An expressively complete local past propositional dynamic logic over Mazurkiewicz traces and its applications

Bharat Adsul, Paul Gastin, Shantanu Kulkarni, Pascal Weil

An expressively complete local past propositional dynamic logic over Mazurkiewicz traces and its applications

Bharat Adsul[*1], Paul Gastin[†2,3], Shantanu Kulkarni[‡1], and Pascal Weil[§3]

[1]IIT Bombay, India
[2]Université Paris-Saclay, ENS Paris-Saclay, CNRS, LMF, 91190 Gif-sur-Yvette, France
[3]ReLaX, CNRS, IRL 2000, Siruseri, India

October 28, 2024

Abstract

We propose a local, past-oriented fragment of propositional dynamic logic to reason about concurrent scenarios modelled as Mazurkiewicz traces, and prove it to be expressively complete with respect to regular trace languages. Because of locality, specifications in this logic are efficiently translated into asynchronous automata, in a way that reflects the structure of formulas. In particular, we obtain a new proof of Zielonka's fundamental theorem and we prove that any regular trace language can be implemented by a cascade product of localized asynchronous automata, which essentially operate on a single process.

These results refine earlier results by Adsul et al. which involved a larger fragment of past propositional dynamic logic and used Mukund and Sohoni's gossip automaton. Our new results avoid using this automaton, or Zielonka's timestamping mechanism and, in particular, they show how to implement a gossip automaton as a cascade product.

Keywords — Mazurkiewicz traces, propositional dynamic logic, regular trace languages, expressive completeness, asynchronous automata, cascade product, Krohn Rhodes theorem, Zielonka's theorem, Gossip automaton

1 Introduction

Mazurkiewicz traces [16] (just called traces in this paper) are a well-established model of concurrent behaviours. Each specifies a partial order on events of a scenario in which a fixed set of processes, organized along a distributed architecture, interact with each other via shared actions. Traces have been extensively studied, with a particular attention to the class of regular trace languages [9, 8]. The significance of that class arises from its expressivity, which is equivalent to Monadic-Second-Order (MSO) logic definability [20, 10, 13], and from the fact that regular trace languages can be implemented by asynchronous automata (also known as Zielonka automata) [21]. Asynchronous automata yield implementations which fully exploit the distributed architecture of the processes.

[*]adsul@cse.iitb.ac.in
[†]paul.gastin@ens-paris-saclay.fr
[‡]shantanu3637@gmail.com
[§]pascal.weil@cnrs.fr

Our main result establishes that a local, past-oriented fragment of Propositional Dynamic Logic (PDL) for traces, called LocPastPDL, is expressively complete with respect to regular trace languages. Let us first comment on the significance of this logic.

PDL was introduced in [11] to reason about arbitrary programs. LDL, a purely future-oriented version of PDL, was developed in [7] to reason about properties of finite words, and was shown to be expressively complete with respect to regular word languages. A more recent work [12] studied a purely past-oriented version of LDL and showed that it is expressively complete and admits a singly exponential translation into deterministic finite-state automata which is an exponential improvement over LDL. PDL has also been applied to message sequence charts (MSC) and related systems in [6, 17]. More recently, a star-free version of PDL interpreted over MSC was shown to be expressively complete with respect to first-order logic [5]. These prior works have successfully demonstrated that PDL is a suitable logical formalism for writing specifications in a variety of contexts. On the one hand, it naturally extends linear temporal logics by permitting richer path formulas to express regular specification patterns easily. On the other hand, it supports efficient translations into automata-theoretic models, which are central to the resolution of many verification or synthesis problems.

The logic LocPastPDL admits three types of local and past-oriented formulas: trace formulas, event formulas and path formulas. A trace formula is a boolean combination of basic trace formulas of the form $\mathsf{EM}_i\,\varphi$ asserting that the last event of process i satisfies the event formula φ. It is local and past in the sense that its satisfaction depends only on the past information available to process i.

The *event formulas* reason about the causal past of events. They are boolean combinations of (a) atomic checks of letters (or actions), (b) formulas of the form $\langle\pi\rangle$ claiming the existence of a path π starting at the current event. The *path formula* π, necessarily *localized at a process*, allows to march along the sequence of past events in which that process participates, checking for regular patterns interspersed with local tests of other event formulas.

A typical event formula contains inside it several localized path formulas which in turn contain other event formulas through embedded tests. This natural hierarchical structure of LocPastPDL formulas renders them easier to design and to understand.

It turns out that our logic LocPastPDL is essentially a fragment of $\mathsf{LocPastPDL}[\mathsf{Y},\mathsf{L}]$ which was introduced in [3] and shown to be expressively complete. The logic $\mathsf{LocPastPDL}[\mathsf{Y},\mathsf{L}]$ also supports *additional constant event/trace formulas* to compare the leading events for each process. The expressive completeness result for $\mathsf{LocPastPDL}[\mathsf{Y},\mathsf{L}]$ crucially exploits the presence of these additional constant formulas. Keeping track of the ordering information between leading process events is a fundamental and very difficult problem in concurrency theory. Its solution as an asynchronous automaton, called a gossip automaton [21, 18], is a key distributed time-stamping mechanism and a crucial ingredient in many asynchronous automata-theoretic constructions [21, 14].

The central result of this work is that LocPastPDL is itself expressively complete. This is done by eliminating constant formulas in $\mathsf{LocPastPDL}[\mathsf{Y},\mathsf{L}]$ using entirely new techniques.

Another important contribution is an efficient translation of LocPastPDL formulas into local cascade products of localized asynchronous automata (this, without any claim about efficiency, was already implicit in [3]). In asynchronous automata, each process runs a finite local-state device and these devices synchronize with each other on shared actions. As in [18, 1, 2], one can use asynchronous automata to also locally compute relabelling functions on input traces, similar in spirit to the sequential letter-to-letter transducers on words.

The *local cascade product* of asynchronous automata (or transducers) from [4, 1, 2] is a natural generalization of cascade product in the sequential setting and, like in the word case [19], it corresponds to compositions of related relabelling functions. Note that, in a cascade

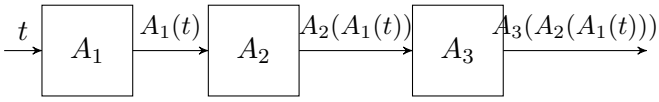

Figure 1: Cascade Product

product, the information flow is unidirectional and hierarchical.

The natural hierarchical structure of LocPastPDL formulas makes it possible to translate them in a modular way into a local cascade product of asynchronous automata. The locality of LocPastPDL is also an asset: each level in the cascade product implementing a formula is a localized asynchronous automaton, that is, one where *all actions* take place on a *single* process, and the other processes do nothing at all. In other words, these automata are essentially classical automata, operating on a single process, and this is much simpler to understand and verify. The past orientation of LocPastPDL makes our implementation deterministic. Finally, a LocPastPDL formula can be computed by a local cascade product of localized asynchronous transducers/automata *whose global-state space is singly exponential in the size of the formula.*

The construction described above, coupled with the expressive completeness of LocPastPDL has some striking applications. It easily provides a new proof of Zielonka's theorem. Another important application is a distributed Krohn-Rhodes theorem. The classical Krohn-Rhodes theorem [15] implies that every regular word language can be accepted by a *cascade product* of simple automata, namely, two-state reset automata and permutation automata. Our distributed version states that every regular trace language can be accepted by a local cascade product of localized two-state asynchronous reset automata and localized asynchronous permutation automata.

Additionally, we show that a gossip automaton/transducer can be implemented as a local cascade product of localized asynchronous automata/transducers. It is important to note that the previously known implementations of gossip as an asynchronous transducer were intrinsically non-local and non-hierarchical. They rely on a delicate reuse of (boundedly many) time-stamps. A process assigning the time-stamps also needs to collect the information about which of its earlier time-stamps are in use by other processes. Given the 'circular/self-referential' nature of this information flow, it appeared rather counter-intuitive that a gossip transducer could be implemented as a local cascade product (which is unidirectional in nature) of localized asynchronous automata (in which only one process is active).

We prove that LocPastPDL is expressively complete by showing that the constant formulas in LocPastPDL[Y, L] can be defined in LocPastPDL itself. To do so, we develop an apparatus of special deterministic path formulas, called sd-path formulas, which suffice to address the leading events of processes in a bounded fashion. Then we reduce the problem of checking the causal ordering of these leading events to the problem of checking the causal ordering of the events addressed by sd-path formulas. We next show that the causal ordering of events addressed by sd-path formulas can be reduced to equality of such events, for possibly different and longer sd-path formulas. Finally, the equality formulas are constructed with the help of *separating* formulas. Given a non-trivial sd-path formula π, we construct a finite set $\Xi(\pi)$ of separating formulas which separate every event e from $\pi(e)$, the event referred from e via address π. The construction of these formulas, while intricate, is entirely novel and constitutes the technical core of this work.

We now describe the organization of the paper. In Section 2, we present the logics of interest, in particular LocPastPDL, after setting up basic preliminary notation. Section 3 carries out the sophisticated task of expressing constant formulas in LocPastPDL. The next Section 4 develops the efficient translation into local cascade product of asynchronous devices and provides

the aforementioned applications of our expressive completeness result. We finally conclude in Section 5.

2 Local Past Propositional Dynamic Logic

2.1 Basic notions about traces

Let $\mathscr{P}$ be a finite, non-empty set of processses. A distributed alphabet over $\mathscr{P}$ is a family $\tilde{\Sigma} = \{\Sigma_i\}_{i\in\mathscr{P}}$ of finite non-empty sets which may have non-empty intersection. The elements of Σ_i are called the *letters*, or *actions*, *that process i participates in*. Let $\Sigma = \bigcup_{i\in\mathscr{P}} \Sigma_i$. Letters $a, b \in \Sigma$ are said to be *dependent* if some process participates in both of them ($\exists i \in \mathscr{P}, a, b \in \Sigma_i$); otherwise they are *independent*.

A poset is a pair $(E, \leq)$ where E is a set and $\leq$ is a partial order on E. For $e, e' \in E$, e' is an *immediate successor* of e (denoted by $e <\cdot e'$) if $e < e'$ and there is no $g \in E$ such that $e < g < e'$. A (*Mazurkiewicz*) *trace* t over $\tilde{\Sigma}$ is a triple $t = (E, \leq, \lambda)$, where $(E, \leq)$ is a finite poset, whose elements are referred to as *events*, and $\lambda\colon E \to \Sigma$ is a labelling function which assigns a letter from Σ to each event, such that

1. for $e, e' \in E$, if $e <\cdot e'$, then $\lambda(e)$ and $\lambda(e')$ are dependent;
2. for $e, e' \in E$, if $\lambda(e)$ and $\lambda(e')$ are dependent, then either $e \leq e'$ or $e' \leq e$.

Let $t = (E, \leq, \lambda)$ be a trace over $\tilde{\Sigma}$. Let $i, j \in \mathscr{P}$ be processes. Events in which process i participates (that is: whose label is in Σ_i), are called *i-events* and the set of such i-events is denoted by E_i. E_i is clearly totally ordered by $\leq$. We define the *location* of an event $e \in E$ to be the set $\mathsf{loc}(e)$ of processes that participate in e, that is, $\mathsf{loc}(e) = \{i \in \mathscr{P} \mid \lambda(e) \in \Sigma_i\}$. We slightly abuse notation to define *location* for letters as well: $\mathsf{loc}(a) = \{i \in \mathscr{P} \mid a \in \Sigma_i\}$. The set of events in the past of e is written $\downarrow e = \{f \in E \mid f \leq e\}$ and the set of events in the strict past of e is written $\Downarrow e = \{f \in E \mid f < e\}$. If $E_i \cap \Downarrow e \neq \emptyset$, we denote by $\mathsf{Y}_i(e)$ the maximum i-event in the strict past of e. We denote by $\mathsf{Y}_{i,j}(e)$ the event $\mathsf{Y}_j(\mathsf{Y}_i(e))$, if it exists. If $E_i \neq \emptyset$, we denote by $\mathsf{L}_i(t)$ the maximum (that is, last) i-event. If in addition $E_j \cap \downarrow \mathsf{L}_i(t) \neq \emptyset$, we denote by $\mathsf{L}_{i,j}(t)$ the maximum j-event in the past of the maximum i-event.

A set of traces over $\tilde{\Sigma}$ is called a *trace language*. Regular trace languages admit several characterizations, in particular in terms of MSO logic [20] or of asynchronous (or Zielonka) automata [21].

Example 2.1. Fig. 2 represents a trace with 11 events over 4 processes. The process set is

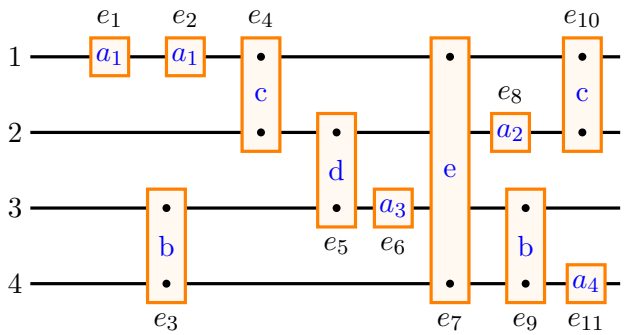

Figure 2: Process trace diagram with labelled events.

$\mathscr{P} = \{1, 2, 3, 4\}$ and the event set is $E = \{e_1, \ldots, e_{11}\}$. Each process is indicated by a horizontal

line and time flows rightward. Each event is represented by a vertical box. Events are either local to a process, or they are non-local. Dots in the non-local events, indicate the participating processes. Each event is labelled by a letter in $\Sigma = \{a_1, a_2, a_3, a_4, b, c, d, e\}$. The distributed alphabet $\tilde{\Sigma} = \{\Sigma_i\}_{i \in \mathscr{P}}$, is clear from the diagram, for example $\Sigma_4 = \{b, e, a_4\}$. It is easy to infer causality relation ($\leq$) looking at the diagram, for example, $e_4 < e_5 < e_6 < e_9$. We can also infer which events are concurrent (not related by $\leq$): for example, event e_7 is concurrent to events e_5, e_6, e_8, while the pairs of events e_5, e_6 and e_5, e_8 are causally ordered by process 3 and process 2, respectively.

