


Characterizations of Fragments of Temporal Logic over Mazurkiewicz Traces

Bharat Adsul 

IIT Bombay, Mumbai, India

Paul Gastin 

Université Paris-Saclay, ENS Paris-Saclay, CNRS, LMF, 91190, Gif-sur-Yvette, France
CNRS, ReLaX, IRL 2000, Siruseri, India

Shantanu Kulkarni 

IIT Bombay, Mumbai, India

Abstract

We study fragments of temporal logics over Mazurkiewicz traces which are well known and established partial-order models of concurrent behaviours. We focus on concurrent versions of “strict past” and “strict future” modalities. Over words, the corresponding fragments have been shown to coincide with natural algebraic conditions on the recognizing monoids. We provide non-trivial generalizations of these classical results to traces. We exploit the local nature of the temporal modalities and obtain modular translations of specifications into asynchronous automata. More specifically, we provide novel characterizations of these fragments via local cascade products of a very simple two-state asynchronous automaton operating on a single process.

2012 ACM Subject Classification Theory of computation → Modal and temporal logics; Theory of computation → Distributed computing models; Theory of computation → Algebraic language theory; Security and privacy → Logic and verification

Keywords and phrases Mazurkiewicz traces, temporal logics, asynchronous automata, cascade product, Green’s relations, algebraic automata theory

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2025.5

1 Introduction

Temporal logics are widely used logical formalisms to reason about properties of computations. Words serve as temporal models of sequential computations/behaviours and linear temporal logic LTL formulas use natural future as well as past temporal modalities to talk about interesting patterns of these behaviours. The celebrated theorem of Kamp [16] showed that, over words, properties definable in LTL coincide with those definable in the first order (FO) logic. By the seminal results of McNaughton-Papert [18] and Schutzenberger [20], these also equal regular languages describable by star-free regular expressions and those recognized by finite aperiodic monoids.

The above mentioned results clearly indicate the important role played by the temporal modalities in the very rich expressive power of LTL. Natural and commonly used temporal modalities include future modalities X (“neXt/tomorrow”), F (“Future”), U (“Until”), the past counterparts Y (“previous/Yesterday”), P (“Past”), S (“Since”) and their strict variants such as XF (“strict future”) and YP (“strict past”). Several works [7], [14], [5] have studied natural fragments of LTL obtained by allowing only some of these modalities and have characterized these fragments in terms of algebraic conditions on the recognizing finite monoids. These algebraic characterizations have been crucially used to show that the computational problems of definability in the associated fragments are decidable.

Let us consider the fragment LTL[YP] obtained by allowing only unary strict past modality YP. When interpreted over words, the formula YP φ , at a given position, asserts that φ holds at some position in the strict past (that is, strictly left) of the current position. It has



© Bharat Adsul, Paul Gastin, and Shantanu Kulkarni;
licensed under Creative Commons License CC-BY 4.0

36th International Conference on Concurrency Theory (CONCUR 2025).

Editors: Patricia Bouyer and Jaco van de Pol; Article No. 5; pp. 5:1–5:20

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

been shown in [6], [7], [14], [5] that a language is definable in $\text{LTL}[\text{YP}]$ iff it can be recognized by a \mathcal{R} -trivial monoid iff the syntactic monoid of the language is \mathcal{R} -trivial. Here \mathcal{R} stands for one of fundamental Green's relations on a monoid. The \mathcal{R} -triviality condition on a monoid M simply means that if two elements of M are right multiples of each other then they are equal (that is, for all $m, n \in M$, $\exists p, q \in M : n = m \cdot p$ and $m = n \cdot q$ implies $m = n$). This characterization can also be stated in simpler automata-theoretic terms [6]. A finite-state automaton is said to be partially ordered if there is a partial order \succeq on the states so that every transition is of the form (p, a, q) where $p \succeq q$. Said differently, an automaton is partially ordered iff the only cycles present in the underlying directed transition graph are self loops. Using this notion, the above mentioned characterization can be stated as follows: a language is $\text{LTL}[\text{YP}]$ -definable iff it can be accepted by a partially ordered automaton.

It turns out that the dual fragment $\text{LTL}[\text{XF}]$ which admits the unary strict future modality XF is characterized by the dual condition of \mathcal{L} -triviality. As expected, a monoid is \mathcal{L} -trivial if no two distinct elements can be left multiples of each other. The fragment $\text{LTL}[\text{XF}, \text{YP}]$ which includes both XF and YP modalities has been well studied and is known to coincide with important first order logic fragments, namely, FO^2 and Δ_2 . It also has an algebraic characterization in terms of monoids from the class DA . See the survey [9] for more details.

In this work we study the above mentioned temporal logic fragments from the concurrency viewpoint. We consider Mazurkiewicz traces as the underlying models of concurrent behaviours. A Mazurkiewicz trace or simply trace, over a set of processes, represents a concurrent behaviour as a labelled partial-order between events. This partial-order between events faithfully captures the *causal* information flow among processes. Traces are extensively studied and there is a rich theory of *regular* trace languages [23], [12]. Regular trace languages are known to coincide with MSO-definable languages as well as languages recognized by finite monoids. We consider natural *event-based* interpretations of temporal modalities XF and YP over traces. For instance, the formula $\text{YP } \varphi$, at a given event, asserts that φ holds at an event which is in the strict causal past of the current event. We denote by $\text{LocTL}[\text{YP}]$ the local/event-based temporal logic over traces which has access to the temporal modality YP . One can similarly define $\text{LocTL}[\text{XF}]$ and $\text{LocTL}[\text{XF}, \text{YP}]$. One of our main results states that a trace language is definable in $\text{LocTL}[\text{YP}]$ iff it can be recognized by a \mathcal{R} -trivial monoid. This can be viewed as a non-trivial generalization of the corresponding result over words. We also obtain an analogous algebraic characterization of $\text{LocTL}[\text{XF}]$ in terms of \mathcal{L} -trivial monoids. An algebraic characterization of $\text{LocTL}[\text{XF}, \text{YP}]$ in terms of monoids from the class DA already appears in [17].

Another important contribution of this work is a *modular* translation of formulas in these fragments into *asynchronous automata*. In an asynchronous automaton, each process runs a finite local-state device and these devices synchronize with each other on shared actions. Asynchronous automata can be naturally used to accept trace languages. Zielonka's theorem states that regular trace languages are precisely the languages accepted by asynchronous automata. As in [19, 2, 3], one can use asynchronous automata as transducers on traces, similar in spirit to the sequential letter-to-letter transducers on words. Unlike these works which use *deterministic* asynchronous transducers to locally compute relabelling functions on input traces, we allow non-deterministic asynchronous transducers. As expected, these transducers compute relabelling relations from input traces to output traces. The *local cascade product* of deterministic asynchronous automata (or transducers) from [2, 3] is a natural generalization of cascade product in the sequential setting [21]. It further extends naturally to the non-deterministic setting and corresponds to compositions of related relabelling relations.

We use this machinery to provide a translation (in fact, a characterization) of $\text{LocTL}[\text{XF}, \text{YP}]$ formulas into local cascade products of very simple two-state asynchronous automata operating on a single process. For $\text{LocTL}[\text{YP}]$ formulas, the resulting atomic two-state transducers are deterministic while for $\text{LocTL}[\text{XF}]$ formulas, they are reverse-deterministic. We remark that our results and their proofs for traces specialize to the setting of words.

The rest of the paper is organized as follows. After setting up the preliminary notions in Section 2, we establish in Section 3 that \mathcal{R} -trivial trace languages are definable in $\text{LocTL}[\text{YP}]$. In Section 4, we provide a translation of $\text{LocTL}[\text{YP}]$ formulas into cascade products of *localized* deterministic asynchronous transducers/automata whose transition monoid is isomorphic to U_1 - the unique two element aperiodic monoid. Section 5 is devoted to dual results concerning \mathcal{L} -trivial and $\text{LocTL}[\text{XF}]$ -definable trace languages. We also sketch a cascade product based characterization of $\text{LocTL}[\text{XF}, \text{YP}]$ in this section.

2 Preliminaries

In this section, we set up the notation and review basics of traces. We also recall the recognizability of trace languages by morphisms into monoids and define the fundamental Green's relations on monoids.

2.1 Trace basics

Let \mathcal{P} be a finite, non-empty set of processes. For every process $i \in \mathcal{P}$, Σ_i is the set of letters/actions in which process i participates. A distributed alphabet over \mathcal{P} is the family $\tilde{\Sigma} = \{\Sigma_i\}_{i \in \mathcal{P}}$ with $\Sigma = \cup_{i \in \mathcal{P}} \Sigma_i$ being the associated total alphabet. Letters $a, b \in \Sigma$ are dependent if there is a process which participates in both of them ($\exists i \in \mathcal{P}, a, b \in \Sigma_i$), otherwise they are independent. For a letter $a \in \Sigma$ we define $\text{loc}(a) = \{i \in \mathcal{P} \mid a \in \Sigma_i\}$ to be the set of processes that participate in a .

A poset is a tuple (E, \leq) with E being the underlying set and \leq being the partial order on E . A Σ -labelled poset over $\tilde{\Sigma}$ is a triple (E, \leq, λ) where (E, \leq) is the underlying poset, and $\lambda: E \rightarrow \Sigma$ is the labelling function which labels each element in E with a letter from Σ . For elements $e, e' \in E$, e' is said to be an *immediate successor* of e , denoted by $e < e'$, if e is below e' ($e < e'$) and there is no event f strictly between e and e' ($\nexists f \in E, e < f < e'$).

