Kamp Theorem for Pomset Languages of Higher Dimensional Automata

Emily Clement
CNRS, LIPN UMR 7030, Université Sorbonne Paris Nord, F-93430 Villetaneuse, France

Enzo Erlich
Université Paris Cité, CNRS, IRIF, F-75013, Paris, France
EPITA Research Laboratory (LRE), Paris, France

Jérémy Ledent
Université Paris Cité, CNRS, IRIF, F-75013, Paris, France

Abstract

Temporal logics are a powerful tool to specify properties of computational systems. For concurrent programs, Higher Dimensional Automata (HDA) are a very expressive model of non-interleaving concurrency. HDA recognize languages of partially ordered multisets, or pomsets. Recent work has shown that Monadic Second Order (MSO) logic is as expressive as HDA for pomset languages. In the case of words, Kamp's theorem states that First Order (FO) logic is as expressive as Linear Temporal Logic (LTL). In this paper, we extend this result to pomsets. To do so, we first investigate the class of pomset languages that are definable in FO. As expected, this is a strict subclass of MSO-definable languages. Then, we define a Linear Temporal Logic for pomsets ($\mathrm{LTL}_{\mathrm{Poms}}$), and show that it is equivalent to FO.

2012 ACM Subject Classification Theory of computation → Modal and temporal logics; Theory of computation → Concurrency

Keywords and phrases Higher dimensional automata, temporal logic, Kamp's theorem

Digital Object Identifier 10.4230/LIPIcs.CSL.2026.43

Related Version *Full Version*: `https://arxiv.org/abs/2410.12493` [4]

Funding This work was partly supported by ANR projects BisoUS (ANR-22-CE48-0012) and PaVeDyS (ANR-23-CE48-0005).

1 Introduction

Context. Higher-Dimensional Automata (HDA) [20] are a powerful model of concurrency that enriches standard finite-state automata with higher-dimensional cells, allowing to specify that some events may occur simultaneously. HDA represent asynchronous concurrent computations where events are non-atomic, and can overlap in various complex patterns. Thus, they are a model of *non-interleaving* concurrency, which is concerned with simultaneity between events rather than commutation of events. HDA have been shown to be a very expressive model of concurrency: comparisons with other models (such as Petri nets, event structures) can be found in [26], based on history-preserving bisimulations, or in [14], based on adjunctions.

A notion of language of HDA was developed only recently [10]. The key idea is that an execution of an HDA may not be represented as a word (where the events/letters are totally ordered); instead, it is a *partially ordered multiset*, or *pomset* for short. This is illustrated in Figure 1. On the left, an HDA is depicted, with two processes running in parallel. One process is executing two events a, then c, while the second process is running b, then d. One possible execution path is depicted in red. This execution can equivalently be depicted using intervals (center of Figure 1): time flows from left to right, and the duration

© Emily Clement, Enzo Erlich, and Jérémy Ledent;
licensed under Creative Commons License CC-BY 4.0

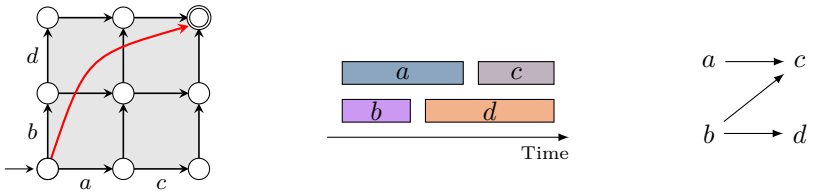


Figure 1 An HDA execution, depicted with intervals, and as a pomset.

of events is depicted by stretching horizontally the boxes representing the events. When two events overlap vertically, this means that they are occurring simultaneously. Finally, a third representation of this execution is depicted on the right of Figure 1, as a pomset. The events are ordered according to Lamport's *happens before* relation [17]: $a \to c$ means that event a terminates before c starts. In particular, overlapping events are not ordered.

Thus, HDA recognize languages of pomsets. Note that, just as sequential executions are a special case of concurrent executions; words are also a special case of pomsets, where the order between events happens to be total. A recent line of work has been extending classic results of automata theory to the setting of pomset languages of HDA. They have been shown to enjoy a variant of Kleene theorem [11], a Myhill-Nerode theorem [12], a pumping Lemma [3], and a Büchi-Elgot-Trakhtenbrot theorem [2]. The latter result will be of particular importance for us: it says that Monadic Second-Order (MSO) logic describes the same class of languages as HDA. To establish this result, the authors used an equivalent representation of pomsets called *ST-sequences* (see Section 2.2), which are words over a finite alphabet. This proof technique, further developed in [1], provides a way to lift some results from words to pomsets.

In this paper, we investigate the class of languages obtained by restricting to First Order (FO) logic instead of MSO. Over finite words, FO-definable languages form a strict subclass of regular languages, called *aperiodic languages* [23]. There are many equivalent characterizations of this class, such as star-free regular expressions and counter-free automata [19]. Of particular interest to us is Kamp's theorem [16, 22], which states that Linear Temporal Logic (LTL) is as expressive as FO logic over finite words.

Contributions: FO and LTL for pomset languages. This paper studies the class of FO-definable languages of pomsets. As expected, they form a strict subclass of HDA languages (Corollary 11). Our main goal is to define an LTL-style logic over pomsets, that defines the same class of languages as FO logic; thus extending Kamp's theorem to pomset languages. To that end, we first show that FO formulas over pomsets can be translated into equivalent FO formulas over ST-sequences (Theorem 10). A similar result for MSO formulas was proved in [2]. However, it cannot be easily adapted: it relies heavily on a relation $\sim$, that relates different occurrences of the same event in an ST-sequence. To show that this relation is definable in MSO, the authors use second-order quantification. In our first technical contribution (Lemma 9), we show that the relation $\sim$ is FO-definable. Instead of giving directly an FO formula (which would be very tedious), we define a counter-free automaton.

We then define a variant of LTL for pomsets that we call $\mathrm{LTL}_{\mathrm{Poms}}$ and prove that $\mathrm{LTL}_{\mathrm{Poms}}$ is equivalent to FO (Theorem 22). We sum up the proof of Theorem 22 by the chain of translations depicted in Figure 2.

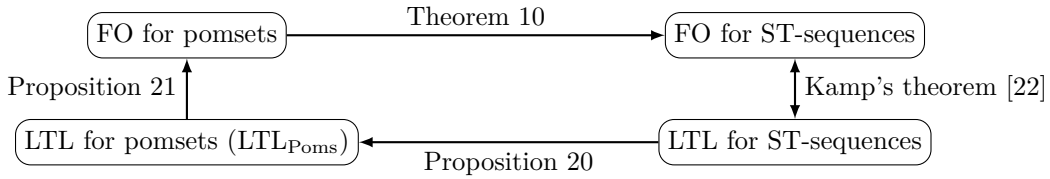


Figure 2 Summary of the proof of Theorem 22.

Relationship with Mazurkiewicz traces. Pomsets also appear in the literature in the context of Mazurkiewicz traces [18], a notion also at the intersection between automata theory and concurrency theory. Many variants of LTL on traces have been defined, and their expressivity has been extensively studied [24, 5, 25, 7, 6, 13]. Thus, it is important to distinguish this line of work with what we are studying in this paper.

The set of Mazurkiewicz traces can be described as the free partially commutative monoid over a dependence alphabet. Thus, a trace is an equivalence class of finite words where some letters are allowed to commute. Equivalently, they can also be regarded as pomsets, using their *dependence graph*. In the dependence graph of a Mazurkiewicz trace, $a \to b$ denotes causal dependency. Two incomparable events are independent, i.e., it does not matter in which order they occur. In contrast, for HDA pomsets, the precedence order $a \to b$ indicates that a terminates before b starts, and two events are incomparable when they are simultaneous (i.e., the time intervals during which they are executed overlap). This semantic distinction results in several technical differences:

- **(i)** HDA pomsets are always required to be *interval orders* (see Definition 1). This is not the case for Mazurkiewicz traces.
- **(ii)** For HDA, it may very well be the case that $a \to b$ (i.e., a happens before b) in one execution, while a and b are incomparable (i.e., simultaneous) in another execution. This is not allowed for Mazurkiewicz traces. Indeed, they are defined with respect to a dependence alphabet, which determines in advance which events may or may not be comparable in the dependence graph.
- **(iii)** Lastly, HDA pomsets actually have an extra relation $\dashrightarrow$ called the *event order*. Indeed, we require simultaneous events (events that are incomparable for the relation $\to$) to be ordered by the event order. This is a way of managing process identity: if two processes are running a in parallel, the event order allows to distinguish the two events a.

Other related work. Modal logics over HDA have been previously investigated. In [21], the author introduces a logic called Higher-Dimensional Modal Logic (HDML), which is interpreted directly on an HDA. It has two variants of the "next" operator: one that can start an event, and one that can terminate an event. A sound and complete axiomatization of this logic is given, as well as tentative definitions of LTL-like and CTL-like extensions. The language theory over pomsets is not investigated since pomset languages of HDA had not been defined at the time. However, an encoding of LTL for Mazurkiewicz traces into HDML is given. A more recent paper [27] introduces IPomset Modal Logic (IPML). This logic features a forward modality and a backwards modality, in the spirit of path logic [15]. No temporal variant of IPML is defined, the focus of this paper being on bisimulation equivalence.

Plan of the paper. In Section 2, we recall the key definitions that we will be using. Then, in Section 3, we investigate FO over pomsets, and show our first technical contribution: FO over pomsets can be translated to FO over ST-sequences (Theorem 10). In Section 4, we define our temporal logic LTL$_{\text{Poms}}$, and study its expressivity in Section 5 (Theorem 22).

2 Preliminaries

In this section, we define several notions that we will need in this paper: interval pomsets with interfaces [10], their ST-decomposition [1], and MSO logic on pomsets [2].

2.1 Interval pomsets with interfaces

Pomsets, interfaces, dimension. Let Σ be a finite alphabet. A partially ordered multiset (or pomset) over Σ is a generalization of words, where the letters need not be totally ordered. This can represent situations where two or more events occur at the same time. An intuitive representation of some pomsets is given in Figure 3, where the events $a, b, c, d \in \Sigma$ are depicted as intervals, and the horizontal axis represents the elapsing time. For instance, in Figure 3a, both events a and b occur before c, which we denote by $a < c$ and $b < c$. However, since the intervals for a and b overlap, the two events are concurrent: they are incomparable for the precedence relation $<$.

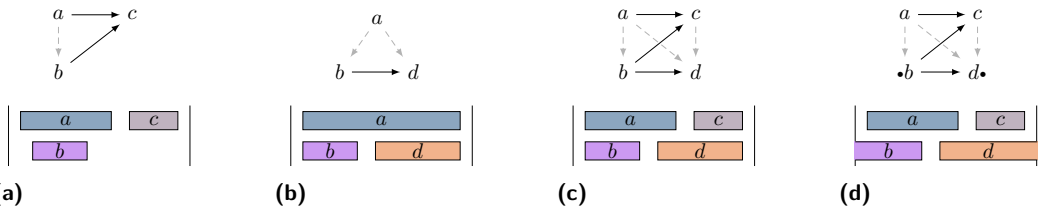


Figure 3 Four interval pomsets of dimension 2. Pomset (d) has an interface.

As seen in Figure 3, the partial orders that we are interested in arise from an interval representation, where $x < y$ means that x's interval terminates before y's interval starts. This is called an interval order.

