfying model.

We start with the model checking problem. We construct a finite state system consisting of two completely independent processes. The process P^1 will describe a sequence of left words from PCP, while P^2 describes a sequence of right words. The automata P^1 and P^2 are completely independent, i.e., represent concurrent processes that never synchronize with each other. Each edge of such an automaton is annotated with predicates from the following set:

b^i A (left when $i = 0$ or write when $i = 1$) word of PCP begins at the current local state.

f^i A (left when $i = 0$ or write when $i = 1$) word of PCP finishes at the current local state.

e^i The last state of process P^i.

l^i The least significant bit of a modulo 4 counter is 1, otherwise, it is 0.

h^i The most significant bit of a modulo 4 counter is 1, otherwise, it is 0.

v^i The letter corresponding to this state is 1, otherwise, it is 0. In a state satisfying e^i, v^i does not hold.

The automaton P^i is a synchronous product of two simple automata $W^i \times M^i$. That is, each step of P^i consists of one step from each component automaton.

The automaton W^i contains a single initial state. For each (left, for $i = 0$ or right, for $i = 1$) word u^i_j appearing in some pair of PCP, the automaton for P^i contains a disjoint sequence of transitions, starting from the initial state, that recognize u^i_j and returns to the initial state. There is an additional edge from the initial state, labeled with a special symbol e^i into a terminal accepting node. Each edge that exists the initial state, except for the latter, is annotated with the predicate b^i. Each edge that enters the initial state is annotated with f^i. In addition, each edge is also annotated with propositions that encode the word's index, 1 to k. The left automaton in Figure 1 describes the structure of this automaton. The letters for the words $u^i_1, \ldots, u^i_k$ are omitted from the figure.

The automaton M^i uses the predicates h^i and l^i, to count modulo 4. This automaton appears on the right side of Figure 1. The synchronous product $W^i \times M^i$ is an automaton that recognizes a sequence of left (when $i = 0$) or right (when $i = 1$) words from PCP, where b^i corresponds to the beginning of each single word from a PCP pair, e^i corresponds to the end of the sequence, and the predicate h^i and l^i count modulo 4.

Figure 2 includes two such words, generated by P^1 and P^2. In each box, representing a local state, we have on top left the modulo 4 count, and top right a binary representation of the index of the word (in the figure, we denote the decimal value). The current letter appears on the bottom left. The bits b^i and e^i, representing the beginning of a word, or the end of the sequence, respectively, appear on the bottom right.

We use an EG formula to assert that these two words match letter by letter. In order to shorten the presentation, we will use some predicates with boolean variables, and assume the knowledge of propositional logic needed to fill in the actual formula. This formula will describe a zigzaging path, as shown using the dotted line in Figure 2. It will use the modulo 4 numbering to impose such a path. (Notice that a modulo 2 numbering would not be sufficient, as the zigzagin path includes all the 4 combinations, and hence cannot be distinguished from any other path.) For any global state where the numbering of the states in both paths is the same, we also require that the letters are the same.

Denote by $bin(h, l)$ a function that returns the value obtained by evaluating the Boolean variable h as the most significant bit, and l as the least

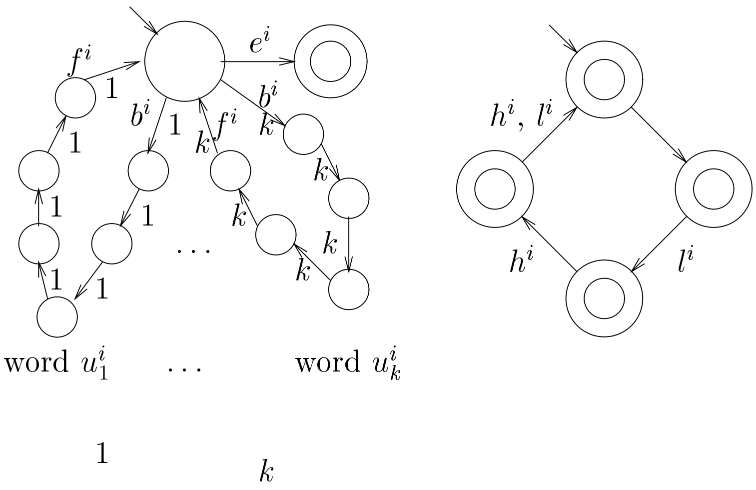

Figure 1: The two automata whose product is P^i

significant bit of a modulo 4 counter. We consider *false* as 0 and *true* as 1. Let $\oplus_4$ be a modulo 4 addition.