A (Mazurkiewicz) trace $t = (E, \leq, \lambda)$ over $\tilde{\Sigma}$ is a finite Σ -labelled poset with λ satisfying the following two properties:

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

For a trace $t = (E, \leq, \lambda)$ we sometimes slightly abuse notation to say event $e \in t$ which we take to mean that event $e \in E$. The set of process i -events in t is defined as $E_i = \{e \in E \mid \lambda(e) \in \Sigma_i\}$. Note that E_i is totally ordered by \leq . For an event $e \in t$, we slightly abuse the notation and define $\text{loc}(e) = \text{loc}(\lambda(e))$ to be the set of processes that participate in e . For an event e we define $\downarrow e = \{f \in t \mid f \leq e\}$ to be the set of events in t below e . Similarly $\Downarrow e = \{f \in t \mid f < e\}$ is the set of events in t strictly below e . It is easy to see that by restricting \leq and λ to events in $\downarrow e$, we obtain a valid trace. By abuse of notation, we denote this trace by $(\downarrow e, \leq, \lambda)$ or simply $\downarrow e$. It will be clear from the context whether we mean $\downarrow e$ to be a trace or merely a set of events. Similarly, $\Downarrow e$ also refers to the trace $(\Downarrow e, \leq, \lambda)$ induced by $\Downarrow e$.

The set of all finite traces over $\tilde{\Sigma}$ is denoted by $TR(\tilde{\Sigma})$. We define the *trace concatenation* operation \cdot on $TR(\tilde{\Sigma})$ as follows. Let $t_1 = (E_1, \leq_1, \lambda_1), t_2 = (E_2, \leq_2, \lambda_2) \in TR(\tilde{\Sigma})$ with $E_1 \cap E_2 = \emptyset$. We define $t_1 \cdot t_2$ or simply $t_1 t_2$ to be the trace $(E_1 \cup E_2, \leq, \lambda) \in TR(\tilde{\Sigma})$ where:

1. \leq is the transitive closure of $\leq_1 \cup \leq_2 \cup \{(e_1, e_2) \in E_1 \times E_2 \mid \lambda_1(e_1), \lambda_2(e_2) \text{ are dependant}\}$.
2. For all $e \in E_1 \cup E_2$, $\lambda(e) = \lambda_1(e)$ if $e \in E_1$ and $\lambda_2(e)$ if $e \in E_2$.

A trace t' is said to be a prefix of trace t if there exists $t'' \in TR(\tilde{\Sigma})$ such that $t = t't''$. It is easy to see that, for $e \in t$, $\downarrow e$ and $\Downarrow e$ are both prefixes of t .

2.2 Recognizable trace languages

Recall that a monoid is a set M equipped with a binary associative operation \cdot and containing an identity element 1_M , such that $\forall m \in M, m \cdot 1_M = 1_M \cdot m = m$. The set $TR(\tilde{\Sigma})$, under the operation of *trace concatenation*, is a monoid with the empty trace ε as the identity element. Let M and N be two monoids with operations \cdot and \circ respectively. A *morphism* h from M to N is a map $h: M \rightarrow N$ such that: $h(1_M) = 1_N$ and for all $m_1, m_2 \in M$, $h(m_1 \cdot m_2) = h(m_1) \circ h(m_2)$.

A subset $L \subseteq TR(\tilde{\Sigma})$ is called a *trace language* over $\tilde{\Sigma}$. We call a morphism from the trace monoid $TR(\tilde{\Sigma})$ to any other monoid M as a trace morphism. A trace language $L \subseteq TR(\tilde{\Sigma})$ is recognizable if there exists a trace morphism h from $TR(\tilde{\Sigma})$ to a *finite* monoid M such that $L = \cup_{m \in P} h^{-1}(m)$ for some $P \subseteq M$. We may refer to h as the recognizing morphism or say that L is recognized by M via recognizing morphism h . The class of recognizable trace languages, also referred to as *regular trace languages*, has several characterizations and we note a few in particular, namely MSO-definable languages over traces [22] and trace languages accepted by asynchronous automata [23].

Classically, the algebraic viewpoint in terms of what are known as *Green's relations* over monoids has been fruitful for characterizing interesting subclasses of regular languages which have an intimate connection with logics. We now briefly recall these fundamental Green's preorders and the associated equivalence relations.

Let M be a monoid with \cdot as the binary monoid operation and let $m_1, m_2 \in M$. The Green's preorder relations are defined as follows:

1. $m_1 \leq_R m_2$ (read as m_1 is R -below m_2) if there exists $m \in M$ such that $m_1 = m_2 \cdot m$.
2. $m_1 \leq_L m_2$ if there exists $m \in M$ such that $m_1 = m \cdot m_2$.
3. $m_1 \leq_J m_2$ if there exist $u, v \in M$ such that $m_1 = u \cdot m_2 \cdot v$
4. $m_1 \leq_H m_2$ if $m_1 \leq_R m_2$ and $m_1 \leq_L m_2$

These preorders naturally give rise to the associated equivalence relations $\mathcal{R}, \mathcal{L}, \mathcal{J}, \mathcal{H}$. We say that $m_1 \mathcal{R} m_2$ (read as m_1 is \mathcal{R} -equivalent to m_2) if $m_1 \leq_R m_2$ and $m_2 \leq_R m_1$. A monoid M is \mathcal{R} -trivial if $\forall m_1, m_2 \in M, m_1 \mathcal{R} m_2$ implies $m_1 = m_2$. In other words, M is \mathcal{R} -trivial iff the preorder \leq_R on M is in fact a partial order on M iff the equivalence relation \mathcal{R} coincides with the identity relation on M .

The other equivalence relations $\mathcal{L}, \mathcal{J}, \mathcal{H}$ and their triviality for M are defined analogously. A trace language $L \subseteq TR(\tilde{\Sigma})$ is called \mathcal{R} -trivial if it can be recognized by a finite \mathcal{R} -trivial monoid. One can similarly define \mathcal{L} -trivial, \mathcal{J} -trivial and \mathcal{H} -trivial trace languages.

► **Example 1.** Consider the monoid $U_1 = \{1, 0\}$ with 1 as the identity and 0 as the zero element. So, $1 \cdot 1 = 1, 0 \cdot 1 = 1 \cdot 0 = 0 \cdot 0 = 0$. It is easily checked that U_1 is \mathcal{R} -trivial, \mathcal{L} -trivial, \mathcal{J} -trivial as well as \mathcal{H} -trivial.

As mentioned in the introduction, for word languages, \mathcal{R} -triviality (\mathcal{L} -triviality) corresponds to definability in LTL with only the strict past (respectively strict future) modality. See [5] for a proof based on Green's relations. It turns out that, for finite monoids,

\mathcal{H} -triviality condition is equivalent to the aperiodicity property. As a consequence, \mathcal{H} -trivial word languages coincide with languages definable in the first order logic and linear temporal logic. These results find interesting generalizations over traces in [15, 13, 8].

3 Algebra to temporal logic

We now state our main theorem and set up the technical machinery and prove it.

► **Theorem 2.** *Let $L \subseteq TR(\tilde{\Sigma})$. Then the following are equivalent.*

1. *L is a \mathcal{R} -trivial trace language.*
2. *L can be expressed using a sentence in $\text{LocTL}[\text{YP}]$.*
3. *L can be accepted by a cascade product of localized U_1 asynchronous automata.*

Proof. $1 \rightarrow 2$ is shown in Theorem 14, $2 \rightarrow 3$ is shown in Theorem 22, $3 \rightarrow 1$ is shown in Theorem 24. ◀

The result we wish to prove in this section is the $1 \rightarrow 2$ for Theorem 2, which says that any trace language which is recognized by a \mathcal{R} -trivial monoid can be expressed as a sentence in $\text{LocTL}[\text{YP}]$. This statement is restated as Theorem 14. We start by defining formally the syntax and semantics of $\text{LocTL}[\text{YP}]$.

3.1 $\text{LocTL}[\text{YP}]$ syntax and semantics

We now define $\text{LocTL}[\text{YP}]$. We have sentences Φ , which are evaluated over finite Mazurkiewicz traces and hence define languages over $TR(\tilde{\Sigma})$ and also event formulas φ which are evaluated at a given event in a trace. The syntax is as follows:

$$\begin{aligned}\Phi &::= E\varphi \mid \Phi_1 \vee \Phi_2 \mid \neg\Phi \\ \varphi &::= a \mid \varphi_1 \vee \varphi_2 \mid \neg\varphi \mid \text{YP}\varphi\end{aligned}$$

The semantics for the boolean operations are as usual. For a trace $t = (E, \leq, \lambda)$ and an event $e \in E$, the following defines the semantics of $\text{LocTL}[\text{YP}]$

$$\begin{aligned}t &\models E\varphi && \text{if there exists an event } e \text{ in } t \text{ such that } t, e \models \varphi \\ t, e &\models a && \text{if } \lambda(e) = a \\ t, e &\models \text{YP}\varphi && \text{if there exists an event } f \text{ in } t \text{ with } f < e \text{ and } t, f \models \varphi.\end{aligned}$$

In the rest of the paper we shall use the following macros liberally:

$$\begin{aligned}A\varphi &= \neg E\neg\varphi, & P\varphi &= \varphi \vee \text{YP}\varphi, \\ YH\varphi &= \neg \text{YP}\neg\varphi, & H\varphi &= \varphi \wedge YH\varphi.\end{aligned}$$

The modalities can be read as Exists (E), All (A), Past (P), Historically (H), Yesterday Past (YP), Yesterday Historically (YH). For a set of letters A , we may abuse notation and use A to denote the event formula $\bigvee_{a \in A} a$ testing whether the event is labelled by a letter in A . For instance, we may use Σ_i as the macro for the event formula $\bigvee_{a \in \Sigma_i} a$ which tests whether the current event is an i -event.

► **Example 3.** Let $t \in TR(\tilde{\Sigma})$ be a trace and $e \in t$ be an event. We design an event formula φ_k^i such that $t, e \models \varphi_k^i$ iff there are at least k process i -events strictly below event e . We define φ_k^i inductively as: $\varphi_0^i = \top$ and for $k > 0$, $\varphi_k^i = \text{YP}(\Sigma_i \wedge \varphi_{k-1}^i)$. Consider the $\text{LocTL}[\text{YP}]$ sentence $\Phi_{m,n} = E(\Sigma_j \wedge \varphi_{n-1}^j \wedge \neg\varphi_n^j \wedge \varphi_m^i)$. Clearly $\Phi_{m,n}$ has the property that trace t satisfies $\Phi_{m,n}$ if and only if the m^{th} process i -event is strictly below the n^{th} process j -event.

