Diagnosis with a Logic of Partial Order Computations

Shaofa Yang[1], Loïc Hélouët[1], Thomas Gazagnaire[2]

1: INRIA Rennes 2: ENS Cachan-Bretagne
shaofa.yang, loic.helouet, thomas.gazagnaire@irisa.fr

Abstract. This paper proposes a diagnosis framework based on a new partial order logic. The purpose of this logic-based diagnosis is to recover some information that was not observed from a set of formulae describing the expected behaviors of a distributed system and an incomplete observation. We first propose a local partial order logic as a way to describe distributed behaviors, and show that without restriction, satisfiability of a set of formulae, and hence diagnosis is not decidable. We show that we can restrict the class of distributed behaviors to models that ensure decidability of satisfyability and of the diagnosis problem. We then show the complexity of diagnosis and unobserved information inference.

1 Introduction

Diagnosis consists in inferring missing information (unobserved events or states, faulty behaviors,...) from a partial observation of a system. It is an important task in telecommunication networks, where the size and heterogeneity of the architectures forces partial observation. When a fault occurs, it is crucial for safety and economical reasons to discover the cause for the failure and to correct it as fast as possible.
Model-based diagnosis brings automated solutions for this problem. Roughly speaking, it consists in comparing runs of a model with the observed behaviors, and infer the missing parts. One might be interested in discovering if an unobserved fault has occurred, as in [26], or in finding all possible runs that may have lead to the observation, as in [3]. Another possibility is to augment observations with useful information such as causal relationships among observed events, known local properties of the system along the run, and events of interest missed by the observation as in [15]. Within this setting, several models have been proposed to represent runs of distributed systems: automata [26], Petri nets [3], Message Sequence Charts (MSCs) [17]. However, diagnosis always needs an up to date model of the running system to be accurate. In nowaday's architectures, new services are added and modified every day, which calls for a daily redesign of models. Clearly, this is not possible with automata or nets, where the smallest modification implies updating many transitions. We propose a new logic-based approach for diagnosis. The main idea is to collect users and designers knowledge with a logic that describes known interactions as partial orders, and temporal relations between these known pieces of behavior. Hence, a model for a running system is

a conjunction of formulae, and one can expect that a modification of the system only implies redefining a limited number of formulae.
The LPOC logic proposed in this paper is a CTL-like logic of partial order, with past and future modalities. It is a local logic, that defines the shape of causal chains in all executions of a system. We first show that satisfiability of this logic is undecidable. This is not really surprising, as similar undecidability results hold for m-LTL [23], a local temporal logic on Lamport diagrams and for template MSCs [16] a logical framework for MSCs. Hence, without restriction on the logic or on the shape of the explanations (i.e. the models of the formulae that are compatible with the observation), satisfiability of LPOC formulae is undecidable (and as a corollary, diagnosis is also an untractable problem).
We then show a restriction on models allowed for LPOC formulae that makes diagnosis a decidable problem, and show how to extract explanations from a partial observation and a set of formulae. When the runs of the system meet a syntactical requirement called "K-influence", then explanations can be computed as the intersection of an automaton representing all runs that are compatible with the observation and an automaton representing all runs that are compatible with the LPOC description of the system.
This paper is organized as follows. The LPOC logic is described in 2. The negative satisfiability result is shown in Section 3. Section 4 shows that assuming that all executions are K-influencing, satisfiability and diagnosis become decidable problems, and establishes complexity results for diagnosis. Section 5 compares our results with some related works. Section 6 concludes this work.

2 Preliminaries

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

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 [22, 23]) 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 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 diagnosis is performed at a given moment after a finite observation period. 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.

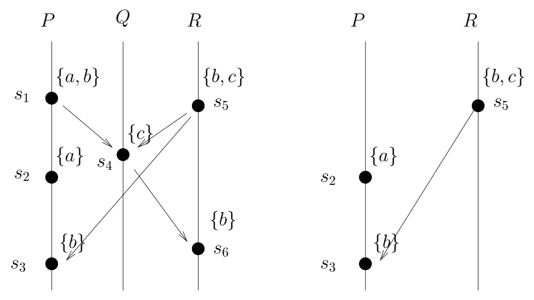

Fig. 1. a) A computation W b) the 1-view of s_3 in W

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 let $\ll_p \subseteq S \times S$ be given by: $s \ll_p s'$ 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 $\ll = (\cup_{p \in \mathcal{P}} \ll_p) \bigcup (\cup_{p,q \in \mathcal{P}, p \neq q} \ll_{pq})$. 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 $\lessdot$. A state s is *minimal* if it has no predecessor. We say s is *maximal* if it has no successor. A *causal chain* is a sequence $s_1 s_2 \dots s_n$ of states where $s_1 \ll s_2 \ll \dots \ll s_n$. A *causal path* is a causal chain $s_1 s_2 \dots s_n$ such that s_n is a maximal state.

We want to 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 which assert that a "pattern" occurs in a bounded past or future of s. Secondly, we consider a branching time framework with CTL-like operators and reason along sequences of causally ordered states. In the sequel, we precise these notions before defining the syntax and the semantics of the logic.

Definition 2. *Let $(S, \eta, \leq, V)$ be a computation, $s \in S$ and m a natural number. The m*-view *of 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'.*

In particular, the 0-view and 0-frontier of s are both the singleton set $\{s\}$. Note that s has at most $|\mathcal{P}|$ successors, one belonging to each S_p. Thus, inductively, the m-view of s contains at most $N_m = \sum_{i=0}^{m} |\mathcal{P}|^i = \frac{1-|\mathcal{P}|^{m+1}}{1-|\mathcal{P}|}$ states, and the same bound holds for the size of m-frontiers. 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', $\leq'$, V' are the respective restrictions of $\eta, \leq, V$ to S'. The* restriction *of W to a subset of states $S' \subseteq S$ is the computation $W' = (S', \widehat{\eta}, \widehat{\leq}, \widehat{V})$ where $\widehat{\eta}$, $\widehat{\leq}$, $\widehat{V}$ are the respective restrictions of η, $\leq$ and V to 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 $LPOC(\mathcal{P}, \mathcal{A})$.*
- *If $\varphi, \varphi' \in LPOC(\mathcal{P}, \mathcal{A})$, then $\neg\varphi$ and $\varphi \vee \varphi'$ are formulae 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 $LPOC(\mathcal{P}, \mathcal{A})$.*

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 path $s_1 s_2 \ldots s_n$ 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. Furthermore, we define some derived CTL temporal operators as follows. First, we have $\mathsf{EF}\varphi \equiv \mathsf{EU}(\mathit{true}, \varphi)$ and $\mathsf{EG}\varphi \equiv \mathsf{EU}(\varphi, \varphi \wedge \bigwedge_{p\in\mathcal{P}} \neg\mathsf{EX}\ loc_p)$. That is, $\mathsf{EF}\varphi$ means that there exists a causal path ρ starting from the current state such that φ holds at some state of ρ. And $\mathsf{EG}\varphi$ demands that there exists a causal path ρ starting from the current state such that φ holds at every state of ρ. Secondly, we define $\mathsf{AX}\varphi \equiv \neg\mathsf{EX}(\neg\varphi)$; $\mathsf{AF}\varphi \equiv \neg\mathsf{EG}\neg\varphi$; and $\mathsf{AG}\varphi \equiv \neg\mathsf{EF}\neg\varphi$. We write $\mathsf{EX}_p(\varphi)$ for the formula $\mathsf{EX}(loc_p \wedge \varphi)$. The notations EF_p, EG_p, AX_p, AF_p, AG_p are defined in the same way. We will say that two formulae φ and φ' are *logically equivalent* iff for any computation W, any state s of W, we have $W, s \models \varphi$ iff $W, s \models \varphi'$.

In LPOC we can also assert the truth of atomic propositions at a state of a computation. Let $a \in \mathcal{A}$. Consider the formula $\varphi_a = \vee_{p\in\mathcal{P}}\big(loc_p \wedge \downarrow_{0,\{a\}}(T_{p,a})\big)$, where each $T_{p,a}$ is the computation containing a singleton state of location p and valuation $\{a\}$. It is clear that for any computation W and a state s, we have $W, s \models \varphi_a$ iff the valuation of s contains a.

Observations of a system do not necessarily record all causal ordering among observed states. More precisely, if in an observation O, a state s is causally earlier than s', then in any explanation W of O, the state in W corresponding to s must be causally earlier than the state in W corresponding to s'. But the converse need not hold. For this reason, we need to introduce the formulae $\Downarrow_{m,A}(T)$ and $\Uparrow_{m,A}(T)$, where m is a natural number, $A \subseteq \mathcal{A}$ and T is a computation such that the valuation of any state s of T is a subset of A. Let $W = (S_W, \eta_W, \leq_W, V_W)$ be a computation and $s \in S_W$. Consider the formula $\Downarrow_{m,A}(T)$ where $T = (S_T, \eta_T, \leq_T, V_T)$. We say $W, s \models \Downarrow_{m,A}(T)$ iff there exists an injective mapping $f : S_T \rightarrow \downarrow_m(s)$ satisfying:

- For every $s \in S_T$, $\eta_T(s) = \eta_W(f(s))$ and $V_T(s) = V_W(f(s)) \cap A$.
- For every $s, s' \in S_T$, $s \leq_T s'$ implies $f(s) \leq_W f(s')$. But we do not demand the converse.

The semantics of $\Uparrow_{m,A}(T)$ is defined in the same way, except the mapping f is from S_T to $\uparrow_m(s)$. Remark that $\Downarrow_{m,A}(T)$ and $\Uparrow_{m,A}(T)$ can be easily defined in LPOC. Recall that the m-view of a state s of a computation contains at most N_m states. Now it is easy to see that $\Downarrow_{m,A}(T)$ is equivalent to the formula $\bigvee_{X\in\mathcal{X}} \downarrow_{m,A}(X)$ where $\mathcal{X}$ is the collection of computations $X = (S_X, \eta_X, \leq_X, V_X)$ containing at most N_m states and for which there exists an injective mapping $g : S_T \rightarrow S_X$ satisfying:

- For every $s \in S_T$, $\eta_T(s) = \eta_X(g(s))$ and $V_T(s) = V_X(g(s))$.
- For every $s, s' \in S_T$, $s \leq_T s'$ implies $g(s) \leq_X g(s')$.

Lastly, 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 $W \models \bigwedge_{p\in\mathcal{P}} \mathsf{EU}(\neg loc_p, loc_p \wedge \varphi_p)$.