Taking a look at the maximum events of each process, we observe that $\mathsf{L}_1(t) = e_{10}$, $\mathsf{L}_2(t) = e_{10}$, $\mathsf{L}_3(t) = e_9$ and $\mathsf{L}_4(t) = e_{11}$. The maximum event for a process need not be a maximal event in the trace: for instance $\mathsf{L}_3(t) = e_9 < e_{11} = \mathsf{L}_4(t)$. By definition, if $\mathsf{L}_i(t)$ is a j-event, then $\mathsf{L}_{i,j}(t) = \mathsf{L}_i(t)$, as in the case of $\mathsf{L}_{1,1}(t) = \mathsf{L}_{1,2}(t) = \mathsf{L}_1(t) = e_{10}$. If $\mathsf{L}_i(t)$ is not a j-event, then $\mathsf{L}_{i,j}(t) = \mathsf{Y}_j(\mathsf{L}_i(t)) < \mathsf{L}_i(t)$: for instance $\mathsf{L}_{1,3}(t) = \mathsf{Y}_3(\mathsf{L}_1(t)) = e_5$ and $\mathsf{L}_{1,4}(t) = \mathsf{Y}_4(\mathsf{L}_1(t)) = e_7$.

Events of the form $\mathsf{Y}_i(e)$ and $\mathsf{Y}_{i,j}(e)$ are also directly visible, e.g. $\mathsf{Y}_4(e_6) = e_3$, $\mathsf{Y}_3(e_8) = e_5$ and $\mathsf{Y}_{3,4}(e_8) = \mathsf{Y}_4(\mathsf{Y}_3(e_8)) = \mathsf{Y}_4(e_5) = e_3$. Note that $\mathsf{Y}_i(e)$ may not exist for every event, for instance $\mathsf{Y}_4(e_4)$ does not exist. □

2.2 Syntax and Semantics of LocPastPDL

We first introduce the *past propositional dynamic logic* PastPDL with the following syntax:

$$\begin{aligned}
\Phi &::= \mathsf{EM}_i\, \varphi \mid \Phi \vee \Phi \mid \neg\Phi \\
\varphi &::= a \mid \varphi \vee \varphi \mid \neg\varphi \mid \langle\pi\rangle\varphi \\
\pi &::= \leftarrow_i \mid \varphi? \mid \pi + \pi \mid \pi \cdot \pi \mid \pi^*
\end{aligned}$$

Trace formulas (of the form Φ) are evaluated over traces, hence they define trace languages. *Event formulas* (of the form φ) are evaluated at an event in a given trace and *path formulas* (of the form π) are evaluated at a pair of events in a given trace.

The semantics of PastPDL is as follows (ommitting the classical boolean connectives): For a trace $t = (E, \leq, \lambda)$, events $e, f \in E$ and process $i \in \mathscr{P}$, we let

$t \models \mathsf{EM}_i\, \varphi$	if $E_i \neq \emptyset$, and $t, \mathsf{L}_i(t) \models \varphi$,
$t, e \models a$	if $\lambda(e) = a$,
$t, e \models \langle\pi\rangle\varphi$	if there exists an event f such that
	$t, e, f \models \pi$ and $t, f \models \varphi$,
$t, e, f \models \leftarrow_i$	if e and f are i-events and
	e is the immediate successor of f on process i,
$t, e, f \models \varphi?$	if $e = f$ and $t, e \models \varphi$,
$t, e, f \models \pi_1 + \pi_2$	if $t, e, f \models \pi_1$ or $t, e, f \models \pi_2$
$t, e, f \models \pi_1 \cdot \pi_2$	if there exists an event g such that
	$t, e, g \models \pi_1$ and $t, g, f \models \pi_2$
$t, e, f \models \pi^*$	if there exist events $e = e_0, e_1, \ldots, e_n = f$
	$(n \geq 0)$ and $t, e_i, e_{i+1} \models \pi$ for each $0 \leq i < n$.

We observe that path formulas π can be seen as regular expressions over the (infinite) alphabet consisting of the left moves $\leftarrow_i$ ($i \in \mathscr{P}$) and the test formulas of the form $\varphi?$. The left moves in this regular expression are called the *top-level moves* of π. We say that π is *i-local* for

some process $i \in \mathscr{P}$ if all its top-level moves are of the form $\leftarrow_i$.[1] We then define LocPastPDL to be the *local* fragment of PastPDL where every path formula is local — that is, i-local for some i (see [3, Section 3]).

Example 2.2. The trace in Fig. 2 satisfies the following trace formula in LocPastPDL:

$$\underbrace{\mathsf{EM}_1(\langle \leftarrow_1 \cdot (d \vee \langle \leftarrow_4 \rangle \top)? \cdot \leftarrow_1 \rangle c)}_{\varphi_1} \vee \underbrace{\mathsf{EM}_3(\langle \leftarrow_4 \rangle b)}_{\varphi_2}.$$

φ_2 evaluates to $\bot$ since if we move one event in the past on the process 4 line from event $\mathsf{L}_3(t) = e_9$, we encounter event e_7 which is not labelled b. Now we examine φ_1. If we move one event in the past on the process 1 line, from event $\mathsf{L}_1(t) = e_{10}$, we are at event e_7. At e_7 we check that backward movement is possible on process 4 line, since e_3 exists. Hence the embedded test formula succeeds, and we move back again on process 1 line from event e_7, to see that we reach event e_4, which is labelled c. This shows that φ_1 evaluates to $\top$ which in turn means the whole formula evaluates to $\top$. □

The main result of this paper (Theorem 3.1 below) is the expresssive completeness of LocPastPDL with respect to regular trace languages. The proof uses the expressive completeness [3] of another variant of PastPDL. First we consider the variant PastPDL[Y, L], built on PastPDL using some additional constants, defined by the following syntax and semantics.

$$\begin{aligned}
\Phi &::= \mathsf{EM}\,\varphi \mid \mathsf{L}_i \le \mathsf{L}_j \mid \mathsf{L}_{i,j} \le \mathsf{L}_k \mid \Phi \vee \Phi \mid \neg\Phi \\
\varphi &::= a \mid \mathsf{Y}_i \le \mathsf{Y}_j \mid \mathsf{Y}_{i,j} \le \mathsf{Y}_k \mid \varphi \vee \varphi \mid \neg\varphi \mid \langle\pi\rangle \\
\pi &::= \leftarrow_i \mid \varphi? \mid \pi + \pi \mid \pi \cdot \pi \mid \pi^*
\end{aligned}$$

$t \models \mathsf{EM}\,\varphi$	if $t, e \models \varphi$ for some maximal event e in t,
$t \models \mathsf{L}_i \le \mathsf{L}_j$	if $\mathsf{L}_i(t), \mathsf{L}_j(t)$ exist and $\mathsf{L}_i(t) \le \mathsf{L}_j(t)$,
$t \models \mathsf{L}_{i,j} \le \mathsf{L}_k$	if $\mathsf{L}_{i,j}(t), \mathsf{L}_k(t)$ exist and $\mathsf{L}_{i,j}(t) \le \mathsf{L}_k(t)$.
$t, e \models \mathsf{Y}_i \le \mathsf{Y}_j$	if $\mathsf{Y}_i(e), Y_j(e)$ exist and $\mathsf{Y}_i(e) \le \mathsf{Y}_j(e)$
$t, e \models \mathsf{Y}_{i,j} \le \mathsf{Y}_k$	if $\mathsf{Y}_{i,j}(e), \mathsf{Y}_k(e)$ exist and $\mathsf{Y}_{i,j}(e) \le \mathsf{Y}_k(e)$
$t, e \models \langle\pi\rangle$	if there exists an event f such that $t, e, f \models \pi$

Finally, we let LocPastPDL[Y, L] be the local fragment of PastPDL[Y, L]. Notice that, apart from the modalities $\mathsf{EM}_i\,\varphi$ and $\langle\pi\rangle\varphi$, LocPastPDL is a fragment of LocPastPDL[Y, L].

Example 2.3. Consider again the trace t from Fig. 2. It satisfies the following trace formulas in LocPastPDL[Y, L]:

- $\mathsf{L}_3 \le \mathsf{L}_4$ since $e_9 \le e_{11}$
- $\neg(\mathsf{L}_2 \le \mathsf{L}_3)$ since e_{10} is concurrent to e_9
- $\mathsf{L}_{2,3} \le \mathsf{L}_4$ since $\mathsf{L}_{2,3}(t) = \mathsf{Y}_3(\mathsf{L}_2(t)) = \mathsf{Y}_3(e_{10}) = e_5$ which is below $\mathsf{L}_4(t) = e_{11}$
- $\mathsf{EM}(\langle\leftarrow_4\rangle)$ since there is a maximal event e_{11} from which a back move is possible on process 4.

[1] Note that left moves $\leftarrow_j$ ($j \neq i$) may be used in the full description of an i-local path formula π, if they occur in other path formulas which are used in event formulas φ that are tested with $\varphi?$ in the regular expression defining π.

The trace t does not satisfy the formula $\mathsf{EM}\, b$ as there is no maximal event labelled b (the only maximal events in t are e_{10} and e_{11}). The event e_8 in trace t, satisfies the following event formulas in $\mathsf{LocPastPDL}[\mathsf{Y}, \mathsf{L}]$:

- $\mathsf{Y}_4 \leq \mathsf{Y}_3$ since $\mathsf{Y}_4(e_8) = e_3 < e_5 = \mathsf{Y}_3(e_8)$
- $\mathsf{Y}_{2,3} \leq \mathsf{Y}_4$ since $\mathsf{Y}_{2,3}(e_8) = e_3 = \mathsf{Y}_4(e_8)$.

□

3 Expressive Completeness of LocPastPDL

The main result of this section is the following.

Theorem 3.1. *A trace language is regular if and only if it can be defined by a* $\mathsf{LocPastPDL}$ *sentence.*

The proof relies in part on the following statement [3].

Theorem 3.2. *A trace language is regular if and only if it can be defined by a sentence in* $\mathsf{LocPastPDL}[\mathsf{Y}, \mathsf{L}]$.

Proof of Theorem 3.1. First, as in [3, Proposition 1], it is easy to see that for all $\mathsf{LocPastPDL}$ sentence Φ we can construct an equivalent MSO sentence $\overline{\Phi}$. Hence, $\mathsf{LocPastPDL}$ sentences define regular trace languages. For the converse, by Theorem 3.2, we only need to prove that the additional formulas in $\mathsf{LocPastPDL}[\mathsf{Y}, \mathsf{L}]$ are definable in $\mathsf{LocPastPDL}$. First observe that the modality $\mathsf{EM}\, \varphi$ is equivalent to

$$\bigvee_{i \in \mathscr{P}} \left(\mathsf{EM}_i\, \varphi \wedge \neg \left(\bigvee_{j \in \mathscr{P}} \mathsf{L}_i < \mathsf{L}_j \right) \right),$$

where $\mathsf{L}_i < \mathsf{L}_j = (\mathsf{L}_i \leq \mathsf{L}_j) \wedge \neg(\mathsf{L}_j \leq \mathsf{L}_i)$. Similarly, formula $\langle \pi \rangle$ is equivalent to formula $\langle \pi \rangle \top$. As a result, we only need to show that the constant formulas of $\mathsf{LocPastPDL}[\mathsf{Y}, \mathsf{L}]$ can be expressed in $\mathsf{LocPastPDL}$, and this is the objective of the rest of this section. The constant event formulas $\mathsf{Y}_i \leq \mathsf{Y}_j$ and $\mathsf{Y}_{i,j} \leq \mathsf{Y}_k$ are dealt with in Theorem 3.8, and the constant trace formulas $\mathsf{L}_i \leq \mathsf{L}_j$ and $\mathsf{L}_{i,j} \leq \mathsf{L}_k$ are dealt with in Theorem 3.18. □

Towards completing the proof and establishing Theorems 3.8 and 3.18, we introduce the class of *sd-path formulas*. These are particular path formulas which uniquely identify certain past events, such as those of the form $\mathsf{Y}_i(e)$ (Proposition 3.6). The constants $\mathsf{Y}_i \leq \mathsf{Y}_j$ and $\mathsf{Y}_{i,j} \leq \mathsf{Y}_k$ can then be expressed by formulas involving inequalities of sd-path formulas (Corollaries 3.10 and 3.11). These inequalities are then expressed by formulas involving equalities of sd-path formulas (Section 3.2.1), and finally by $\mathsf{LocPastPDL}$-formulas (Section 3.2.2).

The last step uses what we call *separating* $\mathsf{LocPastPDL}$ event formulas. The construction of these formulas (Section 3.2.3), while technical, is entirely novel and it replaces the use of timestamping (as in Zielonka's theorem [21]) or the gossip automaton (in Mukund and Sohoni's work [18]).

The constants $\mathsf{L}_i \leq \mathsf{L}_j$ and $\mathsf{L}_{i,j} \leq \mathsf{L}_k$ are expressed as $\mathsf{LocPastPDL}$ trace formulas in a similar fashion, see Section 3.3.

If $\mathscr{P}$ is a singleton, $\mathscr{P} = \{\ell\}$, then $\mathsf{Y}_\ell \leq \mathsf{Y}_\ell$ is equivalent to $\langle \leftarrow_\ell \rangle \top$ and $\mathsf{Y}_{\ell,\ell} \leq \mathsf{Y}_\ell$ is equivalent to $\langle \leftarrow_\ell \cdot \leftarrow_\ell \rangle \top$. Further both $\mathsf{L}_\ell \leq \mathsf{L}_\ell$ and $\mathsf{L}_{\ell,\ell} \leq \mathsf{L}_\ell$ are equivalent to $\mathsf{EM}_\ell\, \top$. Therefore $\mathsf{LocPastPDL}$ and $\mathsf{LocPastPDL}[\mathsf{Y}, \mathsf{L}]$ are clearly equally expressive. In the remainder of this section, we assume that $|\mathscr{P}| > 1$.

3.1 Deterministic Path Formulas and their Properties

Let us first set some notation and definitions. A disjunction $\bigvee_{a\in\Gamma} a$ of letters from some $\Gamma \subseteq \Sigma$ is called an *atomic event formula*. Note that Boolean combinations of atomic event formulas are (equivalent to) atomic event formulas. In particular, $\top$ and $\bot$ are atomic event formulas: $\top = \bigvee_{a\in\Sigma} a$ and $\bot = \bigvee_{a\in\emptyset} a$. Also, if $i \in \mathscr{P}$, then $\mathsf{on}_i = \bigvee_{a\in\Sigma_i} a$ is an atomic event formula, which tests whether an event is on process i.