3.2 Critical structures for a morphism into a \mathcal{R} -trivial monoid

Throughout the rest of this section, we fix a morphism $h: TR(\tilde{\Sigma}) \rightarrow M$ to a fixed finite \mathcal{R} -trivial monoid M .

Let $t = (E, \leq, \lambda)$ be a trace and $e \in E$ be an event of t . Note that, viewing $\downarrow e$ and $\Downarrow e$ as traces, we have $h(\downarrow e) = h(\Downarrow e)h(\lambda(e)) \leq_R h(\Downarrow e)$. The event e is said to be a *critical event* in t if $h(\downarrow e) \neq h(\Downarrow e)$. Since the monoid M is \mathcal{R} -trivial, this is equivalent to $h(\downarrow e) <_R h(\Downarrow e)$. We let $X_t = \{e \in E \mid h(\downarrow e) <_R h(\Downarrow e)\}$ be the set of all critical events of trace t .

► **Lemma 4.** *In the above notation, $|X_t| < |\mathcal{P}| \times |M|$.*

Proof. We show that the number of critical events on a single process line is less than $|M|$. This clearly implies that $|X_t|$ is less than $|\mathcal{P}| \times |M|$. For any $i \in \mathcal{P}$, the set of critical process i events is $X_t \cap E_i$. Assume $X_t \cap E_i = \{e_1, e_2, \dots, e_k\}$ with $e_1 < e_2 < \dots < e_k$ (E_i is totally ordered by \leq). Since e_j are critical events we have $h(\downarrow e_j) <_R h(\Downarrow e_j)$. Since (the trace induced by) $\downarrow e_j$ is a prefix of (the trace induced by) $\Downarrow e_{j+1}$, we have $h(\Downarrow e_{j+1}) \leq_R h(\downarrow e_j)$. We get

$$h(\downarrow e_k) <_R h(\Downarrow e_k) \leq_R h(\downarrow e_{k-1}) <_R h(\Downarrow e_{k-1}) \leq_R \dots h(\downarrow e_1) <_R h(\Downarrow e_1) \leq_R 1_M.$$

Therefore $k < |M|$ which concludes the proof. ◀

A configuration of t is a subset $C \subseteq E$ of events which is downward closed: $C = \downarrow C$. Clearly, for any event $e \in E$, $\downarrow e$ and $\Downarrow e$ are configurations of t . We sometimes abuse the notation and use C to also denote the trace induced by a configuration C of t . It is easy to see that $t = C \cdot t'$ where t' is the trace induced by the set $E \setminus C$ of events of t which do not belong to C .

A configuration C of t is *critical* if for every maximal event $e \in C$, we have $h(C) \neq h(C \setminus \{e\})$, equivalently, $h(C) <_R h(C \setminus \{e\})$. Note that if e is a maximal event of C then $C \setminus \{e\}$ is a configuration of t .

► **Lemma 5.** *Every maximal event of a critical configuration is a critical event.*

Proof. Suppose for contradiction that there exists a critical configuration C and a maximal event $e \in C$ such that $h(C) <_R h(C \setminus \{e\})$ but $h(\downarrow e) = h(\Downarrow e)$. We have that $h(C) = h(\downarrow e) \cdot h(C \setminus \downarrow e)$. Substituting $h(\downarrow e)$ by $h(\Downarrow e)$, we get $h(C) = h(\Downarrow e) \cdot h(C \setminus \downarrow e) = h(C \setminus \{e\})$ giving us a contradiction to the fact that $h(C) \neq h(C \setminus \{e\})$. ◀

► **Example 6.** A configuration in which every maximal event is critical, may not itself be critical. We show this via an example. Consider three processes and the distributed alphabet defined by $\Sigma_1 = \{a\}$, $\Sigma_2 = \{b\}$ and $\Sigma_3 = \{c\}$ (each process has a local action). Consider the \mathcal{R} -trivial (commutative) monoid $M = (\{0, 1, 2\}, \oplus, 0)$ with truncated sum defined as $x \oplus y = \min(2, x + y)$. In other words, 0 is the neutral element, 2 is absorbent, and $1 \oplus 1 = 2$. We have $2 <_R 1 <_R 0$. Consider the (truncated) length trace morphism $h: TR(\tilde{\Sigma}) \rightarrow M$ defined by $h(x) = 1$ for all $x \in \Sigma$. Finally, let $t = abc = (E, \leq, \lambda)$ be the trace with 3 events labelled a , b and c respectively. Each event e in t is critical and is maximal in t . But the configuration E is not critical since for each $e \in E$, we have $h(E \setminus \{e\}) = 2 = h(E)$.

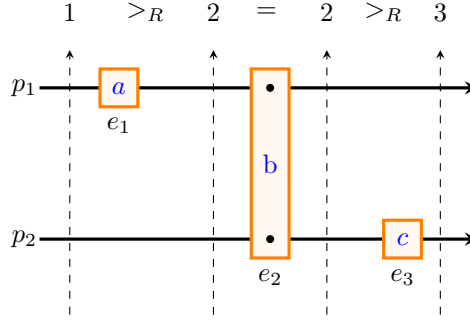
► **Lemma 7.** *Any prefix t' of t containing all the critical events of t , evaluates to the same monoid value as t under h . In other words, we have $h(\downarrow X_t) = h(t)$.*

Proof. We have $h(t) \leq_R h(\downarrow X_t)$ since $\downarrow X_t$ is a prefix of t . For the converse, we first show that there exists a *critical* configuration C of $t = (E, \leq, \lambda)$ such that $h(C) = h(t)$. The configuration $C = E$ satisfies $h(C) = h(t)$. Let C be a configuration with $h(C) = h(t)$.

Either C is critical and we are done, or we can drop a maximal event from C to get a strictly smaller configuration C' satisfying $h(C') = h(t)$. Repeating this, we find a critical configuration C of t such that $h(C) = h(t)$. By Lemma 5, the maximal events of C are critical. Hence we get $C \subseteq \downarrow X_t$. This implies $h(\downarrow X_t) \leq_R h(C) = h(t)$. By \mathcal{R} -triviality, we deduce $h(\downarrow X_t) = h(t)$. \blacktriangleleft

The *type* of a trace $t = (E, \leq, \lambda)$ is the Σ -labelled poset $\text{type}(t) = (X_t, \leq, \lambda)$ obtained as the restriction of the labelled poset (E, \leq, λ) to the critical events of t .

► **Example 8.** It is important to note that the type of a trace need not necessarily be a trace itself. Let $t = abc$ be a trace over the distributed alphabet $\Sigma_1 = \{a, b\}$, $\Sigma_2 = \{b, c\}$ with 3 events $e_1 < e_2 < e_3$ labelled a, b, c respectively, as shown in the figure below. Consider the \mathcal{R} -trivial (commutative) monoid $M = (\{1, 2, 3\}, \max, 1)$ where \max is the binary maximum operation. It is clear that $3 <_R 2 <_R 1$. Consider the trace morphism $h: TR(\tilde{\Sigma}) \rightarrow M$ defined by $h(a) = 2$, $h(b) = 1$ and $h(c) = 3$. We see that only e_1 and e_3 are critical events of t . Then, $\text{type}(t) = (\{e_1, e_3\}, \leq, \lambda)$ with $e_1 < e_3$ which is induced by the original partial order on t . Since the labels a and c are independent, $\text{type}(t)$ is not a trace.



Let $s = (Z, \leq, \lambda)$ be a Σ -labelled poset. A linearization of s is a word¹ $w = (Z, \preceq, \lambda)$ where \preceq is a total order on Z which is consistent with \leq , that is, satisfying $\leq \subseteq \preceq$. We denote by $\text{Lin}(s) \subseteq \Sigma^*$ the set of linearizations of s .

Let $h': \Sigma^* \rightarrow M$ be the *word* morphism defined by $h'(a) = h(a)$ for all $a \in \Sigma$. For all traces $t \in TR(\tilde{\Sigma})$ and for all linearizations $w \in \text{Lin}(t)$, we have $h'(w) = h(t)$.

For every monoid element $m \in M$ we define $\Sigma^m = \{a \in \Sigma \mid m \cdot h(a) = m\}$ as the set of letters stabilizing m . The complement $\Sigma \setminus \Sigma^m = \{a \in \Sigma \mid m \cdot h(a) \neq m\}$ also denoted $\bar{\Sigma}^m$ is the set of letters which cause an \mathcal{R} -descent for m . This means that for every $a \in \bar{\Sigma}^m$ $m \cdot h(a) <_R m$.

► **Lemma 9.** Let $t = (E, \leq, \lambda)$ be a trace and $\text{type}(t) = (X_t, \leq, \lambda)$ be its type. For all linearizations $w \in \text{Lin}(\text{type}(t))$, we have $h'(w) = h(t)$.

Proof. The proof is by induction on the number $k = |X_t|$ of critical events of t .

For the base case $k = 0$ there are no critical events in t . This means that $w = \varepsilon$ and all events in t have labels in Σ^{1_M} . Clearly, we have $h'(\varepsilon) = 1_M = h(t)$.

Assume now $k > 0$ and that the result holds for all traces s with $|X_s| < k$. Consider the linearization $w = a_1 a_2 \dots a_k \in \text{Lin}(\text{type}(t))$. We have $w = (X_t, \preceq, \lambda)$ with $X_t = \{e_1, e_2, \dots, e_k\}$, $e_1 \prec e_2 \prec \dots \prec e_k$ and $\lambda(e_i) = a_i$. Note that the event e_k is maximal in

¹ A word $a_1 a_2 \dots a_n \in \Sigma^*$ is identified with the structure $(\{e_1, e_2, \dots, e_n\}, \preceq, \lambda)$ with $\lambda(e_i) = a_i$ and $e_1 \prec e_2 \prec \dots \prec e_n$.