▶ **Definition 1** (Interval order). *Let $(P, <)$ be a partially ordered set. The relation $<$ is an **interval order** if there exist $I^-, I^+ : P \to \mathbb{R}$, with $I^-(x) \leq I^+(x)$, satisfying the following condition: $\forall x, y \in P.\quad x < y \iff I^+(x) < I^-(y)$*

Not all partial orders are interval orders: for example, the $(2+2)$ pomset depicted in Figure 4 does not have an interval representation. Indeed, if we try to assign intervals to the four events, with a before c and b before d, we always end up with an extra relation: either a before d, or b before c, or both. In fact, interval orders can be characterized as exactly those partial orders that do not have an induced subposet isomorphic to $(2 + 2)$.

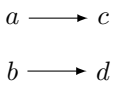


Figure 4 Example of a non-interval pomset: the $2 + 2$ partial order.

We can now define the notion of pomsets, interval pomsets, and their variants with interfaces. Pomsets are also equipped with a binary relation $\dashrightarrow$ called the event order, which orders concurrent events. The intuition of pomsets with interfaces is that we allow some events to be already active at the beginning of a pomset, or still running at the end (see Figure 3d).

▶ **Definition 2** (iiPomset). *A **partially ordered multiset** (also called **pomset**) over an alphabet Σ is a tuple $(P, <_P, \dashrightarrow_P, \lambda_P)$ where P is a finite set, $<_P$ is a strict partial order over P called **precedence**, and $\dashrightarrow_P$ is an acyclic relation on P called the **event order**, and $\lambda_P \colon P \to \Sigma$ is a labeling function, s.t. for all $x, y \in P$, exactly one of the following holds:*

$$x = y, \quad x <_P y, \quad y <_P x, \quad x \dashrightarrow_P y, \quad y \dashrightarrow_P x.$$

We write $x \parallel_P y$ when $x \not<_P y$ and $y \not<_P x$. When there is no ambiguity, we denote $<_P, \parallel_P, \dashrightarrow_P$ and λ_P as $<, \parallel, \dashrightarrow$ and λ.

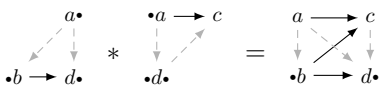
- *A pomset $(P, <_P, \dashrightarrow_P, \lambda_P)$ is an **interval pomset** if $(P, <_P)$ is an interval order.*
- *An **interval pomset with interfaces** is an interval pomset $(P, <_P, \dashrightarrow_P, \lambda_P)$ together with two sets $S_P \subseteq P$ and $T_P \subseteq P$, called the starting (resp. terminating) interfaces. We require elements of S_P (resp. T_P) to be minimal (resp. maximal) elements w.r.t. $<_P$.*

The set of interval pomsets with interfaces is denoted iiPoms. Note that pomsets are a special case of pomsets with interfaces, where the interfaces are $S_P = T_P = \varnothing$. In the rest of the paper, pomsets are always assumed to be interval pomsets with interfaces, so we drop the extra adjectives. Moreover, pomsets are (often implicitly) considered up to isomorphism: the underlying set P itself does not matter as long as the rest of the structure is the same.

The **dimension** of a pomset P is the size of a maximal $<$-antichain in P, that is, a maximal set of elements of P that are pairwise incomparable w.r.t. the precedence relation $<$. Such events are called **concurrent**. Note that any set of concurrent events is totally ordered by the event order $\dashrightarrow$. Intuitively, the dimension of a pomset is the maximal number of processes running concurrently at any time during this execution. We denote by $\text{iiPoms}_{\leq k}$ the set of pomsets of dimension $\leq k$.

When we draw pomsets in pictures, we use a plain arrow $\longrightarrow$ instead of $<$ for the precedence order, and a gray dashed arrow $\dashrightarrow$ for the event order. We represent the interfaces using a bullet symbol $\bullet$. We denote an event a as $\bullet a$ when it belongs to the starting interface S_P; $a\bullet$ when it belongs to the terminating interface T_P; and $\bullet a\bullet$ when it belongs to both. Four pomsets (with and without interface) are depicted in Figure 3, next to their interval representation.

Gluing of pomsets. Gluing is an operation on pomsets that extends word concatenation. The gluing of two pomsets P and Q is a pomset $P * Q$, where all the events of P happen before those of Q. However, we must also take care of the interfaces of P and Q. Indeed, if an event is still active when finishing P, it must be active when beginning Q. Hence, the terminating interface of P and the starting interface of Q must match. Formally, we can view T_P and S_Q as pomsets (inheriting the labeling and event order from P and Q, respectively), and we require that T_P and S_Q must be isomorphic as pomsets (*i.e.*, the labels and event order must be preserved). For the purpose of this paper, we only define gluing when the interfaces match exactly ($T_P = S_Q$ as pomsets); see [11] for a more robust definition. For instance, the pomset of Figure 3d can be obtained as:



▶ **Definition 3** (Gluing). *Let $P, Q \in$ iiPoms be two pomsets such that $P \cap Q = T_P = S_Q$. The gluing of P and Q, denoted by $P * Q$, is defined as $P * Q := (R, <_R, \dashrightarrow_R, S_R, T_R, \lambda_R)$ where:*

$$\begin{aligned} R &= P \cup Q & \lambda_R &= \lambda_P \cup \lambda_Q \\ <_R &= <_P \cup <_Q \cup (P \setminus T_P) \times (Q \setminus S_Q) & S_R &= S_P \\ \dashrightarrow_R &= \dashrightarrow_P \cup \dashrightarrow_Q & T_R &= T_Q \end{aligned}$$

2.2 ST decomposition

Starter-Terminator decomposition of pomsets is a tool introduced in [3, 12] to decompose a pomset as a gluing of elementary elements, *i.e.* pomsets with empty precedence order. This technique allows to describe pomsets over Σ of dimension k as finite words over a finite alphabet $\Omega_{\leq k}$. This makes it possible to lift results from words to pomsets.

A pomset P is called **discrete** when it has an empty precedence order. In that case, the event order $\dashrightarrow$ is a total order. Thus, we will write discrete pomsets as lists of events, between square brackets, where the event order is omitted and goes implicitly from top to bottom. For instance, $P = \left[\begin{smallmatrix} a\bullet \\ \bullet b \end{smallmatrix}\right]$ is a discrete pomset with two concurrent events $a \dashrightarrow b$, where $S_P = \{b\}$ and $T_P = \{a\}$. A discrete pomset P can be:

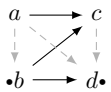
- a **conclist** if $S_P = T_P = \varnothing$,
- a **starter** if $T_P = P$,
- a **terminator** if $S_P = P$,
- an **identity** if it is both a starter and a terminator.

For example, $\left[\begin{smallmatrix} a \\ b \end{smallmatrix}\right]$ is a conclist, $\left[\begin{smallmatrix} a\bullet \\ b\bullet \end{smallmatrix}\right]$ is a starter, $\left[\begin{smallmatrix} \bullet a \\ \bullet b\bullet \end{smallmatrix}\right]$ is a terminator, and $\left[\begin{smallmatrix} \bullet a\bullet \\ \bullet b\bullet \end{smallmatrix}\right]$ is an identity. Intuitively, a starter can only start new events: so, all events must belong to the terminating interface, because they must keep running, hence the $T_P = P$. Conversely, a terminator is allowed to terminate some events, but it cannot start new ones: all events were already running at the start, thus $S_P = P$. Note that the discrete pomset $\left[\begin{smallmatrix} a\bullet \\ \bullet b \end{smallmatrix}\right]$ is neither a starter nor a terminator, since it both starts a and terminates b. A conclist has no interface, it simply denotes an (ordered) list of events that are running concurrently, hence the name, short for concurrency list.

We denote the set of conclists by CList. We write Ω for the set of all starters and terminators, and $\Omega_{\leq k}$ for the ones of dimension at most k. Notice that since the alphabet Σ is finite, the set $\Omega_{\leq k}$ is also finite. A finite word $P_1 P_2 \cdots P_n \in \Omega^*_{\leq k}$ is called **coherent** if $T_{P_i} = S_{P_{i+1}}$ for all $1 \leq i \leq n-1$. When that is the case, we can glue the successive elements in the sequence to obtain a pomset $P_1 * P_2 * \cdots * P_n \in \text{iiPoms}_{\leq k}$. A coherent word on $\Omega_{\leq k}$ is also called an **ST-sequence**. If $w \in \Omega^*_{\leq k}$ is an ST-sequence, we write $\mathsf{glue}(w) \in \text{iiPoms}_{\leq k}$ for its associated pomset.

▶ **Proposition 4** (ST decomposition [12]). *Every pomset $P \in \text{iiPoms}_{\leq k}$ can be decomposed as an ST-sequence: there exists $w \in \Omega^*_{\leq k}$ such that $P = \mathsf{glue}(w)$.*

Let us take our running example of Figure 3d and express an ST-decomposition of this pomset:



$$= \left[\begin{smallmatrix} a\bullet \\ \bullet b\bullet \end{smallmatrix}\right] * \left[\begin{smallmatrix} \bullet a\bullet \\ \bullet b \end{smallmatrix}\right] * \left[\begin{smallmatrix} \bullet a\bullet \\ d\bullet \end{smallmatrix}\right] * \left[\begin{smallmatrix} \bullet a \\ \bullet d\bullet \end{smallmatrix}\right] * \left[\begin{smallmatrix} c\bullet \\ \bullet d\bullet \end{smallmatrix}\right] * \left[\begin{smallmatrix} \bullet c \\ \bullet d\bullet \end{smallmatrix}\right]$$

An ST decomposition is called **sparse** if it alternates between starters and terminators, and contains no identities. For example, the ST decomposition given above is sparse: from left to right, we first start a, terminate b, start d, terminate a, start c, and terminate c.

In general, ST decompositions are not unique. Indeed, one can always add any number of identities (see example on the left); and when several events start at the same time, we can equivalently start one before the other, or both at once (see example on the right).

$$\begin{bmatrix} a\bullet \\ b\bullet \end{bmatrix} = \begin{bmatrix} a\bullet \\ b\bullet \end{bmatrix} * \begin{bmatrix} \bullet a\bullet \\ \bullet b\bullet \end{bmatrix} * \begin{bmatrix} \bullet a\bullet \\ \bullet b\bullet \end{bmatrix} \qquad \begin{bmatrix} a\bullet \\ b\bullet \end{bmatrix} = \begin{bmatrix} a\bullet \end{bmatrix} * \begin{bmatrix} \bullet a\bullet \\ b\bullet \end{bmatrix} = \begin{bmatrix} \\ b\bullet \end{bmatrix} * \begin{bmatrix} a\bullet \\ \bullet b\bullet \end{bmatrix}$$

However, every pomset admits a unique sparse ST decomposition (see [12] for a proof).

2.3 Monadic Second Order logic over pomsets

We now recall the Monadic Second Order (MSO) logic over pomsets introduced in [2]. The main result of [2] is a variant of Büchi's theorem for pomsets, which states that MSO logic captures the same class of pomset languages as higher dimensional automata.

The syntax of MSO formulas for pomsets is generated by the grammar:

$$\varphi, \psi ::= \neg\varphi \mid \varphi \wedge \psi \mid \exists x.\varphi \mid \exists X.\varphi \mid x \in X \mid a(x) \mid \mathbf{s}(x) \mid \mathbf{t}(x) \mid x < y \mid x \dashrightarrow y$$

where $a \in \Sigma$ is a letter of the alphabet, x, y are first order variables and X is a second-order variable. The symbols $\mathbf{s}$ and $\mathbf{t}$ are unary predicates, meaning that x belongs to the starting (resp. terminating) interface. The binary relation symbols $<$ and $\dashrightarrow$ stand for the precedence and event order.