If φ is an event formula, we let $\mathsf{prev}_i(\varphi)$ be the path formula $\leftarrow_i \cdot (\neg\varphi? \cdot \leftarrow_i)^* \cdot \varphi?$. Then $t, e, f \models \mathsf{prev}_i(\varphi)$ if e is an i-event and f is the maximum i-event in the strict past of e which satisfies φ. For instance, $\mathsf{prev}_i(\mathsf{on}_i)$ is equivalent to $\leftarrow_i$.

Finally, we say that a path formula π is *deterministic* if, for each trace t and each event e in t, there exists at most one event f in t such that $t, e, f \models \pi$. We write $\pi(e) = f$ when such an event f exists. It is easily verified that any path formula of the form $\varphi?$, $\leftarrow_i$ or $\mathsf{prev}_i(\varphi)$ is deterministic, and that, if π and π' are deterministic, then so is $\pi \cdot \pi'$.

We now introduce the class of *simple deterministic* path formulas (sd-path formulas), whose syntax is the following:

$$\pi ::= \varphi? \mid \mathsf{prev}_i(\varphi) \mid \pi \cdot \pi,$$

where φ is an atomic event formula and $i \in \mathscr{P}$. By definition, an sd-path formula π can be seen as a word on the (finite) alphabet consisting of tests $\varphi?$ and local path formulas $\mathsf{prev}_i(\varphi)$, where φ is atomic. We then define the *length* of π to be the number $\|\pi\|$ of "letters" of the form $\mathsf{prev}_i(\varphi)$ in π. More formally, we let $\|\varphi?\| = 0$, $\|\mathsf{prev}_i(\varphi)\| = 1$ and $\|\pi \cdot \pi'\| = \|\pi\| + \|\pi'\|$.

We record several important properties of sd-path formulas in the following lemma.

Lemma 3.3. *Let π be an sd-path formula. Then*

1. *π is deterministic.*
2. *π is* monotone*: for each trace t and for all events e, e' in t, if $e \leq e'$ and both $\pi(e)$ and $\pi(e')$ exist, then $\pi(e) \leq \pi(e')$.*
3. *The event formula $\langle\pi\rangle$ is equivalent to a* LocPastPDL *formula.*

Proof. The first two statements are easily proved by structural induction on π. The last statement is proved using the facts that $\langle\pi\rangle$ is equivalent to $\langle\pi\rangle\top?$, that the sd-path formulas $\varphi?$ and $\mathsf{prev}_i(\varphi)$ are local, and that $\langle\pi \cdot \pi'\rangle = \langle\pi\rangle\langle\pi'\rangle = \langle\pi \cdot \langle\pi'\rangle?\rangle$. □

Example 3.4. Note that sd-path formulas used to address past events are not necessarily unique. Referring to Fig. 2, it is easy to verify the following examples.

$$\begin{aligned}
\mathsf{Y}_1(e_8) &= (\mathsf{prev}_2(\top) \cdot \mathsf{prev}_2(\top) \cdot \mathsf{on}_1?)(e_8) = e_4\\
\mathsf{Y}_1(e_8) &= (\mathsf{prev}_2(\mathsf{on}_1))(e_8) = e_4\\
\mathsf{Y}_2(e_8) &= (\mathsf{prev}_2(\top) \cdot \mathsf{on}_2?)(e_8) = e_5\\
e_1 &= (\mathsf{prev}_1(\top) \cdot \mathsf{prev}_1(\top))(e_4)\\
\mathsf{Y}_3(e_8) &= (\mathsf{prev}_2(\top) \cdot \mathsf{on}_3?)(e_8) = e_5\\
\mathsf{Y}_4(e_8) &= (\mathsf{prev}_2(\mathsf{on}_3) \cdot \mathsf{prev}_3(\mathsf{on}_4))(e_8) = e_3
\end{aligned}$$

□

We say that a sd-path formula π is in *standardized form* if it is of the form

$$\varphi_0'? \cdot \mathsf{prev}_{i_1}(\varphi_1) \cdot \varphi_1'? \cdot \cdots \cdot \varphi_{n-1}'? \cdot \mathsf{prev}_{i_n}(\varphi_n) \cdot \varphi_n'?.$$

Since the path formulas $\mathsf{prev}_{i_1}(\varphi_1) \cdot \mathsf{prev}_{i_2}(\varphi_2)$ and $\varphi? \cdot \psi?$ are equivalent to $\mathsf{prev}_{i_1}(\varphi_1) \cdot \top? \cdot \mathsf{prev}_{i_2}(\varphi_2)$ and $(\varphi \wedge \psi)?$, respectively, we see that every sd-path formula is equivalent to one in standardized form, of the same length. The following easy lemma will be useful in the sequel.

Lemma 3.5. *For any $n \geq 0$, there are only finitely many logically distinct sd-path formulas of length n.*

Proof. Let π be an sd-path formula of length n. We may assume that π is in standardized form:

$$\pi = \varphi'_0? \cdot \mathsf{prev}_{i_1}(\varphi_1) \cdot \varphi'_1? \cdot \cdots \cdot \varphi'_{n-1}? \cdot \mathsf{prev}_{i_n}(\varphi_n) \cdot \varphi'_n?.$$

The i_j range over the finite set $\mathscr{P}$. And the φ_i and φ'_i are atomic event formulas, and hence range over a finite set (the power set of Σ). The result follows directly. □

The link between the constants $\mathsf{Y}_i \leq \mathsf{Y}_j$, $\mathsf{Y}_{i,j} \leq \mathsf{Y}_k$ and the sd-path formulas is given by Proposition 3.6.

Proposition 3.6. *Let t be a trace t, e an event and i a process such that $\mathsf{Y}_i(e)$ exists. Then $\mathsf{Y}_i(e) = (\pi \cdot \mathsf{on}_i?)(e)$ for some sd-path formula π satisfying $1 \leq \|\pi\| < |\mathscr{P}|$.*

Proof. Suppose that $f = \mathsf{Y}_i(e)$ exists. By definition of Y_i, we have $f < e$. We construct sequences of m events $f = e_1 < e_2 < e_3 < \cdots < e_m < e$ and m *distinct* processes $i = i_1, i_2, i_3 \ldots, i_m$ such that

1. For $1 \leq j \leq m$, $\mathsf{Y}_{i_j}(e) = e_j$.
2. For $1 < j \leq m$, e_{j-1} and e_j are i_j-events, and $\mathsf{loc}(e_{j-1}) \cap \mathsf{loc}(e) = \emptyset$.
3. There is a process which participates in both e_m and e, that is, $\mathsf{loc}(e_m) \cap \mathsf{loc}(e) \neq \emptyset$.

We begin the construction by letting $e_1 = f$ and $i_1 = i$. In particular, we have $Y_{i_1}(e) = e_1$.

Suppose that we have constructed a length n sequence of events $e_1 < e_2 < e_3 < \cdots < e_n < e$ and a length n sequence of distinct processes $i_1, i_2, i_3 \ldots, i_n$ such that (a) for $1 \leq j \leq n$, $\mathsf{Y}_{i_j}(e) = e_j$ and (b) for $1 < j \leq n$, e_{j-1} and e_j are i_j-events, and $\mathsf{loc}(e_{j-1}) \cap \mathsf{loc}(e) = \emptyset$. If $\mathsf{loc}(e_n) \cap \mathsf{loc}(e) \neq \emptyset$, we let $m = n$ and we are done.

If instead $\mathsf{loc}(e_n) \cap \mathsf{loc}(e) = \emptyset$, we extend these sequences as follows. We have $i_n \notin \mathsf{loc}(e)$, since $i_n \in \mathsf{loc}(e_n)$. Since $e_n < e$, there exists e' such that $e_n <\cdot\, e' \leq e$. As an immediate successor of e_n, e' satisfies $\mathsf{loc}(e_n) \cap \mathsf{loc}(e') \neq \emptyset$ and, as a result, we have $e' < e$. Choose i_{n+1} to be any process in $\mathsf{loc}(e_n) \cap \mathsf{loc}(e')$. As e' is an i_{n+1}-event strictly below e, $\mathsf{Y}_{i_{n+1}}(e)$ exists. We now set $e_{n+1} = \mathsf{Y}_{i_{n+1}}(e)$. Clearly, both e_n and e_{n+1} are i_{n+1}-events and $e_n < e' \leq e_{n+1} < e$. We now argue that, for any $1 \leq j \leq n$, $i_{n+1} \neq i_j$; indeed, if $i_{n+1} = i_j$, then $e_{n+1} = \mathsf{Y}_{i_{n+1}}(e) = \mathsf{Y}_{i_j}(e) = e_j \leq e_n$, a contradiction. Thus the processes $i_1, i_2, \ldots, i_n, i_{n+1}$ are pairwise distinct.

We repeat this procedure as long as $\mathsf{loc}(e)$ is disjoint from the location of the last event constructed. Since $\mathscr{P}$ is finite, the procedure can be repeated only finitely many times. Let m be the length of the final event sequence $e_1 < e_2 < \cdots < e_m$ and the final process sequence $i_1, i_2, \ldots, i_m$. At that stage, we have $\mathsf{loc}(e_m) \cap \mathsf{loc}(e) \neq \emptyset$.

We now use these sequences of events and processes to construct the announced sd-path formula π.

The fact that $\mathsf{Y}_{i_m}(e) = e_m$ implies that no event e' satisfying $e_m < e' < e$ is an i_m-event. Let $\ell \in \mathsf{loc}(e_m) \cap \mathsf{loc}(e)$. Then both e_m and e are ℓ-events and $e_m = \mathsf{Y}_{i_m}(e) = \mathsf{prev}_\ell(\mathsf{on}_{i_m})(e)$.

Similarly, for $1 < j \leq m$, $\mathsf{Y}_{i_{j-1}}(e) = e_{j-1}$ and hence no i_j-event e' satisfying $e_{j-1} < e' < e$ is an i_{j-1}-event. In particular, $\mathsf{Y}_{i_{j-1}}(e_j) = e_{j-1}$. Moreover, both e_{j-1} and e_j are i_j-events, so $e_{j-1} = \mathsf{Y}_{i_{j-1}}(e_j) = \mathsf{prev}_{i_j}(\mathsf{on}_{i_{j-1}})(e_j)$.

Now let $\pi = \mathsf{prev}_\ell(\mathsf{on}_{i_m}) \cdot \mathsf{prev}_{i_m}(\mathsf{on}_{i_{m-1}}) \cdots \mathsf{prev}_{i_2}(\mathsf{on}_{i_1})$. Then π is an sd-path formula and $Y_i(e) = f = e_1 = \pi(e)$. Moreover, since $i_1 = i$, we also have $Y_i(e) = (\pi \cdot \mathsf{on}_i?)(e)$.

To conclude, we only need to verify that $m < |\mathscr{P}|$. Let indeed ℓ be any process in $\mathsf{loc}(e_m) \cap \mathsf{loc}(e)$. Since $\mathsf{loc}(e_j) \cap \mathsf{loc}(e) = \emptyset$ for every $j < m$, we find that $\ell \notin \mathsf{loc}(e_j)$, and hence ℓ is distinct from $i_1, i_2, \ldots, i_m$. Therefore $m + 1 \leq |\mathscr{P}|$, which completes the proof. □

Example 3.7. The proof of Proposition 3.6 shows that, for the trace in Fig. 2, we have

$$\mathsf{Y}_2(e_{11}) = (\mathsf{prev}_4(\mathsf{on}_3) \cdot \mathsf{prev}_3(\mathsf{on}_2) \cdot \mathsf{on}_2?)(e_{11}) = e_5$$

□

3.2 Expressing Constant Event Formulas in LocPastPDL

The main theorem in this section is the following.

Theorem 3.8. *Let $i, j, k \in \mathscr{P}$. The constant event formulas $\mathsf{Y}_i \leq \mathsf{Y}_j$ and $\mathsf{Y}_{i,j} \leq \mathsf{Y}_k$ can be expressed in* LocPastPDL

Overview of the proof. The proof relies on a complex construction, which occupies the rest of Section 3.2. We show (Proposition 3.9 below) that, for each pair (π, π') of sd-path formulas, there exists a LocPastPDL event formula $\mathsf{Leq}(\pi, \pi')$ which expresses the following: $t, e \models \mathsf{Leq}(\pi, \pi')$ if and only if $\pi(e)$ and $\pi'(e)$ exist and $\pi(e) \leq \pi'(e)$. Proposition 3.6 is then used to show that $\mathsf{Y}_i \leq \mathsf{Y}_j$ and $\mathsf{Y}_{i,j} \leq \mathsf{Y}_k$ are logically equivalent to LocPastPDL event formulas using formulas of the form $\mathsf{Leq}(\pi, \pi')$, see Corollaries 3.10 and 3.11.

The proof of Proposition 3.9 (in Section 3.2.1) uses the existence of another class of LocPastPDL formulas, written $\mathsf{Eq}(\pi, \pi')$, which express that $\pi(e)$ and $\pi'(e)$ exist and $\pi(e) = \pi'(e)$ (Proposition 3.12 below). And the proof of Proposition 3.12, in Section 3.2.2, in turn uses the existence of finite sets of so-called separating formulas, which are constructed in Section 3.2.3. □

We now unravel this complex proof structure.

Proposition 3.9. *For each pair of sd-path formulas π and π', there exists a* LocPastPDL *event formula $\mathsf{Leq}(\pi, \pi')$ such that $t, e \models \mathsf{Leq}(\pi, \pi')$ if and only if $\pi(e)$ and $\pi'(e)$ exist and $\pi(e) \leq \pi'(e)$.*

The proof of Proposition 3.9 is deferred to Section 3.2.1. Proposition 3.9 yields the two following corollaries, which establish Theorem 3.8.

