Logic-Based Diagnosis for Distributed Systems*

Shaofa Yang[1†], **Loïc Hélouët**[2] **Thomas Gazagnaire**[3‡]

[1] *UNU-IIST, Macao*
ysf@iist.unu.edu

[2] *INRIA Rennes, France*
loic.helouet @irisa.fr

[3] *Citrix Systems R&D Ltd., UK*
thomas.gazagnaire@citrix.com

Abstract

We address the problem of off-line fault diagnosis for distributed systems. It consists in finding explanations for a given partial observation of abnormal behaviour, using knowledge of system dynamics. For this, a diagnosis algorithm must decide whether there exists an execution that is compatible with our knowledge of the system and with the observation. We represent observations with restricted partial orders which model cause-effect relations among local states, and properties that hold at these states. We capture knowledge of system dynamics with a temporal logic which asserts the evolution of patterns of causal orders. We show that the corresponding diagnosis problem is undecidable. However, if we limit explanations to distributed behaviours in which each process causally influences every other process in a bounded manner, the restricted diagnosis problem becomes decidable.

Keywords: Diagnosis, partial orders, logic.

1 Introduction

For safety and economical reasons, diagnosis of faults is of paramount importance in domains such as telecommunication networks and embedded systems. Diagnosis can be performed off-line or online. In off-line diagnosis, the task is to infer missing information (unobserved

*work supported by the CREATE project of Region Bretagne.

†Work done while this author was at IRISA/INRIA Rennes, France supported by an INRIA post-doctoral fellowship.

‡Work done while this author was at ENS Cachan, antenne de Bretagne, France.

events or states, faulty behaviors,...) from a partial observation of a system, such as a log file. The observation has to be partial for two reasons. Firstly, some state information or actions may not be directly accessible. Secondly, due to the huge size of many distributed systems, it is simply not feasible to keep track of all state information. In online diagnosis, a system is continuously monitored and faults are supposed to be detected as soon as possible after their occurrence.

Traditionally fault diagnosis is performed by inductive reasoning, using expert heuristic rules between faults and observations. However, such expert knowledge is difficult to obtain and easily becomes obsolete when a system's configuration evolves. The so-called *model-based* approach brings more applicable solution to fault diagnosis. In this framework, one captures knowledge of system dynamics in some formal system model such as transition systems [17], Petri nets [2] or message sequence charts [10]. Faults are inferred from observation and the system model. One might be interested in determining if an unobserved fault has occurred, as in [17], or in finding all possible runs that may have led to the observation, as in [2]. Another objective ([8]) is to compute a summary of possible explanations, that is to annotate observations with information that help explaining what might have occurred. The additional information can be causal relations among observed events, known local properties of states or events in the observation, and events of interest not contained in the logged information.

One drawback of model-based diagnosis is that a complete model of the monitored system is not always available. Furthermore, it is difficult to update a system model when a system's configuration changes. Often, the only knowledge available for diagnosis consists of some partial *properties* of a system's behaviour. Considering this, we propose a logic-based approach to diagnosis. More precisely, we capture knowledge of a system's behaviour using formulae in some suitable temporal or modal logics. Temporal or modal logics enables one to specify properties of system's dynamics in a natural way. In many cases, updating properties of a system in the form of logic formulae can be done easily by changing a limited number of subformulae.

In this paper, we represent behaviours of distributed systems with restricted partial orders which define cause-effect relations among local states. These partial orders are called partially ordered computations. We propose a temporal logic over partially ordered computations and call it simply the *logic of partially ordered computations* (LPOC). The main feature of LPOC is to reason about evolution of patterns of causal orders. We use temporal operators

similar to Computation Tree Logic. The design of LPOC is motivated by the fact that, for many distributed systems such as network protocols, properties about their executions are often available in the form of "whenever this pattern of causal ordering occurs, some other pattern will follow in future", where patterns are usually short sequences of message exchanges.

We study off-line diagnosis based on LPOC. The problem is to determine whether there exists an explanation for a given observation and a given LPOC formula Φ describing a system's dynamic properties. An explanation is a distributed behaviour which could have given rise to the observation and which satisfies the formula Φ. We show that this problem is undecidable in general. The undecidability result is mainly due to the undecidability of satisfiability problem of LPOC. We note that the satisfiability problem of several similar temporal logics in the literature are undecidable. These include m-LTL [15], a local temporal logic on Lamport diagrams, and template message sequence charts [9]. However, for a given K, if we limit explanations to so-called K-influencing distributed behaviours in which each process causally influences every other process in a bounded manner, then the restricted diagnosis problem is decidable (for the given K). Furthermore, one can effectively compute a compact summary of all K-influencing explanations.

In the next section, we introduce the syntax and semantics of the logic LPOC. Section 3 defines the diagnosis problem associated with LPOC and show that it is undecidable. Section 4 establishes the decidability of the restricted diagnosis problem where only K-influencing explanations are considered. We also analyze the complexity of the decision algorithm. Section 5 discusses related work and postulates some future directions. To reduce clutter, some proofs are omitted, but can be found in an extended version in [19].

2 Logic of Partially Ordered Computations

Through the rest of the paper, we fix a finite nonempty set $\mathcal{P}$ of process names, and $\mathcal{A}$ a finite nonempty set of atomic propositions. We let p, q range over $\mathcal{P}$.

Definition 1. *A* partially ordered computation *(or computation for short) over $(\mathcal{P}, \mathcal{A})$ is a tuple $(S, \eta, \leq, V)$ where:*

- *S is a finite set of* (local) states.
- *$\eta : S \to \mathcal{P}$ identifies the* location *of each state. For each $p \in \mathcal{P}$, we define $S_p = \{s \in S \mid \eta(s) = p\}$.*

- $\leq \subseteq S \times S$ *is a partial order, called the* causality *relation. Furthermore, for each* p, $\leq$ *restricted to* $S_p \times S_p$ *is a total order.*
- $V : S \to 2^{\mathcal{A}}$ *is a labeling function which assigns a set of atomic propositions to each state. We call* $V(s)$ *the* valuation *of* s.

Intuitively, a computation (also called Lamport diagram in the literature [15]) represents the causal ordering among local states in a distributed execution, in which states of each process are sequentially ordered. The valuation of local state s collects the atomic propositions that hold at s. Figure 1-a) shows a computation. States are designated by black dots with associated name $s_1, \ldots, s_6$. Processes P, Q, R are represented by vertical lines, and states located on a process line are ordered from top to bottom. Finally, valuations of states take value in $\{a, b, c\}$, and are represented between two brackets near the associated state. Note that we consider only *finite* computations, since we address only offline diagnosis in this paper. We will say that two computations $(S, \eta, \leq, V)$ and $(S', \eta', \leq', V')$ are *isomorphic* iff there is a bijection $f : S \to S'$ such that $\eta(s) = \eta'(f(s))$ for any $s \in S$, $s_1 \leq s_2$ iff $f(s_1) \leq' f(s_2)$ for any $s_1, s_2 \in S$, and $V(s) = V'(f(s))$ for any $s \in S$. We identify isomorphic computations and write $W \equiv W'$ if $W$and W' are isomorphic.

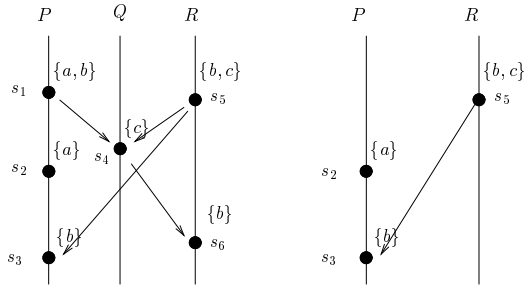

Figure 1: a) A computation W b) the 1-view of s_3 in W

Let $(S, \eta, \leq, V)$ be a computation, and $s, s' \in S$. As usual, we write $s < s'$ when $s \leq s'$ and $s \neq s'$. For each $p \in \mathcal{P}$, we define $\ll_p \subseteq S \times S$ as: $s \ll_p s'$ iff $s, s' \in S_p$, $s < s'$, and there does not exist $s'' \in S_p$ with $s < s'' < s'$. That is, $\ll_p$ is the "immediate" sequential ordering of states belonging to p. We let $\lessdot \subseteq S \times S$ be the least relation such that $\leq$ is the reflexive and transitive closure of $\lessdot$. For each $p, q \in \mathcal{P}$ with $p \neq q$, let us define $\ll_{pq} \subseteq S \times S$ as follows: $s \ll_{pq} s'$ iff $s \in S_p$, $s' \in S_q$, and $s \lessdot s'$. We also define $\ll = (\cup_{p \in \mathcal{P}} \ll_p) \bigcup (\cup_{p,q \in \mathcal{P}, p \neq q} \ll_{pq})$. We note that $<$ is in fact the

transitive closure of $\ll$. If $s \ll s'$, we say s' is a *(causal) successor* of s, and call s a *(causal) predecessor* of s'. We emphasize that $\ll$ is not equal to $\lessdot$. Indeed, $\lessdot$ does not necessarily capture the relations defined by the local ordering on processes. Consider for instance states s_5 and s_6 in Figure 1-a: we have $s_5 \ll s_6$, but not $s_5 \lessdot s_6$. A state s is *minimal* if it has no predecessor, and *maximal* if it has no successor. A *causal chain* is a sequence $s_1 s_2 \ldots s_n$ of states where $s_1 \ll s_2 \ll \ldots \ll s_n$.