$\downarrow X_t$. Consider the trace s which is the prefix of t induced by $(\downarrow X_t) \setminus \{e_k\}$. It is easy to see that $\text{type}(s) = (\{e_1, \dots, e_{k-1}\}, \leq, \lambda)$ and $v = a_1 \dots a_{k-1} \in \text{Lin}(\text{type}(s))$. By induction, we have $h'(v) = h(s)$. Therefore, $h'(w) = h'(v) \cdot h'(a_k) = h(s) \cdot h(\lambda(e_k)) = h(\downarrow X_t) = h(t)$ where the last equality follows from Lemma 7. \blacktriangleleft

It follows immediately that the type of a trace determines its value under h .

► **Corollary 10.** *Let $t, t' \in TR(\tilde{\Sigma})$ be two traces. If $\text{type}(t)$ and $\text{type}(t')$ are isomorphic, denoted $\text{type}(t) \cong \text{type}(t')$, then $h(t) = h(t')$.*

We conclude this subsection with some more definitions. Recall that we have fixed the distributed alphabet $\tilde{\Sigma}$ and the morphism h from $TR(\tilde{\Sigma})$ to the finite \mathcal{R} -trivial monoid M . We define the *type set* of $\tilde{\Sigma}$, with respect to h , to be the set of all possible types of all traces over $\tilde{\Sigma}$. More precisely, $\text{types}(\tilde{\Sigma}) = \{\text{type}(t) \mid t \in TR(\tilde{\Sigma})\}$. Note that, it clearly depends on h . By Lemma 4, every type is a Σ -labelled poset whose underlying set has size less than $|\mathcal{P}| \times |M|$. Therefore $\text{types}(\tilde{\Sigma})$ is a finite set.

By Corollary 10, the type of a trace determines its monoid value under the recognizing morphism h . Hence we can define a function $\pi: \text{types}(\tilde{\Sigma}) \rightarrow M$ by $\pi(\text{type}(t)) = h(t)$ for all $t \in TR(\tilde{\Sigma})$. Note that, by Lemma 9, if $\sigma = (X, \leq, \lambda)$ is a type then $\pi(\sigma) = h'(w)$ for all $w \in \text{Lin}(\sigma)$.

Let $t = (E, \leq, \lambda)$ be a trace and $e \in E$ be an event. The *type of e in t* is the type of the prefix of t induced by the *strict past* of e : $\text{type}(t, e) = \text{type}(\downarrow e) = (X_{\downarrow e}, \leq, \lambda) = (X_t \cap \downarrow e, \leq, \lambda)$. We have $\pi(\text{type}(t, e)) = h(\downarrow e)$.

► **Lemma 11.** *Let $t = (E, \leq, \lambda)$ be a trace and e, f be events in t . If $\text{type}(t, e) = \text{type}(t, f) = \sigma$ and $\lambda(e) = \lambda(f) \in \overline{\Sigma^m}$ where $m = \pi(\sigma)$ then $e = f$.*

Proof. Assume towards a contradiction that there are two events e, f in t with $\text{type}(t, e) = \text{type}(t, f) = \sigma$ and $\lambda(e) = \lambda(f) = a \in \overline{\Sigma^m}$ where $m = \pi(\sigma)$. Without loss of generality we may assume $e < f$ since they are both labelled with the same letter and hence are ordered. Then, e is a critical event: indeed, $\text{type}(t, e) = \sigma$ hence $h(\downarrow e) = \pi(\sigma) = m$ and $\lambda(e) = a \in \overline{\Sigma^m}$ implies that $h(\downarrow e) = m \cdot h(a) \neq m$. We deduce that $X_{\downarrow e} \cup \{e\} \subseteq X_{\downarrow f}$: there are more critical events in the strict past of f than in the strict past of e . This is a contradiction with $\text{type}(t, e) = \text{type}(t, f)$. \blacktriangleleft

3.3 Proof of Theorem 2 [1 \rightarrow 2]

Having done the necessary setup we now prove the first direction of Theorem 2 (restated as Theorem 14 at the end of this subsection). We start with Lemma 12 which supplies the key argument by constructing $\text{LocTL}[\text{YP}]$ event formulas for identifying the type of an event.

An event formula φ of $\text{LocTL}[\text{YP}]$ is called a *strict past* formula if it is a boolean combination of formulas of the form $\text{YP } \varphi'$. A strict past formula does not test the label of the current event.

► **Lemma 12.** *For every $\sigma \in \text{types}(\tilde{\Sigma})$, there exists a strict past event formula φ_σ such that, for all traces $t = (E, \leq, \lambda) \in TR(\tilde{\Sigma})$ and all events e in t , $t, e \models \varphi_\sigma$ iff $\text{type}(t, e) \cong \sigma$.*

Proof. We construct φ_σ by induction and prove that it has the correct property.

Base case: $|\sigma| = 0$, i.e., $\sigma = (\emptyset, \emptyset, \emptyset)$. Note that $\pi((\emptyset, \emptyset, \emptyset)) = 1_M = h(\varepsilon)$ where ε is the empty trace. We define $\varphi_\sigma = \text{YH}(\Sigma^{1_M})$. We now prove that it has the desired property.

If $t, e \models \text{YH}(\Sigma^{1_M})$ then every letter in the strict past of e stabilizes the monoid identity element. We deduce easily that there are no critical events in the strict past of e . Therefore $\text{type}(t, e) = (\emptyset, \emptyset, \emptyset)$.

Conversely, if $\text{type}(t, e) = (\emptyset, \emptyset, \emptyset)$ then there are no critical events in the strict past of e . Since the empty trace is the smallest prefix of a trace and it maps to 1_M under h , we deduce easily that every letter in the strict past of e must stabilize the monoid identity 1_M . Therefore $t, e \models \text{YH}(\Sigma^{1_M})$.

Induction hypothesis: Let $k \geq 0$ and assume that for every type σ with $|\sigma| \leq k$, we have constructed φ_σ satisfying the property of the lemma. Consider an arbitrary type $\tau \in \text{types}(\tilde{\Sigma})$ of size $k + 1$. We construct φ_τ below and prove its correctness.

Let $s = (E_s, \leq_s, \lambda_s) \in TR(\tilde{\Sigma})$ be a trace with $\tau = \text{type}(s) = (X_s, \leq_s, \lambda_s)$. For an element $x \in X_s$, we let $\tau_x = (\downarrow_\tau x, \leq_s, \lambda_s)$ be the restriction of the labelled poset τ to the critical events strictly below x ($\downarrow_\tau x = \{y \in X_s \mid y <_s x\}$). It is easy to see that $\tau_x = \text{type}(s, x) = \text{type}(\downarrow_s x)$ where $\downarrow_s x = \{e \in E_s \mid e <_s x\}$ is (the prefix of s induced by) the strict past of x in s . Let $m_x = h(\downarrow_s x) = \pi(\tau_x)$ and $a_x = \lambda_s(x)$. Since $x \in X_s$ is a critical event of s , we have $h(\downarrow_s x) = h(\downarrow_s x) \cdot h(\lambda_s(x)) = m_x h(a_x) <_R m_x = h(\downarrow_s x)$. Therefore, $a_x \in \overline{\Sigma^{m_x}} = \overline{\Sigma^{\pi(\tau_x)}}$.

Notice that the size of $\tau_x \in \text{types}(\tilde{\Sigma})$ is at most k . By induction hypothesis, we have a formula φ_{τ_x} satisfying the property of the lemma. We use the macro $\varphi(\tau, x)$ to denote the event formula $\varphi_{\tau_x} \wedge a_x$. Finally, if $Z \subseteq X_s$, we let $\tau_Z = (\downarrow_\tau Z, \leq_s, \lambda_s)$. Then, $\tau_Z = \text{type}(\downarrow_s Z)$ is a type and we let $m_Z = \pi(\tau_Z) = h(\downarrow_s Z)$. We define

$$\begin{aligned} \varphi_\tau &= \varphi_1 \wedge \varphi_2 \wedge \varphi_{\text{order}} & \varphi_1 &= \text{YH}(\varphi_{\text{crit}} \vee \varphi_{\text{stab}}) \\ \varphi_2 &= \bigwedge_{x \in X_s} \text{YP } \varphi(\tau, x) & \varphi_{\text{crit}} &= \bigvee_{x \in X_s} \varphi(\tau, x) \\ \varphi_{\text{order}} &= \bigwedge_{x, y \in X_s \mid x <_s y} \text{YP } (\varphi(\tau, y) \wedge \text{YP } \varphi(\tau, x)) \wedge \bigwedge_{x, y \in X_s \mid x \not<_s y} \neg \text{YP } (\varphi(\tau, y) \wedge \text{YP } \varphi(\tau, x)) \\ \varphi_{\text{stab}} &= \bigvee_{Z = \downarrow_\tau Z \subseteq X_s} \Sigma^{m_Z} \wedge \bigwedge_{x \in Z} \text{YP } \varphi(\tau, x) \wedge \bigwedge_{x \in X_s \setminus Z} \neg \text{YP } \varphi(\tau, x) \end{aligned}$$

Before proceeding to the formal proof of correctness of φ_τ , we first provide an intuitive explanation of the various parts of φ_τ .

- φ_2 : asserts that for every “event” x in τ , there is a (unique) corresponding critical event in the strict past whose type is isomorphic to τ_x and whose label is same as that of x .
- φ_1 : asserts that every event in the strict past either corresponds to some “event” x in τ (satisfies φ_{crit}) or is labelled with a stabilizing letter for the type Z arising out of the “events” x in τ
- φ_{order} : asserts that the ordering between “events” in τ and the corresponding critical events is identical.

We now formally show that φ_τ indeed satisfies the property of the lemma. Towards this, fix a trace $t = (E_t, \leq_t, \lambda_t)$ and an event e in t .

(\rightarrow) Assume that $t, e \models \varphi_\tau$. We show that $\text{type}(t, e) \cong \tau$.