Corollary 3.10. *Let $i, j \in \mathscr{P}$ be processes. Then $\mathsf{Y}_i \leq \mathsf{Y}_j$ is logically equivalent to the following* LocPastPDL *event formula:*

$$\underbrace{\bigvee_{\pi'} \langle \pi' \cdot \mathsf{on}_i? \rangle}_{\varphi_1} \wedge \underbrace{\bigvee_{\pi} \langle \pi \cdot \mathsf{on}_j? \rangle}_{\varphi_2} \wedge \underbrace{\bigvee_{\pi} \bigwedge_{\pi'} \Big(\langle \pi' \cdot \mathsf{on}_i? \rangle \implies \mathsf{Leq}(\pi' \cdot \mathsf{on}_i?, \pi \cdot \mathsf{on}_j?) \Big)}_{\varphi_3}$$

where the disjunctions and conjunctions run over sd-path formulas π, π' of length at least 1 and at most $|\mathscr{P}| - 1$

Proof. By definition, $t, e \models \mathsf{Y}_i \leq \mathsf{Y}_j$ if and only if $\mathsf{Y}_i(e)$ and $\mathsf{Y}_j(e)$ exist, and $\mathsf{Y}_i(e) \leq \mathsf{Y}_j(e)$. If π' is an sd-path formula and $\pi' \cdot \mathsf{on}_i?(e)$ exists, then there is an i-event in the strict past of e, and hence $\mathsf{Y}_i(e)$ exists. Conversely, Proposition 3.6 shows that if $\mathsf{Y}_i(e)$ exists, then it is equal to $\pi' \cdot \mathsf{on}_i?(e)$ for some such π', with length between 1 and $|\mathscr{P}| - 1$. Therefore $t, e \models \varphi_1$ if and only if $\mathsf{Y}_i(e)$ exists. Similarly φ_2 expresses the existence of $\mathsf{Y}_j(e)$.

Now assume that $t, e \models \mathsf{Y}_i \leq \mathsf{Y}_j$. Let π be given by Proposition 3.6 such that $\mathsf{Y}_j(e) = \pi \cdot \mathsf{on}_j?(e)$, and let π' be an arbitrary sd-path with $1 \leq \|\pi'\| < |\mathscr{P}|$. Consider the (π, π') implication in φ_3. If $\pi' \cdot \mathsf{on}_i?(e)$ does not exist, this implication is vacously satisfied. If instead $\pi' \cdot \mathsf{on}_i?(e)$ exists, then it is an i-event, so $\pi' \cdot \mathsf{on}_i?(e) \leq \mathsf{Y}_i(e)$ and the same implication is also satisfied. This establishes the fact that $t, e \models \mathsf{Y}_i \leq \mathsf{Y}_j$ implies that $t, e \models \varphi_3$.

Conversely, assume that $\mathsf{Y}_i(e)$ and $\mathsf{Y}_j(e)$ exist, and $t, e \models \varphi_3$. Then there exists an sd-path π which addresses a j-event in the strict past of e (namely $\pi \cdot \mathsf{on}_j?(e)$) such that every i-event of the form $\pi' \cdot \mathsf{on}_i?(e)$ (where π' is an sd-path formula with $1 \leq \|\pi'\| < |\mathscr{P}|$) lies below $\pi \cdot \mathsf{on}_j?(e)$. In particular, $\mathsf{Y}_i(e) \leq \pi \cdot \mathsf{on}_j?(e) \leq Y_j(e)$, and this concludes the proof. □

Corollary 3.11. *Let $i, j, k \in \mathscr{P}$ be processes. Then $\mathsf{Y}_{i,j} \leq \mathsf{Y}_k$ is logically equivalent to the following* LocPastPDL *event formula:*

$$\underbrace{\bigvee_{\pi', \pi''} \langle \pi' \cdot \mathsf{on}_i? \cdot \pi'' \cdot \mathsf{on}_j? \rangle}_{\varphi_1} \wedge \underbrace{\bigvee_{\pi} \langle \pi \cdot \mathsf{on}_k? \rangle}_{\varphi_2} \wedge \underbrace{\left(\bigvee_{\pi} \bigwedge_{\pi', \pi''} \Big(\langle \pi' \cdot \mathsf{on}_i? \cdot \pi'' \cdot \mathsf{on}_j? \rangle \implies \mathsf{Leq}(\pi' \cdot \mathsf{on}_i? \cdot \pi'' \cdot \mathsf{on}_j?, \pi \cdot \mathsf{on}_k?) \Big) \right)}_{\varphi_3}$$

where the disjunctions and conjunctions run over sd-path formulas π, π', π'' of length at least 1 *and at most $|\mathscr{P}| - 1$.*

Proof. By definition, $t, e \models \mathsf{Y}_{i,j} \leq \mathsf{Y}_k$ if and only if $\mathsf{Y}_j(\mathsf{Y}_i(e))$ and $\mathsf{Y}_k(e)$ exist, and $\mathsf{Y}_j(\mathsf{Y}_i(e)) \leq \mathsf{Y}_k(e)$. As in the proof of Corollary 3.10, Proposition 3.6 can be used to prove that φ_1 expresses the existence of $\mathsf{Y}_j(\mathsf{Y}_i(e))$ and φ_2 expresses the existence of $\mathsf{Y}_k(e)$.

Assume that $t, e \models \mathsf{Y}_{i,j} \leq \mathsf{Y}_k$. Let π be given by Proposition 3.6 such that $\mathsf{Y}_k(e) = \pi \cdot \mathsf{on}_k?(e)$, and let π', π'' be arbitrary sd-path formulas with length between 1 and $|\mathscr{P}| - 1$. If $\pi' \cdot \mathsf{on}_i? \cdot \pi'' \cdot \mathsf{on}_j?(e)$ does not exist, then the (π, π', π'') implication in φ_3 is vacously satisfied. If it does exist, then it is a j-event which is strictly below an i-event in the strict past of e. In particular, this event sits below $\mathsf{Y}_j(\mathsf{Y}_i(e)) \leq \mathsf{Y}_k(e) = \pi \cdot \mathsf{on}_k?(e)$, and hence the corresponding implication in φ_3 is again satisfied. Thus $t, e \models \varphi_3$.

Conversely, suppose that $\mathsf{Y}_j(\mathsf{Y}_i(e))$ and $\mathsf{Y}_k(e)$ exist and $t, e \models \varphi_3$. Again as in the proof of Corollary 3.10, there exists an sd-path π such that the k-event $\pi \cdot \mathsf{on}_k?(e)$ sits above any j-event of the form $\pi' \cdot \mathsf{on}_i? \cdot \pi'' \cdot \mathsf{on}_j?(e)$ (where π', π'' are sd-path formulas of length between 1 and $|\mathscr{P}| - 1$). In particular (again using Proposition 3.6), $\mathsf{Y}_j(\mathsf{Y}_i(e)) \leq \pi \cdot \mathsf{on}_k?(e) \leq \mathsf{Y}_k(e)$ and this concludes the proof. □

3.2.1 Reducing inequalities to equalities

Our next step is to prove Proposition 3.9, which relies on the following proposition.

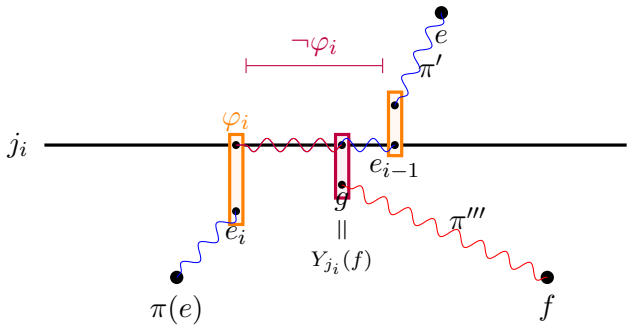

Figure 3: Proof of Lemma 3.13

Proposition 3.12. *For each pair of sd-path formulas π and π', there exists a* LocPastPDL *event formula* $\mathsf{Eq}(\pi, \pi')$ *such that* $t, e \models \mathsf{Eq}(\pi, \pi')$ *if and only if* $\pi(e)$ *and* $\pi'(e)$ *exist and* $\pi(e) = \pi'(e)$.

The proof of Proposition 3.12 is deferred to Section 3.2.2. For now, we show how it is used to establish Proposition 3.9. We first record the following technical lemma.

Lemma 3.13. *Let t be a trace, let e, f be two events in t with $e \not\leq f$ and let π be an sd-path formula such that $\pi(e)$ exists. Then $\pi(e) \leq f$ if and only if there exists a prefix π' of π and a sd-path formula π'' with $\|\pi''\| \leq |\mathscr{P}|$ such that $\pi'(e) = \pi''(f)$.*

Proof. One direction is easy: if $\pi'(e) = \pi''(f)$ for a prefix π' of π, then $\pi(e) \leq \pi'(e) = \pi''(f) \leq f$.

Conversely, suppose that $\pi(e) \leq f$ with $\pi = \varphi'_0? \cdot \mathsf{prev}_{j_1}(\varphi_1) \cdot \varphi'_1? \cdot \dots \cdot \varphi'_{k-1}? \cdot \mathsf{prev}_{j_k}(\varphi_k) \cdot \varphi'_k?$ (we may assume that π is in standardized form).

If $e = f$, the announced result holds with $\pi' = \varphi'_0?$ and $\pi'' = \top?$. We now assume that $e \neq f$.

Let $e_0 = e$ and, for $1 \leq i \leq k$, $e_i = \mathsf{prev}_{j_i}(\varphi_i)(e_{i-1})$. By definition of $\mathsf{prev}_j(\varphi)$, we have $j_i \in \mathsf{loc}(e_i) \cap \mathsf{loc}(e_{i-1})$. And we also have $e_k = \pi(e) \leq f$. Let i be the least index such that $e_i \leq f$; then $1 \leq i \leq k$ since $e_0 = e \not\leq f$. Now let $\pi' = \varphi'_0? \cdot \mathsf{prev}_{j_1}(\varphi_1) \cdot \varphi'_1? \cdot \dots \cdot \varphi'_{i-1}? \cdot \mathsf{prev}_{j_i}(\varphi_i)$. Then π' is a prefix of π and $e_i = \pi'(e)$.

Since e_i and e_{i-1} are j_i-events, the maximal j_i-event such that $g \leq f$ satisfies $e_i \leq g < e_{i-1}$, see Fig. 3. Moreover, $g = \pi'''(f)$ for some sd-path formula π''' with $\|\pi'''\| < |\mathscr{P}|$. Indeed, if $g < f$, we have $g = \mathsf{Y}_{j_i}(f)$ (since $e_{i-1} \not\leq f$) and Proposition 3.6 shows the existence of π'''. If instead $g = f$, then we let $\pi''' = \top?$.

We can now conclude the proof: if $g = e_i$ then we let $\pi'' = \pi'''$. Otherwise, we have $e_i = \mathsf{prev}_{j_i}(\varphi_i)(g)$ and we let $\pi'' = \pi''' \cdot \mathsf{prev}_{j_i}(\varphi_i)$. In either case, we have $\pi'(e) = e_i = \pi''(f)$, as announced. □

Proof of Proposition 3.9. Let π, π' be sd-path formulas. We define $\mathsf{Leq}(\pi, \pi')$ to be the LocPastPDL event formula

$$\mathsf{Leq}(\pi, \pi') = \langle \pi \rangle \wedge \langle \pi' \rangle \wedge \bigvee_{\pi'', \pi'''} \mathsf{Eq}(\pi'', \pi' \pi''')$$

where π'' ranges over prefixes of π and π''' ranges over sd-path formulas of length at most $|\mathscr{P}|$ (so that π'' and $\pi'\pi'''$ have length at most $|\pi|$ and $|\pi'| + |\mathscr{P}|$, respectively).

First suppose that $t, e \models \mathsf{Leq}(\pi, \pi')$. Clearly $\pi(e)$ and $\pi'(e)$ exist. Since $t, e \models \bigvee_{\pi'', \pi'''} \mathsf{Eq}(\pi'', \pi'\pi''')$, there exists a prefix π'' of π and an sd-path formula π''' such that $\pi''(e) = \pi'''(\pi'(e))$. This yields $\pi''(e) \leq \pi'(e)$. Since π'' is a prefix of π, it follows that $\pi(e) \leq \pi''(e)$, and hence $\pi(e) \leq \pi'(e)$.

Conversely, suppose that $\pi(e)$ and $\pi'(e)$ exist and that $\pi(e) \leq \pi'(e)$. The first (existence) condition implies that $t, e \models \langle\pi\rangle \wedge \langle\pi'\rangle$. Since $\pi(e) \leq \pi'(e)$, Lemma 3.13 applied with $f = \pi'(e)$ shows that $\pi''(e) = \pi'''(\pi'(e))$ for some prefix π'' of π and an sd-path formula π''' with length at most $|\mathscr{P}|$. That is, $t, e \models \mathsf{Eq}(\pi'', \pi'\pi''')$ for those particular π'', π''' and this completes the proof. □

3.2.2 Reduction of equalities to separating formulas

Our aim here is to prove Proposition 3.12. For this, we use yet another intermediate result.

Proposition 3.14. *For each sd-path formula π of length at least 1, there exists a finite set $\Xi(\pi)$ of* LocPastPDL *event formulas such that, for every trace t and event e in t such that $\pi(e)$ exists, $\Xi(\pi)$* separates *e and $\pi(e)$: that is, there exist $\xi, \xi' \in \Xi(\pi)$ such that ξ holds at e and not at $\pi(e)$, and ξ' holds at $\pi(e)$ and not at e.*

The proof of Proposition 3.14 is complex and is given in Section 3.2.3. Let us see immediately how that statement is used to prove Proposition 3.12.

Proof of Proposition 3.12. Let π, π' be sd-path formulas. Let Ξ be the union of the set $\{\mathsf{on}_i \mid i \in \mathscr{P}\}$ and of the $\Xi(\sigma)$ where σ is an sd-path formula of length at least 1 and at most $\max(|\pi|, |\pi'|) + |\mathscr{P}|$. Notice that Ξ is a finite set since, up to equivalence, there are finitely many sd-path formulas of bounded length. We define the LocPastPDL event formula $\mathsf{Eq}(\pi, \pi')$ by

$$\mathsf{Eq}(\pi, \pi') = \langle\pi\rangle \wedge \langle\pi'\rangle \;\; \wedge \;\; \neg \bigvee_{\xi \in \Xi} \big(\langle\pi\rangle\xi \wedge \langle\pi'\rangle\neg\xi\big)\,.$$

Suppose first that $\pi(e)$ and $\pi'(e)$ exist and are equal. Clearly $t, e \models \langle\pi\rangle \wedge \langle\pi'\rangle$. Moreover, since $\pi(e) = \pi'(e)$, we have $t, e \models \langle\pi\rangle\varphi$ if and only if $t, e \models \langle\pi'\rangle\varphi$, for every event formula φ. This implies $t, e \models \neg \bigvee_{\xi\in\Xi} \big(\langle\pi\rangle\xi \wedge \langle\pi'\rangle\neg\xi\big)$, and hence $t, e \models \mathsf{Eq}(\pi, \pi')$.