Another EG formula is needed to show that the two words w_1 and w_2 consist of concatenations of left and right words that pairwise agree on their indexes. Thus, this formula will describe that we can form a linearization in which we progress through a single pair of PCP words with the same index at a time. Without loss of generality, we assume that each PCP word is of length at least 3 (otherwise, repeat each letter three consecutive times).

The appropriate sequence satisfies that $\neg(b^2 \wedge f^1) \wedge f^2 \rightarrow (f^1 \vee b^1 \vee e^1) \wedge (b^1 \rightarrow (f^2 \vee b^2))$. Thus, it must start with a state satisfying $b^1 \wedge b^2$. Then, a progress has to occur from the local state satisfying b^1, otherwise $(b^1 \rightarrow (f^2 \vee b^2))$ would be falsified (recall that since each PCP word is at least 3 characters long, after a b^2 state, a state satisfying $\neg b^2 \wedge \neg f^2$ occurs). The b^2 local state must be abandoned before an f^1 state occurs, since $\neg(b^2 \wedge f^1)$ holds. In order to satisfy $f^2 \rightarrow (f^1 \vee b^1 \vee e^1)$, an f^1 state precedes the f^2 state. If the current two PCP words on the sequence are not the last ones, we must have a $f^1 \wedge f^2$ state followed by a $b^1 \wedge f^2$ state, followed by a $b^1 \wedge b^2$ state. The sequence then continues in the same way. Thus, the sequence progresses from one PCP pair (u_l^1, u_l^2) to another PCP pair (u_m^1, u_m^2) via

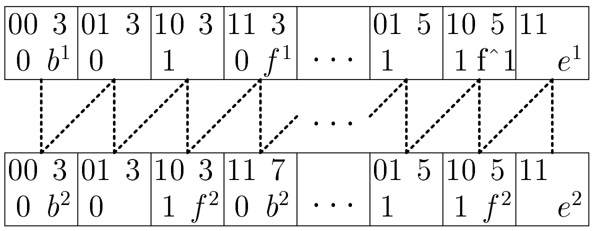

Figure 2: The first linearization

states satisfying $b^1 \wedge f^2$. Such states, and states where either e^1 or e^2 holds are the only ones permitted to have different indexes as shown below. THe conjunct $(e^1 \rightarrow e^2)$ forces that the number of pairs of words on both sides will be the same. Some of the allowed combination of local states that form global states are shown in Figure 3.

Similarly to the use of propositions to encode a binary value using $bin(h, l)$, we define by num^1 an encoding of Boolean propositions that represent the index of the current left word, and similarly by num^2 the index of the right word. A sufficient number of propositions should be available. Then, the required ISTL formula consists of the following two EG conjuncts:

$$
\begin{array}{rl}
EG & (((bin(h^1, l^1) = bin(h^2, l^2)) \vee \\
& (bin(h^1, l^1) = bin(h^2, l^2) \oplus_4 1)) \wedge \\
& ((bin(h^1, l^1) = bin(h^2, l^2) \wedge \neg e^1 \wedge \neg e^2) \rightarrow (v^1 \leftrightarrow v^2))) \\
\wedge EG & (\neg(b^2 \wedge f^1) \wedge (f^2 \rightarrow (f^1 \vee b^1 \vee e^1)) \wedge (b^1 \rightarrow (f^2 \vee b^2)) \\
& (e^1 \vee e^2 \vee (f^2 \wedge b^1))) \vee num^1 = num^2
\end{array}
$$

This formula is of the form $EG\psi_1 \wedge EG\psi_2$. It is easy to see that one can use also the EU modality, instead of the EG. The formula will have the form $\psi_1 EU(e^1 \wedge e^2) \wedge \psi_2 EU(e^1 \wedge e^2)$.

To show undecidability of the satisfiability problem in ISTL, we only need to describe the automata that produces the checked structure by a formula η in the appropriate logic. We then need to check the satisfiability of $\eta \wedge \varphi$, Recall that a process predicate p^i is assigned to each state that belong to a process P^i. We will list the additional conjuncts needed, but will not provide the complete ISTL formulas. These are straightforward, but tedious.