▶ **Definition 5** (Semantics of MSO over pomsets). *An MSO formula φ is evaluated over a pomset $P = (P, <_P, \dashrightarrow_P, S_P, T_P, \lambda_P)$, together with an interpretation function ν. The function ν gives the interpretation of free variables of φ: first-order variables are mapped to events of P, and second-order variables are mapped to sets of events of P. The satisfaction relation $P, \nu \models \varphi$ is defined inductively as follows:*

$P, \nu \models a(x)$ *if* $\lambda_P(\nu(x)) = a$
$P, \nu \models x \in X$ *if* $\nu(x) \in \nu(X)$
$P, \nu \models \mathbf{s}(x)$ *if* $\nu(x) \in S_P$
$P, \nu \models \mathbf{t}(x)$ *if* $\nu(x) \in T_P$
$P, \nu \models \neg\varphi$ *if* $P, \nu \not\models \varphi$
$P, \nu \models \varphi \wedge \psi$ *if* $P, \nu \models \varphi$ *and* $P, \nu \models \psi$
$P, \nu \models x < y$ *if* $\nu(x) <_P \nu(y)$
$P, \nu \models x \dashrightarrow y$ *if* $\nu(x) \dashrightarrow_P \nu(y)$
$P, \nu \models \exists x.\varphi$ *if* $\exists p \in P$ *s.t.* $P, \nu[x \mapsto p] \models \varphi$
$P, \nu \models \exists X.\varphi$ *if* $\exists Q \subseteq P$ *s.t.* $P, \nu[X \mapsto Q] \models \varphi$

We write $P \models \varphi$ when φ does not have any free variables, and $\mathcal{L}(\varphi) = \{P \in \text{iiPoms} \mid P \models \varphi\}$.

In order to prove that MSO over pomsets is as expressive as higher dimensional automata, the authors of [2] used a detour via ST-sequences. An MSO formula φ over pomsets can be translated into a formula $\ulcorner\varphi\urcorner$ over ST-sequences that accepts the representations of the pomsets accepted by φ. The precise statement of this translation is reproduced below, in Lemma 6. Recall that ST-sequences are simply words over a different alphabet $\Omega_{\leq k}$, so MSO logic over ST-sequences is the standard MSO logic over finite words.

▶ **Lemma 6** (cf. [2, Lemma 12]). *Let φ be an MSO formula for pomsets without free variables. Then, for any $k \in \mathbb{N}$, there exists an MSO formula $\ulcorner\varphi\urcorner$ over $\Omega_{\leq k}$, such that:*

$$\mathcal{L}(\ulcorner\varphi\urcorner) = \{w \in (\Omega_{\leq k} \setminus \{\mathrm{Id}_\varnothing\})^+ \mid w \text{ is coherent and } \mathsf{glue}(w) \models \varphi\}$$

where $\mathrm{Id}_\varnothing$ denotes the empty pomset.

3 First Order logic over pomsets and ST-sequences

In this section, our goal is to prove a variant of Lemma 6 for FO formulas. Unfortunately, the proof given in [2] cannot be easily adapted: even if we start with an FO formula φ, the translation that they give yields a formula $\ulcorner\varphi\urcorner$ that contains second-order quantification. This is because their translation makes extensive use of an MSO-definable relation $(x,i) \sim (y,j)$ that keeps track of the position of an event in an ST-sequence. This relation relies on second-order quantification, and is used in several cases of the inductive translation. In Section 3.2, we show that the relation $\sim$ can actually be expressed in first order. The translation from FO formulas on pomsets to FO formulas on ST-sequences then follows in Theorem 10.

3.1 First Order logics for pomsets

First Order (FO) logic is obtained by removing the second-order quantification from the MSO logic described in Section 2.3. Given a finite alphabet Σ, the syntax of **FO formulas over pomsets** is generated by the following grammar:

$$\varphi, \psi ::= \neg\varphi \mid \varphi \wedge \psi \mid \exists x.\varphi \mid a(x) \mid \mathbf{s}(x) \mid \mathbf{t}(x) \mid x < y \mid x \dashrightarrow y$$

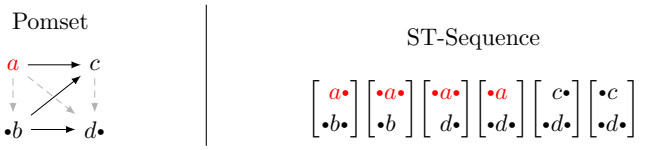
where x is a first order variable and $a \in \Sigma$. The semantics is the same as the one of Definition 5, ignoring the two cases related to second-order variables. When dealing with pomsets of dimension $\leq k$, we also consider **FO formulas over ST-sequences**. Recall that an ST-sequence is simply a word over the finite alphabet $\Omega_{\leq k}$, whose elements are starters/terminators. So this is the usual FO logic over words, whose syntax is generated by:

$$\varphi, \psi ::= \neg\varphi \mid \varphi \wedge \psi \mid \exists x.\varphi \mid P(x) \mid x < y$$

where x is a first order variable and $P \in \Omega_{\leq k}$. It is interpreted over words $w \in \Omega_{\leq k}^*$, with the usual semantics. We write $\mathrm{FO}_{\Omega_{\leq k}}$ for the set of FO formulas over $\Omega_{\leq k}$.

3.2 The same-event relation $\sim$

When translating FO formulas from pomsets to ST-sequences, a key difficulty is that first-order variables contain very different information in those two representations. In an FO formula evaluated over a pomset P, a variable x is interpreted as an element $p \in P$ of the pomset, i.e., an event. However, for an $\mathrm{FO}_{\Omega_{\leq k}}$ formula evaluated over ST-sequences, a variable x is interpreted as a position in the sequence, labeled by a letter $U \in \Omega_{\leq k}$. Each such letter contains several events (up to k, the dimension of the language), but it also contains only a small portion of those events. As seen in the example below, the event a of the pomset depicted on the left spans over 4 letters of the ST-sequence on the right.



Thus, to keep track of events in an ST-sequence, we will use pairs (x,i) where x is a first-order variable, selecting one starter/terminator in the ST-sequence; and i is the position of the event that we are currently tracking in this starter/terminator. Note that, since we

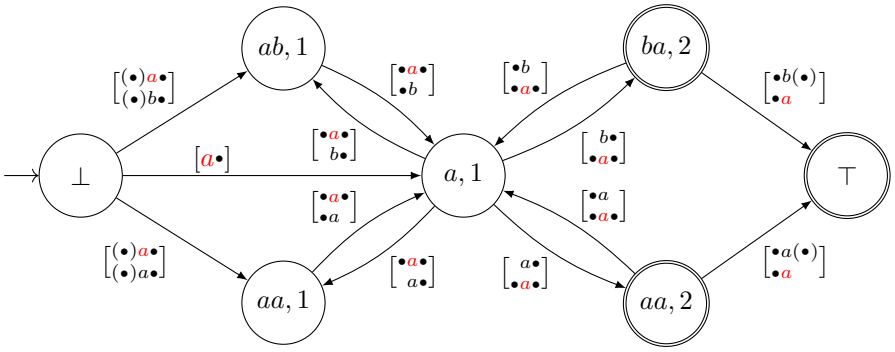


Figure 5 $\mathcal{A}_{1,2,a}$, sink state and identities not drawn.

are interested in pomsets of dimension $\leq k$, there are only finitely many possible values for $1 \leq i \leq k$. Since the same event may span several starters/terminators, it may be the case that two different pairs (x, i) and (y, j) designate the same event, as in the example below:

$$\begin{bmatrix} a\bullet \\ \bullet b\bullet \end{bmatrix} \overbrace{\begin{bmatrix} \bullet a\bullet \\ \bullet b \end{bmatrix}}^{(x,1)} \begin{bmatrix} \bullet a\bullet \\ d\bullet \end{bmatrix} \overbrace{\begin{bmatrix} \bullet a \\ \bullet d\bullet \end{bmatrix}}^{(y,1)} \begin{bmatrix} c\bullet \\ \bullet d\bullet \end{bmatrix} \begin{bmatrix} \bullet c \\ \bullet d\bullet \end{bmatrix}$$

Hence, we will need to use the **same-event relation** $(x, i) \sim (y, j)$, which is true if and only if the i-th event of the evaluation of x is the j-th event of the evaluation of y. This is the same idea as in the proof of Lemma 6 found in [2]; however, we will need to show that this relation $\sim$ can actually be defined using only first order formulas. We do so in Lemma 9, by providing a counter-free automaton recognizing it. Counter-free automata are restrictions of finite state automata whose languages are first-order definable.

▶ **Definition 7** ([19], Counter-Free Automata). *Let $\mathcal{A} = (\Sigma, Q, q_0, F, \delta)$ be a deterministic finite-state automaton. $\mathcal{A}$ is **counter-free** if there exists a positive integer n such that for any non-empty word $w \in \Sigma^*$ and for any state $q \in Q$, the state-transition function satisfies the following equality: $\delta(q, w^n) = \delta(q, w^{n+1})$.*

▶ **Proposition 8** ([19]). *Counter-free automata are as expressive as FO logic over words.*

▶ **Lemma 9.** *Fix $i, j \in \{1, \dots, k\}$. Then, the binary relation $(x, i) \sim (y, j)$ is definable by an $\mathrm{FO}_{\Omega_{\leq k}}$ formula with two free variables x and y.*

Proof (Sketch). Given an event a and two indexes $i, j \in \{1, \dots, k\}$, we build a counter-free automaton $\mathcal{A}_{i,j,a}$ that scans an ST-sequence $P_1 P_2 \cdots P_n$ and accepts if and only if the i-th element of P_1 is the same a-event as the j-th element of P_n. Figure 5 depicts an example of such an automaton, with event alphabet $\Sigma = \{a, b\}$ and pomsets of dimension at most $k = 2$. The a-event that the automaton is currently following is depicted in red on each starter/terminator that it reads. Inside the states, we keep track of the list of currently active events (*i.e.* a conclist $U \in \mathsf{CList}_{\leq k}$), and the position ($i \in \{1, \dots, k\}$) of the followed event. Hence, the set of states is $\mathsf{CList}_{\leq k} \times \{1, \dots, k\}$, plus an initial, a final and a sink state. See Appendix B.1 for the precise formal definition of the automaton.

To prove that all $\mathcal{A}_{i,j,a}$ are counter-free, consider a state (U, ℓ), and an ST-sequence $w = P_1 \cdots P_n$. We need to show that $\mathcal{A}_{i,j,a}$ will never fall in a non-trivial cycle when reading w repeatedly from (U, ℓ). There are several cases:

- If $S_{P_1} \neq U$ or $T_{P_n} \neq U$, then the execution fails and falls in a sink state after one or two iterations.
- If $S_{P_1} = T_{P_n} = U$ and the ℓ-th element of P_1 is the ℓ-th element of P_n, then we are in a trivial cycle (an execution reading w from (U, ℓ) arrives in (U, ℓ))
- If $S_{P_1} = T_{P_n} = U$ and the ℓ-th element of P_1 is the ℓ'-th element of P_n, with $\ell > \ell'$ (the opposite case is similar), then ℓ will keep decreasing strictly as we iterate w. The example below illustrates what might happen, with a pomset of dimension $k = 3$. We start in state $(aaa, 3)$, so three a's are running concurrently, and we are tracking the third one (in red). When the automaton reads the word w below, the first a is terminated, and another a is started, but this new a is placed after the other two according to event order. The tracked a ends up in position 2, so the state of the automaton is now $(aaa, 2)$. After decreasing a finite number of times, the execution arrives in the sink state – and fails.

$$w = \begin{bmatrix} \bullet a \\ \bullet a \bullet \\ \bullet a \bullet \end{bmatrix} \begin{bmatrix} \bullet a \bullet \\ \bullet a \bullet \\ a \bullet \end{bmatrix} \qquad \begin{aligned} \delta((aaa, 3), w) &= (aaa, 2) \\ \delta((aaa, 3), w^2) &= (aaa, 1) \end{aligned}$$

Hence, from any state (U, ℓ), the execution enters a trivial cycle in at most $k+1$ steps. Since any execution leaves the initial state in one step, $\mathcal{A}_{i,j,a}$ is counter-free (by taking $n = k + 2$ in Definition 7). So $\mathcal{A}_{i,j,a}$ can be expressed as an FO formula, and with a disjunction over all $a \in \Sigma$ we get an FO formula for $(x, i) \sim (y, j)$. See Appendix B.1 for details. ◀

3.3 Translation of FO formulas from pomsets to ST-sequences

Now that we have proven that the same-event relation $\sim$ is FO-definable, we can inductively translate FO formulas on pomsets to $\mathrm{FO}_{\Omega_{\leq k}}$ formulas on ST-sequences. The inductive definition of $\ulcorner \varphi \urcorner$ is the same as the one of [2]. We reproduce it in the full version for completeness [4].