Conversely, assume that $t, e \models \langle\pi\rangle \wedge \langle\pi'\rangle$ and let $f = \pi(e)$ and $f' = \pi'(e)$. Suppose $f \neq f'$. We show that $t, e \not\models \mathsf{Eq}(\pi, \pi')$, i.e., $t, e \models \langle\pi\rangle\xi \wedge \langle\pi'\rangle\neg\xi$ for some $\xi \in \Xi$. If $\mathsf{loc}(f) \not\subseteq \mathsf{loc}(f')$ then we choose $\xi = \mathsf{on}_i$ for some $i \in \mathsf{loc}(f) \setminus \mathsf{loc}(f')$.

Suppose now that $\mathsf{loc}(f) \subseteq \mathsf{loc}(f')$. Then f and f' are ℓ-events for any process $\ell \in \mathsf{loc}(f)$, and hence they are ordered. Without loss of generality, we assume that $f < f'$. By Lemma 3.13, there exist sd-path formulas π_1, π_2, π'' such that $\pi = \pi_1 \cdot \pi_2$, $\|\pi''\| \leq |\mathscr{P}|$ and $\pi_1(e) = \pi''(f')$, see Fig. 4. In particular, $f = \pi_2(\pi_1(e)) = \pi_2(\pi''(f'))$. Observe that the path $\pi''\pi_2$ has length

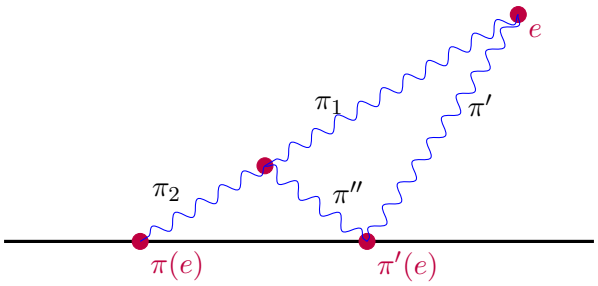

Figure 4: Proof of Proposition 3.12

at most $|\pi| + |\mathscr{P}|$ and at least 1 (since $(\pi''\pi_2)(f') = f < f'$). Proposition 3.14 then implies the existence of a formula $\xi \in \Xi(\pi''\pi_2) \subseteq \Xi$ such that $t, f \models \xi$ and $t, f' \models \neg\xi$. Therefore t, e satisfies $\langle\pi\rangle\xi \wedge \langle\pi'\rangle\neg\xi$, and $t, e \not\models \mathsf{Eq}(\pi, \pi')$. In the symmetric case $f' < f$ we use a formula $\xi' \in \Xi$ such that $t, f' \models \neg\xi'$ and $t, f \models \xi'$. We obtain $t, e \models \langle\pi\rangle\xi' \wedge \langle\pi'\rangle\neg\xi'$, which concludes the proof. □

3.2.3 Construction of separating formulas

Here we establish Proposition 3.14. This is a complex construction which involves constructing, for each sd-path formula π, LocPastPDL event formulas $\mathsf{same}_j(\pi)$ ($j \in \mathscr{P}$) which capture the fact that both $\langle\pi\rangle$ and $\langle\mathsf{prev}_j(\langle\pi\rangle)\rangle$ are defined on a j-event e, and point to the same event in the past of e (Proposition 3.16); and LocPastPDL formulas $\mathsf{Mod}(k,n,\pi)$ ($0 \leq k < n$) which allow counting modulo n the number of events f in the past of the current event such that $\pi(f)$ exists and $\pi(f) \neq \pi(f')$ for all $f' < f$ where π is defined.

We start with an easy lemma.

Lemma 3.15. *Let π be an sd-path formula of length n. Let t be a trace and $e_0, e_1, \ldots, e_n$ be events in t on each of which $\langle\pi\rangle$ holds. If $\pi(e_0) < \pi(e_1) < \cdots < \pi(e_n)$, then $e_0 \leq \pi(e_n)$.*

Proof. The proof is by induction on $n = \|\pi\|$, and the statement is trivial if $n = 0$ since in this case $\pi(e_0) = e_0$.

Assume now that $n > 0$, say, $\pi = \varphi'? \cdot \mathsf{prev}_j(\varphi) \cdot \pi'$. For each $0 \leq i \leq n$, let $f_i = \pi(e_i)$. Note that, since $\langle\pi\rangle$ holds at each e_i, then each e_i is a j-event, and so is each $e'_i = \mathsf{prev}_j(\varphi)(e_i)$. In particular, the e_i and e'_i are totally ordered. In addition, $f_i = \pi'(e'_i)$, see Fig. 5.

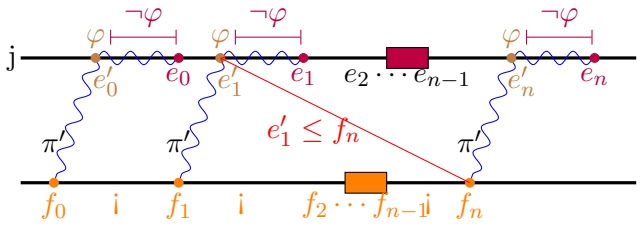

Figure 5: Lemma 3.15

The induction hypothesis implies $e'_1 \leq f_n$. Moreover, $f_0 < f_1$ implies $e'_0 < e'_1$. If $e'_1 < e_0$, we have $e'_1 = e'_0$ by definition of $\mathsf{prev}_j(\varphi)$, a contradiction. So we have $e_0 \leq e'_1 \leq f_n$ and this concludes the proof. □

We now establish the existence of the LocPastPDL formulas $\mathsf{same}_j(\pi)$ announced in the introduction of Section 3.2.3.

Proposition 3.16. *Let π be an sd-path formula and $j \in \mathscr{P}$. There exists a LocPastPDL event formula $\mathsf{same}_j(\pi)$ such that, for every trace t and event e, we have $t, e \models \mathsf{same}_j(\pi)$ if and only if $\langle\pi\rangle$ and $\langle\mathsf{prev}_j(\langle\pi\rangle)\rangle$ hold at e, and $\pi(e) = \pi(\mathsf{prev}_j(\langle\pi\rangle)(e))$.*

Proof. The proof is by structural induction on π. We observe that if t, e satisfies $\langle\mathsf{prev}_j(\langle\pi\rangle)\rangle$, then e is a j-event.

Case 1: $\pi = \varphi?$ In this case, $\pi(e) = e$ if it exists, whereas $\mathsf{prev}_j(\langle\pi\rangle)$ holds only in the strict past of e. So we can never have $\pi(e) = \pi(\mathsf{prev}_j(\langle\pi\rangle)(e))$ and we can let $\mathsf{same}_j(\pi) = \bot$.

Case 2: $\pi = \varphi? \cdot \pi'$ Note that $t, e \models \langle\pi\rangle$ if and only if $t, e \models \varphi \wedge \langle\pi'\rangle$. In that case, $\pi(e) = \pi'(e)$.

Assume that $\pi(e)$ and $\mathsf{prev}_j(\langle\pi\rangle)(e)$ exist. Consider the sequence $e_k < e_{k-1} < \cdots < e_0$ of all j-events satisfying $\langle\pi'\rangle$, starting at $e_k = \mathsf{prev}_j(\langle\pi\rangle)(e)$ and ending at $e_0 = e$, see Fig. 6. In particular, $e_{i+1} = \mathsf{prev}_j(\langle\pi'\rangle)(e_i)$ for every $0 \leq i < k$.

If $\pi(e) = \pi(\mathsf{prev}_j(\langle\pi\rangle)(e))$, then π' also maps e, $\mathsf{prev}_j(\langle\pi\rangle)(e)$, and hence all the e_i, to the same event: equivalently, $\pi'(e_{i+1}) = \pi'(e_i)$ for every $0 \leq i < k$.

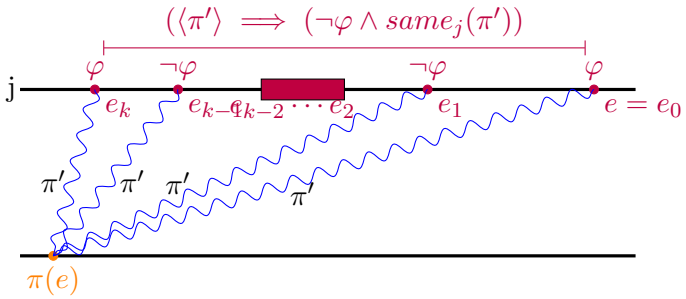

Figure 6: Case $\pi = \varphi? \cdot \pi'$

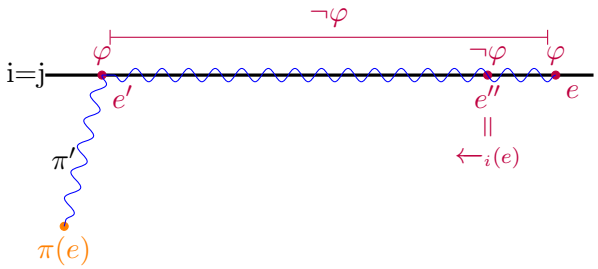

Figure 7: Case $\pi = \mathsf{prev}_i(\varphi) \cdot \pi'$ and $t, e'' \models \neg\varphi$

Conversely, suppose that, for every $0 \leq i < k$, we have $\pi'(e_i) = \pi'(e_{i+1})$. It follows that every $\pi'(e_i)$ is equal to $\pi'(e_0)$. By definition, $\pi'(e_0) = \pi(e)$ and $\pi'(e_k) = \pi(e_k) = \pi(\mathsf{prev}_j(\langle\pi\rangle)(e))$. Therefore, $\mathsf{prev}_j(\langle\pi\rangle)(e) = \pi(e)$.

Thus, if $\pi(e)$ and $\mathsf{prev}_j(\langle\pi\rangle)(e)$ exist, then $\mathsf{same}_j(\pi)$ holds at e if and only if the j-events strictly between $\mathsf{prev}_j(\langle\pi\rangle)(e)$ and e that do satisfy $\langle\pi'\rangle$ (that is: the e_i) do not satisfy φ, and do satisfy $\mathsf{same}_j(\pi')$. This justifies letting $\mathsf{same}_j(\pi)$ be

$$\varphi \wedge \mathsf{same}_j(\pi') \wedge \left\langle \leftarrow_j \cdot \left(\left(\neg\langle\pi'\rangle \vee (\neg\varphi \wedge \mathsf{same}_j(\pi')) \right)? \cdot \leftarrow_j \right)^* \right\rangle \langle\pi\rangle.$$

Case 3: $\pi = \mathsf{prev}_i(\varphi) \cdot \pi'$ An event e satisfies $\langle\pi\rangle$ if and only if $e' = \mathsf{prev}_i(\varphi)(e)$ exists and satisfies $\langle\pi'\rangle$. In that case, $\pi(e) = \pi'(e')$. We first consider the case where $j = i$, which is notationally slightly simpler yet contains the substance of the proof.

Let e'' be the immediate predecessor of e on process i. Suppose first that e'' does not satisfy φ (see Fig. 7). Then $e' < e''$ and $e' = \mathsf{prev}_i(\varphi)(e'')$. It follows that e'' satisfies $\langle\pi\rangle$, $e'' = \mathsf{prev}_i(\langle\pi\rangle)(e)$ and $\pi(e'') = \pi'(e') = \pi(e)$.

Suppose now that e'' satisfies φ (see Fig. 8). Then $e' = e''$. Assume that $f = \mathsf{prev}_i(\langle\pi\rangle)(e)$ exists. Then $f \leq e'$ and f satisfies $\langle\pi\rangle$, so $f' = \mathsf{prev}_i(\varphi)(f)$ exists and satisfies $\langle\pi'\rangle$.

We first verify that $f' = \mathsf{prev}_i(\varphi \wedge \langle\pi'\rangle)(e')$: if it is not the case, there exists an i-event g strictly between f' and e', satisfying $\varphi \wedge \langle\pi'\rangle$. Then the immediate successor of g on the i-line satisfies $\langle\pi\rangle$. This yields a contradiction since $g > f' = \mathsf{prev}_i(\varphi)(\mathsf{prev}_i(\langle\pi\rangle)(e))$.

Let now $f' = e_k < \cdots < e_1 = e'$ be the sequence of i-events between f' and e' which satisfy $\langle\pi'\rangle$. In particular, for each $1 \leq h < k$, $e_{h+1} = \mathsf{prev}_i(\langle\pi'\rangle)(e_h)$.