3 Diagnosis

Our goal is to use LPOC for diagnosis, that is find explanations for an observation from a set of formulae described in LPOC, that represents a specification of the observed system. 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. The sets $\mathcal{A}^{ob}$ and $\mathcal{A}^{ex}$ are assumed to be disjoint. An observable atomic proposition is one whose truth value may be "recorded" during system execution. On the other hand, atomic propositions in $\mathcal{A}^{ex}$ are those that may explain the cause of what have been observed. Thus in typical applications, 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. The observable atomic propositions are in a sense manifestations of the explanatory atomic propositions. To formulate the diagnosis problem, we introduce the notions of *observation* and *explanation*.

Definition 5. *An* observation *is a computation with valuations in* $2^{\mathcal{A}^{ob}}$. *That is,* $(S, \eta, \leq, V)$ *is an observation iff* $V(s) \subseteq \mathcal{A}^{ob}$ *for every* $s \in S$. *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.

Intuitively, an observation describes everything that is "recorded" during an the execution of some prefix of a computation. We assume that the observation is produced by a monitoring mechanisms that observes each process and transmits "recorded" observable atomic propositions to a 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. We emphasize that the observation may have only recorded an initial portion of the explanation. Thus, the image of f may be a proper subset of the states of W whose valuation contains observable atomic propositions. 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 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*. 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.
Condition (ii) asserts that the recorded causality orderings must originate from an execution. However, we do not demand the converse. That is, some causality orderings in the explanation may not be "recorded".
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.

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

Now we can define the diagnosis problem associated with LPOC.

Definition 7. *Let $\mathcal{P}$ be a set of processes, $\mathcal{A}^{ob}$ be a set of observable atomic propositions, and $\mathcal{A}^{ex}$ be a set of atomic explanatory propositions. 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 $\{\Phi_p^{spec}\}_{p\in\mathcal{P}}$ over $(\mathcal{P}, \mathcal{A}^{ob} \cup \mathcal{A}^{ex})$, determine whether there exists a $\{\Phi_p^{spec}\}_{p\in\mathcal{P}}$-explanation for O.*

In what follows, we will often omit the subscript $p \in \mathcal{P}$. The formulae $\{\Phi_p^{spec}\}$ specify the knowledge about execution of the system The objective of diagnosis is to figure out whether there exists an explanation for what have been observed. In case such 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 $\{\Phi_p^{spec}\}_{p\in\mathcal{P}}$ 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 $W \models \{\Phi_p^{spec}\}_{p\in\mathcal{P}}$ and a is in the W-valuation of s. Thus, when the answer to the diagnosis problem is positive, we would 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 1. *The LPOC-diagnosis problem is undecidable.*

proof This theorem is proved by reduction from the Post Correspondence Problem (PCP). The reduction is similar to that used in decision problems related to message sequence charts (see for instance [16]).
Consider an instance of PCP with Σ a finite alphabet such that $|\Sigma| > 1$, and $(g_1, h_1), \ldots, (g_n, h_n)$ a finite sequence of pairs of words over Σ. A solution (if it exists) is a finite sequence of indices $j_1, j_2, \ldots, j_t$ in $\{1, 2, \ldots, n\}$ such that $g_{j_1} g_{j_2} \ldots g_{j_t} = h_{j_1} h_{j_2} \ldots h_{j_t}$. The reduction is as follows. We pick $\mathcal{P} = \{p, q, r\}$. Let $\mathcal{A}_p = \{a^p \mid a \in \Sigma\} \cup \{i^p \mid i \in \{1, 2, \ldots, n\}\}$, $\mathcal{A}_q = \{a^q \mid a \in \Sigma\} \cup \{i^q \mid i \in \{1, 2, \ldots, n\}\}$, and $\mathcal{A}_r = \{i^{pr}, i^{qr} \mid i \in \{1, 2, \ldots, n\}\}$. We take $\mathcal{A}^{ex} = \mathcal{A}_p \cup \mathcal{A}_q \cup \mathcal{A}_r$, and $\mathcal{A}^{ob} = \emptyset$. Let O be the empty observation, that is, a computation with no states. Thus, any computation is an explanation for O.

We encode solutions to the PCP instance by computations which have the form illustrated in Figure 2. The total ordering of states on each process is drawn as a vertical line with the minimum state at the top. The downward-sloping arrows represent pairs (s, s') of states in the successor relation such that s, s' are on different locations. The label of each state indicates its valuation. We can construct formulae Φ_p,Φ_q,Φ_r such that $\{\Phi_p, \Phi_q, \Phi_r\}$ is satisfiable iff there exists a computation W with $W \models \{\Phi_p, \Phi_q, \Phi_r\}$ and W represents a solution to the PCP instance. It will then follow that the PCP instance has a solution iff there exists an explanation for O satisfying $\{\Phi_p, \Phi_q, \Phi_r\}$.

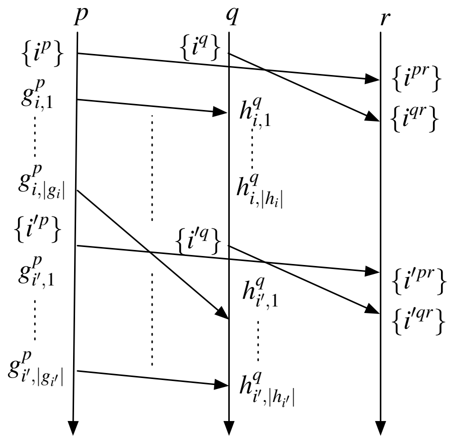

Fig. 2. Encoding solutions to PCP with computations

In the sequel, we outline the construction of Φ_p, Φ_q and Φ_r. For $A \subseteq \mathcal{A}_p$, we write $VAL(p, A)$ for the formula $loc_p \wedge \downarrow_{1,\mathcal{A}_p}(T)$ where T is the computation with a singleton state of location p and valuation A. In other words, the formula $VAL(p, A)$ asserts that the current state has location p and valuation A. We define the notations $VAL(q, A)$, $VAL(r, A)$ in the same way. The formula Φ_p is the conjunction of the following formulae.

(P1) $\bigvee_{i\in\{1...n\}} VAL(p, \{i^p\})$. It asserts that the valuation of the minimum state of S_p is $\{i^p\}$, for some i in $\{1 \dots n\}$.

(P2) $\mathsf{EG}_p(\bigwedge_{i\in\{1...n\}}(VAL(p, \{i^p\}) \rightarrow \varphi_i))$ where each φ_i is given as follows. Let $g_{i,j}$ be the j-th letter of g_i, for $j = 1, \dots, |g_i|$. Then φ_i asserts that there exist m states s_1, s_2, …, $s_{|g_i|}$ in S_p with s_1 being a successor of the current state, and s_{j+1} a successor of s_j for $j = 1, \dots, |g_i| - 1$. Furthermore, the valuation of s_j is $\{g^p_{i,j}\}$ for $j = 1, 2, \dots, |g_i|$, and either $s_{|g_i|}$ has no successor in S_p, or $s_{|g_i|}$ has a successor in S_p whose valuation is $\{\ell^p\}$ for some ℓ in $\{1, \dots, n\}$.
More precisely, $\varphi_i = \downarrow_{|g_i|+1,A_p}(T) \vee (\vee_{\ell=1,2,\dots,n} \downarrow_{|g_i|+1,A_p}(T'_\ell))$, where T is the computation $(\{t_1, t_2, \dots, t_{|g_i|}\}, \eta_T, \leq_T, V_T)$ with $\eta_T(t_j) = p$

for every $j = 1, 2, \ldots, |g_i|$, $t_1 \leq_T t_2 \leq_T \ldots \leq_T t_{|g_i|}$, and $V_T(t_j) = \{g^p_{i,j}\}$ for every $j = 1, 2, \ldots, |g_i|$. Similarly, for each $\ell = 1, 2, \ldots, n$, T'_ℓ is the computation $(\{t_1, t_2, \ldots, t_{|g_i|}, t_{|g_i|+1}\}, \eta_{T'}, \leq_{T'}, V_{T'_\ell})$ with $\eta_{T'_\ell}(t_j) = p$ for every $j = 1, 2, \ldots, |g_i| + 1$, $t_1 \leq_{T'_\ell} t_2 \leq_{T'} \ldots \leq_{T'} t_{|g_i|+1}$, $V_{T'_\ell}(t_j) = \{g^p_{i,j}\}$ for every $j = 1, 2, \ldots, |g_i|$, and $V_{T'_\ell}(t_{|g_i|+1}) = \{\ell^p\}$.

(P3) $\mathsf{EG}_p(\bigwedge_{a\in\Sigma}(\mathit{VAL}(p, \{a^p\}) \rightarrow \mathsf{EX}_q\, \mathit{VAL}(q, \{a^q\})))$. Intuitively, this asserts that every state of valuation $\{a^p\}$ of p is "matched" by a state of q with valuation $\{a^q\}$. We emphasize that the matchings are guaranteed to be one-to-one and order-preserving. This is due to the fact that for every pair p, q of processes $<<_{pq}$ is "fifo", that is one-to-one and order-preserving. More precisely, for each state s of S_p, there is at most one state s' of S_q such that $s << s'$ (and thus $s << pqs'$). Similarly, for each state s of S_q, there is at most one state s' of S_p such that $s' << s$ (and thus $s' << pqs$). Finally, for any s_1,s_2 of S_p, s'_1, s'_2 of S_q such that $s_1 << s'_1$, $s_2 << s'_2$, we have that $s_1 \leq s_2 iff s'_1 \leq s'_2$.

(P4) $\mathsf{EG}_p(\bigwedge_{i\in\{1,2,\ldots,n\}}(\mathit{VAL}(p, \{i^p\}) \rightarrow \mathsf{EX}_r\, \mathit{VAL}(r, \{i^{pr}\})))$. That is, for every state of p with valuation $\{i^p\}$, there is a "matching" state of r with valuation $\{i^{pr}\}$.

The formula Φ_q asserts the conjunction of the following conditions.

(Q1) The valuation of the minimum state of q is $\{i^q\}$ for some i in $\{1, 2, \ldots, n\}$. This is similar to case (P1) in the construction of Φ_p.