▶ **Theorem 10.** *Let φ be an FO formula over pomsets without free variables. Then, for any $k \in \mathbb{N}$, there exists an $\mathrm{FO}_{\Omega_{\leq k}}$ formula $\ulcorner \varphi \urcorner$ over $\Omega_{\leq k}$ such that:*

$$\mathcal{L}(\ulcorner \varphi \urcorner) = \{w \in (\Omega_{\leq k} \setminus \{\mathrm{Id}_\varnothing\})^+ \mid w \text{ is coherent and } \mathsf{glue}(w) \models \varphi\}$$

In the full version [4], we show that in the worst case, the size of $\ulcorner \varphi \urcorner$ is $O(k^{|\varphi|})$, where $|\varphi|$ is the size of φ. As a corollary of Theorem 10, we can can show that FO is strictly weaker than MSO for pomset languages.

▶ **Corollary 11.** *FO is strictly weaker than MSO.*

Proof. Consider the language of pomsets $L = \left\{ \left[\begin{smallmatrix} a \\ b \end{smallmatrix}\right]^{2n} \mid n \in \mathbb{N} \right\}$, where the exponent denotes gluing iteration. Viewed as a language of ST-sequences (i.e., words on $\Omega_{\leq 2}$), this corresponds to $W = \left\{ \left(\left[\begin{smallmatrix} a \bullet \\ b \bullet \end{smallmatrix}\right] \left[\begin{smallmatrix} \bullet a \\ \bullet b \end{smallmatrix}\right] \right)^{2n} \mid n \in \mathbb{N} \right\}$. As a language of words, W is not definable in $\mathrm{FO}_{\Omega_{\leq k}}$ (see e.g. [8] for a proof). By the contrapositive of Theorem 10, L cannot be defined in FO. ◀

4 A Linear Temporal Logic for Pomsets

In this section, we introduce our Linear Temporal Logic for pomsets, $\mathrm{LTL}_{\mathrm{Poms}}$. Using the terminology of [6], it is a *local* temporal logic, meaning that it is interpreted over a single event of the pomset, rather than on a global state. However, since events can span a long period of time, they are split into so-called *sub-events* (see Figure 6). While defining a

temporal logic directly over events might seem more straightforward, we show in Appendix A that such a logic is strictly less expressive than FO over pomsets. Nonetheless, we will show in Section 5, Theorem 22 that with sub-events, $\mathrm{LTL}_{\mathrm{Poms}}$ is as expressive as FO over pomsets of dimension $\leq k$.

4.1 Syntax and Semantics

Let us first give an intuitive explanation of the notion of sub-event, relying on the interval representation of a pomset. Consider the pomset P depicted in Figure 6. The interval representation of P is decomposed in three "slices" (delimited by dotted lines). Each slice corresponds to two consecutive elements of the sparse ST-decomposition of P, i.e., a starter and a terminator. Thus, in every slice, some events start, then some events terminate. Intuitively, a sub-event is given by an event $x \in P$ together with one of the slices it crosses. For instance, event a spans two slices: therefore, it is divided into two sub-events a_1 and a_2.

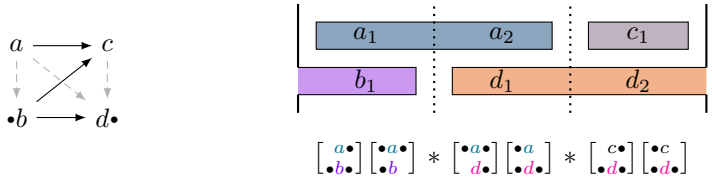


Figure 6 A slicing of a pomset with 4 events (a, b, c, d) into 6 sub-events (a_1, a_2, b_1, c_1, d_1, d_2).

Our formal definition of sub-events does not rely on the ST-decomposition of the pomset. Instead, we define it directly on the pomset itself. A sub-event is a pair of two events, (x, m), where $x \in P$ is the event being considered, and $m \in P$ acts as a timestamp indicating the current slice. One can think of m as *the latest event to have started*. For example, in Figure 6, the sub-event a_1 is given by the pair (a, a); while a_2 is given by the pair (a, d). However, notice that not all pairs of events define sub-events: (c, d) makes no sense since event c is not running when d starts. Definition 12 introduces an order that captures these situations.

▶ **Definition 12.** *Let P be a pomset. We say that $x \in P$ **starts before** $y \in P$, denoted $x <^s y$ if there exists $z \in P$ such that $x \parallel z$ and $z < y$. We write $x \sim^s y$ when $x \not<^s y$ and $y \not<^s x$, and $x \lesssim^s y$ when $x <^s y$ or $x \sim^s y$.*

Intuitively, $x <^s y$ means that in *every* interval representation of P, the interval representing x starts before the one representing y. For example in the pomset of Figure 6, we have $a <^s d$ and $d <^s c$. Observe that $\sim^s$ is an equivalence relation whose equivalence classes correspond intuitively to the slices of P (for example, $a \sim^s b$ in Figure 6). Hence, $\lesssim^s$ is a total preorder on the events of P, and we think of it as a total order on the slices.

▶ **Definition 13** (Sub-event). *A **sub-event** of a pomset P is a pair $(x, m) \in P^2$ where $x \parallel m$ and $x \lesssim^s m$. When $x = y$ and $m \sim^s q$, we write $(x, m) \equiv (y, q)$.*

Note that, when several events start in the same slice, two sub-events may actually represent the same point in the interval representation. This is why we introduce the $\equiv$ relation over sub-events. When $(x, m) \equiv (y, q)$, we think of (x, m) and (y, q) as representing "the same" sub-event. For example, the sub-event denoted by a_1 in Figure 6 can be formalized either by (a, a) or (a, b), since $a \sim^s b$. To simplify the presentation, we do not explicitly quotient by the relation $\equiv$. However, we will make sure that whenever $(x, m) \equiv (y, q)$, the two sub-events satisfy the same formulas, i.e., $P, (x, m) \models \varphi$ iff $P, (y, q) \models \varphi$ (see Proposition 18).

Next, we define an order between sub-events. This order will be crucial to ensure that our temporal logic operators can only go forward in time. There are two ways to advance in time: either stay within the same event x, but move to a later slice; or terminate the current event x and jump to a new event y that occurs after x.

▶ **Definition 14** (Order over sub-events). *Given two sub-events (x,m) and (y,q), we say that (x,m) **precedes** (y,q) in P, denoted $(x,m) \prec (y,q)$, if either $x < y$, or $x = y$ and $m <^s q$.*

We denote by $\preccurlyeq$ the non-strict version: $(x,m) \preccurlyeq (y,q)$ if $(x,m) \prec (y,q)$ or $(x,m) \equiv (y,q)$. Observe that $\preccurlyeq$ is a (partial) preorder on sub-events. In Figure 6, we have for example $a_1 \prec a_2 \prec c_1$ and $b_1 \prec c_1$. However, $a_1 \not\prec d_1$ and $b_1 \not\prec a_2$: it is not allowed to jump to a different event that is concurrent with the current one. Indeed, we want our temporal operators to be able to follow the local view of individual processes. (There will, however, be a different modality allowing to jump to a concurrent event within the same slice.)

Finally, we introduce a one-step version of $\prec$, which plays the role of our "next" modality.

▶ **Definition 15.** *$(x,m) \prec_1 (y,q)$ if $(x,m) \prec (y,q)$ and there is no r such that $m <^s r <^s q$.*

Intuitively, the relation $\prec_1$ only orders events in adjacent slices. For instance, in Figure 6, we have $b_1 \prec_1 d_1$, but $b_1 \not\prec_1 c_1$. Note that Definition 15 is more restrictive than the requirement that "there is no (z,r) such that $(x,m) \prec (z,r) \prec (y,q)$". The latter would allow jumping directly from b_1 to c_1 in Figure 6, skipping a slice. We prefer to ensure that our "next" modality always advances by exactly one slice.

We are now ready to introduce our logic $\mathrm{LTL}_{\mathrm{Poms}}$. The "next" modality, denoted $\langle \prec_1 \rangle$, jumps to a successor sub-event in the next slice. Note that it is existential, since there may be more than one successor. The modalities $\langle \dashrightarrow \rangle$ and $\langle \dashrightarrow^{-1} \rangle$ jump to a concurrent sub-event within the same slice. Recall that in our pomsets, concurrent events are totally ordered by the event order $\dashrightarrow$. Finally, the atomic formula $\mathbf{s}$ (resp. $\mathbf{t}$) checks whether the current sub-event is being started (resp. terminated) in the current slice.

▶ **Definition 16** ($\mathrm{LTL}_{\mathrm{Poms}}$). *The syntax of $\mathrm{LTL}_{\mathrm{Poms}}$ is generated by the following grammar:*

$$\varphi, \psi ::= a \mid \mathbf{s} \mid \mathbf{t} \mid \neg\varphi \mid \varphi \wedge \psi \mid \langle \prec_1 \rangle \varphi \mid \langle \dashrightarrow \rangle \varphi \mid \langle \dashrightarrow^{-1} \rangle \varphi \mid \varphi \,\mathbf{U}\, \psi$$

where $a \in \Sigma$. Given a $\mathrm{LTL}_{\mathrm{Poms}}$ formula φ and a sub-event (x,m), we define the satisfaction relation $P,(x,m) \models \varphi$ by induction on the formula φ:

1. *$P,(x,m) \models a$ if $\lambda_P(x) = a$*
2. *$P,(x,m) \models \mathbf{s}$ if $x \notin S_P$ and $x \sim^s m$*
3. *$P,(x,m) \models \mathbf{t}$ if $x \notin T_P$ and there is no $q \in P$ s.t. $x \parallel q$ and $m <^s q$*
4. *$P,(x,m) \models \neg\varphi$ if $P,(x,m) \not\models \varphi$*
5. *$P,(x,m) \models \varphi \wedge \psi$ if $P,(x,m) \models \varphi$ and $P,(x,m) \models \psi$*
6. *$P,(x,m) \models \langle \prec_1 \rangle \varphi$ if there is a sub-event (y,q) such that $(x,m) \prec_1 (y,q)$ and $P,(y,q) \models \varphi$.*
7. *$P,(x,m) \models \langle \dashrightarrow \rangle \varphi$ if there is $y \in P$ such that (y,m) is a sub-event, y is a direct successor of x by $\dashrightarrow$, and $P,(y,m) \models \varphi$*
8. *$P,(x,m) \models \langle \dashrightarrow^{-1} \rangle \varphi$ if there is $y \in P$ such that (y,m) is a sub-event, y is a direct predecessor of x by $\dashrightarrow$, and $P,(y,m) \models \varphi$*
9. *$P,(x,m) \models \varphi \,\mathbf{U}\, \psi$ if there exists (y,n) such that $(x,m) \preccurlyeq (y,n)$, $P,(y,n) \models \psi$ and, for all (z,q) such that $(x,m) \preccurlyeq (z,q) \prec (y,n)$, $P,(z,q) \models \varphi$.*

To define what "P satisfies φ" means for a given pomset P and $\mathrm{LTL}_{\mathrm{Poms}}$ formula φ, we need to choose a canonical "source" sub-event of the pomset. Let $M_P \subseteq P$ be the set of events that are minimal according to $<_P$. Notice that M_P is totally ordered by the event-order relation $\dashrightarrow_P$. Then we let $\mathrm{source}(P) = (x, x)$, where $x = \min_{\dashrightarrow_P} M_P$. Intuitively, this is the top-left sub-event in an interval representation. Finally, define $P \models \varphi$ iff $P, \mathrm{source}(P) \models \varphi$.