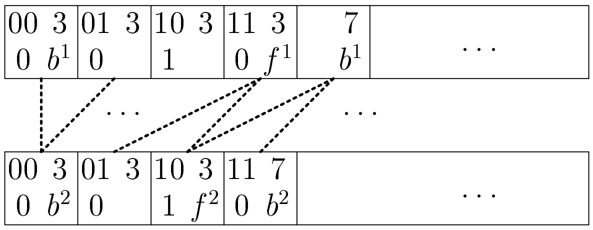

Figure 3: The second linearization

- The bits h^i and l^i on the states of process P^i (namely, those labeled with p^i) constitute a modulo 4 counter.
- The processes P^1 and P^2 are completely concurrent. One can express this by asserting that in every state that does not include e^1 or e^2, one can make both a P^1 move *and* a P^2 move.
- The initial state is annotated with b^1 and with b^2. A state annotated by f^i must follow a state annotated by b^i or by e^i. A state annotated with e^i has no successor under process p^i.
- Consider the states of one process, from any state where b^i holds to a state before b^i or e^i holds, Then the annotation of these states with proposition v^i representing 1, and without v^i, representing 0 forms a (left for $i = 0$ or right for $i = 1$) word of PCP.

4 Conclusions

We showed that the model checking and the satisfiability problem for some logics that express properties of partial order executions is undecidable. This is true even under a simple and natural locality assumption, that allow assigning propositions to the local states.

In a previous proof [9], it was shown that a logic such as ISTL is undecidable, even when using only the AG, EF and EX operator are used (the proof uses some index on EX to denote the process that is making the progress, but this can be eliminated with appropriate process predicates). However, this was done without the locality assumption. In [2, 14], logics with the locality assumption that include the AG, EF and EX operator where shown to be decidable. In fact, in [2], it was shown that using only these modalities, we obtain an EXPSPACE decision procedure. In [15], this was shown to be also the lower bound for such a logic. The logic TRPTL of [14] adds a decidable *until* operator, which is shown in [15] to make the decidability problem of this logic nonelementary. The question of a logic with only the EG, AG and EX was left open in [2].

In this paper, we showed that various attempts to add a modality that would reason about the states of a *possible path of global states* in a partial order execution result in an undecidable logic. This does not rule out the possibility to decide related properties such as serializability. In [1] it was shown how to use automata based decision procedure for deciding serializability, while the decidable local temporal logic of [3, 11], which uses the local states of partial ordered structures $(S, \prec)$, can also be used to express the property.

The subtle observation is that one can assert about the global states of partial order executions, as long as there is no need to assert about the *continuous* behavior of execution sequences. Properties such as serializability conform with this restriction. What is required is to assert that the system moves from one state in which no transaction is active, to another state in which no transaction is active, by doing the transitions of a single transaction. When one wants also to assert that during this progress some global predicate holds during one interleaving consisting with the system behavior, (but may fail to hold in another), this property becomes undecidable.

Acknowledgement

As in many cases, this paper begun with a discussion and a question posed by Amir Pnueli.

References

[1] R. Alur, K. McMillan and D. Peled, Model-checking of Correctness Conditions for Concurrent Objects, *LICS96, 11th IEEE Symposium on Logic in Computer Science*, 1996, 219–228, New Brunswick, NJ, USA.

[2] R. Alur, K. McMillan and D. Peled, Deciding global partial-order properties, ICALP'98, Aalborg, Denmark, July, 1998, Lecture Notes in Computer Science 1443, Springer, 41-52.

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

[4] P.A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency control and Recovery in Database Systems.* Addison-Wesley, 1987.

[5] E. A. Emerson and E. M. Clarke, Characterizing Correctness Properties of Parallel Programs Using Fixpoints, Automata, Languages and Programming, LNCS 85, Springer, 1980, 169-181.

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

[7] S. Katz and D. Peled, Verification of Distributed Programs Using Representative Interleaving Sequences, *Distributed Computing*, 6(1992), 107–120.

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

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

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

[11] S. Pinter and P. Wolper. A temporal logic for reasoning about partially ordered computations. *3rd ACM Symposium on Principles of Distributed Computing*, 28–37, 1984.

[12] A. Pnueli, The temporal logic of programs, *18th FOCS, IEEE Symposium on Foundation of Computer Science*, 1977, 46–57.

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

[14] P.S. Thiagarajan, I. Walukiewicz, An expressively complete linear time temporal logic for Mazurkiewicz traces, LICS'97, IEEE, 1997, 183–194.

[15] I. Walukiewicz, Difficult Configurations – On the Complexity of LTrL, ICALP'98, Aalborg, Denmark, July, 1998, Lecture Notes in Computer Science 1443, Springer, 140-151.