In the sequel, we define a temporal logic of partial-order computations (called "LPOC" for short) to reason about distributed behaviors. It has two basic features. First, at a state s of a computation, atomic formulae assert that a "pattern" occurs in a bounded past or bounded future of s. Secondly, we consider a branching time framework with CTL-like operators and reason along sequences of causally ordered states.

Definition 2. *Let $(S, \eta, \leq, V)$ be a computation, and m be a natural number. The* m-view *of $s \in S$, denoted $\downarrow_m(s)$, is the collection of states s' in S such that there exists a causal chain of length at most m starting from s' and ending at s. More precisely, $\downarrow_m(s) = \{s' \mid \exists s_0, \ldots s_n \in S, n \leq m$ and $s' = s_0 \ll s_1 \ll \ldots \ll s_n = s\}$. Similarly, the* m-frontier *of s, denoted by $\uparrow_m(s)$, is the collection of states s' in S such that there exists a causal chain of length at most m starting from s and ending at s'.*

Figure 1-b) shows an example of m-view. Note that the 0-view and 0-frontier of a state s are both the singleton set $\{s\}$. Each state s has at most $|\mathcal{P}|$ successors, one belonging to each S_p. Thus, inductively, the m-view and m-frontier of s contains at most $\mathcal{N}_m = \sum_{i=0}^{m} |\mathcal{P}|^i = \frac{1-|\mathcal{P}|^{m+1}}{1-|\mathcal{P}|}$ states. In order to reason about the "pattern" of a computation, we also need a notion of *projection.*

Definition 3. *Let $W = (S, \eta, \leq, V)$ be a computation over $(\mathcal{P}, \mathcal{A})$, and let $A \subseteq \mathcal{A}$. The* projection *of W onto A is the computation $W' = (S', \eta', \leq', V')$ where $S' = \{s \in S \mid V(s) \cap A \neq \emptyset\}$, and η', $\leq'$, are the respective restrictions of $\eta, \leq$ to S' and $V'(s) = V(s) \cap A$ for every $s \in S'$.*

The atomic formulae of our logic will assert that the projection of the computation formed from the m-view or the m-frontier of a state is isomorphic to a given computation. We are now ready to define the logic LPOC.

Definition 4. *The set of LPOC formulae over a set of processes $\mathcal{P}$ and a set of atomic propositions $\mathcal{A}$, is denoted by $LPOC(\mathcal{P}, \mathcal{A})$, and is inductively defined as follows:*

- *For each $p \in \mathcal{P}$, the symbol loc_p is a formula in $LPOC(\mathcal{P}, \mathcal{A})$.*
- *Let m be a natural number, A be a subset of $\mathcal{A}$, and $T = (S, \eta, \leq, V)$ be a computation such that $V(s) \subseteq A$ for every $s \in S$. Then $\downarrow_{m,A}(T)$, $\uparrow_{m,A}(T)$ are formulae in $LPOC(\mathcal{P}, \mathcal{A})$.*
- *If φ, φ' are formulae in $LPOC(\mathcal{P}, \mathcal{A})$, then $\mathsf{EX}\varphi$, $\mathsf{EU}(\varphi, \varphi')$ are formulae in $LPOC(\mathcal{P}, \mathcal{A})$.*
- *If $\varphi, \varphi' \in LPOC(\mathcal{P}, \mathcal{A})$, then $\neg\varphi$ and $\varphi \vee \varphi'$ are formulae in $LPOC(\mathcal{P}, \mathcal{A})$.*

From now on, we shall refer to formulae in $LPOC(\mathcal{P}, \mathcal{A})$ simply as *formulae*. Their semantics is interpreted at local states of a computation. For a computation W and a given state s of W, we write $W, s \models \phi$ when W satisfies ϕ, which is defined inductively as follows:

- $W, s \models loc_p$ iff the location of s is p (i.e. $\eta(s) = p$),
- $W, s \models \downarrow_{m,A}(T)$ iff the projection of $\downarrow_m(s)$ onto A is isomorphic to T.
- $W, s \models \uparrow_{m,A}(T)$ iff the projection of $\uparrow_m(s)$ onto A is isomorphic to T.
- $W, s \models \mathsf{EX}\varphi$ iff there exists a state s' in W such that s' is a causal successor of s and $W, s' \models \varphi$.
- $W, s \models \mathsf{EU}(\varphi, \varphi')$ iff there exists a causal chain $s_1 s_2 \ldots s_n$ in W with $s = s_1$. Further, there exists an index i in $\{1, 2, \ldots, n\}$ with $W, s_i \models \varphi'$, and $W, s_j \models \varphi$ for every j in $\{1, 2, \ldots, i-1\}$.

The semantics for boolean combinations and negations of formulae is as usual. We assume the standard boolean operators. We define some derived temporal operators as follows: $\mathsf{EF}\varphi \equiv \mathsf{EU}(\mathit{true}, \varphi)$, $\mathsf{EG}\varphi \equiv \mathsf{EU}(\varphi, \varphi \wedge \neg\mathsf{EX}\mathit{true})$, $\mathsf{AX}\varphi \equiv \neg\mathsf{EX}(\neg\varphi)$, $\mathsf{AF}\varphi \equiv \neg\mathsf{EG}(\neg\varphi)$, and $\mathsf{AG}\varphi \equiv \neg\mathsf{EF}\neg\varphi$. We can also assert the truth of an atomic proposition a at a state of a computation with $\varphi_a = \bigvee_{p \in \mathcal{P}} \big(loc_p \wedge \downarrow_{0,\{a\}}(T_{p,a})\big)$, where each $T_{p,a}$ is the computation containing a singleton state of location p and valuation $\{a\}$.

For a computation W and a LPOC formula φ, we say that W *satisfies* φ, written $W \models \varphi$, iff there exists some minimal state s_{min} of W such that $W, s_{min} \models \varphi$. We say that φ is *satisfiable* iff there exists a computation W such that $W \models \varphi$.

For application to diagnosis, it is useful to define the notion of a computations satisfying a collection of formulae, one for each process. Formally, for a computation $W = (S, \eta, \leq, V)$ and a $\mathcal{P}$-indexed family of formulae $\{\varphi_p\}_{p \in \mathcal{P}}$, we say W satisfies $\{\varphi_p\}_{p \in \mathcal{P}}$, written $W \models \{\varphi_p\}_{p \in \mathcal{P}}$ by abuse of notation, iff the following condition

holds: for each $p \in \mathcal{P}$, $S_p \neq \emptyset$ and $W, s_p \models \varphi_p$ where s_p is the minimum state in S_p (i.e. $s_p \leq s$ for every $s \in S_p$). Note that $W \models \{\varphi_p\}_{p\in\mathcal{P}}$ iff $W \models \bigwedge_{p\in\mathcal{P}} \mathsf{EU}(\neg loc_p, loc_p \wedge \varphi_p)$.

We have chosen the existential until operator because it is essential in asserting properties such as "whenever some pattern T occurs, some other pattern T' will follow". More precisely, this demands that along every causal chain, whenever a pattern T occurs, pattern T' should occur later *and* no more pattern T can occur again before the point at which the pattern T' has occurred. This kind of properties are commonly needed in practical applications.