First consider the fact that $t, e \models \varphi_2$. This means that for all $x \in X_s$, there exists an event $e_x \in \downarrow_t e$, such that $t, e_x \models \varphi(\tau, x)$. By the inductive hypothesis, we have $\tau_x \cong \text{type}(t, e_x) = \text{type}(\downarrow_t e_x)$ and $\lambda_t(e_x) = a_x \in \overline{\Sigma^{m_x}}$. From Lemma 11, e_x is the unique event in t satisfying $\varphi(\tau, x)$. Using $h(\downarrow_t e_x) = \pi(\tau_x) = m_x$ and $\lambda_t(e_x) = a_x \in \overline{\Sigma^{m_x}}$, we deduce that e_x is critical in $\downarrow_t e$. Therefore, $Y = \{e_x \mid x \in X_s\} \subseteq X_{\downarrow_t e}$.

Now, using the fact that $t, e \models \varphi_{order}$ and the unicity of the events e_x , we deduce that for all $x, y \in X_s$, we have $x \leq_s y$ if and only if $e_x \leq_t e_y$. We also know that for all $x \in X_s$, we have $\lambda_s(x) = a_x = \lambda_t(e_x)$. Therefore, $\tau \cong (Y, \leq_t, \lambda_t)$.

In order to obtain $\tau \cong \text{type}(t, e) = (X_{\downarrow_t e}, \leq_t, \lambda_t)$ as desired, it remains to prove that $Y = X_{\downarrow_t e}$. Towards a contradiction, assume that this is not the case and consider a minimal event $f \in X_{\downarrow_t e} \setminus Y$. Since $t, e \models \varphi_1$, we have $t, f \models \varphi_{crit} \vee \varphi_{stab}$. If $t, f \models \varphi_{crit}$ then $t, f \models \varphi(\tau, x)$ for some $x \in X_s$ and by the unicity constraint $f = e_x \in Y$, a contradiction. Hence, $t, f \models \varphi_{stab}$ and there is a downward closed set $Z = \downarrow_\tau Z \subseteq X_s$ such that $\lambda_t(f) \in \Sigma^{m_Z}$ and $\downarrow_t f \cap Y = \{e_x \mid x \in Z\}$. By minimality of f , we get $\downarrow_t f \cap X_{\downarrow_t e} = \downarrow_t f \cap Y = \{e_x \mid x \in Z\}$. We deduce that $\text{type}(t, f) = (\{e_x \mid x \in Z\}, \leq_t, \lambda_t) \cong \tau_Z$ and $h(\downarrow_t f) = m_Z$. Using $\lambda_t(f) \in \Sigma^{m_Z}$ we conclude that f is not critical, a contradiction.

(\leftarrow) Conversely, assume that $\tau \cong \text{type}(t, e)$. We show that $t, e \models \varphi_\tau$.

From the isomorphism between $\tau = (X_s, \leq_s, \lambda_s)$ and $\text{type}(t, e) = (X_{\downarrow_t e}, \leq_t, \lambda_t)$ we get a one-one correspondence $x \in X_s \mapsto e_x \in X_{\downarrow_t e}$ such that for all $x, y \in X_s$ we have $x \leq_s y$ if and only if $e_x \leq_t e_y$ and $\lambda_t(e_x) = \lambda_s(x)$. We deduce that $\text{type}(t, e_x) \cong \tau_x$ and $\lambda_t(e_x) = a_x$ for all $x \in X_s$. Therefore, $t, e_x \models \varphi(\tau, x)$ for all $x \in X_s$ and $t, e \models \varphi_2$.

Next, it is easy to check that $t, e \models \varphi_{order}$ using the unicity property (Lemma 11) and the fact that $x <_s y$ if and only if $e_x <_t e_y$ for all $x, y \in X_s$.

Let $f \in \downarrow_t e$ be an event in the strict past of e . Either $f \in X_{\downarrow_t e}$ is a critical event and $f = e_x$ for some $x \in X_s$. In this case, we get $t, f \models \varphi(\tau, x)$ and $t, f \models \varphi_{crit}$. Or $f \in \downarrow_t e \setminus X_{\downarrow_t e}$ is a non-critical event. Let $Z = \{x \in X_s \mid e_x < f\}$. Then, $Z = \downarrow_\tau Z$ is downward closed and using the unicity property we deduce that $t, f \models \bigwedge_{x \in Z} \text{YP } \varphi(\tau, x) \wedge \bigwedge_{x \in X_s \setminus Z} \neg \text{YP } \varphi(\tau, x)$. Now, due to the fact that $\tau \cong \text{type}(t, e)$ it is clear that $\tau_Z \cong \text{type}(t, f)$. We deduce that $m_Z = \pi(\tau_Z) = h(\downarrow_t f)$. Since f is not critical, we have $\lambda_t(f) \in \Sigma^{m_Z}$. Therefore, $t, f \models \varphi_{stab}$. We have shown that $t, e \models \varphi_1$. \blacktriangleleft

► **Lemma 13.** *For every element $m \in M$ we can define a strict past event formula φ_m in $\text{LocTL}[\text{YP}]$ such that for every trace t and event e of t , we have $t, e \models \varphi_m$ iff $h(\downarrow e) = m$.*

Proof. We define $\varphi_m = \bigvee_{\sigma \in \text{types}(\tilde{\Sigma}) : \pi(\sigma) = m} \varphi_\sigma$ and show that it has the desired property.

Let t be a trace and e be an event in t . If $t, e \models \varphi_m$ then there exists some $\sigma \in \text{types}(\tilde{\Sigma})$ such that $\pi(\sigma) = m$ and $t, e \models \varphi_\sigma$. By Lemma 12, $\sigma \cong \text{type}(t, e) = \text{type}(\downarrow e)$. By definition of π , we have $h(\downarrow e) = \pi(\sigma)$. As $\pi(\sigma) = m$, we conclude that $h(\downarrow e) = m$. Now for the other direction assume $h(\downarrow e) = m$. By Lemma 12, $t, e \models \varphi_{\text{type}(t, e)}$ and by definition of π , $\pi(\text{type}(t, e)) = h(\downarrow e)$. As $h(\downarrow e) = m$, $\pi(\text{type}(t, e)) = m$ and hence $t, e \models \varphi_m$. \blacktriangleleft

► **Theorem 14.** *Let $L \subseteq \text{TR}(\tilde{\Sigma})$ be recognized by the morphism $h : \text{TR}(\tilde{\Sigma}) \rightarrow M$. Then there is a sentence Φ_L of $\text{LocTL}[\text{YP}]$ which defines L : for all traces $t \in \text{TR}(\tilde{\Sigma})$, $t \models \Phi_L \iff t \in L$.*

Proof. As L is recognized by h , there is a subset $P \subseteq M$ such that $L = h^{-1}(P)$. For each monoid element $m \in M$, we define below a sentence Φ_m of $\text{LocTL}[\text{YP}]$ which defines the trace language $h^{-1}(m)$: for all traces t we have $t \models \Phi_m \iff h(t) = m$. The sentence defining L is $\Phi_L = \bigvee_{m \in P} \Phi_m$.

Let $\# \notin \Sigma$ be a new letter. We extend the distributed alphabet $\tilde{\Sigma}$ by making $\#$ an action shared by all processes: $\tilde{\Sigma}_\# = \{\Sigma_i \cup \{\#\}\}_{i \in \mathcal{P}}$. For each trace $t \in \text{TR}(\tilde{\Sigma})$, we consider the augmented trace $\bar{t} = t \cdot \# \in \text{TR}(\tilde{\Sigma}_\#)$ obtained by adding a maximum event $e_\#$ labelled $\#$ to t . Note that all processes participate in $e_\#$ and $t = \downarrow e_\#$. Now by Lemma 13, we have a

strict past event formula² φ_m , such that $\bar{t}, e_{\#} \models \varphi_m \iff h(\Downarrow e_{\#}) = m \iff h(t) = m$. We can now set Φ_m as the sentence obtained by replacing every outermost occurrence of YP in φ_m by E . It is easy to see that $t \models \Phi_m$ iff $\bar{t}, e_{\#} \models \varphi_m$. As a result, $t \models \Phi_m$ iff $h(t) = m$. ◀

4 Logic to cascade products to algebra

In this section, we first introduce a localized version of $\text{LocTL}[\text{YP}]$ namely $\text{LocTL}[\text{YP}_i]$ and show that both have the same expressive power. $\text{LocTL}[\text{YP}_i]$ is more suited for the direction $2 \rightarrow 3$ of Theorem 2 where we connect logic to cascade products of asynchronous automata. Subsequently, in Section 4.2 we introduce asynchronous devices and their cascade products and prove Theorem 22 in Section 4.3. Finally we show the $3 \rightarrow 1$ direction of Theorem 2 by connecting cascade products to algebra in Theorem 24.

4.1 $\text{LocTL}[\text{YP}_i]$ and its equivalence with $\text{LocTL}[\text{YP}]$

We define *local* modalities E_i and YP_i for each process $i \in \mathcal{P}$ as follows:

$$\text{E}_i \varphi = \text{E}(\Sigma_i \wedge \varphi) \quad \text{and} \quad \text{YP}_i \varphi = \Sigma_i \wedge \text{YP}(\Sigma_i \wedge \varphi)$$

Let t be a trace and e be an event of t . It is clear that, $t, e \models \text{YP}_i \varphi$ iff e is an i -event and there exists an i -event e' in the strict past of e such that $t, e' \models \varphi$. Further $t \models \text{E}_i \varphi$ iff there exists an i -event f of t such that $t, f \models \varphi$.

We define $\text{LocTL}[\text{YP}_i]$ as a fragment of $\text{LocTL}[\text{YP}]$ in which only above local modalities are allowed. More precisely, the syntax of $\text{LocTL}[\text{YP}_i]$ logic is as follows:

$$\begin{aligned} \Phi &::= \text{E}_i \varphi \mid \Phi_1 \vee \Phi_2 \mid \neg \Phi \\ \varphi &::= a \mid \varphi_1 \vee \varphi_2 \mid \neg \varphi \mid \text{YP}_i \varphi \end{aligned}$$

► **Lemma 15.** $\text{LocTL}[\text{YP}]$ and $\text{LocTL}[\text{YP}_i]$ express the same class of trace languages over $\tilde{\Sigma}$.

Proof. We only need to show how to express the modalities E and YP of $\text{LocTL}[\text{YP}]$ with the *local* modalities of $\text{LocTL}[\text{YP}_i]$. We have $\text{E} \varphi = \bigvee_{i \in \mathcal{P}} \text{E}_i \varphi$.