(Q2) For each $i \in \{1, 2, \ldots, n\}$, if a state of q has valuation $\{i^q\}$, then there exist $|h_i|$ subsequent states s_1, s_2, ..., $s_{|h_i|}$ of q, whose valuations are, respectively, $\{h^q_{i,j}\}$, with $h_{i,j}$ being the j-th letter of h_i, for $j = 1, 2, \ldots, |h_i|$. Further, either $s_{|h_i|}$ has no successor of location q, or $s_{|h_i|}$ has a successor of location q and valuation $\{\ell^q\}$ for some ℓ in $\{1, 2, \ldots, n\}$. The detailed formula of this case can be constructed in the same way as case (P2) in the construction of Φ_p.

(Q3) For each $a \in \Sigma$, if a state s of q has valuation $\{a^q\}$, then $\downarrow_{1,\mathcal{A}_p}(T)$ holds at s, where T is the computation containing a singleton state of location p and valuation $\{a^p\}$. That is, every state of valuation $\{a^q\}$ of q is "matched" by a state of valuation $\{a^p\}$ of p.

(Q4) For each $i \in \{1, 2, \ldots, n\}$, if a state of q has valuation $\{i^q\}$, then it has a successor of location r and valuation $\{i^{qr}\}$. This is similar to case (P4) of Φ_p.

Finally, Φ_r asserts the conjunction of the following conditions.

(R1) The minimum state of r has valuation $\{i^{pr}\}$ for some i in $\{1, 2, \ldots, n\}$.

(R2) For each index $i \in \{1, 2, \ldots, n\}$, if a state of r has valuation $\{i^{pr}\}$, then it has a successor state, say, s, of location r and valuation $\{i^{qr}\}$. Further, either s has no successor of location r, or s has a successor of location r and valuation $\{\ell^{pr}\}$ for some $\ell \in \{1, 2, \ldots, n\}$. The detailed formula can be constructed in the same way as case (P2) of Φ_p.

(R3) For each $i \in \{1, 2, \ldots, n\}$, if a state s of r has valuation $\{i^{pr}\}$, then $\downarrow_{1,\mathcal{A}_p}(T)$ holds at s, where T is the computation containing a singleton state of location p and valuation $\{i^p\}$.

(R4) For each $i \in \{1, 2, \ldots, n\}$, if a state s of r has valuation $\{i^{qr}\}$, then $\downarrow_{1,\mathcal{A}_q}(T)$ holds at s, where T is the computation containing a singleton state of location p and valuation $\{i^q\}$.

It is easy to see that the constructions of Φ_p, Φ_q, Φ_r encode a PCP instance, and that there exists an explanation for O satisfying Φ_p, Φ_q, Φ_r if and only if the corresponding instance of PCP has a solution. This completes the proof for Theorem 1. □

4 Diagnosis with *K*-influencing computations

We have shown in previous section (Theorem 1) that the diagnosis problem is undecidable in general. A question that naturally arises is whether we can identify a subclass of computations for which the problem becomes tractable. For this, we identify a subclass of computations called K-influencing computations, and show that the LPOC diagnosis problem (and thus computing the summary) is decidable within this class.

Definition 8. *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) *(iv)* $s'_1 \not< s_n$*.*

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 3. We now state the main result of this section.

Theorem 2. *Given an observation O, a $\mathcal{P}$-indexed family $\{\Phi_p^{spec}\}_{p\in\mathcal{P}}$ 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 $\{\Phi_p^{spec}\}_{p\in\mathcal{P}}$-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 $\{\Phi_p^{spec}\}_{p\in\mathcal{P}}$ a $\mathcal{P}$-indexed family of formulae. We can slightly adapt the definition of summaries in section 3 to $K-$ influencing computations: the K-*summary* of O under $\{\Phi_p^{spec}\}_{p\in\mathcal{P}}$ is the mapping $g : S_O \rightarrow 2^{\mathcal{A}^{ex}}$ 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 $W \models \{\Phi_p^{spec}\}_{p\in\mathcal{P}}$ and a is in the W-valuation of s.

Corollary 1. *Given an observation O, a $\mathcal{P}$-indexed family $\{\Phi_p^{spec}\}_{p\in\mathcal{P}}$ of LPOC formulae, and an integer $K \in \mathbb{N}$, one can effectively compute the K-summary of O under $\{\Phi_p^{spec}\}_{p\in\mathcal{P}}$.*

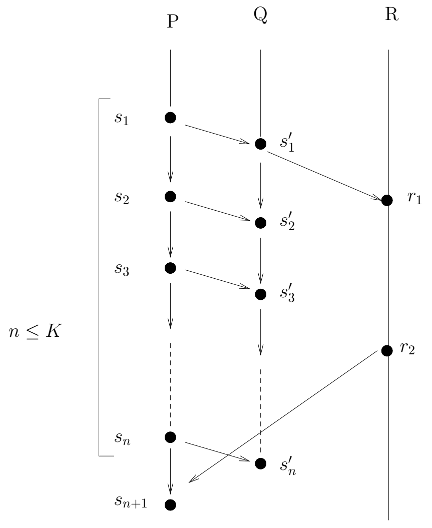

Fig. 3. K-influencing computations

Through the rest of this section, we prove Theorem 2 and Corollary 1. We fix the integer K, observation O and the formulae $\{\Phi_p^{spec}\}_{p\in\mathcal{P}}$. Recall from section 2 that we can easily construct a single formula Φ^{spec} such that for any computation W, $W \models \Phi^{spec}$ iff $W \models \{\Phi_p^{spec}\}_{p\in\mathcal{P}}$. In what follows, we fix Φ^{spec}. 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 2 consists of two steps. Firstly, we show that K-influencing computations can be identified with Mazurkiewicz traces ([7]) 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 [20]. Secondly, we construct three finite state automata Aut_K, $Aut_{\Phi^{spec}}$, 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 $\mathcal{W}_K$, $Aut_{\Phi^{spec}}$ accepts σ iff $W_\sigma \models \Phi^{spec}$. And Aut_O accepts σ iff W_σ is an explanation for O. The crux is the construction of $Aut_{\Phi^{spec}}$. We shall give $Aut_{\Phi^{spec}}$ in the form of a two-way alternating automaton, which can be transformed to a finite state automaton. The basic idea is similar to [14] and the usual translation from LTL to alternating automata [29]. The new technicality in our construction is in checking conformance with formulae of the form $\downarrow_{m,A}(T)$, $\uparrow_{m,A}(T)$. With Aut_K, Aut_O, $Aut_{\Phi^{spec}}$, it follows that there exists a sequence in $\Sigma^\star$ accepted by Aut_K, Aut_O, $Aut_{\Phi^{spec}}$ iff $\mathcal{W}_K$ contains a computation W such that $W \models \Phi^{spec}$ and W is an explanation for O. This then establishes Theorem 2. As for Corollary 1, we shall 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 $W_\sigma \models \Phi^{spec}$, 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 $\{\Phi_p^{spec}\}_{p\in\mathcal{P}}$.

4.1 Encoding *K*-influencing computations with Traces

We recall that a Mazurkiewicz trace ([7]) 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 $\Gamma = \mathcal{P} \times \Gamma_{pre} \times \Gamma_{suc} \times 2^{\mathcal{A}}$. 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 $D \subseteq \Sigma \times \Sigma$ via: $(p, PRE, SUC, A)\ D\ (p', PRE', SUC', A')$ iff one of the following conditions holds:

- $p = p'$.
- $p \neq p'$. For some $i \in [K]$, $pre(p', i) \in PRE$ and $suc(p,i) \in SUC'$.
- $p \neq p'$. 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, $suc(\hat{q}, \hat{\imath}) \in \Gamma_{suc}$ is in SUC iff s has a successor $\hat{s}'$ with $\eta(\hat{s}') = \hat{q}$ (such a $\hat{s}'$ is unique) and $\hat{\imath} = \hat{\jmath} \mod K$ where $\hat{\jmath}$ 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 $tr(W) = (S, \leq, \lambda_W)$. The following results follow easily from the definitions.

Proposition 1. *Let $W \in \mathcal{W}_K$. Then $tr(W)$ is a trace over (Σ, I). Furthermore, if W' is a K-influencing computation, then $W = W'$ iff $tr(W) = tr(W')$.*

Figure 4 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 $A_1, \dots A_9$.

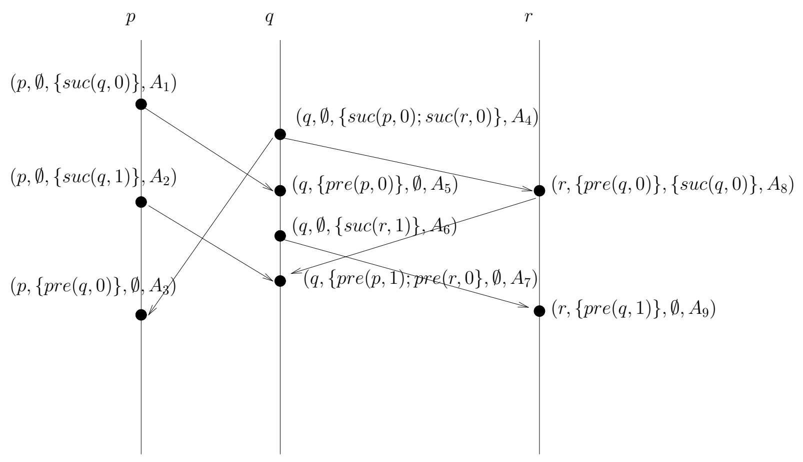
p
q
r
$(p, \emptyset, \{suc(q,0)\}, A_1)$
$(q, \emptyset, \{suc(p,0); suc(r,0)\}, A_4)$
$(p, \emptyset, \{suc(q,1)\}, A_2)$
$(q, \{pre(p,0)\}, \emptyset, A_5)$
$(r, \{pre(q,0)\}, \{suc(q,0)\}, A_8)$
$(q, \emptyset, \{suc(r,1)\}, A_6)$
$(p, \{pre(q,0)\}, \emptyset, A_3)$
$(q, \{pre(p,1); pre(r,0\}, \emptyset, A_7)$
$(r, \{pre(q,1)\}, \emptyset, A_9)$