Let us define a simple example with LPOC. We define a formula meaning that whenever a connection phase described by a pattern T_{conn} occurs between two processes *Client* and *Server*, then a data transfer described by a pattern T_{data} necessarily occurs later. This formula can be expressed by $AG\varphi$, where :
$\varphi = (loc_{Client} \wedge \uparrow_{2,A}(T_{conn})) \Longrightarrow$
$(EX(loc_{Client} \wedge EX(EU(loc_{Client}, loc_{Client} \wedge \uparrow_{2,A'}(T_{data}))))$, where the patterns T_{conn} and T_{data} are described in Figure 2,
$A = \{disc, noclient, client, connected\}$, and
$A' = \{DataSent, DataRecv, DataAck\}$.

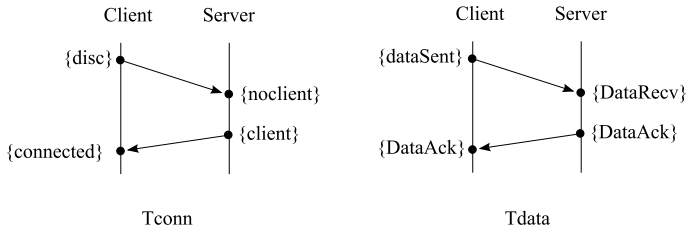

Figure 2: Two patterns

3 Diagnosis

An usual framework to perform diagnosis (see for instance [2, 10, 17]) in a distributed system is as follows. A central agent, called the *diagnoser*, collects information from some processes in the system. Each process is equipped with mechanisms that signal information to the diagnoser. This equipment can be implemented by means of code instrumentation, or by hardware mechanisms that raise alarms and sends them to the diagnoser. Of course, due to the size of runs of real systems, the diagnoser only collects a limited subset of what really

occurs in the system. The collected information on observed states and causal dependencies is called an *observation*. From this observation and a model of the system, the diagnoser can then output an explanation of what have been observed. This generic framework is depicted in Figure 3: processes are represented by squares, connected by communication links. Black squares symbolize the local observation mechanisms, that send their observations to the diagnoser, symbolized by the central ellipse.

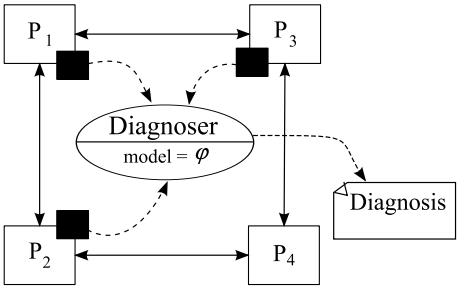

Figure 3: A diagnosis framework

Intuitively, an observation describes everything that is "recorded" during the execution of some prefix of a computation. We assume that the monitoring mechanisms transmit "recorded" observable atomic propositions to the central diagnosis system. Only records of the same process are guaranteed to be received by the central diagnosis system in the order they were sent. In addition to local observation on each process, observations contain some causal ordering among observed states located on different processes. This allows finest descriptions of observed behaviors, and to rule out some explanations that might be compatible with the shuffle of local observations during diagnosis. This causal ordering can be inferred if observations are tagged for instance with vectorial clocks [14, 6]. Nevertheless, the observation architecture and the way computations are logged is out of the scope of this work.

Our goal is to use LPOC for diagnosis. We shall represent knowledge of system dynamics by a $\mathcal{P}$-indexed family $\{\Phi_p\}_{p\in\mathcal{P}}$ of formulae. We also record observation of partial execution in the form of a computation. The objective of diagnosis is to find explanations, also in the form of computations, which could have led to the observation and satisfies the $\mathcal{P}$-indexed family of formulae. As there could be many explanations, it is also desirable to compute summarative information of the collection of explanations.

To this end, we fix $\mathcal{A}_{ob} \subseteq \mathcal{A}$ the subset of *observable* atomic propositions, and $\mathcal{A}_{ex} \subseteq \mathcal{A}$ the subset of *explanatory* atomic propositions. Typically, explanatory atomic propositions correspond to the faults or state information that cannot be directly accessed. On the other hand, observable atomic proposition indicate state information that can be directly "recorded", for instance, alarms and abnormal behavior. To formulate the diagnosis problem, we can now formalize the notions of *observation* and *explanation*.

Definition 5. *An* observation *is a computation* $(S, \eta, \leq, V)$ *such that for each state* s *in* S, $V(s) \subseteq \mathcal{A}_{ob}$. *Let* $O = (S_O, \eta_O, \leq_O, V_O)$ *be an observation and* $W = (S_W, \eta_W, \leq_W, V_W)$ *a computation. Then,* W *is an* explanation *for* O *iff there exists an injective mapping* $f : S_O \to S_W$ *such that:*

(i) $\forall s \in S_O$, $\eta_O(s) = \eta_W(f(s))$ *and* $V_O(s) = V_W(f(s)) \cap \mathcal{A}_{ob}$.
(ii) $\forall s, s' \in S_O$, *if* $s \leq_O s'$, *then* $f(s) \leq_W f(s')$.
(iii) For every s *in the image of* f, $V_W(s) \cap \mathcal{A}_{ob} \neq \emptyset$. *Furthermore, for any* $s' \in S_W$, *if* $\eta_W(s') = \eta_W(s)$, $s' \leq_W s$ *and* $V_W(s') \cap \mathcal{A}_{ob} \neq \emptyset$, *then* s' *is also in the image of* f.

Conditions *i*) to *iii*) come from the supposed faithful and non-lossy nature of the observation mechanism. More precisely, condition (*i*) means that the location mapping should be respected by the observation mechanism, and that in explanations, only states at which at least one observable atomic proposition holds may be "recorded", and hence appear in the observation. Intuitively, (the truth of) an observable atomic proposition corresponds typically to the *presence* of some alarm or observed abnormal behaviour. And the monitoring mechanism could only detect the *presence* of observable atomic propositions, but not their absence. We also suppose that the observation mechanisms do not produce atomic propositions that were not observed. Hence, if no observable proposition hold at a state s of an execution, then s is not recorded in the observation.

Condition (*ii*) asserts that the recorded causal orderings must originate from an actual dependence in the execution. The converse property (recording all causal dependencies) is *not* demanded, since some causal orderings in the explanation may not be "recorded". For generality, we do not impose any more specific condition on the recording of causality ordering. We remark however that our results could be extended easily to deal with more specific condition on recording of causalities, which may arise from particular application domains.

Condition (*iii*) states that there is no loss during recording of states: if a state of process p is recorded, then any causally preceding states s' of p must be recorded in case the valuation of s' contains observable atomic propositions. In other words, we assume that the monitoring mechanism is itself free from faults and thus do not miss any presence of observable abnormal signals. In some application domains such as telecommunication networks, the monitoring mechanism is also part of the observed system. In such situations, it may be sometimes necessary to consider also the potential faults incurred by the monitoring mechanism. However, in this paper, we will consider that the observation mechanisms is non-lossy.

Let us illustrate the notions of observation and explanation on the examples of Figure 4. In this example, there are two processes "Client" and "Server", and the observable atomic propositions consist of "Reset" and "Reboot" (shown in bold in Figure 4. Both W, W' are explanations for O in Figure 4. In W, the server glitches, reboots itself and requests the client to reset. Upon receiving the request from the server, the client then resets itself. In W', the server breaks down and reboots itself after repairing, while the client resets itself following detection of loss of connection ("LostConn") and failure to re-establish the connection.