For the strict past modality YP we use a well-known fact on traces. Let $t = (E, \leq, \lambda)$ be a trace and e, f be events in t . Then $e < f$ if and only if there is a sequence of events $e = e_0 < e_1 < e_2 < \dots < e_k = f$ with $1 \leq k \leq |\mathcal{P}|$ and for each $1 \leq j \leq k$, there is a process $i_j \in \mathcal{P}$ such that e_{j-1}, e_j are both i_j -events. It is easy to deduce from here that

$$\text{YP} \varphi = \bigvee_{\substack{i_1, i_2, \dots, i_k \in \mathcal{P} \\ 1 \leq k \leq |\mathcal{P}|}} \text{YP}_{i_k} \dots \text{YP}_{i_2} \text{YP}_{i_1} \varphi.$$

This shows that YP modality can be expressed in $\text{LocTL}[\text{YP}_i]$ and concludes the proof. ◀

► **Remark 16.** Referring to the syntax and semantics from [1] (Section 2.2), it is clear that $\text{LocTL}[\text{YP}_i]$ is a sublogic of LocPastPDL . Indeed, let φ be a $\text{LocTL}[\text{YP}_i]$ event formula and assume there exists an equivalent event formula $\bar{\varphi}$ in LocPastPDL . Then $\text{YP}_i \varphi$ is implemented with $\langle \leftarrow_i^+ \rangle \bar{\varphi}$ and $\text{E}_i \varphi$ is implemented with $\text{EM}_i(\langle \leftarrow_i^* \rangle \bar{\varphi})$

² Formally the formula φ_m may use $\#$. We can take care of this by using $\neg \Sigma$ wherever it occurs in the formula.

4.2 Asynchronous devices and cascade product

We now recall the model of asynchronous automata due to Zielonka [23].

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

- for each $i \in \mathcal{P}$, S_i is a finite non-empty set of local i -states.
- for each $a \in \Sigma$, $\delta_a: S_a \rightarrow S_a$ is a deterministic joint transition function where $S_a = \prod_{i \in \text{loc}(a)} S_i$ is the set of a -states.
- with $S = \prod_{i \in \mathcal{P}} S_i$ as the set of all global states, $s_{\text{in}} \in S$ is a designated initial global state.

Let $s \in S$ be a global state. We can write s as (s_a, s_{-a}) where s_a is the projection of s onto $\text{loc}(a)$ and s_{-a} is the projection onto $\mathcal{P} \setminus \text{loc}(a)$. For any letter $a \in \Sigma$, we can extend the joint transition function $\delta_a: S_a \rightarrow S_a$ to a global transition function $\Delta_a: S \rightarrow S$ on global states as follows: $\Delta_a((s_a, s_{-a})) = (\delta_a(s_a), s_{-a})$. For a trace $t \in TR(\tilde{\Sigma})$, we let Δ_ε be the identity function on S , and define $\Delta_t: S \rightarrow S$ as the composition $\Delta_t = \Delta_a \circ \Delta_{t'}$ when $t = t'a$. Note that Δ_t is well defined, since for any pair (a, b) of independent letters, $\Delta_a \circ \Delta_b = \Delta_b \circ \Delta_a$. We denote by $A(t)$ the global state $\Delta_t(s_{\text{in}})$ reached when A runs on t .

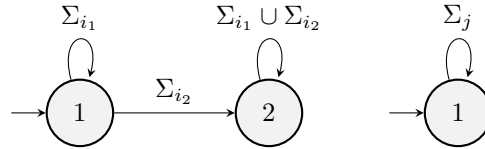
At this point we define the transition monoid $M(A)$ associated with the asynchronous automaton A . Let $M(A)$ be the finite set of functions $\{\Delta_t \mid t \in TR(\tilde{\Sigma})\}$. This set under the usual function composition operation is a monoid with Δ_ε as its identity element.

For any asynchronous automaton A , if we fix some $F \subseteq S$ to be the set of accepting global states then we say $L(A, F) = \{t \mid A(t) \in F\}$ is the *trace language accepted by A* .

We also use asynchronous automata as letter-to-letter asynchronous transducers as in [19, 2, 3] which compute a relabelling function. Let Γ be a finite non-empty set. We can define a natural extended distributed alphabet, $\widetilde{\Sigma \times \Gamma}$ by inheriting the distribution from $\tilde{\Sigma}$ that is, $\text{loc}((a, \gamma)) = \text{loc}(a)$. $TR(\tilde{\Sigma})$ and $TR(\widetilde{\Sigma \times \Gamma})$ are the set of traces over the original and extended distributed alphabets. A function $\theta: TR(\tilde{\Sigma}) \rightarrow TR(\widetilde{\Sigma \times \Gamma})$ is called a Γ -labelling function if, for every $t = (E, \leq, \lambda) \in TR(\tilde{\Sigma})$, $\theta(t) = (E, \leq, (\lambda, \mu)) \in TR(\widetilde{\Sigma \times \Gamma})$. A Γ -labelling function decorates each event e of the trace t with a label $\mu(e)$ from Γ .

An *asynchronous Γ -transducer* over $\tilde{\Sigma}$ (or simply Σ) is a tuple $\hat{A} = (A, \{\mu_a\})$ where $A = (\{S_i\}, \{\delta_a\}, s_{\text{in}})$ is an asynchronous automaton and each μ_a ($a \in \Sigma$) is a map $\mu_a: S_a \rightarrow \Gamma$. We overload notation and use $\hat{A}: TR(\tilde{\Sigma}) \rightarrow TR(\widetilde{\Sigma \times \Gamma})$ to also denote the Γ -labelling function computed by the transducer \hat{A} . For $t = (E, \leq, \lambda) \in TR(\tilde{\Sigma})$, we define $\hat{A}(t) = (E, \leq, (\lambda, \mu)) \in TR(\widetilde{\Sigma \times \Gamma})$ such that, for every $e \in E$ with $\lambda(e) = a$, $\mu(e) = \mu_a(s_a)$ where $s = A(\downarrow e)$.

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. For an automaton/transducer localized at i , all non-trivial transitions are on letters in which process i participates.



■ **Figure 1** $U_1[i]$ automaton.

► **Example 17.** We define $U_1[i]$ to be a deterministic asynchronous automaton which is localized at process i . In $U_1[i]$, the local state set of process i has two states: $S_i = \{1, 2\}$ while all other processes have a single state. In Figure 1 we show the local automaton for

process i , which operates only on $\Sigma_i = \Sigma_{i_1} \uplus \Sigma_{i_2}$. Note that, a letter in Σ_{i_2} allows to change the local state from 1 to 2 while the rest of the transitions are self-loops. Clearly, we can identify the global-states of $U_1[i]$ with S_i . With this convention, we always use 1 as the initial global state for $U_1[i]$. Note that the transition monoid of $U_1[i]$ is isomorphic to a submonoid of U_1 from Example 1. For a trace t , $U_1[i](t)$, the global state reached by $U_1[i]$ when run on t , is 2 iff there exists an i -event in t with label in Σ_{i_2} .

A *degenerate* $U_1[i]$ automaton is a $U_1[i]$ automaton for which $\Sigma_{i_2} = \emptyset$. As state 2 is not reachable from the initial state 1, we can safely assume that $S_i = \{1\}$. Observe that the transition monoid of a degenerate $U_1[i]$ is the trivial monoid.

► **Example 18.** An asynchronous Γ -transducer over Σ whose underlying automaton is $U_1[i]$ from Example 17, is denoted by $\widehat{U_1[i]}$. More precisely, $\widehat{U_1[i]} = (U_1[i], \{\mu_a\})$. Recall that in $U_1[i]$, $S_i = \{1, 2\}$ and $S_j = \{1\}$ for $j \neq i$. Further, for each $a \in \Sigma$, $\mu_a: S_a \rightarrow \Gamma$. These output functions $\{\mu_a\}$ can be viewed as follows.

1. For each a such that $i \in \text{loc}(a)$, we identify S_a with S_i . Thus, $\mu_a: S_i \rightarrow \Gamma$.
2. For each a such that $i \notin \text{loc}(a)$, S_a is a singleton. As a result, μ_a is a constant function and can be simply thought of as an element of Γ .

Note that, $\widehat{U_1[i]}$ always uses 1 as the initial global state and it has an associated Γ -labelling function $\widehat{U_1[i]}: TR(\widetilde{\Sigma}) \rightarrow TR(\widetilde{\Sigma} \times \Gamma)$.

A *degenerate* $\widehat{U_1[i]}$ Γ -transducer has a degenerate $U_1[i]$ as its underlying automaton and has effectively only one global state. As a result, its output function can be simply specified as a output function $\mu: \Sigma \rightarrow \Gamma$.

Now we introduce the local cascade product of asynchronous transducers [4, 3] which will play an important role in later sections.

► **Definition 19.** Let $\widehat{A} = (\{S_i\}, \{\delta_a\}, s_{\text{in}}, \{\mu_a\})$ be a Γ -transducer over Σ and $\widehat{B} = (\{Q_i\}, \{\delta_{(a,\gamma)}\}, q_{\text{in}}, \{\nu_{(a,\gamma)}\})$ be Π -transducer over $\Sigma \times \Gamma$. We define the local cascade product of \widehat{A} and \widehat{B} to be the Π -transducer $\widehat{A} \circ_\ell \widehat{B} = (\{S_i \times Q_i\}, \{\nabla_a\}, (s_{\text{in}}, q_{\text{in}}), \{\tau_a\})$ over Σ where $\nabla_a((s_a, q_a)) = (\delta_a(s_a), \delta_{(a, \mu_a(s_a))}(q_a))$ and $\tau_a: S_a \times Q_a \rightarrow \Pi$ is defined by $\tau_a((s_a, q_a)) = \nu_{(a, \mu_a(s_a))}(q_a)$.

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

The following lemma is easily verified. See [3] for more details.