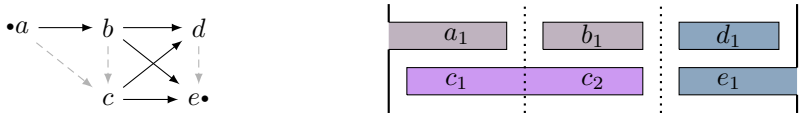


Figure 7 A pomset containing 5 events a, b, c, d, e, where c is split into two sub-events c_1 and c_2. Note that formally, c_1 has two representatives $(c, a) \equiv (c, c)$, while c_2 is represented by (c, b).

▶ **Example 17.** In the pomset represented in Figure 7, $c_1 \models \mathbf{s} \wedge \neg \mathbf{t}$ while $c_2 \models \neg \mathbf{s} \wedge \mathbf{t}$. Notice that $a_1 \not\models \mathbf{s}$: since event a belongs to the starting interface of the pomset, it was already started in the first slice. Similarly, $e_1 \not\models \mathbf{t}$. The $\langle \prec_1 \rangle$ modality allows to jump from c_2 to either d_1 or e_1, with an existential quantification. It can be read as an "exists next" modality. For instance, we have $c_2 \models \langle \prec_1 \rangle\, d$ and $c_2 \models \langle \prec_1 \rangle\, e$, but $c_2 \not\models \langle \prec_1 \rangle\, d \wedge e$. However, note that $c_1 \not\models b$: the "next" modality does not allow to jump to a different event since event c is still running (cf. Definition 15). The dual "for all next" modality can be defined as $[\prec_1]\, \varphi := \neg \langle \prec_1 \rangle\, \neg \varphi$.

The operators $\langle \dashrightarrow \rangle$ and $\langle \dashrightarrow^{-1} \rangle$ allow to move to a concurrent event following the event order relation, while staying within the current slice. So, crucially, $b_1 \not\models \langle \dashrightarrow \rangle \langle \dashrightarrow^{-1} \rangle\, a$. This was our motivation to introduce the notion of sub-events. Without them, one could inadvertently move forwards or backwards in time by only following event order relations. Note that, within a slice, the event order relation is total. Thus, there can be at most one direct successor. The operator $\langle \dashrightarrow \rangle$ is still existential in a degenerate sense: if there is no successor, the formula is not satisfied. For instance, $e_1 \not\models \langle \dashrightarrow \rangle\, \top$.

The operator $\mathbf{U}$ is the usual Until modality with regard to $\preccurlyeq$. However, it might be slightly counter-intuitive. Despite the universal quantification over all intermediary sub-events, it may seemingly "miss" events that are concurrent with one of the two endpoints. For instance, in the pomset of Figure 7, $a_1 \models (a \vee b)\, \mathbf{U}\, d$: since event c is parallel with a, the sub-event c_2 is not reachable from a_1. In the next section, we will define a variant of the Until operator that also takes into account these parallel events.

As stated earlier, our logic is consistent with the $\equiv$ relation. This is proven by induction over the formula; the full proof can be found in Appendix B.2.

▶ **Proposition 18.** *Let P be a pomset, and let $(x, m) \equiv (y, q)$ two equivalent sub-events of P, then for every formula φ of $\mathrm{LTL}_{\mathrm{Poms}}$, $P, (x, m) \models \varphi$ if and only if $P, (y, q) \models \varphi$.*

4.2 Derived operators

The precise choice of operators for the logic $\mathrm{LTL}_{\mathrm{Poms}}$ may seem somewhat arbitrary. It will be justified in Section 5, where we show that it is equivalent to FO. So, any first-order definable temporal operators can also be defined in $\mathrm{LTL}_{\mathrm{Poms}}$. We give a few useful examples below. Recall that in the whole paper, we are working with pomsets of bounded dimension k. So there can be at most k parallel events at any time.

- **Exists parallel:** $\langle\!\|\rangle\,\varphi := \bigvee_{i=0}^{k}(\langle \dashrightarrow\rangle^i\,\varphi \vee \langle \dashrightarrow^{-1}\rangle^i\,\varphi)$. This operator jumps to some sub-event in the current slice. $P,(x,m) \models \langle\!\|\rangle\,\varphi$ iff there exists $y \in P$ such that (y,m) is a subevent and $P,(y,m) \models \varphi$. The dual universally quantified operator is $[\!|\!|]\,\varphi := \neg\,\langle\!\|\rangle\,\neg\varphi$.
- **Exists strict parallel:** $\langle\!\|^{\neq}\rangle\,\varphi := \bigvee_{i=1}^{k}(\langle \dashrightarrow\rangle^i\,\varphi \vee \langle \dashrightarrow^{-1}\rangle^i\,\varphi)$. Similar to the previous one, but this operator must jump to a different sub-event. In particular, if there is no other event currently running, this formula is always false.
- **Finally and Globally:** $\mathbf{F}\varphi := \top\,\mathbf{U}\,\varphi$. This is the usual Finally operator of temporal logics: $P,(x,m) \models \mathbf{F}\varphi$ iff there exists a sub-event (y,q) of P such that $(x,m) \preccurlyeq (y,q)$ and $P,(y,q) \models \varphi$. Its dual operator is Globally, defined as $\mathbf{G}\varphi := \neg\,\mathbf{F}\,\neg\varphi$.
- **Finally parallel and Globally parallel:** $\mathbf{F}^{\|}\varphi := \mathbf{F}\,\langle\!\|\rangle\,\varphi$. This variant of $\mathbf{F}$ covers all sub-events that happen later or at the same time as the current sub-event, even those that are on parallel events. $P,(x,m) \models \mathbf{F}^{\|}\varphi$ iff there exists a sub-event (y,q) of P such that $m \lesssim^s q$ and $P,(y,q) \models \varphi$. The dual universally quantified operator is $\mathbf{G}^{\|}\varphi := \neg\,\mathbf{F}^{\|}\,\neg\varphi$. Equivalently, one can also define it as $\mathbf{G}^{\|}\varphi := \mathbf{G}\,[\!|\!|]\,\varphi$.
- **Until parallel:** $\varphi\,\mathbf{U}^{\|}\,\psi := ([\!|\!|]\,\varphi)\,\mathbf{U}\,(\langle\!\|\rangle\,\psi)$. This variant of the Until modality takes into account events that are concurrent with the two endpoints. More formally, $P,(x,m) \models \varphi\,\mathbf{U}^{\|}\,\psi$ iff there exists a sub-event of the form (y,q) with $m \lesssim^s q$ such that $P,(y,q) \models \psi$ and for every sub-event (z,r) such that $m \lesssim^s r <^s q$, $P,(z,r) \models \varphi$.

4.3 Toy example

To illustrate how our logic $\mathrm{LTL}_{\mathrm{Poms}}$ allows to concisely express properties of concurrent systems, let us consider a very simple example: specifying the correctness of a mutual exclusion algorithm using locks. Thus, suppose that we have a lock mechanism available, with two operations: action P to *acquire* the lock, and action V to *release* it [9]. Now assume that we want our processes to use some critical resource a (perhaps a shared data structure) that cannot be accessed concurrently. So, we want to ensure that there can never be two processes executing action a concurrently. The obvious solution to this problem is the following: every process runs the program $(P;a;V)^*$. That is, first acquire the lock, then perform action a, and then release the lock (and repeat).

We would like to specify that this implementation of a critical section is correct. For that, we need to specify (i) the behavior of the lock, and (ii) the expected behavior of the mutual exclusion algorithm. We express both of those properties in the language of $\mathrm{LTL}_{\mathrm{Poms}}$.

(i) Lock specification. $\varphi_{\mathrm{lock}} = \mathbf{G}^{\|}((P \wedge \mathbf{t}) \Rightarrow (\neg\,\langle\!\|^{\neq}\rangle(P \wedge \mathbf{t}) \wedge \langle \prec_1\rangle(\neg(P \wedge \mathbf{t})\,\mathbf{U}^{\|}\,(V \wedge \mathbf{s}))))$.
This formula expresses that at any point during the execution of the program, whenever an action P terminates (i.e., a process acquires the lock), then no other process can acquire the lock $(P \wedge \mathbf{t})$ until the lock is released $(V \wedge \mathbf{s})$.

(ii) Mutual exclusion specification. $\varphi_{\mathrm{exclusion}} = \mathbf{G}^{\|}(a \Rightarrow \neg\,\langle\!\|^{\neq}\rangle\,a)$.
This formula expresses that there are never two overlapping events a.

What we want to verify is that, assuming the P/V actions behave according to the lock specification, our algorithm ensures the mutual exclusion property is satisfied. The algorithm itself (say, for k processes) can be modeled as an HDA, obtained as the parallel composition of k copies of one process performing a loop $(P;a;V)^*$. Executions of this HDA are pomsets, but not all of them satisfy the lock specification. We want to check that every execution that satisfies φ_{lock} also satisfies $\varphi_{\mathrm{exclusion}}$. This amounts to model-checking that every pomset in the language of the HDA satisfies the formula $\varphi_{\mathrm{lock}} \Rightarrow \varphi_{\mathrm{exclusion}}$.

5 Expressivity

In this section, we show that $\mathrm{LTL}_{\mathrm{Poms}}$ has the same expressive power as first order logic for pomsets of bounded dimension. We prove this by providing well-chosen translations between pomset and word logics, thus allowing us to use directly the results of Kamp's theorem. The translation from $\mathrm{LTL}_{\mathrm{Poms}}$ to FO can be done by writing the semantics of $\mathrm{LTL}_{\mathrm{Poms}}$ in FO formulas (see Proposition 21). Since we already gave the translation from FO on pomsets to FO on ST-sequences in Theorem 10, we now have to prove that any LTL formula over ST-sequences can be translated to an equivalent $\mathrm{LTL}_{\mathrm{Poms}}$ formula.

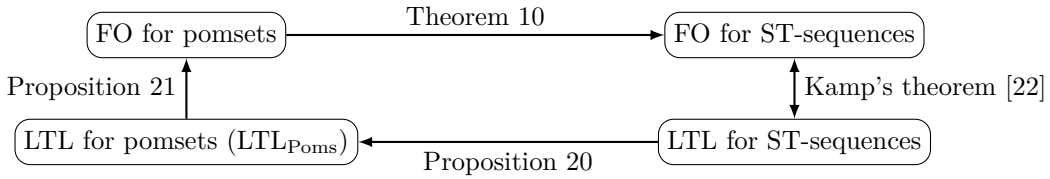


Figure 8 Summary of the proof of Theorem 22.