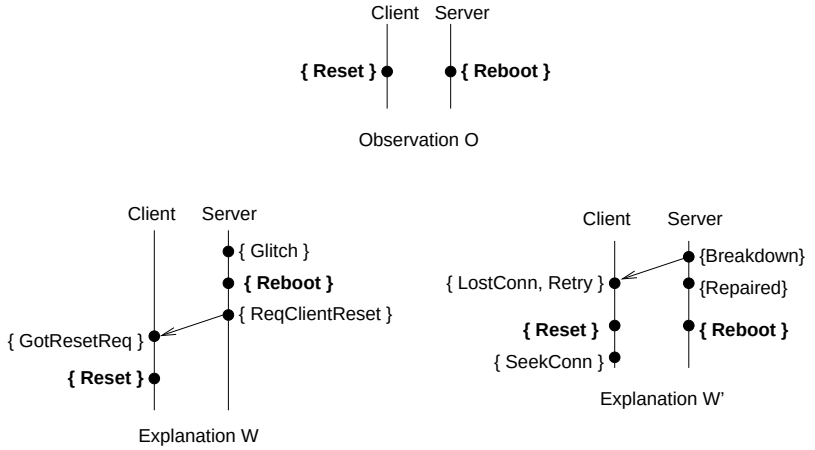

Figure 4: Two computations W and W', and an observation O

We emphasize that while an explanation W represents a full execution of a system, an observation induced by W, may have recorded information from a part of W. Thus, the image of f may be a proper subset of the states of W whose valuation contains observable atom-

ic propositions. Similarly, some causal dependencies among observed states may not have been saved during the observation process. On the examples of Figure 4, we can notice that there is a causal dependency between the states of W labeled by *reboot* and *reset*, but that this causality does not appear in the observation O. However, we suppose the observer is faithful, that is, it does not create wrong states or causalities. Note that for given O and W, if the mapping f from O to W exists, then it is necessarily unique. Thus, for each state $s \in S_O$, it makes sense to call $V_W(f(s))$ the *W-valuation of s*.

Definition 6. *Let O be an observation, and $\{\Phi_p\}_{p\in\mathcal{P}}$ be a $\mathcal{P}$-index family of formulae. W is a* $\{\Phi_p\}_{p\in\mathcal{P}}$-explanation *for O iff W is an explanation for O and $W \models \{\Phi_p\}_{p\in\mathcal{P}}$.*

Now we can define the diagnosis problem associated with LPOC.

Definition 7. *Let $\mathcal{P}$ be a set of processes, $\mathcal{A}_{ob}$ and $\mathcal{A}_{ex}$ be two sets of atomic propositions as before. The* LPOC-diagnosis *problem is defined as follows: given an observation $O = (S_O, \eta_O, \leq_O, V_O)$ and a $\mathcal{P}$-indexed family of formulae $\{\Phi_p\}_{p\in\mathcal{P}}$ over $(\mathcal{P}, \mathcal{A}_{ob}, \mathcal{A}_{ex})$, determine whether there exists a $\{\Phi_p\}_{p\in\mathcal{P}}$-explanation for O.*

In what follows, we will often omit the subscript $p \in \mathcal{P}$. The formulae $\{\Phi_p\}$ specify some knowledge about executions of the system, for instance some faulty behaviors. The objective of diagnosis is to figure out whether there exists an explanation for what has been observed, and hence detect if the actual behavior of the whole system was faulty or not. In case an explanation exists, we also want to obtain more detailed information about the possible truth values of propositions in $\mathcal{A}_{ex}$ at each observed state. We define the *(explanatory) summary* of O under $\{\Phi_p\}_{p\in\mathcal{P}}$ as the mapping $g : S_O \to 2^{\mathcal{A}_{ex}}$ such that for every $s \in S_O$, $g(s) \subseteq \mathcal{A}_{ex}$, and $a \in g(s)$ iff there exists an explanation W for O with $W \models \{\Phi_p\}_{p\in\mathcal{P}}$ and a is in the W-valuation of s. Thus, when the answer to the diagnosis problem is positive, we want further to compute the summary of O. Unfortunately, in the general case, the LPOC-diagnosis problem is undecidable (and so is the computation of summaries).

Theorem 8. *The LPOC-diagnosis problem is undecidable.*

Proof sketch: By a reduction from the Post Correspondence Problem (PCP), similar to that used in decision problems related to message sequence charts [9]. The complete proof of this theorem can be found in the extended version.

4 Diagnosis with K-Influencing Explanations

We have shown in previous section (Theorem 1) that the diagnosis problem is undecidable in general. There are two usual ways to overcome this problem. The first one is to consider a decidable fragment of the logic. Note however, that encoding a PCP becomes possible as soon as there is a way to describe sequences of properties located on a given process, and to define a mapping of states on different processes that respects the ordering. This is why in most cases very small fragment of partial order logics become undecidable when no restriction is imposed on the kind of model considered. Then, the question that naturally arises is whether we can identify a subclass of computations for which LPOC diagnosis is tractable. For this, we identify the subclass of K-influencing computations, and show that the LPOC diagnosis problem (and thus computing the summary) is decidable within this class.

Definition 9. *Let $\mathcal{P}$ be a set of processes, $\mathcal{A}_{ob}$ be a set of observable atomic propositions, $\mathcal{A}_{ex}$ be a set of atomic explanatory propositions. Let $W = (S, \eta, \leq, V)$ be a computation, and $p, q \in \mathcal{P}$ with $p \neq q$. The* causal degree *of p towards q in W is the maximum integer $n \in \mathbb{N}$ for which there exist s_1, s_2, ..., s_n in S_p, and s'_1, s'_2, ..., s'_n in S_q such that:*

(i) $s_1 < s_2 < \ldots < s_n$ *and* $s'_1 < s_2 < \ldots < s'_n$.
(ii) for $i = 1, 2, \ldots, n$, $s_i \ll s'_i$, *that is, s_i is a predecessor of s'_i.*
(iii) $s'_1 \not< s_n$.

For $K \in \mathbb{N}$, W is K-influencing *iff for any pair of processes p, q in $\mathcal{P}$ with $p \neq q$, the causal degree of p towards q is at most K.*

Intuitively, the causal degree of p towards q is the maximal number of events that precede some event on q that p can execute without having to wait for q. The general shape of $K-$influencing computations is illustrated in Figure 5. We now state the main result of this section.

Theorem 10. *Given an observation O, a $\mathcal{P}$-indexed family $\{\Phi_p\}_{p \in \mathcal{P}}$ of LPOC formulae, and an integer $K \in \mathbb{N}$, one can effectively determine whether there exists a $K-$influencing computation W which is a $\{\Phi_p\}_{p \in \mathcal{P}}$-explanation for O.*

An important consequence of the above theorem is that one can effectively compute a summary of the K-influencing explanations of O. Let O be an observation and $\{\Phi_p\}_{p \in \mathcal{P}}$ a $\mathcal{P}$-indexed family of

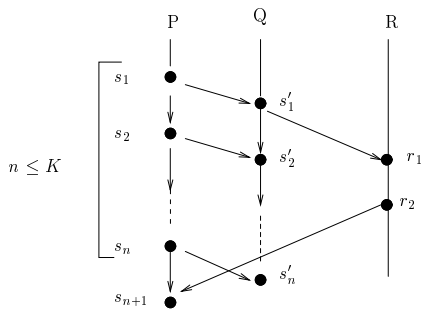

Figure 5: $K-$influencing computations

formulae. We can slightly adapt the definition of summaries in section 3 to K-influencing computations: the *K-summary* of O under $\{\Phi_p\}_{p\in\mathcal{P}}$ is the mapping $g : S_O \to 2^{\mathcal{A}_{ex}}$ such that for every s in S_O, $g(s) \subseteq \mathcal{A}_{ex}$, and $a \in g(s)$ iff there exists a K-influencing explanation W for O with $W \models \{\Phi_p\}_{p\in\mathcal{P}}$ and a is in the W-valuation of s.

Corollary 11. *Given an observation O, a $\mathcal{P}$-indexed family $\{\Phi_p\}_{p\in\mathcal{P}}$ of LPOC formulae, and an integer $K \in \mathbb{N}$, one can effectively compute the K-summary of O under $\{\Phi_p\}_{p\in\mathcal{P}}$.*

Through the rest of this section, we prove Theorem 10 and Corollary 11. We fix the integer K, the observation O and the formulae $\{\Phi_p\}_{p\in\mathcal{P}}$. Recall from section 2 that we can easily construct a single formula Φ such that for any computation W, $W \models \Phi$ iff $W \models \{\Phi_p\}_{p\in\mathcal{P}}$. In what follows, we fix Φ. We will assume that the computations used hereafter are nonempty. It will be clear from the proof that this involves no loss of generality. We let $\mathcal{W}_K$ denote the set of K-influencing computations.