Fig. 4. A 2-influencing computation and its Σ-labeling

A *linearization* of a trace $(E, \sqsubseteq, \lambda)$ is a sequence $\lambda(e_1)\lambda(e_2)\dots\lambda(e_n)$ where $e_1, \dots, e_n$ are distinct members of E, $E = \{e_1, \dots, e_n\}$, and for any $i, j \in \{1, \dots, n\}$, $e_i \leq e_j$ implies $i \leq j$. For any computation $W \in \mathcal{W}_K$, we will call *Σ-linearization of* W a linearization of $tr(W)$, and denote by $Lin^{\Sigma}(W)$ the set of Σ-linearizations of W. Let $Lin^{\Sigma}_K = \bigcup_{W \in \mathcal{W}_K} Lin^{\Sigma}(W)$.
Note that computations in $\mathcal{W}_K$ can be uniquely constructed from sequences in Lin^{Σ}_K. 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 $\sigma_1, \sigma_2, \dots, \sigma_h$, $h \leq |\Sigma|$, such that $\sigma \preccurlyeq \sigma_1 \preccurlyeq \sigma_2 \dots \preccurlyeq \sigma_h \preccurlyeq \sigma$ and $last(\sigma) D last(\sigma_1) D last(\sigma_2) \dots last(\sigma_h) D last(\sigma)$. For every $\sigma \in Lin^{\Sigma}_K$, 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, we have:

Proposition 2. *Let $\sigma \in Lin_K^\Sigma$ and $W = poc(\sigma)$. Then $W \in \mathcal{W}_K$ and $\sigma \in Lin^\Sigma(W)$.*

4.2 Automata construction

Here we construct three finite state automata Aut_K, Aut_O, $Aut_{\Phi^{spec}}$ which have the following properties:

- For $\sigma \in \Sigma^\star$, σ is accepted by Aut_K iff $\sigma \in Lin_K^\Sigma$.
- Suppose $\sigma \in \Sigma^\star$ is in Lin_K^Σ. Then σ is accepted by Aut_O iff $poc(\sigma)$ is a K-influencing explanation for O. Note that we will not consider sequences outside Lin_K^Σ. Aut_O will then be constructed from Aut_K.
- Suppose $\sigma \in \Sigma^\star$ is in Lin_K^Σ. Then σ is accepted by $Aut_{\Phi^{spec}}$ iff $poc(\sigma)$ satisfies Φ^{spec}. Again, we will not consider sequences outside Lin_K^Σ.

It follows that a sequence $\sigma \in \Sigma^\star$ is accepted by the product of Aut_K, Aut_O, $Aut_{\Phi^{spec}}$ iff $poc(\sigma)$ is a K-influencing $\{\Phi_p^{spec}\}$-explanation for O. The construction of Aut_K and Aut_O is obvious from the definitions of K-influencing computations and basic properties of traces (see [7] for more details).

Construction of Aut_K: For a given K, a set of processes $\mathcal{P}$, and a set of atomic propositions $\mathcal{A}$, the automaton $Aut_K = (Q, q^i, \Sigma, \delta, F)$ can be constructed as follows.
For a given word $\sigma = a_1 \dots a_n$ we will say that letter $a_i = (p, PRE, SUC, A)$ is *acknowledged* by $q \in \mathcal{P}$ in σ if and only if

- for every $j \in [K]$, $suc(q, j)$ is not in SUC, or
- there exists a_j, a_k, $i < j < k \leq n$ such that in $poc(\sigma)$, $x = a_1..a_j \in S_q$ $y = a_1..a_i \in S_p$ $z = a_1..a_i \in S_p$, and $x \leq y \leq z$. a_i

When a letter of σ is acknowledged for every $q \in \mathcal{P}$, we will say that it is *acknowledged.* Note that from the definition of $K-$influencing computations, if σ is a prefix of a linearization of some $K-$influencing computation, then the number of letters of σ located on a process p that are not acknowledged by q is necessarily lower or equal to K. We are now ready to define the components of Aut_K :

- Σ is defined as before
- elements of Q are a tuples of the form (σ, cnt) where σ is a word in Σ^* such that for each pair of processes p, q the number of letters of the form (p, PRE, SUC, A) where SUC contains $suq(q, i)$ for $i \in [K]$ is at most K, and $cnt : \mathcal{P} \times \mathcal{P} \to 0 \dots K$ is a function that associates a number in $0 \dots K$ to each pair p, q in $\mathcal{P}$. This number corresponds to the number of letters that are not yet acknowledged between two processes. The initial state of the automaton is defined by $q^i = (\epsilon, cnt_0)$ where ϵ is the empty word and cnt_0 associates value 0 to each pair $p, q \in \mathcal{P}$.

- $\delta \subseteq Q \times \Sigma \times Q$ and
  - $\big(\sigma, cnt), a, (\sigma', cnt)\big) \in \delta$ for any a of the form $(p, \emptyset, \emptyset, A)$ with $p \in \mathcal{P}$ and $A \subseteq \mathcal{A}$.
  - $\big((\sigma, cnt), a, (\sigma', cnt')\big) \in \delta$ if σ' is the projection of $\sigma.a$ on non-acknowledged events, and $a = (p, PRE, SUC, A)$ for some $p \in \mathcal{P}$ and $A \subseteq \mathcal{A}$, and furthermore, for every $pre(p', i) \in PRE$, there exists a letter $a' = (p', PRE', SUC', A')$ in σ for which $suc(p, i) \in SUC'$ and a' is the first letter that contains $suc(p, j)$ for some $j \in [K]$. For every $suc(p', i) \in SUC$, we have $cnt'(p, p') = cnt(p, p')+1$, and for every $pre(p', i) \in PRE$, we have $cnt'(p', p) = cnt(p', p) - 1$.
- The set of accepting states is $F = Q$. Any state is accepting, as Lin_K^{Σ} is prefix-closed by definition.

This construction of Aut_K immediately provide indications on the size of the model.

Proposition 3. *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, the size of the automaton Aut_K that recognizes linearizations of $K-$influencing computations over $\mathcal{P}$ with valuations in $\mathcal{A}$ is in $\mathcal{O}(|\Sigma|^{K.|\mathcal{P}|^2})$.*

proof: Note that in any state (σ, cnt) the function $cnt(p, p')$ count the letters of σ located on process p that are not acknowledged by p'. That is, function cnt does not add new information that can not be computed directly from σ, and is just introduced for convenience. As there can be at most K letters per pair of processes that are not acknowledged in a word labeling a state of Aut_K the size of such word is bounded by $K.|\mathcal{P}|^2$. The size of Aut_K is then not greater than $|\Sigma|^{K.|\mathcal{P}|^2}$.$\square$

Construction of Aut_O: 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. Note that it is not sufficient to memorize only the subset of states of O that have been recognized so far, as we must also ensure that two states ordered in O are also ordered in $poc(\sigma)$, for every linearization σ accepted by Aut_O. Hence, Aut_O can not be built as a product between Aut_K and an automaton that recognizes linearizations of O (and where states are cuts of O), as this product allows for computations that are not explanations of O.
At the same time as we play a linearization accepted by Aut_K, we need to memorize states of the observation that have already been seen, their respective ordering, but also the ordering imposed by unobserved states and causalities. Let W be a computation, and let s be a sate of W such that $V(s) \cap \mathcal{A}^{ob} \neq \emptyset$, that is if O_W is the restriction of W to observed states corresponding to computation W, then s appears in O_W. The future processes of s (or simply future of s) is the set of processes where a causal successor of s appears. It is defined as

$$F(s) = \{p \in \mathcal{P} \mid \exists s', s \leq s' \wedge \eta(s') = p\}$$

Then, if a new observable state s occurs after W, and if $\eta(s') \in F(s)$, we necessarily have $s \leq s'$.
Computing $F(s)$ for each observed state on a complete computation is easy, but we need to do it on the fly for arbitrary long computations. This information will be sufficient to ensure that the ordering in an observation is preserved by a $K-$influencing computation. For this, we define a new data structure that abstracts most of the computation and recalls causal dependencies. Recall that relation $<<_{pq}$ behaves as a FIFO channel. For each state s of S, and every run σ we define a function $SINCE_{\sigma}(s,p,q)$ that recalls actions of σ located on process p that must have an immediate successor on process q, that have been played after p has appeared in the future of s but have not yet been acknowledged by q. By construction, as we are only considering linearizations of $K-$influenced computations, $SINCE_{\sigma}(s,p,q)$ is of size at most K. It can be inductively defined as follows:

- $SINCE_{\epsilon}(s,p,q) = \epsilon$ for every s,p,q
- $SINCE_{\sigma.a}(s,p,q) = SINCE_{\sigma}(s,p,q)$ if p is not in $F(s)$.
- $SINCE_{\sigma.a}(s,p,q) = SINCE_{\sigma}(s,p,q).succ(i,p)$ if p is in $F(s)$ and $a = (p, SUC, PRE, A)$, with $suc(q,i) \in SUC$ (intuitively, add a reference to an unmatched event that has no direct causal successor in the word $\sigma.a$)
- $succ(i,p).SINCE_{\sigma.a}(s,p,q) = SINCE_{\sigma}(s,p,q)$ if p is in $F(e)$ and $a = (q, SUC, PRE, A)$, with $pre(p,i) \in PRE$ (intuitively, remove the first letter of the word when a matching event on process q is found).

When a new observed action of Aut_K is played, leading to the creation of a new observed state s in the associated computation, we set $F(s) = \{p\}$ and $SINCE(e,q,q') = \emptyset$ for every $q \neq p$. If we call $Z_q = \{suc(q,i), i \in [K]\}$, $SINCE(e,p,q)$ associates a word in Z_q^*, to each pair of state and process of S. Note that due to K-influence, this word is of size at most K.