First, we formalize the intuition given at the start of Section 4.1: the "slices" of a pomset can be obtained by gluing two consecutive elements (a starter and a terminator) of its sparse ST-decomposition. The following definition allows padding the sparse ST-decomposition with identity elements, in order to make sure that it always starts with a starter, and ends with a terminator. We write Id_U for the identity pomset with starting and terminating interface U.

▶ **Definition 19.** *Given a pomset P with sparse ST-decomposition $P_1P_2\cdots P_n$, the **even-sparse** ST-decomposition of P is $SP_1P_2\cdots P_nT$ where $S = \mathrm{Id}_{S_{P_1}}$ if P_1 is a terminator, or $S = \varepsilon$ otherwise, and $T = \mathrm{Id}_{T_{P_n}}$ if P_n is a starter, or $T = \varepsilon$ otherwise.*

▶ **Proposition 20.** *For any k, for any LTL formula φ over k-dimensional ST-sequences, there exists an $\mathrm{LTL}_{\mathrm{Poms}}$ formula $\ulcorner\varphi\urcorner$ such that for any pomset P with even-sparse decomposition $P_1P_2\cdots P_{2n}$, we have $P \models \ulcorner\varphi\urcorner$ if and only if $P_1P_2\cdots P_{2n} \models \varphi$.*

Proof sketch. Notice that, since one "slice" of a pomset corresponds to two symbols in the ST-decomposition, one application of the Next modality $\langle\prec_1\rangle$ of $\mathrm{LTL}_{\mathrm{Poms}}$ corresponds to two applications of the Next modality $\mathbf{X}$ of LTL. Thus, we actually define two translations $\ulcorner\varphi\urcorner_{\mathbf{s}}$ and $\ulcorner\varphi\urcorner_{\mathbf{t}}$ by mutual induction. The first translation is the one required to prove the Proposition, $\ulcorner\varphi\urcorner = \ulcorner\varphi\urcorner_{\mathbf{s}}$. The second translation $\ulcorner\varphi\urcorner_{\mathbf{t}}$ is used after one application of $\mathbf{X}$, when the ST-decomposition starts with a terminator.

For the base case where $\varphi = P$ for some starter or terminator P, $\ulcorner\varphi\urcorner_{\mathbf{s}}$ checks that P is indeed a starter, and that the elements in the current slice are exactly those of P. Similarly for $\ulcorner\varphi\urcorner_{\mathbf{t}}$, we check that P is a terminator. To translate the $\mathbf{X}$ modality, if the current element is a starter, then the next one is the terminator of the same slice: $\ulcorner\mathbf{X}\psi\urcorner_{\mathbf{s}} = \ulcorner\psi\urcorner_{\mathbf{t}}$. If, on the contrary, the current element is a terminator, we need to jump to the next slice: $\ulcorner\mathbf{X}\psi\urcorner_{\mathbf{t}} = \langle\prec_1\rangle\ulcorner\psi\urcorner_{\mathbf{s}}$. For the Until modality, we need to ensure that any intermediate slice satisfies ψ_1 for both the starter and terminator parts, until the final slice which satisfies ψ_2 either in its starter or terminator part: $\ulcorner\psi_1\,\mathbf{U}\,\psi_2\urcorner_{\mathbf{s}} = (\ulcorner\psi_1\urcorner_{\mathbf{s}} \wedge \ulcorner\psi_1\urcorner_{\mathbf{t}})\,\mathbf{U}\,(\ulcorner\psi_2\urcorner_{\mathbf{s}} \vee (\ulcorner\psi_1\urcorner_{\mathbf{s}} \wedge \ulcorner\psi_2\urcorner_{\mathbf{t}}))$ and $\ulcorner\psi_1\,\mathbf{U}\,\psi_2\urcorner_{\mathbf{t}} = \ulcorner\psi_2\urcorner_{\mathbf{t}} \vee (\ulcorner\psi_1\urcorner_{\mathbf{t}} \wedge \langle\prec_1\rangle((\ulcorner\psi_1\urcorner_{\mathbf{s}} \wedge \ulcorner\psi_1\urcorner_{\mathbf{t}})\,\mathbf{U}\,(\ulcorner\psi_2\urcorner_{\mathbf{s}} \vee (\ulcorner\psi_1\urcorner_{\mathbf{s}} \wedge \ulcorner\psi_2\urcorner_{\mathbf{t}}))))$.

The complete proof is in the full version [4]. ◀

Proposition 21 is proved by expressing the semantics of Definition 16 in FO. All details are in the full version [4]. Then we conclude the proof of expressivity of $\mathrm{LTL}_{\mathrm{Poms}}$ in Theorem 22.

▶ **Proposition 21.** *For any* $\mathrm{LTL}_{\mathrm{Poms}}$ *formula* φ*, there exists an FO formula over pomsets* $\ulcorner\varphi\urcorner$ *such that, for any pomset* P*,* $P \models \varphi$ *if and only if* $P \models \ulcorner\varphi\urcorner$*.*

▶ **Theorem 22.** *For any pomset language* L *of bounded dimension* k*, the two following statements are equivalent:*

1. *There exists* φ*, an FO formula, such that for any pomset* P*,* $P \in L$ *if and only if* $P \models \varphi$*.*
2. *There exists* ψ*, an* $\mathrm{LTL}_{\mathrm{Poms}}$ *formula, such that for any pomset* P*,* $P \in L$ *if and only if* $P \models \psi$*.*

Proof. Let us proceed as depicted in Figure 8. By Theorem 10, any FO formula φ over pomsets can be translated into an equivalent FO formula φ' over ST-sequences, accepting exactly the ST-decompositions of pomsets accepted by φ. This formula can in turn be translated into an equivalent LTL formula φ'' by Kamp's theorem [22]. Finally, by Proposition 20, there is an $\mathrm{LTL}_{\mathrm{Poms}}$ formula φ''', such that for any pomset P, $P \models \varphi'''$ if and only if P's even-sparse ST-decomposition validates φ'', which in turn is equivalent to the fact that $P \models \varphi$. This proves that FO is at most as expressive as $\mathrm{LTL}_{\mathrm{Poms}}$. Proposition 21 ensure that they are in fact equivalent by providing the converse translation. ◀

▶ Remark 23. All our proofs are constructive, in the sense that algorithms can easily be extracted from our translations and from Kamp's theorem [22]. Given an FO formula φ, the size of the associated $\mathrm{LTL}_{\mathrm{Poms}}$ formula is non-elementary in the size of φ. This is because there is a non-elementary succinctness gap between FO and LTL over words [22]. For the converse translation, given an $\mathrm{LTL}_{\mathrm{Poms}}$ formula ψ, the size of the translated formula FO is linear in the size of ψ. The complexity for each translation depicted in Figure 8 can be found alongside the associated proofs in the full version [4].

6 Conclusion and future work

This paper is a first step towards understanding temporal logics on HDA pomset languages. We have established two key results. The first one, Theorem 10, shows that FO formulas on pomsets can be translated to FO on ST-sequences. Our second result is Theorem 22, asserting that the temporal logic $\mathrm{LTL}_{\mathrm{Poms}}$ is equivalent to FO over pomsets, thus extending Kamp's theorem to pomset languages.

As future work, it would be insightful to find other characterizations of the class of FO-definable pomset languages, such as defining a notion of counter-free higher dimensional automata. More importantly, a central purpose of temporal logics is the specification and verification of program properties. Thus, the obvious next steps of this work is to design efficient algorithms for deciding satisfiability and model-checking of $\mathrm{LTL}_{\mathrm{Poms}}$ formulas. It is already known (see [2], Corollary 10) that these problems are decidable for MSO formulas, and thus also for FO formulas. We hope however that verifying $\mathrm{LTL}_{\mathrm{Poms}}$ formulas directly should yield a better complexity than relying on their FO translation.

References

1 Amazigh Amrane, Hugo Bazille, Emily Clement, Uli Fahrenberg, and Krzysztof Ziemianski. Presenting interval pomsets with interfaces. In Uli Fahrenberg, Wesley Fussner, and Roland Glück, editors, *Relational and Algebraic Methods in Computer Science - 21st International Conference, RAMiCS 2024, Prague, Czech Republic, August 19-22, 2024, Proceedings*, volume 14787 of *Lecture Notes in Computer Science*, pages 28–45. Springer, 2024. doi:10.1007/978-3-031-68279-7_3.

2 Amazigh Amrane, Hugo Bazille, Uli Fahrenberg, and Marie Fortin. Logic and languages of higher-dimensional automata. In Joel D. Day and Florin Manea, editors, *Developments in Language Theory - 28th International Conference, DLT 2024, Göttingen, Germany, August 12-16, 2024, Proceedings*, volume 14791 of *Lecture Notes in Computer Science*, pages 51–67. Springer, 2024. doi:10.1007/978-3-031-66159-4_5.

3 Amazigh Amrane, Hugo Bazille, Uli Fahrenberg, and Krzysztof Ziemianski. Closure and decision properties for higher-dimensional automata. In Erika Ábrahám, Clemens Dubslaff, and Silvia Lizeth Tapia Tarifa, editors, *Theoretical Aspects of Computing - ICTAC 2023 - 20th International Colloquium, Lima, Peru, December 4-8, 2023, Proceedings*, volume 14446 of *Lecture Notes in Computer Science*, pages 295–312. Springer, 2023. doi:10.1007/978-3-031-47963-2_18.

4 Emily Clement, Enzo Erlich, and Jérémy Ledent. Expressivity of linear temporal logic for pomset languages of higher dimensional automata. *CoRR*, abs/2410.12493, 2024. doi:10.48550/arXiv.2410.12493.

5 Volker Diekert and Paul Gastin. LTL is expressively complete for mazurkiewicz traces. *J. Comput. Syst. Sci.*, 64(2):396–418, 2002. doi:10.1006/jcss.2001.1817.

6 Volker Diekert and Paul Gastin. From local to global temporal logics over mazurkiewicz traces. *Theoretical Computer Science*, 356(1):126–135, 2006. In honour of Professor Christian Choffrut on the occasion of his 60th birthday. doi:10.1016/j.tcs.2006.01.035.

7 Volker Diekert and Paul Gastin. Pure future local temporal logics are expressively complete for mazurkiewicz traces. *Inf. Comput.*, 204(11):1597–1619, 2006. doi:10.1016/j.ic.2006.07.002.

8 Volker Diekert and Paul Gastin. First-order definable languages. In Jörg Flum, Erich Grädel, and Thomas Wilke, editors, *Logic and Automata: History and Perspectives [in Honor of Wolfgang Thomas]*, volume 2 of *Texts in Logic and Games*, pages 261–306. Amsterdam University Press, 2008.

9 Edsger W. Dijkstra. The structure of "the"-multiprogramming system. *Commun. ACM*, 11(5):341–346, 1968. doi:10.1145/363095.363143.

10 Uli Fahrenberg, Christian Johansen, Georg Struth, and Krzysztof Ziemianski. Languages of higher-dimensional automata. *Math. Struct. Comput. Sci.*, 31(5):575–613, 2021. doi:10.1017/S0960129521000293.

11 Uli Fahrenberg, Christian Johansen, Georg Struth, and Krzysztof Ziemianski. A kleene theorem for higher-dimensional automata. In Bartek Klin, Slawomir Lasota, and Anca Muscholl, editors, *33rd International Conference on Concurrency Theory, CONCUR 2022, September 12-16, 2022, Warsaw, Poland*, volume 243 of *LIPIcs*, pages 29:1–29:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.CONCUR.2022.29.