The proof for Theorem 10 consists of two steps. Firstly, we show that K-influencing computations can be identified with Mazurkiewicz traces [5] over a suitable trace alphabet (Σ, I). This way, we can identify K-influencing computations with equivalence classes of finite sequences in $\Sigma^\star$. This encoding is in spirit the same as the of encoding of universally K-bounded message sequence charts with traces in [12]. Secondly, we construct three finite state automata Aut_K, Aut_Φ, Aut_O, running over linearizations of traces of (Σ, I). Aut_K checks if an input sequence represents a computation of $\mathcal{W}_K$. For a sequence σ representing a computation W_σ in $\mathcal{W}_K$, Aut_Φ accepts σ iff $W_\sigma \models \Phi$, and Aut_O accepts σ iff W_σ is an explanation for O. The crux is the construction of Aut_Φ. We shall give Aut_Φ in the form of a two-way

alternating automaton, which can be transformed to a finite state automaton. The basic idea is similar to [7] and the usual translation from LTL to alternating automata [18]. The new technicality in our construction is in checking conformance with formulae of the form $\downarrow_{m,A}(T)$, $\uparrow_{m,A}(T)$. Then, there exists a sequence in $\Sigma^\star$ accepted by Aut_K, Aut_O, Aut_Φ iff $\mathcal{W}_K$ contains a computation W such that $W \models \Phi$ and W is an explanation for O. This then establishes Theorem 10. For Corollary 11, we will show that for any state s of O and any atomic proposition $a \in \mathcal{A}_{ex}$, one can find a finite state automaton $Aut_{s,a}$ which has the following property: if a sequence σ represents a computation W_σ in $\mathcal{W}_K$ such that $W_\sigma \models \Phi$, and W_σ is an explanation of O, $Aut_{s,a}$ accepts σ iff a is in the W_σ-valuation of O. As a result, one can then effectively compute the K-summary of O under $\{\Phi_p\}_{p\in\mathcal{P}}$.

Encoding K-influencing computations with Traces

We recall that a Mazurkiewicz trace [5] alphabet is a pair (Σ, I) where Σ is a finite alphabet, and $I \subseteq \Sigma \times \Sigma$ is an irreflexive and symmetric relation called the *independence relation*. The *dependence relation* D is given by $(\Sigma \times \Sigma) \setminus I$. A (finite) Σ-labelled poset is a pair $(E, \sqsubseteq, \lambda)$, where E is a finite set, $\sqsubseteq \subseteq E \times E$ a partial order, and $\lambda : E \to \Sigma$ a labeling function. As usual, we write $e \sqsubset e'$ if $e \sqsubseteq e'$ and $e \neq e'$. We let $\widehat{\sqsubset} \subseteq E \times E$ denote the least relation whose reflexive and transitive closure is equal to $\sqsubseteq$. A (Mazurkiewicz) trace over (Σ, I) is a Σ-labelled poset $tr = (E, \sqsubseteq, \lambda)$ satisfying: (i) for any $e, e' \in E$, $\lambda(e) D \lambda(e')$ implies $e \sqsubseteq e'$ or $e' \sqsubseteq e$; (ii) for any $e, e' \in E$, $e \widehat{\sqsubset} e'$ implies $\lambda(e) D \lambda(e')$. We define isomorphism of traces in the obvious way and write $tr = tr'$ if the traces tr, tr' are isomorphic.

Let $[K] = \{0, 1, \ldots, K-1\}$. We define the alphabets $\Gamma_{pre} = \{pre(p,i) \mid p \in \mathcal{P}, i \in [K]\}$, and $\Gamma_{suc} = \{suc(p,i) \mid p \in \mathcal{P}, i \in [K]\}$. Let $\Gamma = \mathcal{P} \times 2^{\Gamma_{pre}} \times 2^{\Gamma_{suc}} \times 2^{\mathcal{A}}$. We define Σ as the subset of Γ satisfying: $\big(p, \{pre(p_1, i_1), \ldots, pre(p_g, i_g)\}, \{suc(p'_1, i'_1), \ldots, suc(p'_h, i'_h)\}, A\big) \in \Sigma$ iff $p_1, \ldots, p_g$ are distinct members of $\mathcal{P} \setminus \{p\}$, and $p'_1, \ldots, p'_h$ are distinct members of $\mathcal{P} \setminus \{p\}$. We now define the dependence relation $D \subseteq \Sigma \times \Sigma$ as: $(p, PRE, SUC, A)\ D\ (p', PRE', SUC', A')$ iff one of the following conditions holds:

- $p = p'$.
- $p \neq p'$, and for some $i \in [K]$, $pre(p', i) \in PRE$ and $suc(p, i) \in SUC'$.
- $p \neq p'$, and for some $i \in [K]$, $suc(p', i) \in SUC$ and $pre(p, i) \in PRE'$.

We set the independence relation $I = \Sigma \times \Sigma - D$. It is trivial to verify that (Σ, I) is a trace alphabet. From now on, we fix the trace alphabet (Σ, I).

Let $W = (S, \eta, \leq, V)$ be a K-influencing computation. Let us define the Σ-labeling of W, denoted λ_W^Σ (or simply λ_W), as the following function from S to Σ: for $s \in S$, $\lambda_W(s) = (p, PRE, SUC, A)$ where:

- $p = \eta(s)$.
- $pre(q, i) \in \Gamma_{pre}$ is in PRE iff s has predecessor s' with $\eta(s') = q$ (such a s' is necessarily unique by the definition of predecessor) and $i = j \mod K$ where j is the number of states s'' in S_p satisfying $s'' < s$ and that s'' has a predecessor of location q.
- Further, $suc(\hat{q}, \hat{i}) \in \Gamma_{suc}$ is in SUC iff s has a successor $\hat{s}'$ with $\eta(\hat{s}') = \hat{q}$ (such a $\hat{s}'$ is also unique) and $\hat{i} = \hat{j} \mod K$ where $\hat{j}$ is the number of states $\hat{s}''$ in S_p satisfying $s'' < s$ and that s'' has a successor of location q.
- $A = V(s)$.

Note that PRE and SUC are not necessarily singletons, as a state may have one successor (reps. predecessor) on each process. Let us define $tr(W) = (S, \leq, \lambda_W)$. Remark that for $W \in \mathcal{W}_K$, $tr(W)$ is a trace over (Σ, I). Furthermore, if W' is a K-influencing computation, then $W = W'$ iff $tr(W) = tr(W')$. Figure 6 below shows an example of a 2-influencing computation with the associated labeling. For the sake of clarity, the subsets of atomic propositions that are true at each state are not explicitly given, but only described by $A_1, \ldots A_9$.

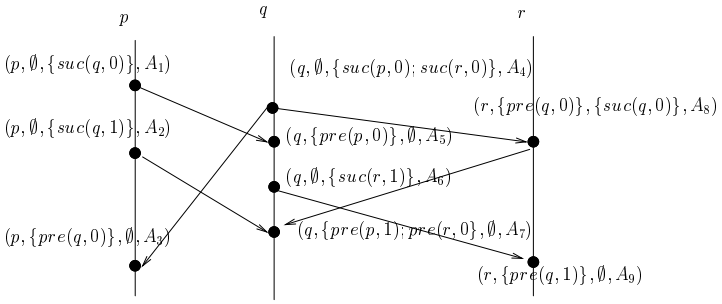

Figure 6: A 2-influencing computation and its Σ-labeling

A *linearization* of a trace $(E, \sqsubseteq, \lambda)$ is a sequence $\lambda(e_1) \ldots \lambda(e_n)$ where $e_1, \ldots, e_n$ are distinct members of E, $E = \{e_1, \ldots, e_n\}$, and

for any $i, j \in \{1, \ldots, n\}$, $e_i \sqsubseteq e_j$ implies $i \leq j$. For any computation $W \in \mathcal{W}_K$, by a Σ-*linearization of* W, we refer to a linearization of $tr(W)$. We denote by $Lin^{\Sigma}(W)$ the set of Σ-linearizations of W. Let $Lin_K^{\Sigma} = \bigcup_{W \in \mathcal{W}_K} Lin^{\Sigma}(W)$. Note that computations in $\mathcal{W}_K$ can be uniquely constructed from sequences in Lin_K^{Σ}. For a non-null sequence σ in $\Sigma^\star$, let $last(\sigma)$ denote the last letter of σ. For σ, σ' in $\Sigma^\star$, we write $\sigma \preccurlyeq \sigma'$ iff σ is a prefix of σ'. We define the partial order $\sqsubseteq_{\Sigma^\star} \subseteq \Sigma^\star \times \Sigma^\star$ via: $\sigma \sqsubseteq_{\Sigma^\star} \sigma'$ iff σ, σ' are non-null, $\sigma \preccurlyeq \sigma'$ and there exist $\sigma_1, \sigma_2, \ldots, \sigma_h$, $h \leq |\Sigma|$, such that $\sigma \preccurlyeq \sigma_1 \preccurlyeq \sigma_2 \ldots \preccurlyeq \sigma_h \preccurlyeq \sigma'$ and $last(\sigma)Dlast(\sigma_1)Dlast(\sigma_2)\ldots last(\sigma_h)Dlast(\sigma')$. For every $\sigma \in Lin_K^{\Sigma}$, we define the computation $poc(\sigma) = (S_\sigma, \eta_\sigma, \leq_\sigma, V_\sigma)$, where:

- S_σ is the set of non-empty prefixes of σ.
- For $\tau \in S_\sigma$, $\eta_\sigma(\tau) = p$ iff $last(\tau) = (p, PRE, SUC, A)$ for some PRE, SUC, A.
- $\leq_\sigma$ is the restriction of $\sqsubseteq_{\Sigma^\star}$ to S_σ.
- For $\tau \in S_\sigma$, $V_\sigma(\tau) = A$ iff $last(\tau) = (p, PRE, SUC, A)$ for some p, PRE, SUC.