Now $\pi(e) = \pi(f)$ if and only if $\pi'(e') = \pi'(f')$. This is equivalent to requiring that $\pi'(e_h) = \pi'(e') = \pi'(e_1)$ for all h, which in turn is equivalent to each e_h $(1 \leq h < k)$ satisfying $\mathsf{same}_i(\pi')$. Since $f' = \mathsf{prev}_i(\varphi \wedge \langle\pi'\rangle)(e')$ note that all events in the interval (f', e') either satisfy $\neg\langle\pi'\rangle$ or they are among e_h hence they satisfy $\neg\varphi \wedge \mathsf{same}_i(\pi')$

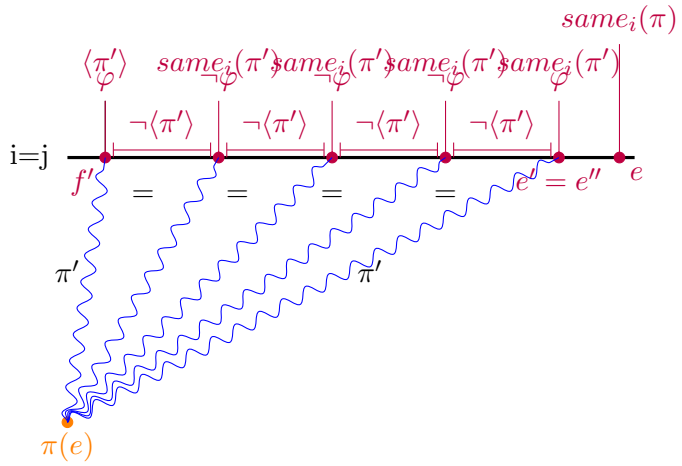

Figure 8: Case $\pi = \mathsf{prev}_i(\varphi) \cdot \pi'$ and $t, e'' \models \varphi$

This justifies letting $\mathsf{same}_j(\pi)$ be the conjunction of $\langle\pi\rangle$ and

$$\langle{\leftarrow_i}\rangle\neg\varphi \vee \Big\langle {\leftarrow_i} \cdot (\varphi \wedge \mathsf{same}_i(\pi'))? \cdot {\leftarrow_i} \cdot \Big(\big(\langle\pi'\rangle \implies (\neg\varphi \wedge \mathsf{same}_i(\pi'))\big)? \cdot {\leftarrow_i} \Big)^* \cdot \big(\langle\pi'\rangle \wedge \varphi\big)? \Big\rangle.$$

Sub-case 3.2: $i \neq j$ Here we let $\mathsf{same}_j(\pi) = \mathsf{same}_i(\mathsf{on}_j? \cdot \pi)$. Note that Case 2 gives a formula for $\mathsf{same}_i(\mathsf{on}_j? \cdot \pi)$ in terms of $\mathsf{same}_i(\pi)$, which in turn is constructed in Case 3.1 above.

To justify this choice for $\mathsf{same}_j(\pi)$, we observe that any event satisfying $\langle\pi\rangle$ is an i-event, since π starts with ${\leftarrow_i}$. Similarly, if $\mathsf{prev}_j(\langle\pi\rangle)$ holds at an event e, then both e and $f = \mathsf{prev}_j(\langle\pi\rangle)(e)$ are j-events (since $\mathsf{prev}_j(\langle\pi\rangle)$ starts and ends with ${\leftarrow_j}$). As a result, π and $\mathsf{prev}_j(\langle\pi\rangle)$ are defined at e if and only if $\mathsf{on}_j? \cdot \pi$ and $\mathsf{prev}_i(\langle \mathsf{on}_j? \cdot \pi\rangle)$ are. Moreover, in that case, we have $f = \mathsf{prev}_j(\langle\pi\rangle)(e) = \mathsf{prev}_i(\langle \mathsf{on}_j? \cdot \pi\rangle)(e)$, $\pi(e) = (\mathsf{on}_j? \cdot \pi)(e)$ and $\pi(f) = (\mathsf{on}_j? \cdot \pi)(f)$. Therefore $\pi(e) = \pi(f)$ if and only if $(\mathsf{on}_j? \cdot \pi)(e) = (\mathsf{on}_j? \cdot \pi)(f)$. This concludes the proof. □

We now use Lemma 3.16 to construct, for each sd-path formula π, LocPastPDL event formulas which count modulo n the number of events f in the past of the current event e with different π-images.

Lemma 3.17. *Let π be an sd-path formula of positive length, say $\pi = \varphi'? \cdot \mathsf{prev}_i(\varphi) \cdot \pi'$. For every $n > 1$ and $0 \leq k < n$, there exists a* LocPastPDL *event formula $\mathsf{Mod}(k, n, \pi)$ such that, for any trace t and event e in t, $t, e \models \mathsf{Mod}(k, n, \pi)$ if and only if $t, e \models \langle\pi\rangle$ and*

$$\Big|\Big\{ f < e \mid t, f \models \langle\pi\rangle \wedge \neg\mathsf{same}_i(\pi) \Big\}\Big| \;=\; k \pmod n.$$

Proof. We note that every event satisfying $\langle\pi\rangle$ is an i-event. Let $\psi = \langle\pi\rangle \wedge \neg\mathsf{same}_i(\pi)$.

If $t, e \models \langle\pi\rangle$, then the events $f < e$ satisfying ψ are the events $(\mathsf{prev}_i(\psi))^m(e)$ $(m > 0)$, and the minimal such event (which does not satisfy $\mathsf{prev}_i(\psi)$) satisfies $\neg\big(\langle{\leftarrow_i^+}\rangle\psi\big)$, see Fig. 9. As a result, choosing the following formulas for the $\mathsf{Mod}(k, n, \pi)$

$$\mathsf{Mod}(0, n, \pi) = \langle\pi\rangle \wedge \Big\langle \Big(\big(\mathsf{prev}_i(\psi)\big)^n \Big)^* \Big\rangle \neg\big(\langle{\leftarrow_i^+}\rangle\psi\big)$$
$$\mathsf{Mod}(k, n, \pi) = \langle\pi\rangle \wedge \langle\mathsf{prev}_i(\psi)\rangle \mathsf{Mod}(k-1, n, \pi) \text{ for every } 1 \leq k < n.$$

proves the statement. □

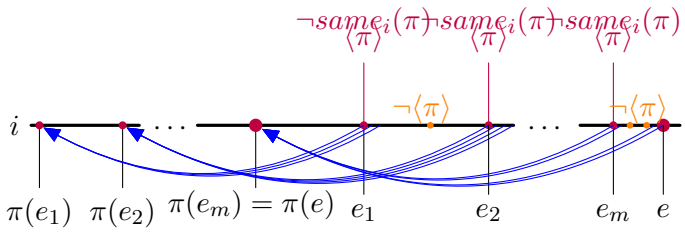

Figure 9: Mod Counting

We can finally complete the task of Section 3.2.

Proof of Proposition 3.14. Let π be an sd-path formula of positive length, say $\pi = \varphi'?\cdot \mathsf{prev}_i(\varphi)\cdot \pi'$, and let $n = \|\pi\| + 1$. In particular, $\langle\pi\rangle$ holds only at i-events. Let

$$\Xi(\pi) \;=\; \Big\{\neg\langle\pi\rangle, \langle\pi\rangle, \mathsf{same}_i(\pi), \neg\mathsf{same}_i(\pi)\Big\} \cup \Big\{\mathsf{Mod}(k,n,\pi) \mid 0 \le k < n\Big\}.$$

We want to show that $\Xi(\pi)$ is a separating set of formulas for π, that is, if t is a trace, e is an event and $\pi(e)$ exists, then there exist $\xi \in \Xi(\pi)$ such that ξ holds at e and not at $\pi(e)$. It follows easily from the definition of $\Xi(\pi)$ that there is also a formula $\xi' \in \Xi(\pi)$ such that ξ' holds at $\pi(e)$ and not at e.

Suppose that this is not the case: there exists a trace t and an event e such that $\pi(e)$ exists and, for every formula $\xi \in \Xi(\pi)$, if ξ holds at e, then it holds at $\pi(e)$ as well. Since $\langle\pi\rangle \in \Xi(\pi)$ holds at e, the event $\pi(e)$ also satisfies $\langle\pi\rangle$.

Let $e_1 < \cdots < e_m$ be the i-events f such that $\pi(e) < f \le e$ and $t, f \models \langle\pi\rangle \wedge \neg\mathsf{same}_i(\pi)$. If $m = 0$, then every i-event f such that $\pi(e) < f \le e$ which satisfies $\langle\pi\rangle$, satisfies $\mathsf{same}_i(\pi)$ as well. In particular, e satisfies $\mathsf{same}_i(\pi)$ and it follows that $\pi(\pi(e)) = \pi(e)$. This cannot be since $\|\pi\| > 0$. Therefore $m > 0$.

By construction, $\pi(e) < e_1$ and $e_{h-1} \le \mathsf{prev}_i(\langle\pi\rangle)(e_h)$ for each $1 < h \le m$. By definition of $\mathsf{same}_i(\pi)$, we have $\pi(e_1) < \cdots < \pi(e_m) \le \pi(e) < e_1$. If $m \ge n = \|\pi\| + 1$, then Lemma 3.15 yields $e_1 \le \pi(e_m)$, a contradiction. Therefore, $1 \le m < n$.

Now consider the formula $\mathsf{same}_i(\pi)$ and $\neg\mathsf{same}_i(\pi)$ in $\Xi(\pi)$. Assuming, as we do, that $\Xi(\pi)$ is not separating, shows that e satisfies $\mathsf{same}_i(\pi)$ if and only if $\pi(e)$ does. Therefore e satisfies $\langle\pi\rangle \wedge \neg\mathsf{same}_i(\pi)$ if and only $\pi(e)$ does. As a result, m is also the number of events f satisfying $\langle\pi\rangle \wedge \neg\mathsf{same}_i(\pi)$ and $\pi(e) \le f < e$.

Finally, let k be such that $t, e \models \mathsf{Mod}(k,n,\pi)$ (there is necessarily such a k). Since $\mathsf{Mod}(k,n,\pi) \in \Xi(\pi)$, $\pi(e)$ too satisfies $\mathsf{Mod}(k,n,\pi)$, and this implies that $m = 0 \mod n$ — again a contradiction since we verified that $1 \le m < n$. This concludes the proof that $\Xi(\pi)$ is a separating set. □

3.3 Expressing Constant Trace Formulas in LocPastPDL

This subsection is dedicated to proving the analog of Theorem 3.8 for constant trace formulas.

Theorem 3.18. *The constant trace formulas* $\mathsf{L}_i \le \mathsf{L}_j$ *and* $\mathsf{L}_{i,j} \le \mathsf{L}_k$ *can be expressed in* LocPastPDL

The proof strategy is the same as for Theorem 3.8. We start by strengthening Lemma 3.13, using the fact that we are now only concerned with events at the end of the trace.

Lemma 3.19. *Let i be a process, t be a trace, e be the maximum i-event of t and f be any event. Let π be an sd-path formula such that $\pi(e)$ is defined. Then $\pi(e) \leq f$ if and only if π can be factored as $\pi = \pi_1 \cdot \pi_2$ and there exists an sd-path formula π_3 of length at most $|\mathscr{P}|$ such that $\pi_1(e) = \pi_3(f)$.*

Proof. If π_1, π_2, π_3 exist such that $\pi = \pi_1 \cdot \pi_2$ and $\pi_3(f) = \pi_1(e)$ then concatenating both sides by π_2 we have $\pi(e) = \pi_2(\pi_1(e)) \leq \pi_1(e) = \pi_3(f) \leq f$.

Conversely, suppose that $\pi(e) \leq f$. If $e \not< f$, we apply Lemma 3.13 to get the expected result. Suppose now that $e < f$. Since e is the maximum i-event, Proposition 3.6 shows that there exists an sd-path formula π' of length at most $|\mathscr{P}|$ such that $e = \pi'(f)$. We conclude by letting $\pi_3 = \pi'$, $\pi_1 = \top?$ and $\pi_2 = \pi$. □

Lemma 3.20. *Let i, j be processes and let π_1, π_2 be sd-path formulas. There exist* LocPastPDL *trace formulas $\mathsf{EQ}_{i,j}(\pi_1, \pi_2)$ and $\mathsf{LEQ}_{i,j}(\pi_1, \pi_2)$ such that for all traces t, t satisfies $\mathsf{EQ}_{i,j}(\pi_1, \pi_2)$ (resp. $\mathsf{LEQ}_{i,j}(\pi_1, \pi_2)$) if and only if $\pi_1(\mathsf{L}_i(t))$ and $\pi_2(\mathsf{L}_j(t))$ exist, and $\pi_1(\mathsf{L}_i(t)) = \pi_2(\mathsf{L}_j(t))$ (resp. $\pi_1(\mathsf{L}_i(t)) \leq \pi_2(\mathsf{L}_j(t))$).*

Proof. Let $\mathsf{EQ}_{i,j}(\pi_1, \pi_2)$ and $\mathsf{LEQ}_{i,j}(\pi_1, \pi_2)$ be the following formulas:

$$\begin{aligned}
\mathsf{EQ}_{i,j}(\pi_1, \pi_2) = {} & \mathsf{EM}_i(\langle \pi_1 \rangle) \wedge \mathsf{EM}_j(\langle \pi_2 \rangle) \\
& \wedge \neg \bigvee_{\xi \in \Xi} \mathsf{EM}_i(\langle \pi_1 \rangle \xi) \wedge \mathsf{EM}_j(\langle \pi_2 \rangle \neg \xi) \\
\mathsf{LEQ}_{i,j}(\pi_1, \pi_2) = {} & \mathsf{EM}_i(\langle \pi_1 \rangle) \wedge \mathsf{EM}_j(\langle \pi_2 \rangle) \\
& \wedge \bigvee_{\pi_1', \pi_3} \mathsf{EQ}_{i,j}(\pi_1', \pi_2 \cdot \pi_3)
\end{aligned}$$

with $\Xi = \{\mathsf{on}_i? \mid i \in \mathscr{P}\} \cup \bigcup_\pi \Xi(\pi)$ where the union runs over the sd-path formulas π with length at least 1 and at most $|\mathscr{P}| + \max(||\pi_1||, ||\pi_2||)$ (and $\Xi(\pi)$ is given by Proposition 3.14); and where the last disjunction is over the prefixes π_1' of π_1 and the sd-path formulas π_3 of length at most $|\mathscr{P}|$.

Observe first that $t \models \mathsf{EM}_i(\langle \pi_1 \rangle) \wedge \mathsf{EM}_j(\langle \pi_2 \rangle)$ if and only if $\pi_1(\mathsf{L}_i(t))$, $\pi_2(\mathsf{L}_j(t))$ exist. If in addition $\pi_1(\mathsf{L}_i(t)) = \pi_2(\mathsf{L}_j(t))$, then, for every event formula φ, we have $t \models \mathsf{EM}_i(\langle \pi_1 \rangle \varphi)$ if and only if $t \models \mathsf{EM}_j(\langle \pi_2 \rangle \varphi)$. In particular, t satisfies $\mathsf{EQ}_{i,j}(\pi_1, \pi_2)$.