12 Uli Fahrenberg and Krzysztof Ziemianski. A myhill-nerode theorem for higher-dimensional automata. In Luís Gomes and Robert Lorenz, editors, *Application and Theory of Petri Nets and Concurrency - 44th International Conference, PETRI NETS 2023, Lisbon, Portugal, June 25-30, 2023, Proceedings*, volume 13929 of *Lecture Notes in Computer Science*, pages 167–188. Springer, 2023. doi:10.1007/978-3-031-33620-1_9.

13 Paul Gastin and Dietrich Kuske. Uniform satisfiability in PSPACE for local temporal logics over mazurkiewicz traces. *Fundam. Informaticae*, 80(1-3):169–197, 2007. doi:10.3233/FUN-2007-801-310.

14 Eric Goubault and Samuel Mimram. Formal relationships between geometrical and classical models for concurrency. In Lisbeth Fajstrup, Eric Goubault, and Martin Raussen, editors, *Proceedings of the workshop on Geometric and Topological Methods in Computer Science, GETCO 2010, Aalborg, Denmark, January 11-15, 2010*, volume 283 of *Electronic Notes in Theoretical Computer Science*, pages 77–109. Elsevier, 2010. doi:10.1016/j.entcs.2012.05.007.

15 André Joyal, Mogens Nielsen, and Glynn Winskel. Bisimulation from open maps. *Inf. Comput.*, 127(2):164–185, 1996. doi:10.1006/inco.1996.0057.

16 Johan Anthony Willem Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, Computer Science Department, University of California at Los Angeles, USA, 1968.

17 Leslie Lamport. On interprocess communication. part I: basic formalism. *Distributed Comput.*, 1(2):77–85, 1986. doi:10.1007/BF01786227.

18 Antoni W. Mazurkiewicz. Trace theory. In Wilfried Brauer, Wolfgang Reisig, and Grzegorz Rozenberg, editors, *Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986, Part II, Proceedings of an Advanced Course, Bad Honnef, Germany, 8-19 September 1986*, volume 255 of *Lecture Notes in Computer Science*, pages 279–324. Springer, 1986. doi:10.1007/3-540-17906-2_30.

19 Robert McNaughton and Seymour Papert. *Counter-Free Automata*, volume Research Monograph, 65. The M.I.T. Press, Cambridge, Mass., 1971.

20 Vaughan R. Pratt. Modeling concurrency with geometry. In David S. Wise, editor, *Conference Record of the Eighteenth Annual ACM Symposium on Principles of Programming Languages, Orlando, Florida, USA, January 21-23, 1991*, pages 311–322. ACM Press, 1991. doi:10.1145/99583.99625.

21 Cristian Prisacariu. Modal logic over higher dimensional automata. In Paul Gastin and François Laroussinie, editors, *CONCUR 2010 - Concurrency Theory, 21th International Conference, CONCUR 2010, Paris, France, August 31-September 3, 2010. Proceedings*, volume 6269 of *Lecture Notes in Computer Science*, pages 494–508. Springer, 2010. doi:10.1007/978-3-642-15375-4_34.

22 Alexander Rabinovich. A proof of kamp's theorem. *Logical Methods in Computer Science*, Volume 10, Issue 1:730, 2014. doi:10.2168/LMCS-10(1:14)2014.

23 Marcel Paul Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8(2):190–194, 1965. doi:10.1016/S0019-9958(65)90108-7.

24 P. S. Thiagarajan. A trace based extension of linear time temporal logic. In *Proceedings of the Ninth Annual Symposium on Logic in Computer Science (LICS '94), Paris, France, July 4-7, 1994*, pages 438–447. IEEE Computer Society, 1994. doi:10.1109/LICS.1994.316047.

25 P.S. Thiagarajan and I. Walukiewicz. An expressively complete linear time temporal logic for mazurkiewicz traces. *Information and Computation*, 179(2):230–249, 2002. doi:10.1006/inco.2001.2956.

26 Rob J. van Glabbeek. On the expressiveness of higher dimensional automata. *Theor. Comput. Sci.*, 356(3):265–290, 2006. doi:10.1016/j.tcs.2006.02.012.

27 Safa Zouari, Krzysztof Ziemianski, and Uli Fahrenberg. Bisimulations and logics for higher-dimensional automata. *CoRR*, abs/2402.01589, 2024. doi:10.48550/arXiv.2402.01589.

A Discussion: Why we need sub-events

In this section, we present a temporal logic that is weaker than FO, which we call **Event-based Pomset Temporal Logic** (EPTL). This is a straightforward attempt at defining a local temporal logic for pomsets, with two "Exists Next" operators for the precedence order $<_P$ and for the event order $\dashrightarrow_P$. It is similar to the local temporal logic over Mazurkiewicz traces defined in [7]. This example showcases why it is not easy to design an LTL-like logic over pomsets that is equivalent to FO, and why we had to introduce the notion of sub-events.

▶ **Definition 24** (Events-based Pomset Temporal Logic (EPTL)). *The syntax of EPTL formulas is defined by the following grammar, where $a \in \Sigma$:*

$$\varphi, \psi ::= a \mid \mathbf{s} \mid \mathbf{t} \mid \neg\varphi \mid \varphi \wedge \psi \mid \langle < \rangle \varphi \mid \langle \dashrightarrow \rangle \varphi \mid \varphi \,\mathbf{U}\, \psi$$

An EPTL formula is evaluated over a single event x of a pomset P:

- $P, x \models a$ *if* $\lambda_P(x) = a$, $P, x \models \mathbf{s}$ *if* $x \in S_P$, $P, x \models \mathbf{t}$ *if* $x \in T_P$,
- $P, x \models \neg\varphi$ *if* $P, x \not\models \varphi$, $P, x \models \varphi \wedge \psi$ *if* $P, x \models \varphi$ *and* $P, x \models \psi$,
- $P, x \models \langle < \rangle \varphi$ *if* $\exists y \in P$, y *is a direct successor of* x *for* $<_P$ *and* $P, y \models \varphi$,
- $P, x \models \langle \dashrightarrow \rangle \varphi$ *if* $\exists y \in P$, y *is a direct successor of* x *for* $\dashrightarrow_P$ *and* $P, y \models \varphi$,
- $P, x \models \varphi \,\mathbf{U}\, \psi$ *if* $\exists y >_P x$ $P, y \models \psi$ *and* $\forall z \in P$ *such that* $x \leq_P z <_P y$, $P, z \models \varphi$.

In order to define $P \models \varphi$, where φ is an EPTL formula, we need to fix a canonical $e \in P$ from which φ will be interpreted. This motivates Definition 25.

▶ **Definition 25.** *Given a non-empty pomset P, the* ***source*** *of P, denoted* $\mathrm{source}(P)$*, is the minimal event according to* $\dashrightarrow$ *of* $\{e \in P \mid \forall f \in P, f \not< e\}$.

We write $P \models \varphi$ if $P, \mathrm{source}(P) \models \varphi$. Note that the unary predicate verifying whether x is a source is FO-definable, define $\mathrm{source}(x) := \forall y.(\forall z.y \not< z) \Rightarrow x \dashrightarrow y$.

▶ **Proposition 26.** *EPTL is stricly less expressive than FO.*

Proof (Sketch). First, EPTL is at most as expressive as FO since its semantics expressed in Definition 24 can be translated into first order formulas. To show that the inclusion is strict, consider the following two families of pomsets, depicted in Figure 9:

$$P_n = \begin{bmatrix} a\bullet \\ a\bullet \end{bmatrix} \left(\begin{bmatrix} \bullet a \\ \bullet a \bullet \end{bmatrix} \begin{bmatrix} \bullet a \bullet \\ a\bullet \end{bmatrix} \right)^n \begin{bmatrix} \bullet a \\ \bullet a \end{bmatrix} \quad \text{and} \quad P'_n = \begin{bmatrix} a\bullet \\ a\bullet \end{bmatrix} \begin{bmatrix} \bullet a \\ \bullet a \bullet \end{bmatrix} \begin{bmatrix} a\bullet \\ \bullet a \bullet \end{bmatrix} \left(\begin{bmatrix} \bullet a \\ \bullet a \bullet \end{bmatrix} \begin{bmatrix} \bullet a \bullet \\ a\bullet \end{bmatrix} \right)^n \begin{bmatrix} \bullet a \\ \bullet a \end{bmatrix}$$

Figure 9 Pomsets P_2 and P'_2.

Consider the FO formula $\varphi = \exists x.\exists y.\exists z.\, \mathrm{source}(x) \wedge x \to y \wedge x \dashrightarrow^d z \dashrightarrow^d y$, where $x \to y$ and $x \dashrightarrow^d y$ denote the direct successor w.r.t. $<$ and $\dashrightarrow$, respectively. The formula φ separates the two languages, i.e., $P_n \models \varphi$ but $P'_n \not\models \varphi$ for all $n \in \mathbb{N}$. Now we must show that any EPTL formula of size t cannot distinguish the pomsets P_t and P'_t. First, atomic formulas ($a \in \Sigma$, $\mathbf{s}$ and $\mathbf{t}$) cannot distinguish between elements of P_t and P'_t, since all events are labeled by a, and the interfaces are empty. Moreover, notice that starting from $\mathrm{source}(P_t)$ (in orange in Figure 9) or $\mathrm{source}(P'_t)$ (in purple), there is a chain of length $\geq t$ for both relations $\to$ and $\dashrightarrow$. Thus, using modalities $\langle < \rangle$ and $\langle \dashrightarrow \rangle$ to distinguish $\mathrm{source}(P_t)$ from $\mathrm{source}(P'_t)$ would require to reach the last elements of the pomset, which cannot be done in less than t steps. Finally, the until modality cannot distinguish them either: this is a bit more technical but fairly standard. So we conclude that the language $\mathcal{L}(\varphi)$ is not definable in EPTL. ◀

▶ Remark 27. To separate the two languages in $\mathrm{LTL}_{\mathrm{Poms}}$, one can remark that in the second slice of the pomsets P'_n, there is an event a (in orange) which is both started and terminated. Thus, the formula $\varphi = \langle \prec_1 \rangle (\mathbf{s} \wedge \mathbf{t})$ is satisfied in all pomsets of the form P'_n, but in none of the pomsets P_n.

B Omitted proofs

B.1 Proof of Lemma 9

▶ **Lemma 9.** *Fix $i, j \in \{1, \dots, k\}$. Then, the binary relation $(x, i) \sim (y, j)$ is definable by an $\mathrm{FO}_{\Omega_{\leq k}}$ formula with two free variables x and y.*

Proof. We define a finite state automaton that scans the ST-sequence between the two positions x and y. During all the execution, the automaton keeps track the position of the event represented by the i-th position of x. Showing that this automaton is counter-free yields an $\mathrm{FO}_{\Omega_{\leq k}}$ formula for the $\sim$ relation, thus proving the lemma. Formally, given $a \in \Sigma$ and $i, j \in \{1, \dots, k\}$, we define a finite state automaton $\mathcal{A}_{i,j,a} = (\Omega_{\leq k}, Q, q_0, F, \delta)$ over the alphabet $\Omega_{\leq k}$, parameterized by i, j, a, as follows:

- The set of states Q is given by:

$$Q = \{\bot, \top, \sqcup\} \cup (\mathsf{CList}_{\leq k} \times \{1, \dots, k\})$$

  The states $\bot, \top, \sqcup$ represent the initial state, final state, and sink state, respectively. All other states are of the form (C, ℓ), where C is the list of currently active events, and the event a that we are following is the ℓ-th element in this conclist.
- The initial state is $q_0 = \bot$.
- The set of accepting states is $F = \{\top\} \cup \{(q, j) \mid q \in Q\}$ (recall that j is a parameter fixed in advance).
- The transition function δ is defined as follows, for any $P \in \Omega_{\leq k}$:

$$\begin{aligned}
\delta(\bot, P) &= (T_P, i) && \text{if } T_P[i] \text{ is a non-terminating } a \\
\delta(\bot, P) &= \top && \text{if } T_P[i] \text{ is a terminating } a \text{ and } i = j \\
\delta((S_P, \ell), P) &= (T_P, \ell') && \text{for any } \ell, \ell' \text{ when } S_P[\ell] = T_P[\ell'] \text{ and is an } a \\
\delta((S_P, j), P) &= \top && \text{if } S_P[j] \text{ is a terminating } a \\
\delta(q, P) &= \sqcup \text{ (the sink state)} && \text{otherwise}
\end{aligned}$$

  where $C[i]$ denotes is the i-th element of the conclist C.