It is easy to verify that $poc(\sigma)$ is well-defined. Furthermore, for every $\sigma \in Lin_K^{\Sigma}$ and $W = poc(\sigma)$, we have $W \in \mathcal{W}_K$ and $\sigma \in Lin^{\Sigma}(W)$.

Automata construction.

We can build three finite state automata Aut_K, Aut_O, Aut_Φ which have the following properties:

- For $\sigma \in \Sigma^\star$, σ is accepted by Aut_K iff $\sigma \in Lin_K^{\Sigma}$.
- $\sigma \in Lin_K^{\Sigma}$ is accepted by Aut_O iff $poc(\sigma)$ is a K-influencing explanation for O.
- $\sigma \in Lin_K^{\Sigma}$ is accepted by Aut_Φ iff $poc(\sigma)$ satisfies Φ.

It follows that a sequence $\sigma \in \Sigma^\star$ is accepted by the product of Aut_K, Aut_O, Aut_Φ iff $poc(\sigma)$ is a K-influencing $\{\Phi_p\}$-explanation for O.

We do not detail the construction of Aut_K and Aut_O, that can be found in the extended version of this paper [19].

Proposition 12. *Let $\mathcal{P}$ be a set of processes, $\mathcal{A}$ be a set of atomic proposition, K be an integer, and Σ be Mazurkiewicz trace alphabet computed from $\mathcal{P}$, $\mathcal{A}$ and K. Then, there exists an automaton Aut_K of size $\mathcal{O}(|\Sigma|^{K \cdot |\mathcal{P}|^2})$ that recognizes linearizations of K-influencing computations over $\mathcal{P}$ with valuations in $\mathcal{A}$.*

We can reuse the construction of Aut_K to build Aut_O, the automaton that recognizes K-influencing explanations of O. At each

state of this automaton, one must recall a state of Aut_K reached (ie, the current $K-$influencing linearization explored), the part of O that is embedded in this explanation, and some additional information about the causalities that may appear in the future.

Proposition 13. *Let O be an observation over a set of processes $\mathcal{P}$, with valuations in an alphabet $\mathcal{A}_{ob}$. Let K be an integer and $\mathcal{A} \supseteq \mathcal{A}_{ob}$ be a set of atomic propositions. The size of Aut_O is at most in $\mathcal{O}(|Aut_K| \cdot 2^{|O|} \cdot 2^{|\mathcal{P}|} \cdot (|O| \cdot |\mathcal{P}|)^2 \cdot (|\mathcal{P}| \cdot K + 1)^K)$.*

Construction of Aut_Φ.

We next give the description of Alt_Φ, a two-way alternating automaton [13] that recognizes linearizations of $K-$influencing computations satisfying Φ. This automaton can then be transformed into a standard finite state automaton Aut_Φ.

We introduce some new atomic formulae in order to simplify the structure of Φ. Recall that for a computation W and a state s of W, the m-view or the m-frontier of s contains at most $\mathcal{N}_m = \sum_{i=0}^{m} |\mathcal{P}|^i$ states. We introduce formulae of the form $\downarrow_m(T), \uparrow_m(T)$, where $m \in \mathbb{N}$ and T is a computation containing at most $\mathcal{N}_m$ states. Let $W = (S, \eta, \leq, V)$ be a computation and $s \in S$. Then $W, s \models \downarrow_m(T)$ iff the m-view of s is isomorphic to T. The semantics of $\uparrow_m(T)$ is given similarly. We note that a formula $\downarrow_{m,A}(T)$ is equivalent to $\vee_{T' \in \mathcal{T}} \downarrow_m(T')$, where $\mathcal{T}$ is the collection of computation T' such that T' contains at most $\mathcal{N}_m$ states and the projection of T' onto A is isomorphic to T. Similarly, we can write $\uparrow_{m,A}(T)$ as a disjunction of formulae $\uparrow_m(T)$ with an analogous semantics.

Let $W = (S, \eta, \leq, V)$ be a computation and $s \in S$. Let $s' \in S$ and $\tau = a_1 a_2 \ldots a_n$ be a non-null sequence in $\Sigma^\star$. If there exist $s_1, \ldots, s_n \in S$ such that $s' = s_n$, $s_n \ll s_{n-1} \ll \ldots s_1 \ll s$ and $\lambda_W(s_i) = a_i$ for $i = 1, \ldots, n$, then we say s' is a *τ-ancestor* of s. Recall that each state in W has at most $|\mathcal{P}|$ predecessors, one belonging to each S_p. Thus, we can in fact say s' is *the* τ-ancestor of s. We introduce formulae of the form $\downarrow(\tau, \tau')$ where τ, τ' are non-null sequences in $\Sigma^\star$. We define $W, s \models \downarrow(\tau, \tau')$ iff there exist states $\hat{s}, \hat{s}'$ such that $\hat{s}$ is the τ-ancestor of s, $\hat{s}'$ is the τ'-ancestor of s, and $\hat{s} \leq \hat{s}'$.

We argue that a formula $\downarrow_m(T)$ can be equivalently written as a boolean combination of formulae of the form $\downarrow(\tau, \tau')$. Assume without loss of generality of T contains a maximum state s_{max}. Thus, every state in T is the τ-ancestor of s_{max} for some τ of length at most the number of states of T. Hence, $\downarrow_m(T)$ is equivalent to asserting for each pair of states s, s' in T, whether $\downarrow(\tau, \tau')$, $\downarrow(\tau', \tau)$, or

$\neg \downarrow(\tau, \tau') \wedge \neg \downarrow(\tau', \tau)$, where s, s' are the respectively the τ-ancestor and τ'-ancestor of s_{max}.

Analogously, we define *τ-descendants* and introduce formulae of the form $\uparrow(\tau, \tau')$ where τ, τ' are non-null sequences in $\Sigma^\star$. It follows that a formula $\uparrow_m(T)$ is equivalent to a boolean combination of formulae of the form $\uparrow(\tau, \tau')$.

With the new formulae introduced above, we can assume without loss of generality that Φ is formed from $\neg, \wedge, \vee$ and the atomic formulae loc_p, $\downarrow(\tau, \tau')$, $\uparrow(\tau, \tau')$, $\mathsf{EX}\varphi$, $\mathsf{EU}(\varphi, \varphi')$. Furthermore, negations in Φ only apply to atomic formulae.

Now we are ready to describe the two-way alternating automaton Alt_Φ. The basic elements of Alt_Φ are similar as in usual translations of temporal logics to alternating automata (see e.g. [18]). The main difficulty is to deal with atomic formulae of the form $\downarrow(\tau, \tau')$, $\uparrow(\tau, \tau')$.

We informally recall some basics of two-way alternating automata and refer to [4, 13] for details. Let *Alt* be a two-way alternating automaton. An input word is delimited on the left by a left marker and on the right by a right marker. Initially, *Alt* is at the initial state with the head at the first letter of the input word. Upon reading the letter of the current head position, *Alt* can spawn several copies where each copy can move the head left or right and go to a new control state. Which combination of copies can be spawned are pre-determined by a transition relation. A run of *Alt* over an input word σ is a (finite) tree, where each branch terminates upon reaching the left or the right marker. And *Alt* accepts σ iff there exists a run over σ such that every leaf contains an accepting state.