Conversely, suppose that $f_1 = \pi_1(\mathsf{L}_i(t))$ and $f_2 = \pi_2(\mathsf{L}_j(t))$ exist and $f_1 \neq f_2$. If $\mathsf{loc}(f_1) \not\subseteq \mathsf{loc}(f_2)$, let ℓ be a process in $\mathsf{loc}(f_1) \setminus \mathsf{loc}(f_2)$. Then $\xi = \mathsf{on}_\ell? \in \Xi$ and t satisfies $\mathsf{EM}_i(\langle \pi_1 \rangle \xi) \wedge \mathsf{EM}_j(\langle \pi_2 \rangle \neg \xi)$. Thus t does not satisfy $\mathsf{EQ}_{i,j}(\pi_1, \pi_2)$.

If instead $\mathsf{loc}(f_1) \subseteq \mathsf{loc}(f_2)$, let $\ell \in \mathsf{loc}(f_1)$. Then f_1 and f_2 are ℓ-events, and hence $f_1 < f_2$ or $f_2 < f_1$. Assume $f_1 < f_2$ (the proof is similar if $f_2 < f_1$). By Lemma 3.19, there exist sd-path formulas π_1', π_1'', π_3 such that $\pi_1 = \pi_1' \cdot \pi_1''$, $\|\pi_3\| \leq |\mathscr{P}|$ and $\pi_1'(\mathsf{L}_i(t)) = \pi_3(f_2)$, see Fig. 10. In particular $f_1 = \pi_1''(\pi_3(f_2))$. Let then $\pi' = \pi_3 \cdot \pi_1''$. In particular $\|\pi'\| \geq 1$ (since $f_1 = \pi'(f_2) < f_2$) and $\|\pi_1'\| \leq \|\pi_1\|$, so $\Xi(\pi') \subseteq \Xi$. Therefore, we find $\xi \in \Xi(\pi')$ such that t satisfies $\mathsf{EM}_i(\langle \pi_1 \rangle \xi) \wedge \mathsf{EM}_j(\langle \pi_2 \rangle \neg \xi)$, and hence it does not satisfy $\mathsf{EQ}_{i,j}(\pi_1, \pi_2)$.

This concludes the proof of the statement on $\mathsf{EQ}_{i,j}(\pi_1, \pi_2)$. The proof of the statement on $\mathsf{LEQ}_{i,j}(\pi_1, \pi_2)$ is a direct application of Lemma 3.19, which states that $\pi_1(\mathsf{L}_i(t)) \leq \pi_2(\mathsf{L}_j(t))$ if and only if there exist sd-path formulas π_1', π_1'' and π_3 such that $\pi_1 = \pi_1' \cdot \pi_1''$, $\|\pi_3\| \leq |\mathscr{P}|$ and $\pi_1'(\mathsf{L}_i(t)) = \pi_3(\pi_2(\mathsf{L}_j(t)))$. This is true if and only if t satisfies $\bigvee_{\pi_1', \pi_3} \mathsf{EQ}_{i,j}(\pi_1', \pi_2 \cdot \pi_3)$. □

We can now complete the proof of the expressive completeness of LocPastPDL.

Proof of Theorem 3.18. We want to show that the constant trace formulas $\mathsf{L}_i \leq \mathsf{L}_j$ and $\mathsf{L}_{i,j} \leq \mathsf{L}_k$ are logically equivalent to LocPastPDL formulas.

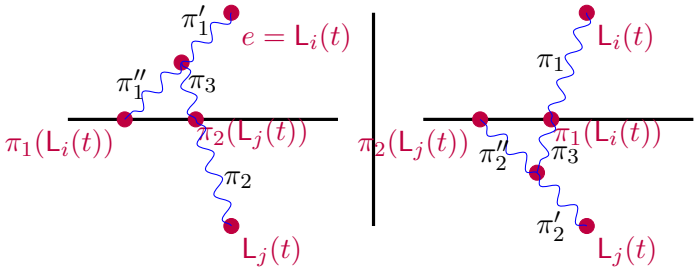

Figure 10: Applying Lemma 3.19 when $e = \mathsf{L}_i(t)$, $f_1 = \pi_1(\mathsf{L}_i(t))$, $f_2 = \pi_2(\mathsf{L}_i(t))$ and $f_1 < f_2$ (left) or $f_2 < f_1$ (right)

It is immediate that $\mathsf{L}_i \leq \mathsf{L}_j$ is equivalent to the LocPastPDL formula $\mathsf{LEQ}_{i,j}(\top?, \top?)$ from Lemma 3.20.

Now recall that t satisfies $\mathsf{L}_{i,j} \leq \mathsf{L}_k$ if and only if $\mathsf{L}_k(t)$, $\mathsf{L}_{i,j}(t)$ exist and $\mathsf{L}_{i,j}(t) \leq \mathsf{L}_k(t)$.

Notice that $\mathsf{L}_i(t)$ is a j-event if and only if $t \models \mathsf{EM}_i(\mathsf{on}_j)$. In this case, $\mathsf{L}_{i,j}(t) = \mathsf{L}_i(t)$. If $\mathsf{L}_i(t)$ is not a j-event, then $\mathsf{L}_{i,j}(t)$ is equal to $\mathsf{Y}_j(\mathsf{L}_i(t))$ and Proposition 3.6 asserts that $\mathsf{L}_{i,j}(t) = (\pi \cdot \mathsf{on}_j?)(\mathsf{L}_i(t))$ for some sd-path formula π of length at least 1 and at most $|\mathscr{P}| - 1$.

The existence of $\mathsf{L}_{i,j}(t)$ and $\mathsf{L}_k(t)$ is asserted by the sentence $\bigvee_\pi \mathsf{EM}_i(\langle\pi\rangle\mathsf{on}_j) \wedge \mathsf{EM}_k(\top)$, where the disjunction runs over the sd-path formulas π of length at most $|\mathscr{P}| - 1$.

This justifies considering the following LocPastPDL formula:

$$\Phi = \mathsf{EM}_k(\top) \wedge \bigvee_\pi \mathsf{EM}_i(\langle\pi\rangle\mathsf{on}_j) \wedge \left(\bigwedge_\pi \mathsf{EM}_i(\langle\pi\rangle\mathsf{on}_j) \Longrightarrow \mathsf{LEQ}_{i,k}(\pi \cdot \mathsf{on}_j?, \top?) \right),$$

where both the disjunction and the conjunction run over the sd-path formulas π of length at most $|\mathscr{P}| - 1$. As discussed, if t satisfies Φ, then it satisfies $\mathsf{L}_{i,j} \leq \mathsf{L}_k$.

Conversely, suppose that t satisfies $\mathsf{L}_{i,j} \leq \mathsf{L}_k$ and that $\mathsf{L}_k(t)$ and $\mathsf{L}_{i,j}(t)$ exist. Then every event of the form $(\pi \cdot \mathsf{on}_j?)(\mathsf{L}_i(t))$ lies below $\mathsf{L}_{i,j}(t)$, and hence below $\mathsf{L}_k(t)$. Thus t satisfies Φ. This establishes that $\mathsf{L}_{i,j} \leq \mathsf{L}_k$ is logically equivalent to Φ, which concludes the proof. □

4 Applications

4.1 Asynchronous devices and cascade product

We first quickly review the model of asynchronous automata due to Zielonka [21]. Besides using these automata to accept trace languages, we also use them, as in [18, 1, 2], to locally compute relabelling functions on input traces, similar in spirit to the sequential letter-to-letter transducers on words. The *local cascade product* of asynchronous automata from [1, 2] is a natural generalization of cascade product of sequential automata and, like in the word case [19], it corresponds to compositions of related relabelling functions.

Recall that, in keeping with our earlier notation, $\tilde{\Sigma} = \{\Sigma_i\}_{i \in \mathscr{P}}$ is a distributed alphabet over the set $\mathscr{P}$ of processes with total alphabet $\Sigma = \cup_{i \in \mathscr{P}} \Sigma_i$.

An *asynchronous automaton* $\mathcal{A}$ over $\tilde{\Sigma}$ (or simply over Σ) is a tuple $\mathcal{A} = (\{S_i\}_{i \in \mathscr{P}}, \{\delta_a\}_{a \in \Sigma}, s_{\text{in}})$ where

- for each $i \in \mathscr{P}$, S_i is a finite non-empty set of local i-states.

- for each $a \in \Sigma$, $\delta_a \colon S_a \to S_a$ is a deterministic joint transition function where $S_a = \prod_{i \in \mathsf{loc}(a)} S_i$ is the set of a-states.
- with $S = \prod_{i \in \mathscr{P}} S_i$ as the set of all global states, $s_{\mathrm{in}} \in S$ is a designated initial global state.

For a global state $s \in S$, we write $s = (s_a, s_{-a})$ where s_a is the projection of s on $\mathsf{loc}(a)$ and s_{-a} is the projection on the complement $\mathscr{P} \setminus \mathsf{loc}(a)$. For $a \in \Sigma$, the joint transition function $\delta_a \colon S_a \to S_a$ can be naturally extended to a global transition function $\Delta_a \colon S \to S$, on global states as follows: $\Delta_a((s_a, s_{-a})) = (\delta_a(s_a), s_{-a})$. Further we define, for a trace t over Σ, $\Delta_t \colon S \to S$ by letting Δ_ε be the identity function on S, the composition $\Delta_t = \Delta_a \circ \Delta_{t'}$ when $t = t'a$. The well-definedness of Δ_t follows easily from the fact that, for a pair (a, b) of independent letters, $\Delta_a \circ \Delta_b = \Delta_b \circ \Delta_a$. We denote by $\mathcal{A}(t)$ the global state $\Delta_t(s_{\mathrm{in}})$ reached when running $\mathcal{A}$ on t.

Asynchronous automata are used to accept trace languages. If F is a subset of the set S of global states, we say that $L(\mathcal{A}, F) = \{t \mid \mathcal{A}(t) \in F\}$ is the *trace language accepted by $\mathcal{A}$ with final states F*. A trace language is said to be *accepted by $\mathcal{A}$* if it is equal to $L(\mathcal{A}, F)$ for some $F \subseteq S$.

In this work, we use asynchronous automata also as letter-to-letter asynchronous transducers, which compute a relabelling function. Let Γ be a finite non-empty set. The set $\Sigma \times \Gamma$ naturally inherits a distribution from $\tilde{\Sigma}$: $\mathsf{loc}((a, \gamma)) = \mathsf{loc}(a)$. We denote by $\mathrm{Tr}(\Sigma)$ (resp. $\mathrm{Tr}(\Sigma \times \Gamma)$) the set of traces over corresponding alphabets. A function $\theta \colon \mathrm{Tr}(\Sigma) \to \mathrm{Tr}(\Sigma \times \Gamma)$ is called a *Γ-labelling function* if, for every $t = (E, \leq, \lambda) \in \mathrm{Tr}(\Sigma)$, $\theta(t) = (E, \leq, (\lambda, \mu)) \in \mathrm{Tr}(\Sigma \times \Gamma)$. Thus a Γ-labelling function simply decorates each event e of the trace t with a label $\mu(e)$ from Γ.

An *asynchronous Γ-transducer* over Σ is a tuple $\widehat{\mathcal{A}} = (\mathcal{A}, \{\mu_a\})$ where $\mathcal{A} = (\{S_i\}, \{\delta_a\}, s_{\mathrm{in}})$ is an asynchronous automaton and each μ_a ($a \in \Sigma$) is a map $\mu_a \colon S_a \to \Gamma$. The Γ-labelling function *computed* by $\widehat{\mathcal{A}}$ is the following map from $\mathrm{Tr}(\Sigma)$ to $\mathrm{Tr}(\Sigma \times \Gamma)$, also denoted by $\widehat{\mathcal{A}}$: for $t = (E, \leq, \lambda) \in \mathrm{Tr}(\Sigma)$, let $\widehat{\mathcal{A}}(t) = (E, \leq, (\lambda, \mu)) \in \mathrm{Tr}(\Sigma \times \Gamma)$ such that, for every $e \in E$ with $\lambda(e) = a$, $\mu(e) = \mu_a(s_a)$ where $s = \mathcal{A}(\Downarrow e)$.

An asynchronous automaton (resp. transducer) with local state sets $\{S_i\}$ is said to be *localized at process i* if all local state sets S_j with $j \neq i$ are singletons. It is *localized* if it is localized at some process. Note that, in a device localized at i, only process i carries non-trivial information and all non-trivial transitions are on letters in which process i participates.

Now we introduce the important notion of local cascade product of asynchronous transducers [4, 2].

Definition 4.1. Let $\widehat{\mathcal{A}}$ be a Γ-transducer over Σ and let $\widehat{\mathcal{B}}$ be a Π-transducer over $\Sigma \times \Gamma$, say, $\widehat{\mathcal{A}} = (\{S_i\}, \{\delta_a\}, s_{\mathrm{in}}, \{\mu_a\})$ and $\widehat{\mathcal{B}} = (\{Q_i\}, \{\delta_{(a,\gamma)}\}, q_{\mathrm{in}}, \{\nu_{(a,\gamma)}\})$. We define the local cascade product of $\widehat{\mathcal{A}}$ and $\widehat{\mathcal{B}}$ to be the $(\Gamma \times \Pi)$-transducer $\widehat{\mathcal{A}} \circ_\ell \widehat{\mathcal{B}} = (\{S_i \times Q_i\}, \{\nabla_a\}, (s_{\mathrm{in}}, q_{\mathrm{in}}), \{\tau_a\})$ over Σ where $\nabla_a((s_a, q_a)) = (\delta_a(s_a), \delta_{(a,\mu_a(s_a))}(q_a))$ and $\tau_a \colon S_a \times Q_a \to \Gamma \times \Pi$ is defined by $\tau_a((s_a, q_a)) = (\mu_a(s_a), \nu_{(a,\mu_a(s_a))}(q_a))$.

In the sequential case, that is, when $|\mathscr{P}| = 1$, the local cascade product coincides with the well-known operation of cascade product of sequential letter-to-letter transducers.

The following lemma (see [2]) is easily verified.

Lemma 4.2. *The $(\Gamma \times \Pi)$-labelling function computed by $\widehat{\mathcal{A}} \circ_\ell \widehat{\mathcal{B}}$ is the composition of the Γ-labelling function computed by $\widehat{\mathcal{A}}$ and the Π-labelling function computed by $\widehat{\mathcal{B}}$: for every $t \in \mathrm{Tr}(\Sigma)$,*

$$(\widehat{\mathcal{A}} \circ_\ell \widehat{\mathcal{B}})(t) = \widehat{\mathcal{B}}(\widehat{\mathcal{A}}(t))$$

4.2 LocPastPDL translation into cascade product

Now we exploit the locality and the natural hierarchical structure of LocPastPDL formulas to translate them into local cascade products of localized devices.