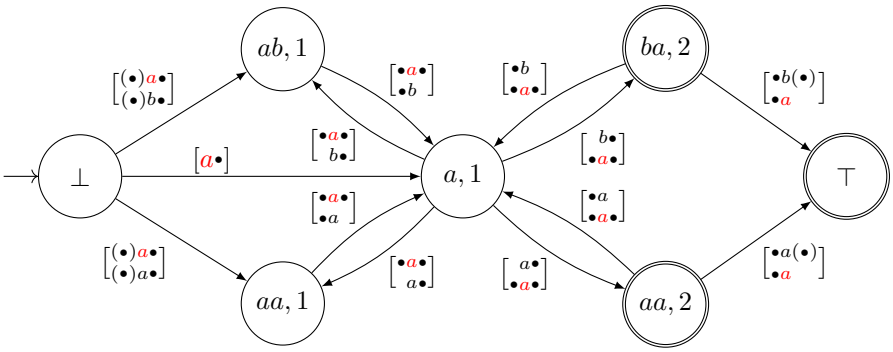


Figure 10 $\mathcal{A}_{1,2,a}$, sink state and identities not drawn.

Figure 10 shows the automaton $\mathcal{A}_{1,2,a}$, for $\Sigma = \{a, b\}$ and $k = 2$. For the sake of readability, when an interface is represented enclosed in parenthesis (e.g. $[(\bullet)a]$), we mean that both transitions exist, with and without the interface (e.g. $[a]$ and $[\bullet a]$). The event a that we are tracking is colored in red.

Note that $\mathcal{A}_{i,j,a}$ is deterministic and complete. We now show that it is counter-free. So we need to find an $n \in \mathbb{N}$ such that, for any non-empty word $w \in (\Omega_{\leq k})^*$ and any state q, we have $\delta(q, w^n) = \delta(q, w^{n+1})$. Choose $n = k + 2$. Consider an arbitrary state q and a non-empty word w. We have three cases:

1. If $q = \top$ or $q = \sqcup$, the only accessible state on a non-empty word is the sink state $\sqcup$, verifying the property.
2. If $q = (c, \ell) \in \mathsf{CList}_{\leq k} \times \{1, \dots, k\}$, let us consider $\delta((c, \ell), w)$.
   a. If $\delta((c, \ell), w) = (c, \ell)$, we indeed have $\delta((c, \ell), w^{n+1}) = \delta((c, \ell), w^n) = (c, \ell)$.
   b. If $\delta((c, \ell), w) = (c', \ell')$, with $c' \neq c$, then $\delta((c', \ell'), w) = \sqcup$ since either the starting interface or the terminating interface of w will be incompatible with c'. Thus, the property is verified.
   c. If $\delta((c, \ell), w) = (c, \ell')$, with $\ell \neq \ell'$, assume that $\ell > \ell'$ (the opposite case is similar). We claim that, as we keep iterating the word w, the index ℓ will keep decreasing strictly, until we reach either the state $\top$ or $\sqcup$ after at most k iterations. So at iteration $k + 1$, we end up in $\sqcup$, where we stay, making the property true. To prove our claim, let us write $(c, \ell_i) = \delta((c, \ell), w^i)$ the sequence of visited states. We need to show that the sequence (ℓ_i) is strictly decreasing.
   The example below illustrates what might happen, with a pomset of dimension $k = 3$. We start in state $(aaa, 3)$, so three a's are running concurrently, and we are tracking the third one (in red). When the automaton reads the word w below, the first a is terminated, and another a is started, but this new a happens to be placed after the other two according to event order. The tracked a ends up in position 2, so the state of the automaton is now $(aaa, 2)$.

$$w = \begin{bmatrix} \bullet a \\ \bullet a \bullet \\ \bullet a \bullet \end{bmatrix} \begin{bmatrix} \bullet a \bullet \\ \bullet a \bullet \\ a \bullet \end{bmatrix} \qquad \begin{aligned} \delta((aaa, 3), w) &= (aaa, 2) \\ \delta((aaa, 3), w^2) &= (aaa, 1) \end{aligned}$$

   Let us assume that for some i, $\ell_i > \ell_{i+1}$, we will show that at the next step, $\ell_{i+1} > \ell_{i+2}$. Let $P_w \in \mathrm{iiPoms}$ be the pomset generated by gluing w (the automaton ensures that w is coherent, otherwise we end up in the sink state $\sqcup$). Let $p_i, p_{i+1} \in P_w$ denote the events of P_w at position ℓ_i and ℓ_{i+1} respectively, in the starting interface S_{P_w}. (In the example above, they are the red and blue events, respectively.) Since $\ell_i > \ell_{i+1}$, we must have $p_{i+1} \dashrightarrow p_i$ according to event order. In the terminating interface T_{P_w}, both events are still active. Event p_i ends up at position ℓ_{i+1} and p_{i+1} ends up at position ℓ_{i+2}. Since these are still the same events, the event order did not change, so we still have $p_{i+1} \dashrightarrow p_i$, i.e., $\ell_{i+1} > \ell_{i+2}$ as required.
3. If $q = \bot$, we have $\delta(\bot, w) \neq \bot$ since w is non-empty and there is no incoming transition in $\bot$. So we land in one in the previous cases, with one extra step. Hence, $\delta(\bot, w^{k+3}) = \delta(\bot, w^{k+2})$.

This proves that $\mathcal{A}_{i,j,a}$ is counter-free. By Proposition 8, let $\theta_{i,j,a}$ be a closed first-order formula equivalent that recognizes the same language as $\mathcal{A}_{i,j,a}$. By $\theta_{i,j,a}^{x \to y}$, we denote the restriction of $\theta_{i,j,a}$ to the portion of the ST-sequence located between x and y, i.e. where subformulas of the form $\exists z.\varphi$ are inductively replaced by $\exists z.x \leq z \wedge z \leq y \wedge \varphi$. Then the following $\mathrm{FO}_{\Omega_{\leq k}}$ formula defines $(x, i) \sim (y, j)$:

$$(x, i) \sim (y, j) := \bigvee_{a \in \Sigma} (x < y \wedge \theta_{i,j,a}^{x \to y}) \vee (y < x \wedge \theta_{j,i,a}^{y \to x}) \qquad \blacktriangleleft$$

B.2 Proof of Proposition 18

▶ **Proposition 18.** *Let P be a pomset, and let $(x,m) \equiv (y,q)$ two equivalent sub-events of P, then for every formula φ of* $\mathrm{LTL}_{\mathrm{Poms}}$*, $P,(x,m) \models \varphi$ if and only if $P,(y,q) \models \varphi$.*

Proof. First observe that $\sim^s$ is an equivalence relation over events. Indeed, it is reflexive since $m <^s q$ implies $m \neq q$ and symmetric by construction. As for transitivity, assume that $m \sim^s q \sim^s r$. To show that $m \sim^s r$, it is enough to show $m \not<^s r$. Assume by contradiction that there is $t \in P$ s.t. $t \parallel m$, and $t < r$. Then, we prove $t \parallel q$ as follows:

- If $t < q$, then $m <^s q$ because $t \parallel m$. But this is false since $m \sim^s q$.
- If $q < t$, then $q < r$ since $t < r$. This implies that $q <^s r$, which is false by hypothesis.

Since $q \sim^s r$, $q \not<^s r$. Therefore, as $t \parallel q$, it cannot be that $t < r$. Since this was a hypothesis, we have a contradiction, and $m \sim^s r$.

We can now proceed to the main proof. It is sufficient to prove that $P,(x,m) \models \varphi$ implies $P,(y,q) \models \varphi$. First, observe that $x = y$ by definition of $\equiv$, hence we write (x,q) from now on. We proceed by induction over φ.

- $P,(x,m) \models a$ implies $\lambda(x) = a$, thus $P,(x,q) \models a$
- $P,(x,m) \models \mathbf{s}$ implies $x \sim^s m$. Since $m \sim^s q$, this yields $x \sim^s q$, which in turn becomes $P,(x,q) \models \mathbf{s}$
- $P,(x,m) \models \mathbf{t}$ implies that there is no $r \in P$ such that $x \parallel r$ and $m <^s r$. Since $m \sim^s q$, $m <^s r$ is equivalent to $q <^s r$. This in turn implies $P,(x,q) \models \mathbf{t}$.
- $P,(x,m) \models \neg\psi$ implies $P,(x,m) \not\models \psi$, which implies, by induction hypothesis, $P,(x,q) \not\models \psi$. Thus, $P,(x,q) \models \neg\psi$.
- $P,(x,m) \models \psi_1 \wedge \psi_2$ implies $P,(x,m) \models \psi_1$ and $P,(x,m) \models \psi_2$. Hence, by induction hypothesis, $P,(x,q) \models \psi_1$ and $P,(x,q) \models \psi_2$. It comes that $P,(x,q) \models \psi_1 \wedge \psi_2$.
- $P,(x,m) \models \langle \prec_1 \rangle \psi$ implies that there exists a sub-event (y,r) such that $(x,m) \prec_1 (y,r)$ and $P,(y,r) \models \psi$. Because $(x,m) \prec_1 (y,r)$, we have that $m <^s r$ and there is no $s \in P$ such that $m <^s s <^s r$. Since $m \sim^s q$, it comes that $q <^s r$ and there is no s such that $q <^s s <^s r$. Thus, $(x,q) \prec_1 (y,r)$ and $P,(x,q) \models \langle \prec_1 \rangle \psi$.
- $P,(x,m) \models \langle \dashrightarrow \rangle \psi$ implies that there is y such that $x \sim^s y$, y is a direct successor of x by $\dashrightarrow$ and $P,(y,m) \models \psi$. Since $m \sim^s q$, (y,q) is a sub-event and $P,(y,q) \models \psi$ by induction. Hence, $P,(x,q) \models \langle \dashrightarrow \rangle \psi$.
- For $\langle \dashrightarrow^{-1} \rangle \psi$, the argument is identical.
- Finally, for $P,(x,m) \models \psi_1 \mathbf{U} \psi_2$, fix (y,r) to be such that $P,(y,r) \models \psi_2$ and $(x,m) \preccurlyeq (y,r)$ and any (z,s) such that $(x,m) \preccurlyeq (z,s) \prec (y,r)$ verifies $P,(x,s) \models \psi_1$. Since $m \sim^s q$, we have that $(x,m) \preccurlyeq (y,r)$ implies $(x,q) \preccurlyeq (y,r)$, and the same goes for the intermediary (z,s). Hence, $P,(x,q) \models \psi_1 \mathbf{U} \psi_2$. ◀