Lemma 1. *Let $\sigma = a_1...a_k$ be a run of Aut_K, $b = (p, SUC, PRE, A)$ be a letter of Σ such that $\sigma.b$ is also accepted by Aut_K. Let $a_i = (q, SUC_i, PRE_i, A_i)$ be an observable letter of σ such that $q \neq p$ and $p \notin F(a_1 \ldots a_i)$ in $poc(\sigma)$. Then, $p \in F(a_1 \ldots a_i)$ in $poc(\sigma.b)$, if and only if there exists $q' \in \mathcal{P}$, such that $q' \in F(a_1 \ldots a_i)$, $SINCE(a_i, q', p) = suc(p,i).w$ for some w, and $pre(q',i) \in PRE$.*

proof : first of all, as $p \notin F(a_1 \ldots a_i)$ in $poc(\sigma)$, we know that there is no causal chain from q to p in $poc(\sigma)$. Hence, if $p \in F(a_1 \ldots a_i)$ in $poc(\sigma.b)$, then b is located on process p, and adding b to σ creates a new causal dependency from a process $q' \neq p$ to p (otherwise, for every observed event e of sigma, $F(e)$ remains unchanged. Note also that we necessarily have, $q' \in F(a_1 \ldots a_i)$. b is of the form (p, SUC, PRE, A). Now let us suppose that $pre(q',i) \in PRE$ and $SINCE(a_i, q', p) = suc(p,i).w$. Then, it means that there exists a letter $a_j = (q', SUC_j, PRE_j, A_j)$ such that state $a_1 \ldots a_j$ is a causal successor of $a_1 \ldots a_i$ in $poc(\sigma)$, and such that $suc(p,i) \in SUC_j$. According to the dependence relation on Σ, we have that $a_1 \ldots b$ is a causal successor of $a_1 \ldots a_j$ in $poc(\sigma.b)$. Then there is a causal chain from a_i to b, and $p \in F(a_i)$ in $poc(\sigma.b)$. Now let us suppose that $SINCE(a_i, q', p) = suc(p, i).w$ and that for every $pre(x, j) \in PRE$, either $x \neq q'$ or $i \neq j$. Then, b and every action played since $q' \in F(a_i)$ are independent, and $p \notin F(a_1 \dots a_i)$. $\square$

The intuition behind this lemma is simple. To memorize future processes along a potentially infinite computation, we only need to maintain a finite set of words (one for each pair $p, q \in \mathcal{P}$ per observed event) and a function that recalls for each p and each observed event s if $p \in F(s)$. This finite information is sufficient to update $F(e)$ at each step of a $K-$influencing computation, and then decide whether the ordering among states of the observations O is respected in current computation.

Figure 5 illustrates the online construction of F and $SINCE$. In this figure, we suppose that we study a system composed of processes $\mathcal{P} = \{P, Q, R\}$, and that a word σ of Aut_k have been played. Figure 5 depicts the computation $poc(\sigma)$: black circles represent observed states, and black crosses represent unobserved states. In this computation, we have $P \in F(o_1), F(o_2), F(o_3)$ and $F(o_4)$, $Q \in F(o_1), F(0_2), F(o_4)$ and $F(o_5)$, $R \in F(o_1), F(o_4), F(o_5)$ and $F(o_6)$. Let us suppose that $\lambda(u_6) = (Q, SUC_6, PRE_6, A_6)$, with $suc(R, i) \notin SUC$ for every $i \in [K]$. Suppose that $\lambda(u_7) = (Q, SUC_7, PRE_7, A)$, with $suc(R, 0) \in SUC_7$. We have $SINCE(o_2, Q, R) = suc(R, 0)$. Let us append a new unobserved event u_{10} to the computation, such that $\lambda(u_{10}) = (R, SUC_{10}, PRE_{10}, A_{10})$, with $pre(q, 0) \in PRE_{10}$. Then, a new causal dependency between u_7 and u_{10} exists, and $R \in F(e)$ in $poc(\sigma.u_{10})$.

We are now ready to build Aut_O, the automaton that recognizes all linearizations of $K-$influencing explanations of an observation $O = (S_O, \eta_O, \leq_O, V_O)$. Let us denote by $\mathcal{SI}_O$ the set of function that associate to each triple s, p, q in $S_O \times \mathcal{P} \times \mathcal{P}$ a word in Z_q^* of size at most K, and by $\mathcal{F}_O$ the set of functions that associate to each observed state of O a subset of $\mathcal{P}$. We define $Aut_O = (Q_O, q_O^i, \Sigma, \delta_O, F_O)$, where:

- Σ is defined as usual
- $Q_O \subseteq Q \times 2^{S_0} \times \mathcal{F}_O \times \mathcal{SI}_O$
- $F_O = Q \times \{S_O\} \times \mathcal{F}_{O,\mathcal{P}} \times \mathcal{S}$ that is a linearization is accepted if an only if all states of S have been observed
- $q_O^i = (q^i, \emptyset, f^i, SINCE^i)$, where f^i is the function that associates $\emptyset$ to any state of S_O, and $SINCE^i$ the function that associates an empty word to every state of S_O and every pair of processes of $\mathcal{P}$.
- $\big((q, S1, f1, SINCE1), a, (q', S2, f2, SINCE2)\big) \in \delta_O$ if and only if (q, a, q') is a transition of Aut_K, with $a = (p, PRE, SUC, A)$ and either
  - $S1 = S2 = S_O$ (any transition allowed by Aut_k is allowed when the observation O is completely embedded in the explanation that was built), or
  - $A \cap \mathcal{A}^{ob} = \emptyset$, $S1 = S2 \subset S_O$, and for every s in $S1$, $F2(s) = F1(s) \cup \{p\}$ if $\exists q \in F1(s)$ and $SINCE1(s, q, p) = suc(p, i).w$ for some w and $pre(q, i) \in PRE$. Furthermore, $SINCE2(s, q, p) = w$, and for every $q' \in f2(s)$, if $suc(q', i) \in SUC$ then $SINCE2(s, p, q') = SINCE1(s, p, q').suc(q', i)$. Other components of $SINCE1$ remain unchanged, or

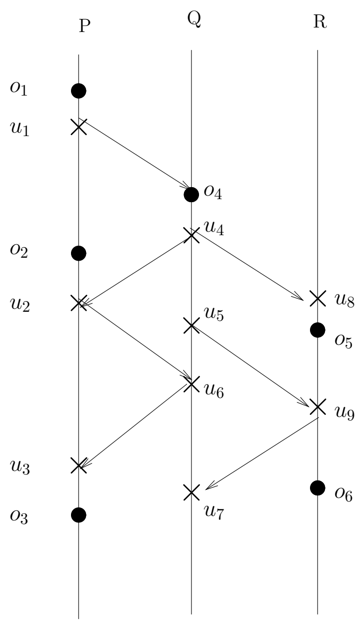

Fig. 5. Construction of SINCE() and F()

- $A \cap \mathcal{A}^{ob} \neq \emptyset$, and there exists s' minimal in the restriction of O to $S_O \setminus S1$ such that $V(s') = A \cap \mathcal{A}^{ob}$, and for every $s \leq_O s'$, we have that either $p \in F(s)$, or there exists a process $q \in \mathcal{P}$ such that $SINCE1(s, q, p) = suc(p, i).w$ and $pre(q, i) \in PRE$. We also set $F2(s') = \{p\}$, and other components of $f1, SINCE1$ are updated as in previous rule. We also set $S2 = S1 \cup \{s'\}$.

Proposition 4. *Let O be an observation over a set of processes $\mathcal{P}$, with valuations in and 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 $\mathcal{O}(|Aut_K|.2^{|O|}.2^{|\mathcal{P}|}.(|O|.|\mathcal{P}|)^2.(|\mathcal{P}|.K+1)^K)$*

proof: Considering that we can encode any word of size at most K in Σ^* as a word of size exactly K using an additional filling symbol, we obtain that function $SINCE()$ can be encoded as a relation of size $|O|.|\mathcal{P}|^2.(|\Gamma_{suc}|+1)^K$ with $|\Gamma_{suc}| = |\mathcal{P}|.K$. The rest is obvious from the description of the construction of Aut_O. $\square$

Construction of $Aut_{\Phi^{spec}}$.
We next give the description of $Alt_{\Phi^{spec}}$, a two-way alternating automaton ([21]) that recognizes linearizations of $K-$influencing computations satisfying Φ^{spec}. This automaton can then be transformed into a standard finite state automaton $Aut_{\Phi^{spec}}$.

We introduce some new atomic formulae in order to simplify the structure of Φ^{spec}. We introduce formulae of the form $\downarrow_m(T)$, where $m \in \mathbb{N}$ and T a computation containing at most 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 equal (i.e. isomorphic) to T. 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 N_m states and the projection of T' onto A is equal 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 an 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 $\neg(\tau, \tau') \wedge \neg(\tau', \tau)$, 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^{spec} 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^{spec} only apply to atomic formulae.

Now we are ready to describe the two-way alternating automaton $Alt_{\Phi^{spec}}$. The basic elements of $Alt_{\Phi^{spec}}$ are similar as in usual translations of temporal logics to alternating automata (see e.g. [29]). The main difficulty is to deal with atomic formulae of the form $\downarrow(\tau, \tau')$, $\uparrow(\tau, \tau')$.

For the sake of clarity, we informally recall some basics of two-way alternating automata ([6, 21]) before presenting their precise definitions. 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 span 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 spanned 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.

Formally, a *two-way alternating automaton* ([6, 21]) is a structure $Alt = (Z, z_{in}, Z_{fi}, Alph, \delta)$, where Z is a finite set of states, $z_{in} \in Z$ the initial state, $Z_{fi} \subseteq Z$ the set of final states, $Alph$ a finite alphabet, and δ is a function from $Z \times (Alph \cup \{\sharp_{lft}, \sharp_{rgt}\})$ to $\mathcal{B}^+(\{lft, rgt\} \times Z)$. Here, the special symbols $\sharp_{lft}$, $\sharp_{rgt}$ are the left and the right marker. The letters lft, rgt are to indicate whether the head should move left or right. The notation $\mathcal{B}^+(X)$ denote the set of positive boolean combination of elements from a finite set X. Namely, $\mathcal{B}^+(X)$ consists of formulas formed X and $\wedge, \vee$, where logically equivalent members of $\mathcal{B}^+(X)$ are identified in the obvious way. Thus, $\mathcal{B}^+(X)$ is also a finite set. We assume tacitly that $\mathcal{B}^+(X)$ also contains the special formulas *true* and *false*. For $\theta \in \mathcal{B}^+(X)$ and $X' \subseteq X$, we say that X' satisfies θ iff θ evaluates to true under the truth assignment f mapping members of X' to *true* and members of $X - X'$ to *false*. Thus, we can associate each formula θ in $\mathcal{B}^+(X)$ by the collection of subsets $X' \subseteq X$ such that X' satisfies θ.

A *run* ρ of Alt over $\sigma = a_1a_2 \dots a_n$ in $\Sigma^\star$ is finite tree whose nodes are labelled with $Z \times \{0, 1, 2, \dots, n, n+1\}$ and the labelling inductively satisfies the following:

- The root of ρ is labelled $(z_{in}, 1)$.
- Suppose a node Nod of ρ is labelled (z, j) with $1 \leq j \leq n$. If $\delta(z, a_j) \neq true$, then there exists some $\{(d_1, z_1), \dots, (d_h, z_h)\}$, $h \geq 1$, satisfying $\delta(z, a_j)$ and Nod has exactly h child nodes whose labellings are $(z_1, j_1), \dots, (z_h, j_h)$. Further, for $i = 1, \dots, h$, if $d_i = lft$, then $j_i = j - 1$; otherwise $(d_i = rgt)$, $j_i = j + 1$.
  If $\delta(z, j) = true$, then there is no need to expand further the node Nod. That is, Nod should become a leaf node (and the path from the root to node Nod is considered accepting).