► **Lemma 20.** The Π -labelling function computed by $\widehat{A} \circ_\ell \widehat{B}$ is the composition of the Γ -labelling function computed by \widehat{A} and the Π -labelling function computed by \widehat{B} : for every $t \in TR(\widetilde{\Sigma})$,

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

Although we have defined the local cascade product of transducers, it is equally easy to define the local cascade product of asynchronous automata. We refer to [3] for precise details of this construction and its connection with the wreath product operation and the resulting local cascade product principle.

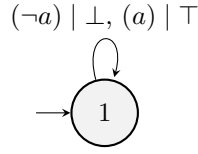
4.3 Proof of Theorem 2 [2 \rightarrow 3]

In this section, we provide a proof of 2 \rightarrow 3 of Theorem 2. In view of Lemma 15, it suffices to show that languages expressed by $\text{LocTL}[\text{YP}_i]$ sentences are accepted by local cascade product of $U_1[i]$ automata. This is precisely the content of Theorem 22.

Let φ be a $\text{LocTL}[\text{YP}_i]$ event formula. For each trace $t \in TR(\tilde{\Sigma})$ and event e in t , we let $\mu_\varphi(e) \in \{\top, \perp\}$ be the truth value of φ at e , and we let θ^φ be the $\{\top, \perp\}$ -labelling function given by $\theta^\varphi(t) = (E, \leq, (\lambda, \mu_\varphi))$, for each $t = (E, \leq, \lambda) \in TR(\tilde{\Sigma})$.

► **Lemma 21.** *Let φ be a $\text{LocTL}[\text{YP}_i]$ event formula and let θ^φ be the corresponding $\{\top, \perp\}$ -labelling function. Then we can construct a local cascade product \hat{A}_φ of $\widehat{U_1[i]}$ transducers which computes θ^φ .*

Proof. The construction of \hat{A}_φ proceeds by structural induction on φ . We first consider the base case with $\varphi = a$. Fix some process $i \in \text{loc}(a)$. We let \hat{A}_φ be the degenerate $\{\top, \perp\}$ -transducer $\widehat{U_1[i]}$ over Σ whose output function is $\mu: \Sigma \rightarrow \{\top, \perp\}$ where $\mu(b) = \top$ if $b = a$ and $\mu(b) = \perp$ otherwise. Clearly, the $\{\top, \perp\}$ -labelling function computed by \hat{A}_φ coincides with θ^φ .

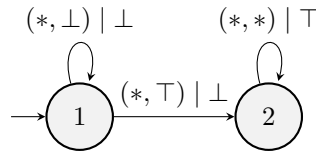


Now we consider the inductive case where $\varphi = \text{YP}_i \varphi'$. By induction, we have $\{\top, \perp\}$ -transducer $\hat{A}_{\varphi'}$ which computes $\theta^{\varphi'}$. Note that the output alphabet of $\hat{A}_{\varphi'}$ is $\Pi = \Sigma \times \{\top, \perp\}$ with the distribution $\Pi_i = \Sigma_i \times \{\top, \perp\}$.

We construct \hat{A}_φ as a local cascade product $\hat{A}_{\varphi'} \circ_\ell \hat{B}$, where \hat{B} is a $\{\top, \perp\}$ -transducer $\widehat{U_1[i]}$ over Π defined below, so that \hat{A}_φ computes θ^φ . In view of Lemma 20, $\hat{A}_\varphi(t) = \hat{B}(\hat{A}_{\varphi'}(t))$. By induction on φ' , $\hat{A}_{\varphi'} = \theta^{\varphi'}$. Thus, the requirement that \hat{A}_φ computes θ^φ simply translates to the requirement that $\hat{B}(\theta^{\varphi'}(t)) = \theta^\varphi(t)$.

Let $t = (E, \leq, \lambda)$ be a trace over Σ and $t' = \theta^{\varphi'}(t) = (E, \leq, (\lambda, \mu'))$. Further, let $t'' = \theta^\varphi(t) = (E, \leq, (\lambda, \mu))$. As $\varphi = \text{YP}_i \varphi'$, for an event $e \in E$, $\mu(e) = \top$ iff e is an i -event and there is an i -event f in the strict past of e such that $\mu'(f) = \top$.

We are now ready to construct \hat{B} as a $\widehat{U_1[i]}$ transducer over Π so as $\hat{B}(t') = t''$. The local component for process i in $\hat{B} = \widehat{U_1[i]}$ is as depicted in the figure below.



More precisely, for \hat{B} over Π , with $S_i = \{1, 2\}$, the only non-self-loop transitions (from local state 1 to local state 2) are on letters of the form (a, \top) with $a \in \Sigma_i$. The output functions are as follows. For $a \in \Sigma_i$, $\mu_{(a, *)}(1) = \perp$, $\mu_{(a, *)}(2) = \top$ and for $a \notin \Sigma_i$, $\mu_{(a, *)} = \perp$.

Clearly, \hat{B} outputs \perp on events in which process i does not participate. Further, it outputs \top on an i -event only after reaching state 2 which ensures that there is an i -event in the strict past where the input label (the truth value of φ') carries the value \top . It is easy to see from here that, for every $t \in TR(\tilde{\Sigma})$, $\hat{B}(\theta^{\varphi'}(t)) = \theta^\varphi(t)$. This in turn implies that $\hat{A}_\varphi = \hat{A}_{\varphi'} \circ_\ell \hat{B}$ correctly computes θ^φ .

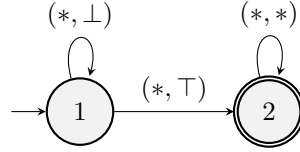
We now consider the case where $\varphi = \varphi_1 \vee \varphi_2$. By induction, we have a transducer \hat{A}_{φ_1} (\hat{A}_{φ_2}) which computes θ^{φ_1} (respectively θ^{φ_2}). We consider the natural direct product $\hat{A} = \hat{A}_{\varphi_1} \times \hat{A}_{\varphi_2}$ which is a $(\{\top, \perp\} \times \{\top, \perp\})$ -transducer which simultaneously computes the truth values of φ_1 as well as φ_2 . We can easily see that the direct product is a special

case of the cascade product. We can simply define $\hat{A}_\varphi = \hat{A} \circ_\ell \hat{B}$ where \hat{B} is a degenerate $\{\top, \perp\}$ -transducer $\widehat{U_1[i]}$ (for any process i) over $\Sigma \times (\{\top, \perp\} \times \{\top, \perp\})$ whose output function μ is as follows: $\mu((a, b_1, b_2)) = b_1 \vee b_2$. It is easy to see that \hat{A} computes θ^φ .

The remaining case where $\varphi = \neg\varphi'$ can be similarly handled by a local cascade product of $\hat{A}_{\varphi'}$ with a degenerate $\widehat{U_1[i]}$ transducer. \blacktriangleleft

► **Theorem 22.** *Let Φ be a $\text{LocTL}[\text{YP}_i]$ sentence. There exists an asynchronous automaton A_Φ which is a cascade product of $U_1[i]$ automata and it accepts exactly the trace language defined by Φ . In other words, A_Φ accepts $L_\Phi = \{t \in \text{TR}(\tilde{\Sigma}) \mid t \models \Phi\}$.*

Proof. A basic $\text{LocTL}[\text{YP}_i]$ trace formula Φ is of the form $E_j \varphi$, where φ is a $\text{LocTL}[\text{YP}_i]$ event formula. For a trace t , we have that $t \models E_j \varphi$ iff there exists a j -event e in the trace such that $t, e \models \varphi$. From Lemma 21 we have a $\{\top, \perp\}$ -transducer \hat{A}_φ which is a local cascade product of $\widehat{U_1[i]}$ transducers such that $\hat{A}_\varphi = \theta^\varphi$. Let $t = (E, \leq, \lambda)$ be any trace over Σ . It is clear that $t \models E_j \varphi$ iff some j -event in the output trace $t' = \hat{A}_\varphi(t)$ is labelled by \top . Therefore the desired property of A_Φ is that it should accept the trace t iff t' has a \top -labelled j -event. To achieve this, we simply take the cascade product of the $\{\top, \perp\}$ -transducer \hat{A}_φ with an asynchronous automaton $U_1[j]$ over $\Sigma \times \{\top, \perp\}$. Beginning in the initial global state 1, the non-trivial component of $U_1[j]$ automaton owned by process j waits at local state 1 for the label \top to occur on j -events and transits to local state 2 after seeing the label \top and stays there thereafter. This $U_1[j]$ automaton is depicted in the figure below.



In order to accept L_Φ , in the automaton $A_\Phi = \hat{A}_\varphi \circ_\ell U_1[j]$, we define a global state of A_Φ to be *accepting* if its component for the last (in the cascade product) $U_1[j]$ global state is 2. As \hat{A}_φ is a local cascade product of $\widehat{U_1[i]}$ transducers and the underlying automaton for each $\widehat{U_1[i]}$ is simply a $U_1[i]$ automaton, it is not difficult to see that A_Φ is in fact a local cascade product of $U_1[i]$ automata.

This takes care of basic primitive languages defined by sentences of the form $E_i \varphi$. As a general sentence Φ is a boolean combination of sentences of the form $E_i \varphi$, L_Φ is also a boolean combination of basic primitive languages. The result for general Φ follows as boolean combinations can be handled by direct product construction which can be easily implemented by the local cascade product construction. \blacktriangleleft

4.4 Proof of Theorem 2 [3 \rightarrow 1]

In this subsection, we provide a proof of 3 \rightarrow 1 of Theorem 2 in the form of Theorem 24.

► **Lemma 23.** *Let A be an asynchronous automaton over Σ which is a local cascade product of $U_1[i]$ automata. Then the transition monoid of A is \mathcal{R} -trivial.*

Proof. Let $A = (\{S_i\}, \{\delta_a\}, s_{in})$ be a local cascade product of $U_1[i]$ automata. We show that there is a partial order \leq on the global state set $S = \prod_{i \in \mathcal{P}} S_i$ such that, for every $(s, a) \in S \times \Sigma$, $\Delta_a(s) \leq s$ (recall that, $\Delta_a: S \rightarrow S$ is the natural global transition function induced by the joint transition function $\delta_a: S_a \rightarrow S_a$). It is known [6] and easy to see that the existence of such a partial order implies that the transition monoid of A is \mathcal{R} -trivial.