For clarity, we describe only informally the operations of Alt_Φ. The exact construction of Alt_Φ can be found in the extended version. For illustration purpose, we fix an input word $\sigma = a_1 a_2 \ldots a_n$ in Lin_K^Σ. We write $\sigma, i \models \varphi$ iff $poc(\sigma), a_1 \ldots a_i \models \varphi$.

Let $SF(\Phi)$ be the set of subformulae of Φ and their negations, where $\neg\neg\varphi$ is identified with φ. A state z of Alt_Φ consists of a formula φ in $SF(\Phi)$ and some alphabetic constraints. Such a state z must verify that φ holds at the current head position i and that the alphabetic constraints should be satisfied subsequently.

Note that $poc(\sigma) \models \Phi$ iff $\sigma, h \models \Phi$ where $a_1 a_2 \ldots a_h$ is a minimal state in $poc(\sigma)$. Thus, at the initial state, Alt_Φ searches for position h such that $a_j \; I \; a_h$ for $j = 1, \ldots, h-1$, and upon reaching position h, it verifies that Φ holds at h.

It now suffices to explain how Alt_Φ verifies that a formula in $SF(\Phi)$ holds at the current head position. We proceed inductively from the atomic formulae of forms loc_p, $\downarrow(\tau, \tau')$, $\uparrow(\tau, \tau')$ and their negations, then to formulae of forms $\mathsf{EX}\varphi$, $\mathsf{EU}(\varphi, \varphi')$, and their nega-

tions. We then study conjunction and disjunction of formulae.

Firstly, Alt_Φ can easily check if loc_p or $\neg loc_p$ holds at the current head position, simply from the letter at the head position. Next we consider atomic formulae of the form $\downarrow(\tau, \tau')$, $\uparrow(\tau, \tau')$ and their negations. For the input sequence σ, we let $a_i = (p_i, PRE_i, SUC_i, A_i)$ for each i. Recall that $poc(\sigma) = (S_\sigma, \eta_\sigma, \leq_\sigma, V_\sigma)$ where S_σ is the set of prefixes of σ. For $s, s' \in S_\sigma$, we write $s \ll_\sigma s'$ iff s is a predecessor of s' in $poc(\sigma)$. Consider $g, h \in \{1, 2, \ldots, n\}$, it is easy to see that $a_1 \ldots a_g \ll_\sigma a_1 \ldots a_h$ iff $g < h$ and one of the following conditions holds:

- $p_g = p_h$. And for each index i with $g < i < h$, $p_i \neq p_g$.
- $p_g \neq p_h$ and $a_g \; D \; a_h$. Further, there do *not* exist indices $i_1, i_2, \ldots, i_t$, $t \leq |\Sigma|$, such that $g < i_1 < i_2 < \ldots < i_t < h$, and $a_g \; D \; a_{i_1} \; D \; a_{i_2} \ldots a_{i_t} \; D \; a_h$.

For a formula $\downarrow(\tau, \tau')$, where $\tau = b_1 b_2 \ldots b_m$, $\tau' = b'_1 b'_2 \ldots b'_{m'}$, we note that $\sigma, \ell \models \downarrow(\tau, \tau')$ iff there exist indices $\ell_1, \ldots, \ell_m$, $\ell'_1, \ldots, \ell'_{m'}$, in $\{\ell + 1, \ell + 2, \ldots, n\}$ such that:

- $a_1 a_2 \ldots a_\ell \ll_\sigma \rho_1 \ll_\sigma \rho_2 \ll_\sigma \ldots \ll_\sigma \rho_m$, where $\rho_i = a_1 a_2 \ldots a_{\ell_i}$ for $i = 1, 2, \ldots, m$. And $a_{\ell_i} = b_i$ for $i = 1, 2, \ldots, m$. This asserts that the τ-ancestor of $a_1 a_2 \ldots a_l$ exists.
- $a_1 a_2 \ldots a_\ell \ll_\sigma \rho'_1 \ll_\sigma \rho'_2 \ll_\sigma \ldots \ll_\sigma \rho'_{m'}$, where $\rho'_i = a_1 a_2 \ldots a_{\ell'_i}$ for $i = 1, 2, \ldots, m'$. And $a_{\ell'_i} = b'_i$ for $i = 1, 2, \ldots, m'$. This asserts that the τ'-ancestor of $a_1 a_2 \ldots a_l$ exists.
- $a_1 a_2 \ldots a_{\ell_m} \leq_\sigma a_1 a_2 \ldots a_{\ell'_{m'}}$, that is, $a_1 a_2 \ldots a_{\ell_m} \sqsubseteq_{\Sigma^*} a_1 a_2 \ldots a_{\ell'_{m'}}$. This asserts that the τ-ancestor of $a_1 a_2 \ldots a_l$ causally precedes the τ'-ancestor of $a_1 a_2 \ldots a_l$.

Thus, to verify that a formula $\downarrow(\tau, \tau')$ or its negation holds at the current position, Alt_Φ moves to the left until it hits the left end marker and along the way checks the existence of indices $\ell_1, \ldots, \ell_m, \ell'_1, \ldots, \ell'_{m'}$ satisfying the above conditions. Analogously, it is clear how Alt_Φ can verify if a formula $\uparrow(\tau, \tau')$ or its negation holds at the current head position.

Finally, we note that formulae of forms $\mathsf{EX}\varphi$, $\mathsf{EU}(\varphi, \varphi')$ and their negation can be handled as in usual translations of temporal logics over traces to alternating automata (e.g. [7]). This is also the case for conjunction and disjunction of formulae. This completes the description of Alt_Φ.

It is not difficult to see that the number of states of Alt_Φ is of complexity $O(2^{|\Phi|} \cdot |\Sigma|^{|\Sigma| \cdot m})$, where m is the maximum length of

τ, τ' for all atomic formulae of the form $\uparrow(\tau, \tau')$, $\downarrow(\tau, \tau')$. It follows from [13] that Alt_Φ can be transformed to a finite state automaton Aut_Φ with $2^{N \cdot 2^N}$ states where N is the number of states of Alt_Φ. Checking for the existence of an explanation then consists in checking the emptiness of the intersection of Aut_Φ and Aut_O built in proposition 13. The proof of Theorem 10 is now completed. □

Proof of corollary 11: To prove Corollary 11, we first recall that if W is an explanation for O, then the injective mapping from the states of O to the states of W dictated in the definition of explanation is unique. Thus, it is easy to see that for any state s of O and any atomic proposition $a \in \mathcal{A}_{ex}$, one can construct a finite state automaton $Aut_{s,a}$ which has the following property: if σ a sequence σ representing a computation W_σ in $\mathcal{W}_K$ where $W_\sigma \models \Phi$ and W_σ is an explanation of O, $Aut_{s,a}$ accepts σ iff a is in the W_σ-valuation of s. $Aut_{s,a}$ can then be easily constructed from Aut_O by requiring that transitions that add s to the subset of observed states of O are labelled by letters with valuations that contain a. As a result, one can then effectively compute the K-summary of O under $\{\Phi_p\}_{p \in \mathcal{P}}$, by testing for each state s of O, each a in $\mathcal{A}_{ex}$, the non-emptiness of the product of Aut_Φ and $Aut_{s,a}$. □

If the formula ϕ is such that all frontiers and views used are at most m-frontiers or m-views, then one can determine whether there exists a K-influencing explanation W for an observation O with complexity $\mathcal{O}(W_1.2^{W_2.2^{W_2}})$, where:

$$W_1 = |\Sigma|^{K.|\mathcal{P}|^2} \cdot 2^{|O|} \cdot 2^{|\mathcal{P}|} \cdot (|O| \cdot |\mathcal{P}|)^2 \cdot (|\mathcal{P}| \cdot K + 1)^K$$

$$W_2 = 2^{\left(|\phi| \cdot \mathcal{N}_m^2 \cdot 2^{\mathcal{A}^{ex}} \cdot \sum_{i=0}^{\mathcal{N}_m} f(i)\right)} \cdot |\Sigma|^{|\Sigma| \cdot m}$$

with $f(i) = i \cdot 2^{\frac{i^2}{4} + \frac{3i}{2} + ln(i)}$, and $\mathcal{N}_m = \frac{1 - |\mathcal{P}|^{m+1}}{1 - |\mathcal{P}|}$. From the definition of summaries, computing a summary for O can then be done in $\mathcal{O}(|O| \cdot |\mathcal{A}^{ex}| \cdot W_1 \cdot 2^{W_2 \cdot 2^{W_2}})$. The proof of these complexity results is not provided here, but can be found in the extended version of this paper [19].