In the run ρ, a path from the root to a leaf node Nod is *accepting* iff Nod is labelled (z, j) where either $\delta(z, a_j) = true$, or $z \in Z_{fi}$ and $j \in \{0, n+1\}$. The run ρ is *accepting* iff every path from the root to a leaf node is accepting. We say σ is accepted by Alt iff there exists some accepting run over σ.

We are now ready to describe the two-way alternating automaton $Alt_{\Phi^{spec}}$. Let $SF(\Phi^{spec})$ be the set of subformulae of Φ^{spec} and their negations, where $\neg\neg\varphi$ is identified with φ. As indicated earlier, $Alt_{\Phi^{spec}}$ runs over sequences in $\Sigma^\star$. The set of states of $Alt_{\Phi^{spec}}$ is the union of the following pairwise disjoint sets:

- $\{z_{init}\}$, where z_{init} is the *initial* state of $Alt_{\Phi^{spec}}$.
- $Z_1 = \{Min_a \mid a \in \Sigma\}$. These states are auxiliary states needed to search for a position (corresponding to a minimal state) at which Φ^{spec} holds.
- $Z_2 = \{Hold_\varphi \mid \varphi \in SF(\Phi^{spec})\}$, where intuitively z_φ is the state which expects to verify that φ holds at the current head position.
- $Z_3 = \bigcup AUX_\varphi$, where φ ranges over formulas in $SF(\Phi^{spec})$. For each such φ, the states in AUX_φ are auxiliary states needed *during the process* of verifying that φ holds at some position.

In what follows, we describe the transition function δ of $Alt_{\Phi^{spec}}$ and give the details of the sets in Z_3 as we go along. It will turn out that $Alt_{\Phi^{spec}}$ needs no final states. That is, for a sequence $\sigma = a_1a_2 \dots a_n$ in $\Sigma^\star$, if ρ

is an accepting run over σ, then every leaf node is labelled (z, j) with $\delta(z, a_j) = true$.
To clarify the intuition behind the transitions of $Alt_{\Phi^{spec}}$, we shall fix a sequence $\sigma = a_1 a_2 \ldots a_n \in \Sigma^\star$ and often informally describe the operation of $Alt_{\Phi^{spec}}$ before stating the precise transitions of $Alt_{\Phi^{spec}}$.
Firstly, note that $poc(\sigma) \models \Phi^{spec}$ iff $\sigma, h \models \Phi^{spec}$ where $a_1 a_2 \ldots a_h$ is a minimal state in $poc(\sigma)$. Thus, at the initial state, $Alt_{\Phi^{spec}}$ 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^{spec} holds at h. Intuitively, we designate a state z_a, $a \in \Sigma$, to indicate the search for a position h with $a_h = a$ and $a_1 a_2 \ldots a_h$ being a minimal state. Thus, for $a \in \Sigma$, we have, $\delta(z_{init}, a) = \vee_{b \in \Sigma} \delta(Min_b, a)$. And for $a, b \in \Sigma$, we have

$$\delta(Min_b, a) = \begin{cases} \delta(Hold_{\Phi^{spec}}, a) & \text{if } a = b \\ (rgt, Min_b) & \text{if } a \; I \; b \\ false & \text{if } a \; D \; b \text{ and } a \neq b \; . \end{cases}$$

Next, we explain how $Alt_{\Phi^{spec}}$ verifies that a formula in $SF(\Phi^{spec})$ holds at the current head position. We proceed inductively to describe how $Alt_{\Phi^{spec}}$ verifies a formula of forms loc_p, $\downarrow(\tau, \tau')$, $\uparrow(\tau, \tau')$, $\mathsf{EX}\varphi$, $\mathsf{EU}(\varphi, \varphi')$. We then study negation, conjunction and disjunction of formulas.

—**Formulas of form** loc_p
Clearly $Alt_{\Phi^{spec}}$ can easily check if loc_p holds at the current head position, simply from the letter at the head position. Thus, we have $\delta(loc_p, a) = true$ if $a = (\hat{p}, PRE, SUC, A)$ with $\hat{p} = p$; and $\delta(loc_p, a) = false$ if otherwise.

—**Formulas of forms** $\downarrow(\tau, \tau'), \uparrow(\tau, \tau')$
For the input sequence σ, we let $a_i = (p_i, PRE_i, SUC_i, A_i)$ for each $i = 1, 2, \ldots, n$. 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 $s \lll_\sigma s'$ iff s is a predecessor of s' in $poc(\sigma)$. Consider $g, h \in \{1, 2, \ldots, n\}$, it is easy to see that $a_1 \ldots a_g \lll_\sigma a_1 \ldots a_h$ 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 $a_g \; D \; a_{i_1} \; D \; a_{i_2} \ldots a_{i_t} \; D \; a_h$.

With this, we explain how $Alt_{\Phi^{spec}}$ can verify that $a_1 \ldots a_g \lll_\sigma a_1 \ldots a_h$. This utility will used subsequently to verify formulas $\downarrow(\tau, \tau')$, $\uparrow(\tau, \tau')$. If $a_g = a_h$ and $p = p_g = p_h$, then $Alt_{\Phi^{spec}}$ just need to ensure that between positions g and h (excluding g, h), there are no letters $(\hat{p}, PRE, SUC, A)$ with $\hat{p} = p$. Now suppose $a_g \neq a_h$. Let Seq be the collection of sequences $b_1 b_2 \ldots b_t$ over Σ with $t \leq |\Sigma| - 2$, $a_g \; D \; b_1 \; D \; b_2 \ldots \; D \; b_t \; a_h$. Then $Alt_{\Phi^{spec}}$ checks that $a_g \; D \; a_h$ and furthermore for every sequence $b_1 b_2 \ldots b_t$ in Seq, there do *not* exist positions $i_1, i_2, \ldots, i_t$ with $g < i_1 < i_2 < \ldots < i_t < h$ and $a_{i_1} = b_1$, $a_{i_2} = b_2$, $\ldots$, $a_{i_t} = b_t$.
For convenience, we describe how $Alt_{\Phi^{spec}}$ verifies a formula $\uparrow(\tau, \tau')$. Formulas $\downarrow(\tau, \tau')$ can be similarly handled. Fix a formula $\uparrow(\tau, \tau')$. Let $\tau = b_1 b_2 \ldots b_m$, $\tau' = b'_1 b'_2 \ldots b'_{m'}$. We note that for a position ℓ, $\sigma, \ell \models \uparrow(\tau, \tau')$ iff there exist indices $\ell_1, \ldots, \ell_m$, $\ell'_1, \ldots, \ell'_{m'}$, in $\{\ell+1, \ell+2, \ldots, n\}$ satisfying the following conditions:

Condition (i) $a_1a_2\dots a_\ell \ll_\sigma \rho_1 \ll_\sigma \rho_2 \ll_\sigma \dots \ll_\sigma \rho_m$, where $\rho_i = a_1a_2\dots a_{\ell_i}$ for $i = 1, 2, \dots, m$. And $a_{\ell_i} = b_i$ for $i = 1, 2, \dots, m$. This asserts that the τ-descendant of $a_1a_2\dots a_l$ exists.

Condition (ii) $a_1a_2\dots a_\ell \ll_\sigma \rho'_1 \ll_\sigma \rho'_2 \ll_\sigma \dots \ll_\sigma \rho'_{m'}$, where $\rho'_i = a_1a_2\dots a_{\ell'_i}$ for $i = 1, 2, \dots, m'$. And $a_{\ell'_i} = b'_i$ for $i = 1, 2, \dots, m'$. This asserts that the τ'-descendant of $a_1a_2\dots a_l$ exists.

Condition (iii) $a_1a_2\dots a_{\ell_m} \leq_\sigma a_1a_2\dots a_{\ell'_{m'}}$, that is, $a_1a_2\dots a_{\ell_m} \sqsubseteq_{\Sigma^\star} a_1a_2\dots a_{\ell'_{m'}}$. This asserts that the τ-descendant of $a_1a_2\dots a_l$ causally precedes the τ'-descendant of $a_1a_2\dots a_l$.

Thus, to verify that a formula $\uparrow(\tau,\tau')$ holds at position ℓ, we move towards the right and verifying the existence of indices ℓ_1, ℓ_2, …, ℓ_m, ℓ'_1, ℓ'_2, …, $\ell'_{m'}$ satisfying the above conditions. We stop at the position $\ell'_{m'}$ if these indices are successfully verified, or when some position violating the above conditions are found, or when we hit the right end marker before verifying all the indices. In the first case, $\Uparrow(\tau,\tau')$ indeed holds at position ℓ and in the latter two cases, the formula $\Uparrow(\tau,\tau')$ does not hold at position ℓ.

We describe subsequently the auxiliary states in $AUX_{\Uparrow(\tau,\tau')}$ that aid in the process of verifying $\Uparrow(\tau,\tau')$ holds at position ℓ.

We note that the indices ℓ_1, ℓ_2, …, ℓ_m are *uniquely* determined by a_ℓ and (the letters of) τ. In particular, $a_{\ell_1} = b_1$, $a_{\ell_2} = b_2$, …, $a_{\ell_m} = b_m$. Similarly, the indices ℓ'_1, ℓ'_2, …, $\ell'_{m'}$ are *uniquely* determined by a_ℓ and (the letters of) τ'. In particular, $a_{\ell_1} = b_1$, $a_{\ell_2} = b_2$, …, $a_{\ell_m} = b_m$. We recall from above the description of how $Alt_{\Phi^{spec}}$ can verify $a_1a_2\dots a_g \ll_\sigma a_1a_2\dots a_h$ for two positions g, h. Let *Seq* be the collection of sequences over Σ of the form $a_{\ell_1}\xi_1 a_{\ell_2}\xi_2\dots\xi_{m-1}a_{\ell_m}\lambda a_{\ell'_{m'}}$, where ξ_1,ξ_2,…, ξ_{m-1},λ are sequences over Σ of length at most $|\Sigma|-2$. Each ξ_t sequence will aid in verifying that $a_1a_2\dots a_{\ell_t} \ll_\sigma a_1a_2\dots a_{\ell_{t+1}}$ (cf. Condition (i) above). The sequence λ will help verify Condition (iii) above. Let *Seq'* be the collection of sequences over Σ of the form $a_{\ell'_1}\xi'_1 a_{\ell'_2}\xi'_2\dots\xi'_{m'-1}a_{\ell'_{m'}}$ where each ξ'_1, ξ'_2, …, $\xi'_{m'-1}$ are sequences over Σ of length at most $|\Sigma|-2$. Sequences in *Seq'* will be used to help verifying Condition (ii) above.