We start with the definition of a natural relabelling function associated with a collection F of LocPastPDL event formulas. Let $\Gamma_F = \{\top, \bot\}^F$. For each trace $t \in \mathrm{Tr}(\Sigma)$ and event e in t, we let $\mu_F(e) \in \Gamma_F$ be the tuple of truth-values of the formulas $\varphi \in F$ at e, and we let θ^F be the Γ_F-labelling function given by $\theta^F(t) = (E, \leq, (\lambda, \mu_F))$, for each $t = (E, \leq, \lambda) \in \mathrm{Tr}(\Sigma)$.

We can now state an important result on event formulas in LocPastPDL. This result, without an analysis of global states of the resulting construction, is already implicit in [3].

Theorem 4.3. *Let φ be a* LocPastPDL *event formula and θ^φ be the corresponding $\{\top, \bot\}$-labelling function. Then θ^φ can be computed by a local cascade product of localized asynchronous transducers. The number of global states in this transducer is $2^{\mathcal{O}(|\varphi|)}$.*

Let us highlight the main ideas required to prove Theorem 4.3. The proof proceeds by structural induction on φ, and constructs a cascade product $\mathcal{A}_\varphi$ of localized transducers which computes θ^φ. The most non-trivial case is when $\varphi = \langle \pi \rangle$ where π is an i-local path formula. Recall that π is a regular expression involving top-level moves $\leftarrow_i$ and test formulas. Let F be the set of event formulas which appear in these test formulas. By induction, for each $\psi \in F$, θ^ψ can be computed by cascade product $\mathcal{A}_\psi$ of localized transducers. By a direct product construction, which may be seen as a special case of cascade product, we can construct a transducer of the desired form, say $\mathcal{A}_F$, which computes Γ_F-labelling function θ^F. We finally construct $\mathcal{A}_\varphi$ as a cascade product of $\mathcal{A}_F \circ_\ell \mathcal{B}$ where $\mathcal{B}$ is localized at i. In order to construct $\mathcal{B}$, we first convert π into a deterministic finite-state automata $\mathcal{B}_i$ over alphabet Γ_F. Now we obtain the asynchronous transducer by localizing $\mathcal{B}_i$ at process i and suitably designing a labelling function which computes θ^φ. See [3] for more details.

The complexity bound is also proved by structural induction. The sizes $|\varphi|$ and $|\pi|$ of event and path formulas are defined inductively as expected, with $|\psi?| = 1 + |\psi|$. For a path expression π, we also define the top-level size $||\pi||$ which does not take into account the size of the event formulas tested in π: $||\psi?|| = 1$. The transducers for atomic formulas $\varphi = a$ with $a \in \Sigma$ have a single global state. The transducers for the boolean connectives $\neg, \vee, \wedge$ also have a single global state. Consider now the non-trivial case $\varphi = \langle \pi \rangle$ described above. By induction, the number of global states of each $\mathcal{A}_\psi$ is $2^{\mathcal{O}(|\psi|)}$. We get the local cascade product $\mathcal{A}_F$ with $2^{\mathcal{O}(||F||)}$ global states, where $||F|| = \sum_{\psi \in F} |\psi|$. Finally, we translate the regular expression π, considering test formulas $\psi?$ for $\psi \in F$ as uninterpreted symbols, to a DFA with $2^{\mathcal{O}(||\pi||)}$ many states. A final cascade product yields $\mathcal{A}_\varphi$ with $2^{\mathcal{O}(|\varphi|)}$ many global states.

Now we turn our attention to translating LocPastPDL sentences into automata. A basic trace formula Φ is of the form $\mathsf{EM}_i\, \varphi$ where φ is an event formula. A trace t satisfies $\mathsf{EM}_i\, \varphi$ if the last i-event exists and φ holds at this event. By Theorem 4.3, we have a local cascade product $\widehat{\mathcal{A}}$ of localized transducers which computes θ^φ and thus, records the truth value of φ at every event. It is easy to convert $\widehat{\mathcal{A}}$ into an asynchronous automaton which checks at the final global state if the last i-event exists and whether φ evaluates to $\top$ at this event. As an arbitrary trace formula is simply a boolean combination of formulas of the form $\mathsf{EM}_i\, \varphi$, which can be handled by a direct product construction, we get the following result.

Theorem 4.4. *Let Φ be a* LocPastPDL *sentence. We can construct a local cascade product of localized asynchronous automata which precisely accepts the language defined by Φ. The number of global states is $2^{\mathcal{O}(|\Phi|)}$.*

4.3 Zielonka's theorem, distributed Krohn-Rhodes theorem and a Gossip implementation

Now we present some important applications of the expressive completeness of LocPastPDL. The next theorem provides a new proof of the fundamental theorem of Zielonka which characterizes regular trace languages using asynchronous automata.

Theorem 4.5. *A trace language is regular if and only if it is accepted by a local cascade product of localized asynchronous automata. In particular, every regular trace language can be accepted by an asynchronous automaton.*

Proof. Let L be a regular trace language. By Theorem 3.1, there is a LocPastPDL sentence Φ which defines L. By Theorem 4.4, we can construct a local cascade product of localized automata which accepts the language defined by Φ, that is, L.

Conversely, a local cascade product of localized automata is an asynchronous automaton, which is known to accept a regular trace language. □

Another important application is a novel distributed Krohn-Rhodes theorem. Recall that the classical Krohn-Rhodes theorem [15] states that every sequential automaton can be simulated by a *cascade product* of simple automata, namely, two-state reset automata and permutation automata. In a two-state reset automaton, the transition function of each letter is either the identity function or a constant function. In contrast, in a permutation automaton, each letter induces a permutation of the state set.

Let $i \in \mathscr{P}$. A two-state reset automaton localized at process i is an asynchronous automaton localized at i which has two i-local states and the local transition on each letter is either the identity function or a constant. Similarly, in a localized permutation automaton, each letter of the active process induces a permutation of its local state set while the other local state sets are singletons.

Theorem 4.6. *Every regular trace language can be accepted by a local cascade product of localized two-state reset automata and localized permutation automata.*

Proof. Note that, by essentially using the classical Krohn-Rhodes theorem, an asynchronous automaton localized at process i can be simulated by a local cascade product of two-state reset automata localized at i and permutation automata localized at i. Therefore, the theorem follows by Theorem 4.5. □

Our final application concerns an implementation of a gossip automaton/transducer. Let $\mathcal{G}$ be the collection of constant event formulas $\{\mathsf{Y}_i \leq \mathsf{Y}_j\}_{i,j \in \mathscr{P}}$. Consider the $\Gamma_{\mathcal{G}}$-labelling function. Note that $\Gamma_{\mathcal{G}}$ records, for every trace and at each event of that trace, the truth values of the formulas in $\mathcal{G}$ in the form of an additional label from $\Gamma_{\mathcal{G}}$. So $\theta^{\mathcal{G}}$ keeps track of the ordering information between leading process events. By a gossip transducer, we mean any asynchronous transducer which computes $\theta^{\mathcal{G}}$ and hence keeps track of the latest gossip/information [18] in a distributed environment.

Theorem 4.7. *The $\Gamma_{\mathcal{G}}$-labelling function $\theta^{\mathcal{G}}$ can be computed by an asynchronous transducer which is a local cascade product of localized asynchronous transducers.*

Proof. By Theorem 3.8, event formulas in $\mathcal{G}$ can be expressed in LocPastPDL. Further, by Theorem 4.3, for each of these LocPastPDL formulas φ, the corresponding θ^{φ} can be computed by a local cascade product of localized asynchronous transducers. Combining these together through a direct product, we obtain an asynchronous transducer of the desired form which computes $\theta^{\mathcal{G}}$. □

The existing constructions [21, 18] of a gossip transducer are intrinsically non-local and non-hierarchical. It is important to note that our construction is quite inefficient in terms of the size of the resulting transducer. The key point of our construction is to show the existence of a gossip implementation using compositions of labelling functions each of which is localized at some process. It was not at all clear that such an implementation could exist!

5 Conclusion

Our main result is the identification of a local, past-oriented fragment of propositional dynamic logic for traces called LocPastPDL, which is expressively complete with respect to regular trace languages. The natural hierarchical structure of LocPastPDL formulas allows a modular translation into a cascade product of localized automata. We have also given many important applications such as a new proof of Zielonka's theorem, a novel distributed version of Krohn-Rhodes theorem and a hierarchical implementation of a gossip automaton.

A natural question is to identify a fragment of LocPastPDL which matches the first-order logic in expressive power. Another promising future direction is to characterize the precise power of local cascade products of localized two-state reset automata. Extending these results with future-oriented path formulas, and to the setting of infinite traces are also interesting directions.

It remains to investigate the utility of LocPastPDL specifications from a more practical and empirical viewpoint. Our modular and efficient translation of these specifications should provide significant improvements in applications such as synthesizing distributed monitors and verifying properties of concurrent programs.

References

[1] Bharat Adsul, Paul Gastin, Saptarshi Sarkar, and Pascal Weil. Wreath/cascade products and related decomposition results for the concurrent setting of Mazurkiewicz traces. In Igor Konnov and Laura Kovács, editors, *31st International Conference on Concurrency Theory, CONCUR 2020, September 1-4, 2020, Vienna, Austria (Virtual Conference)*, volume 171 of *LIPIcs*, pages 19:1–19:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[2] Bharat Adsul, Paul Gastin, Saptarshi Sarkar, and Pascal Weil. Asynchronous wreath product and cascade decompositions for concurrent behaviours. *Log. Methods Comput. Sci.*, 18, 2022.

[3] Bharat Adsul, Paul Gastin, Saptarshi Sarkar, and Pascal Weil. Propositional dynamic logic and asynchronous cascade decompositions for regular trace languages. 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 28:1–28:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. URL: `https://doi.org/10.4230/LIPIcs.CONCUR.2022.28`, `doi:10.4230/LIPICS.CONCUR.2022.28`.

[4] Bharat Adsul and Milind A. Sohoni. Asynchronous automata-theoretic characterization of aperiodic trace languages. In *FSTTCS 2004: Foundations of Software Technology and Theoretical Computer Science, 24th International Conference, Chennai, India, December 16-18, 2004, Proceedings*, pages 84–96, 2004. `doi:10.1007/978-3-540-30538-5\_8`.

[5] Benedikt Bollig, Marie Fortin, and Paul Gastin. Communicating finite-state machines, first-order logic, and star-free propositional dynamic logic. *J. Comput. Syst. Sci.*, 115:22–53, 2021. doi:10.1016/j.jcss.2020.06.006.

[6] Benedikt Bollig, Dietrich Kuske, and Ingmar Meinecke. Propositional dynamic logic for message-passing systems. *Log. Methods Comput. Sci.*, 6(3), 2010. URL: http://arxiv.org/abs/1007.4764.

[7] Giuseppe De Giacomo and Moshe Y Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *Twenty-Third International Joint Conference on Artificial Intelligence*, 2013.

[8] Volker Diekert and Yves Métivier. Partial commutation and traces. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages, Volume 3: Beyond Words*, pages 457–533. Springer, 1997. doi:10.1007/978-3-642-59126-6_8.

[9] Volker Diekert and Grzegorz Rozenberg, editors. *The Book of Traces*. World Scientific, 1995. doi:10.1142/2563.

[10] Werner Ebinger and Anca Muscholl. Logical definability on infinite traces. *Theor. Comput. Sci.*, 154(1):67–84, 1996.

[11] Michael J Fischer and Richard E Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, 18(2):194–211, 1979.

[12] Giuseppe De Giacomo, Antonio Di Stasio, Francesco Fuggitti, and Sasha Rubin. Pure-past linear temporal and dynamic logic on finite traces. In Christian Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 4959–4965. ijcai.org, 2020. URL: https://doi.org/10.24963/ijcai.2020/690, doi:10.24963/IJCAI.2020/690.

[13] Giovanna Guaiana, Antonio Restivo, and Sergio Salemi. Star-free trace languages. *Theor. Comput. Sci.*, 97(2):301–311, 1992.

[14] Nils Klarlund, Madhavan Mukund, and Milind A. Sohoni. Determinizing asynchronous automata. In Serge Abiteboul and Eli Shamir, editors, *Automata, Languages and Programming, 21st International Colloquium, ICALP94, Jerusalem, Israel, July 11-14, 1994, Proceedings*, volume 820 of *Lecture Notes in Computer Science*, pages 130–141. Springer, 1994. doi:10.1007/3-540-58201-0_63.

[15] Kenneth Krohn and John Rhodes. Algebraic theory of machines I. Prime decomposition theorem for finite semigroups and machines. *Transactions of The American Mathematical Society*, 116, 04 1965. doi:10.2307/1994127.

[16] Antoni Mazurkiewicz. Concurrent program schemes and their interpretations. *DAIMI Report Series*, 6(78), Jul. 1977. doi:10.7146/dpb.v6i78.7691.

[17] Roy Mennicke. Propositional dynamic logic with converse and repeat for message-passing systems. *Log. Methods Comput. Sci.*, 9(2), 2013. doi:10.2168/LMCS-9(2:12)2013.

[18] Madhavan Mukund and Milind A. Sohoni. Keeping track of the latest gossip in a distributed system. *Distributed Comput.*, 10(3):137–148, 1997. doi:10.1007/s004460050031.

[19] Howard Straubing. *Finite automata, formal logic, and circuit complexity*. Birkhaüser Verlag, Basel, Switzerland, 1994.

[20] Wolfgang Thomas. On logical definability of trace languages. In V. Diekert, editor, *Proceedings of a workshop of the ESPRIT Basic Research Action No 3166: Algebraic and Syntactic Methods in Computer Science (ASMICS), Kochel am See, Bavaria, FRG (1989)*, Report TUM-I9002, Technical University of Munich, pages 172–182, 1990.

[21] Wieslaw Zielonka. Notes on finite asynchronous automata. *RAIRO Theor. Informatics Appl.*, 21(2):99–135, 1987. doi:10.1051/ita/1987210200991.