5 Related Work and Conclusion

We have proposed a diagnosis framework based on a new partial order logic (LPOC) over partial orders (i.e. the truth of formulae is evaluated at local states). Unsuprisingly, satisfiability of LPOC formulae, and hence diagnosis are not decidable without restriction. To keep decidability of diagnosis, a restriction called K-influence is imposed on the models. As LPOC uses the existential until operator, for a given K, LPOC restricted to K-influencing computations is not de-

finable in the first order logic over the Mazurkiewicz traces encoding K-influencing computations. However, it can be easily translated to MSO formulae. An interesting work would be to look for a fragment that is expressively complete for the first order logic over the traces encoding K-influencing computations.

Even with the restriction to K-influencing computations, diagnosis is very expensive (several exponential in the size of the formula and exponential in the size of the observed behavior). This high complexity could mean that diagnosis with LPOC is unfeasible. Note however that this complexity is in the worst cases. For instance, the exponential in the size of the observation comes from the maximal number of configurations in a partial order. In practice, for an observation with a bounded number of processes, the number of configurations can be much slower. The other costly part of the diagnosis problem comes from the translation from alternating automata to finite state automata. Again this is a worst case complexity. Note also that the translation of LPOC formulae into conjunction of formulae of the form $\uparrow(\tau, \tau')$, $\downarrow(\tau, \tau')$ is costly only when the pattern considered in the formulae are large. In general, basic patterns used in partial order languages such as message sequence charts are rather small, and we argue that this should also be the case with LPOC formulae. Some complexity gains can hence be expected by restricting the size and the number of partial order templates considered, but also the modalities of the formulae. Note however that most of the modalities chosen for LPOC seem important. The simple example of section 2 shows that the Until operator is essential to express properties of the form "when T1 occurs, T2 will occur later". One may also try to restrict the use of negation, that is LPOC formulae would only be conjunctions of positive assertions on the occurrence of patterns. Note however that the translation of an LPOC formula to a simplified formula on causal chains uses negation when two states of a pattern are not causally related. Hence, even in a restricted setting, negation of some properties will have to be checked. So, the small complexity gain that could occur may not justify the loss of expressiveness due to a restriction on negations.

In [16], D.Peled shows that model checking TLC$^-$ formulae on High-level Message Sequence charts (HMSCs) is decidable. TLC$^-$ is a subset of TLC that only contains next and until temporal operators, and describes the shape of causal chains in all the partial orders generated by a HMSC. TLC^- is clearly less expressive than LPOC.

The Propositional Dynamic Logic (PDL) for message passing systems proposed by [3], extends dynamic LTL for traces [11]. Model checking PDL properties over HMSCs is PSPACE complete. [15]

proposes a local logic LD0 and several extensions over computations, with future and past modalities, and show that in the general case, satisfiability is undecidable. However, these logics become decidable when considering models of bounded size, or when computations can be organized as successive layers of finite message exchanges. LD0 only describes chains of causally related events occurring in the future or in the past of a local state, while the template matching in LPOC allows to describe a complete partial order in a bounded future or past of a local state. LPOC is then more discriminating than LD0, and if we restrict our models to Message Sequence Charts (a partial order where locality of events and messages are explicitly represented), it is also more expressive and discriminating than TLC^- and PDL.

Note also that for TLC^-, PDL, or $LD0$, partial orders are seen as models of formulae, but not as elements of the logic itself. The closest approach mixing logic and partial orders is called "Template Message Sequence Charts" [9]. A template MSC is an MSC that comports some "hole" and incomplete messages. Roughly speaking, models for a template MSC are obtained by filling the holes with new partial orders, and matching sendings and receptions of messages. The authors increase the power of template MSCs with pre/post condition operators. The models of these formulae are MSCs. This logic is very expressive, but satisfiability is undecidable when no bound is assumed on the set of models considered. However, a restricted fragment of the logic is proposed to model check existentially bounded Communicating Finite State Machines. Note however that models for template MSC formulae are MSCs, while models for LPOC formulae are arbitrary computations. Even if we only consider LPOC formulae over MSCs, LPOC and template MSCs remain uncomparable. On one hand, holes in template MSCs are not necessarily descriptions of what happens in the future or in the past of an event. By filling hole, one may add concurrent events, i.e. it is possible to say with template MSCs that whenever an action a occurs on process p, a concurrent action b occurs on process q. Clearly, this kind of formula can not be expressed with LPOC. On the other hand, some LPOC formulae that use the until operator do not find their equivalent in template MSC.

Note also that the works in [16],[3] and [9] rely on the existentially bounded nature of models to ensure decidability of model checking (that is, there is a bound b such that every MSC considered possess a linearization where the size of communication channel never exceeds b). This is not sufficient in our case to obtain decidability of diagnosis, as the PCP encoding of section 3 is existentially bounded. The K-influencing restriction is then closer to the universal bound on

MSCs (the contents of communication channels in all linearizations of MSCs is bounded by some integer b) needed to model check HMSCs with global logics [1]. It might be interesting to see whether the layered computation restriction of [15] is sufficient to make diagnosis with LPOC formulae decidable.

References

[1] Rajeev Alur and Mihalis Yannakakis Model Checking of Message Sequence Charts. *CONCUR*, (1999) 114–129.

[2] Albert Benveniste, Eric Fabre, Claude Jard, and Stefan Haar. Diagnosis of asynchronous discrete event systems, a net unfolding approach. *IEEE Transactions on Automatic Control*, **48**(5),(2003) 714–727.

[3] Benedikt Bollig, Dietrich Kuske, and Ingmar Meinecke. Propositional Dynamic Logic for Message-Passing Systems. *FSTTCS*, (2007) 303–315.

[4] J.A. Brzozowski and E.L Leiss. On Equations for Regular Languages, Finite Automata, and Sequential Networks. *TCS*, **10**, (1980) 19–35.

[5] Volker Diekert and Gregor Rozenberg, editors. Book of Traces. *World Scientific*, Singapore, (1995).

[6] Colin Fidge. Logical time in distributed computing systems. *Computer*, **24**(8), (1991) 28–33.

[7] Paul Gastin and Madhavan Mukund. An Elementary Expressively Complete Temporal Logic for Mazurkiewicz Traces. *ICALP*, (2002) 938–949.

[8] Thomas Gazagnaire and Loïc Hélouët. Event Correlation with Boxed Pomsets. *FORTE*, (2007) 160–176.

[9] Blaise Genest, Markus Minea, Anca Muscholl, and Doron Peled. Specifying and Verifying Partial Order Properties Using Template MSCs. *FoSSaCS*, (2004) 195–210.

[10] Loïc Hélouët, Thomas Gazagnaire, and Blaise Genest. Diagnosis from Scenarios. *WODES*, (2006) 307–312.

[11] Jesper.G. Henriksen and P.S. Thiagarajan. Dynamic Linear Time Temporal Logic. *Ann. Pure Appl. Logic*, **96**(1-3), (1999) 187–207.

[12] Dietrich Kuske. Regular sets of infinite message sequence charts. *Information and Computation*, **187**(1), (2003) 80–109.

[13] Richard.E. Ladner, Richard.J. Lipton, and Larry.J. Stockmeyer. Alternating Pushdown and Stack Automata. *SIAM J. Comput.*, **13**(1), (1984) 135–155.

[14] Friedemann Mattern. On the relativistic structure of logical time in distributed systems. *Parallel and Distributed Algorithms*, (1989) 215–226.

[15] B. Meenakshi and Ramaswamy Ramanujam. Reasoning about Layered Message Passing Systems. *VMCAI*, (2003) 268–282.

[16] Doron Peled. Specification and Verification of Message Sequence Charts. *FORTE*, (2000) 139–154.

[17] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D.C Teneketzis. Failure diagnosis using discrete-event models. *IEEE Transactions on Control Systems Technology*, **4**(2), (1996) 105–124.

[18] Moshe.Y Vardi. An Automata-Theoretic Approach to Linear Temporal Logic. *Banff Higher Order Workshop*, (1995) 238–266.

[19] Shaofa Yang, Loïc Hélouët, and Thomas Gazagnaire. Logic based diagnosis for distributed systems. *INRIA Technical report*, (2009).