We have that $AUX_{\Uparrow(\tau,\tau')} = AUX^1_{\Uparrow(\tau,\tau')} \cup AUX^2_{\Uparrow(\tau,\tau')}$, where a state in $AUX^1_{\Uparrow(\tau,\tau')}$ consists of two sequences over Σ, one from *Seq* and one from *Seq'*. Using states in $AUX^1_{\Uparrow(\tau,\tau')}$, we can then verify $\Uparrow(\tau,\tau')$ by checking Conditions (i)(ii)(iii) above. Further, the states in $AUX^2_{\Uparrow(\tau,\tau')}$ are auxiliary states needed to implement the transitions associated with states in $AUX^1_{\Uparrow(\tau,\tau')}$. It is now easy, though tedious, to write the exact transitions of $Alt_{\Phi^{spec}}$ associated with state $Hold_{\Uparrow(\tau,\tau')}$.

—**Formulas of forms $\mathsf{EX}\varphi$, $\mathsf{EU}(\varphi,\varphi')$**

Here we consider formuls of forms $\mathsf{EX}\varphi$, $\mathsf{EU}(\varphi,\varphi')$. Note that $\sigma, i \models \mathsf{EX}\varphi$ iff there exists an index j in $\{i+1,\dots,n\}$ such that $a_1a_2\dots a_i \ll_\sigma a_1a_2\dots a_j$ and $\sigma, j \models \varphi$. Thus, to verify that $\mathsf{EX}\varphi$ holds at position i, $Alt_{\Phi^{spec}}$ moves to the right searching for such a j. Similarly, we note that $\sigma, i \models \mathsf{EU}(\varphi,\varphi')$ iff one of the following conditions holds:

- $\sigma, i \models \varphi'$.
- $\sigma, i \models \varphi$. And there exists an index j satisfying $a_1a_2\dots a_i \ll_\sigma a_1a_2\dots a_j$ and $\sigma, j \models \mathsf{EU}(\varphi,\varphi')$.

With this, it is clear how $Alt_{\Phi^{spec}}$ can verify the formula $\mathsf{EU}(\varphi, \varphi')$ holds at the current head position. Again, it is straightforward, though tedious, to provide the exact transitions of $Alt_{\Phi^{spec}}$ associated with states $Hold_{\mathsf{EX}\varphi}$, $Hold_{\mathsf{EU}(\varphi,\varphi')}$.

—**Negation, conjunction, disjunction of formulas**
Finally, negation, conjunction and disjunction of formulas are handled as in usual translations of temporal logics to alternating automata (see e.g. [29]). Namely, for $a \in \Sigma$, $\varphi, \varphi' \in SF(\Phi^{spec})$, we have $\delta(Hold_{\neg\varphi}, a) = \neg\delta(Hold_{\varphi}, a)$, $\delta(Hold_{\varphi\wedge\varphi'}, a) = \delta(\varphi, a) \wedge \delta(\varphi', a)$ and $\delta(Hold_{\varphi\vee\varphi'}, a) = \delta(\varphi, a) \vee \delta(\varphi', a)$.
We have now completed the description of $Alt_{\Phi^{spec}}$. It is not difficult to see that the number of states of $Alt_{\Phi^{spec}}$ is of complexity $O(2^{|\Phi^{spec}|} \cdot |\Sigma|^{|\Sigma|\cdot m})$, 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 [21] that $Alt_{\Phi^{spec}}$ can be transformed to a finite state automaton $Aut_{\Phi^{spec}}$ with $2^{N \cdot 2^N}$ states where N is the number of states of $Alt_{\Phi^{spec}}$. The proof of Theorem 2 is now completed. $\Box$
To prove Corollary 1, 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 $W_\sigma \models \Phi^{spec}$ 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 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 $\{\Phi_p^{spec}\}_{p\in\mathcal{P}}$, by testing for each state s of O, each a in $\mathcal{A}^{ex}$, the non-emptiness of the product of $Aut_{\Phi^{spec}}$ and $Aut_{s,a}$. $\Box$

Theorem 3. *Let O be an observation of a system composed of a set $\mathcal{P}$ of processes, $\mathcal{A} = \mathcal{A}^{ob} \cup \mathcal{A}^{ex}$ be the set of observable and explanatory atomic propositions, $\phi = \{\Phi_p^{spec}\}_{p\in\mathcal{P}}$ be a $\mathcal{P}$-indexed family of LPOC formulae such that all frontiers and views used in ϕ are at most m-frontiers or m-views. Let K be an integer, and Σ be the Mazurkiewicz trace alphabet computed from $\mathcal{A}$ and K. Then one can determine whether there exists a K-influencing explanation W for O with complexity $\mathcal{O}(W_1.2^{W_2.2^{W_2}})$, where:*
- $W_1 = |\Sigma|^{K.|\mathcal{P}|^2}.2^{|O|}.2^{|\mathcal{P}|}.(|O|.|\mathcal{P}|)^2.(|\mathcal{P}|.K+1)^K$, *and*
- $W_2 = 2^{|\phi|.Nm^2 \sum_{i\in 1..Nm} 2^{\frac{i^2}{4}\cdot\frac{3i}{2}.ln(i)}.i.2^{\mathcal{A}^{ex}}}.|\Sigma|^{|\Sigma|\cdot m}$.
- $Nm = \frac{1-|\mathcal{P}|^{m+1}}{1-|\mathcal{P}|}$

proof: First every sub-formula of the form $\uparrow_{m,A}(T)$ must be translated into a disjunction of formulae of the form $\uparrow_m(T)$. Then every sub-formula of the form $\uparrow_m(T)$ is translated into conjunction of formulae of the form $\uparrow(\sigma, \sigma')$. Then a 2-way alternating automaton over a set of formulae of the form $\uparrow(\sigma, \sigma')$, $EX\phi$, $AX\phi$, ... is built and transformed into an equivalent (but exponentially larger) standard automaton.

The first step means computing all partial orders of size up to Nm. Recall that according to Kleitman & Rotschild [19], the number of partial orders of size n is in $O(2^{\frac{n^2}{4}\cdot\frac{3n}{2}.ln(n)})$. Hence, a formula of the form $\uparrow_{m,A}(T)$ can be translated as a disjunction of up to $\underset{i\in 1..Nm}{\Sigma} 2^{\frac{i^2}{4}\cdot\frac{3i}{2}.ln(i)}.i.2^{\mathcal{A}^{ex}}$ formulae of the form $\uparrow_m (T_{ij})$, where each T_{ij} is a template of size lower than Nm that allows an embedding of T into T_{ij}.
Then, each template of the form $\uparrow_m (T_{ij})$, can be defined by a conjunction of up to Nm^2 formulae of the form $\uparrow(\sigma,\sigma')$. So, ϕ is equivalent to a formula ϕ'' that is formed from $\neg,\wedge,\vee$ and the atomic formulae loc_p, $\downarrow(\sigma,\sigma')$, $\uparrow(\sigma,\sigma')$, $\mathsf{EX}\varphi$, $\mathsf{EU}(\varphi,\varphi')$, and of size at most $|\phi|.Nm^2 \underset{i\in 1..Nm}{\Sigma} 2^{\frac{i^2}{4}\cdot\frac{3i}{2}.ln(i)}.i.2^{\mathcal{A}^{ex}}$. So, the size of the 2-way alternating automaton Alt_ϕ computed for the formula ϕ is in $W_2 = 2^{|\phi|.Nm^2 \underset{i\in 1..Nm}{\Sigma} 2^{\frac{i^2}{4}\cdot\frac{3i}{2}.ln(i)}.i.2^{\mathcal{A}^{ex}}}.|\Sigma|^{|\Sigma|\cdot m}$ Then, from [21], a two-way alternating automaton with N states can be simulated by a deterministic (one-way) automaton with $2^{N\cdot 2^N}$ states.
The size W_1 of Aut_O has been computed in proposition 4. Once the automaton Aut_ϕ that recognizes $K-$influencing computations satisfying ϕ has been computed, then the diagnosis problem can be reduced to the vacuity of intersection of Aut_O and Aut_ϕ. The diagnosis problem can hence be solved in $O(W_1.2^{W_2.2^{W_2}})$.
$\square$

Corollary 2. *Let O be an observation, and ϕ be a formula describing a distributed system. Computing a summary for O can be done in* $\mathcal{O}(|O|.|\mathcal{A}^{ex}|.W_1.2^{W_2.2^{W_2}})$

proof: Computing a summary is equivalent to labeling each state s of O with each single unobserved proposition p of $\mathcal{A}^{ex}$ in addition to the observed propositions, and running diagnosis. If the obtained product computed is empty, then there is no computation is the set of explanations provided by ϕ such that s is labelled by p, and p should not appear in the summary labeling of s. This operation is repeated for every $p \in \mathcal{A}^{ex}$ and every $s \in O$. $\square$

5 Related work

Several logical formalisms have been proposed for traces or partial orders. Among them, we can distinguish *local logics*, where the truth of formulae are evaluated at local states, and *global logics*, where properties are evaluated at *configurations* (i.e collections of local states of processes) of the models. For global logics, [28] has shown that LTrL, an LTL-like logic over traces with a past operator is equal in expressive power to the first order theory of traces. This result is generalized for pure future LTL on traces by [8, 10]. TrPTL, proposed by [27] is a logic with a next and a special until operator restricted to events of a process. Its expressive power is included in FO logic [13], and satisfiability of TrPTL formulae is decidable [27]. Note that for traces, satisfiability and model-checking of LTL-like logics with "universal until" is non elementary [30] (see [5] for an automaton-based proof of this result) and are undecidable with "existential until" [25]. In local logics, formulae are interpreted over local states of a partial order. Without being exhaustive, one can cite several results: TLC, proposed by [1] (a local logic over partial orders with local next, existential until, and similar past operators, plus a parallel operator) has the same expressive power as FO [9], and local logics with strict universal until and universal until have the same expressive power as FO on traces [11]. Model-checking and satisfiability of MSO-definable local logics over traces is in PSPACE [12]. Moreover, for traces every MSO-definable local logics is complete for FO, and uniform satisfiability for MSO-definable local logics is in PSPACE [13].

The LPOC logic proposed in this paper is a local logic over partial orders. Clearly, it is not FO definable, as the existential until can not be defined with FO. However, it can be easily translated to MSO formulae. To keep decidability of the diagnosis problem, some restrictions are imposed on the models. It is frequent with partial order models, as defining a model checking as the intersection of two automata, one representing a formula, and the other one representing the model often allows for an encoding of the PCP.

In [24], D.Peled proposes an algorithm to model check formulae in TLC$^-$ (a subset of TLC) on High-level Message Sequence charts (HMSCs). Model checking HMSCs with global logics such as LTL or CTL is reputed undecidable, except for restricted classes of the language [2]. TLC$^-$ formulae describe the shape of causal chains in all the partial orders generated by a HMSC, i.e. they express properties on sequences of events that are ordered by $<$, the covering relation of a partial order. The model-checking algorithm for TLC$^-$ transforms a formula ϕ and a HMSC H into Buchi automata and verify the emptiness of the intersection of both automata. If the intersection of both models is not empty, then $H \models \phi$. More precisely, the Buchi automaton computed from H selects one representant linearization for each MSC generated by H, and the automaton computed from ϕ describes linearization that satisfy $\neg\phi$. Due to communications in MSCs, the automata computed comport two transition relations. Emptiness of the intersection of automata with two transition relations is in general undecidable, but for the specific case of HMSCs, the second transition relation, that represents messages, is contained in the transitive closure of the first one, and emptiness becomes decidable. As TLC^- only contains next and until temporal operators it is clearly less expressive than LPOC. it .

Bollig proposes a propositional dynamic logic (PDL) for message passing systems see [4], extending the Henriksen and Thiagarajan's dynamic LTL for traces [18], and show that model checking HMSCs wit PDL properties is PSPACE complete.

[22, 23] proposes several temporal logics over Lamport diagrams, with future and past modalities, and show that in the general case, these logics are undecidable. However, these logics become decidable when considering models of bounded size, and for communication closed layered b-bounded diagrams, that is partial orders that can be organized as suc-

cessive layers of finite message exchanges among fixed sets of processes. 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 allow to describe a complete partial order in a bounded future or past of a local state. The LPOC logic is then slightly more discriminating than for instance the LD0 logic proposed by [23], as shown in Figure 6. It is possible to say that a computation starts with template T_1 with the LPOC formula $\phi_P =\downarrow_{3,\{a,b,c\}} (T_1)$, but LD0 formulae describing such computations also allow computations starting with T_2.

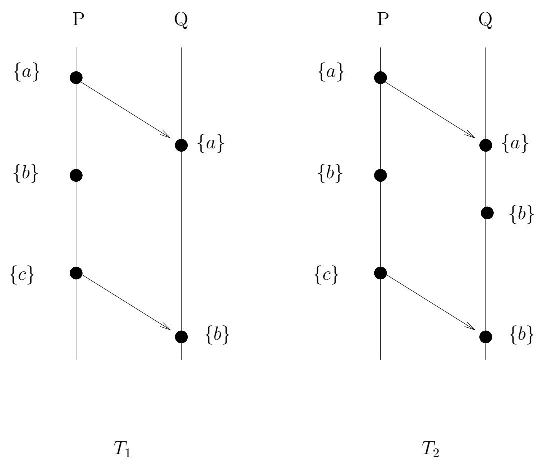

Fig. 6. LPOC is more discriminating than LD0

Note however that for all the local logics described above, 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" [16]. A Message Sequence Chart is a partial order where locality of events and messages are explicitly represented. A template MSC is an incomplete MSC, i.e a MSC with some "holes" and that may also comport 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 operator of the form $Ma \longrightarrow Mb$,and $Ma \longrightarrow \neg Mb$ where Ma and Mb are template MSCs, and conjunctions of templates of the form $\wedge_i(Ma^i \longrightarrow (\vee_j \pm M^{ij}))$. 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 Lamport diagrams. Hence, in LPOC, causal precedence is not restricted to local ordering on processes and message relation, and a state might have up to $|\mathcal{P}|$ predecessors, even in K-influenced computations. Even if we only consider LPOC formulae over MSCs, LPOC and template MSC-s 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, then 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 do not find their equivalent in template MSC. Even if we impose models of LPOC formulae to be MSCs, there is no way to encode the until operator of LPOC with template MSCs. Both logics are hence uncomparable.
Note also that the works in [24] and [16] 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 linearization of every MSC considered is bounded buy some integer b) that is needed by [2] to model check High level Message Sequence Charts. It might be interesting to see whether the restriction of [22, 23] that imposes layered computations is sufficient to make diagnosis with LPOC formulae a decidable problem.

6 Conclusion

We have proposed a diagnosis framework based on partial order logic. Like most of powerful logics such as Template MSCs, satisfiability of formulae and hence diagnosis turns to be undecidable problems without restrictions on the kind of models considered. By imposing computations to be K-influenced, diagnosis finds a solution. Even within this setting, diagnosis turns out to be very expensive (exponential in the size of the formula and in the size of the observed behavior). Some complexity gains can be expected by restricting the size and the number of partial order templates considered, but also the modalities of the formulae.

References

1. Rajeev Alur, Doron Peled, and Wojciech Penczek. Model-Checking of Causality Properties. In *LICS*, pages 90–100, 1995.
2. Rajeev Alur and Mihalis Yannakakis. Model Checking of Message Sequence Charts. In *CONCUR*, pages 114–129, 1999.
3. A. Benveniste, E. Fabre, C. Jard, and S. Haar. Diagnosis of asynchronous discrete event systems, a net unfolding approach. *IEEE Transactions on Automatic Control*, 48(5):714–727, May 2003.

4. Benedikt Bollig, Dietrich Kuske, and Ingmar Meinecke. Propositional Dynamic Logic for Message-Passing Systems. In *FSTTCS*, pages 303–315, 2007.
5. Benedikt Bollig and Martin Leucker. Deciding LTL over Mazurkiewicz Traces. In *TIME*, pages 189–197, 2001.
6. Janusz A. Brzozowski and Ernst L. Leiss. On Equations for Regular Languages, Finite Automata, and Sequential Networks. *Theor. Comput. Sci.*, 10:19–35, 1980.
7. V. Diekert and G. Rozenberg, editors. *Book of Traces.* World Scientific, Singapore, 1995.
8. Volker Diekert and Paul Gastin. LTL Is Expressively Complete for Mazurkiewicz Traces. In *ICALP*, pages 211–222, 2000.
9. Volker Diekert and Paul Gastin. Local Temporal Logic is Expressively Complete for Cograph Dependence Alphabets. In *LPAR*, pages 55–69, 2001.
10. Volker Diekert and Paul Gastin. LTL Is Expressively Complete for Mazurkiewicz Traces. *J. Comput. Syst. Sci.*, 64(2):396–418, 2002.
11. Volker Diekert and Paul Gastin. Pure Future Local Temporal Logics Are Expressively Complete for Mazurkiewicz Traces. In *LATIN*, pages 232–241, 2004.
12. Paul Gastin and Dietrich Kuske. Satisfiability and Model Checking for MSO-definable Temporal Logics are in PSPACE. In *CONCUR*, pages 218–232, 2003.
13. Paul Gastin and Dietrich Kuske. Uniform satisfiability in PSPACE for local temporal logics over Mazurkiewicz traces. *Fundamenta Informaticae*, 80(1-3):169–197, November 2007.
14. Paul Gastin and Madhavan Mukund. An Elementary Expressively Complete Temporal Logic for Mazurkiewicz Traces. In *ICALP*, pages 938–949, 2002.
15. Thomas Gazagnaire and Loïc Hélouët. Event Correlation with Boxed Pomsets. In *FORTE*, pages 160–176, 2007.
16. Blaise Genest, Marius Minea, Anca Muscholl, and Doron Peled. Specifying and Verifying Partial Order Properties Using Template MSCs. In *FoSSaCS*, pages 195–210, 2004.
17. Loïc Hélouët, Thomas Gazagnaire, and Blaise Genest. Diagnosis from Scenarios. In *WODES*, 2006.
18. Jesper G. Henriksen and P. S. Thiagarajan. Dynamic Linear Time Temporal Logic. *Ann. Pure Appl. Logic*, 96(1-3):187–207, 1999.
19. D.J Kleitman and B.L Rotschild. Asymptotic enumeration of partial orders. *Transactions of the American Mathematical Society*, (205):205–220, 1975.
20. D. Kuske. Regular sets of infinite message sequence charts. *Information and Computation*, 187(1):80–109, 2003.
21. Richard E. Ladner, Richard J. Lipton, and Larry J. Stockmeyer. Alternating Pushdown and Stack Automata. *SIAM J. Comput.*, 13(1):135–155, 1984.
22. B. Meenakshi and Ramaswamy Ramanujam. Reasoning about Message Passing in Finite State Environments. In *ICALP*, pages 487–498, 2000.
23. B. Meenakshi and Ramaswamy Ramanujam. Reasoning about Layered Message Passing Systems. In *VMCAI*, pages 268–282, 2003.

24. Doron Peled. Specification and Verification of Message Sequence Charts. In *FORTE*, pages 139–154, 2000.
25. Wojciech Penczek. On Undecidability of Propositional Temporal Logics on Trace Systems. *Inf. Process. Lett.*, 43(3):147–153, 1992.
26. 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):105–124, 1996.
27. P. S. Thiagarajan. A Trace Based Extension of Linear Time Temporal Logic. In *LICS*, pages 438–447, 1994.
28. P. S. Thiagarajan and Igor Walukiewicz. An Expressively Complete Linear Time Temporal Logic for Mazurkiewicz Traces. In *LICS*, pages 183–194, 1997.
29. Moshe Y. Vardi. An Automata-Theoretic Approach to Linear Temporal Logic. In *Banff Higher Order Workshop*, pages 238–266, 1995.
30. Igor Walukiewicz. Difficult Configurations - On the Complexity of LTrL. In *ICALP*, pages 140–151, 1998.