We show the existence of the required partial order by induction on the number of constituent $U_1[i]$ automata in A . In the base case, A itself is a $U_1[i]$. As the global state set of $U_1[i]$ is $\{1, 2\}$ (see Example 17), by setting \leq to be the total order with $2 < 1$, we get the desired property.

To complete the inductive step, we assume the existence of a partial order \preceq on the global state set S of A . We now use it to construct the required partial order \leq on the global state set of $A \circ_\ell U_1[i]$. Note that the global state set of $A \circ_\ell U_1[i]$ is simply $S \times \{1, 2\}$. We define \leq as follows: for $s, s' \in S$ and $k, k' \in \{1, 2\}$, $(s', k') \leq (s, k)$ iff $s' \preceq s$, and $k' \leq k$. In other words, \leq is simply the direct product of \preceq on S and the total order on $\{1, 2\}$ with $2 < 1$. Let $a \in \Sigma$, then the (global) deterministic transition on a of $A \circ_\ell U_1[i]$ takes the state (s, k) to the state $(s', k') = (\Delta_a(s), \Delta_{(a, s_a)}(k))$. It is clear from this description that $(s', k') \leq (s, k)$. This shows that \leq on $S \times \{1, 2\}$ has the desired property and we are done. \blacktriangleleft

► **Theorem 24.** *Let $L \subseteq TR(\tilde{\Sigma})$ be accepted by an asynchronous automaton A which is a local cascade product of $U_1[i]$ automata. Then L is \mathcal{R} -trivial.*

Proof. We consider the transition monoid $M(A)$ associated with A . Recall that it is the finite monoid consisting of $\{\Delta_t : t \in TR(\tilde{\Sigma})\}$ where Δ_t is the global transition function on the global state set induced by the trace t . By the previous lemma, this monoid is \mathcal{R} -trivial. Due to the local nature of the transitions of A , if a and b are independent then Δ_a and Δ_b commute. As a result, we have a well defined morphism $h : TR(\tilde{\Sigma}) \rightarrow M(A)$ which sends t to Δ_t . Suppose that L is accepted by A with s_{in} as the initial global state and F as a subset of global accepting states of A . Then L is recognized by the trace morphism h via the set $\{\Delta_t \in M(A) \mid \Delta_t(s_{in}) \in F\}$. Hence L is \mathcal{R} -trivial. \blacktriangleleft

5 Results on related fragments

In this section we briefly present other results which are of the same form as Theorem 2 but for trace languages which are \mathcal{L} -trivial or recognized by monoids from the class DA. First we shall define the necessary terminology and then subsequently state the results.

$\text{LocTL}[\text{XF}, \text{YP}]$ is the fragment of temporal logic over traces which contains both the strict future (XF) and the strict past (YP) modality. We give below the syntax for $\text{LocTL}[\text{XF}, \text{YP}]$ and only explain the semantics of the new modality XF.

$$\begin{aligned} \Phi &::= E\varphi \mid \Phi_1 \vee \Phi_2 \mid \neg\Phi \\ \varphi &::= a \mid \varphi_1 \vee \varphi_2 \mid \neg\varphi \mid \text{XF}\varphi \mid \text{YP}\varphi \end{aligned}$$

For a trace $t = (E, \leq, \lambda)$ and an event $e \in E$, we have $t, e \models \text{XF}\varphi$ if there exists an event f in t with $e < f$ and $t, f \models \varphi$. Observe that $\text{LocTL}[\text{YP}]$ is the past fragment of $\text{LocTL}[\text{XF}, \text{YP}]$. We can similarly define $\text{LocTL}[\text{XF}]$ is as the future fragment of $\text{LocTL}[\text{XF}, \text{YP}]$ which has access to only XF modality.

For a trace $t = (E, \leq, \lambda)$, the reverse trace $t^r = (E, \leq', \lambda)$ is defined to be the trace where, for $e, f \in E$, $e \leq' f$ iff $f \leq e$. For a trace language $L \subseteq TR(\tilde{\Sigma})$, the reverse language L^r is defined as $L^r = \{t^r \mid t \in L\}$.

For a monoid M , M^r denotes the opposite monoid with multiplication \circ^r defined as $m \circ^r n := n \circ m$ for all $m, n \in M^r$. It is easy to see that L is recognized by M iff L^r is recognized by M^r . Further, it is easy to deduce that M is \mathcal{L} -trivial iff M^r is \mathcal{R} -trivial. It follows that L is \mathcal{L} -trivial iff L^r is \mathcal{R} -trivial.

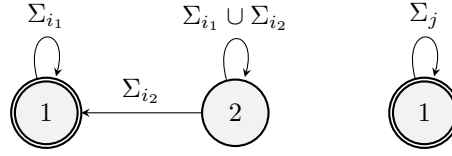
It is also easy to verify that L is $\text{LocTL}[\text{XF}]$ definable iff L^r is $\text{LocTL}[\text{YP}]$ definable. Given a sentence Φ in $\text{LocTL}[\text{XF}]$ defining L , we can obtain a $\text{LocTL}[\text{YP}]$ sentence Φ^r which defines L^r by simply replacing all occurrences of XF operators in Φ by YP , and vice versa. In view of the above discussion, it follows from Theorem 2 that a trace language L is \mathcal{L} -trivial iff it is definable in $\text{LocTL}[\text{XF}]$.

In order to obtain a translation of $\text{LocTL}[\text{XF}]$, or more generally, $\text{LocTL}[\text{XF}, \text{YP}]$ event formulas and sentences into cascade products of asynchronous devices, as in Lemma 21 and Theorem 22 for $\text{LocTL}[\text{YP}]$, we work with *localized* modalities XF_i and YP_i and use *non-deterministic* asynchronous automata/transducers.

We can define the localized logic $\text{LocTL}[\text{XF}_i, \text{YP}_i]$ by using the process indexed modalities XF_i , YP_i and E_i instead of XF , YP and E as in Section 4.1. The semantics of XF_i modality is as follows: for a trace $t = (E, \leq, \lambda)$ and $e \in E$, $t, e \models \text{XF}_i \varphi$ iff e is an i -event and there exists an i -event f with $e < f$ such that $t, f \models \varphi$. It can be shown that $\text{LocTL}[\text{XF}_i, \text{YP}_i]$ has the same expressive power as $\text{LocTL}[\text{XF}, \text{YP}]$. Observe that $\text{LocTL}[\text{YP}_i]$ is a fragment of $\text{LocTL}[\text{XF}_i, \text{YP}_i]$. One can similarly define the fragment $\text{LocTL}[\text{XF}_i]$ of $\text{LocTL}[\text{XF}_i, \text{YP}_i]$ which has the same expressive power as $\text{LocTL}[\text{XF}]$.

Now we turn our attention to local cascade product of non-deterministic asynchronous automata and transducers. For the purpose of this work, it suffices to work with deterministic and *reverse* deterministic versions and their cascade products.

We begin with the key example of the reverse deterministic asynchronous automata $U_1^r[i]$. In $U_1^r[i]$, the local state set of process i has two states, that is, $S_i = \{1, 2\}$ while all other processes have a singleton state set $S_j = \{1\}$ for $j \neq i$. For a letter $a \notin \Sigma_i$, the processes participating in a have a deterministic transition, namely, a self-loop.



■ **Figure 2** $U_1^r[i]$ automata: on the left is the local automaton for process i , on the right is the local single state automaton for every other process $j \neq i$.

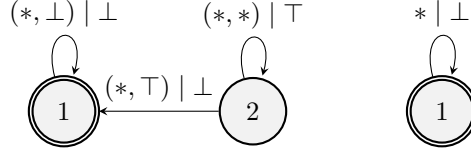
If $a \in \Sigma_i = \Sigma_{i1} \uplus \Sigma_{i2}$, we can identify a joint a -state with the local state of process i . With this identification, process i has non-deterministic transitions as shown in Figure 2. For instance, assume $a \in \Sigma_{i2}$. Then there is no transition on a at state 1. Further, at state 2, on a , the next state can be either state 1 or state 2. Note that these local transitions are obtained by simply *reversing* the local transitions in the *deterministic* $U_1[i]$ automata.

We *always* use the global state 1 as the only *final/accepting* state. It is important to observe that, for a trace t , $U_1^r[i]$ admits a *unique accepting run* ρ_t (which ends in global state 1). If t has no i -event with label in Σ_{i2} , then ρ_t must begin in global state 1 and stays there until the end. However, if t has an i -event with label in Σ_{i2} , then ρ_t must begin in the global state 2 and process i component changes its local state from 2 to 1 on the *last* occurrence of an i -event with label in Σ_{i2} . After this crucial event, process i stays at local state 1 as all subsequent “future” i -events are labelled by letters in Σ_{i1} . This unambiguity property of $U_1^r[i]$ is crucial. While using $U_1^r[i]$ as a language acceptor, we allow a *set* of initial global states. Thanks to the unambiguity property, the language accepted with single initial global state 1 is the complement of the language accepted with single initial global state 2.

Now we introduce the reverse deterministic asynchronous Γ -transducers $\widehat{U_1^r[i]}$. The underlying automaton of $\widehat{U_1^r[i]}$ is $U_1^r[i]$ and the output function provides a label from Γ on each local transition of $U_1^r[i]$. In $\widehat{U_1^r[i]}$, we allow any initial global state but insist

that 1 is the only accepting state. We also associate with $\widehat{U_1^r[i]}$ a Γ -labelling function $\widehat{U_1^r[i]}: TR(\tilde{\Sigma}) \rightarrow TR(\tilde{\Sigma} \times \Gamma)$ as follows: let $t = (E, \leq, \lambda)$ and ρ_t be the unique accepting run of the underlying $U_1^r[i]$ on t . We define $\widehat{U_1^r[i]}(t)$ to be the trace $t' = (E, \leq, (\lambda, \mu))$ where, for $e \in E$, $\mu(e)$ is the label from Γ on the transition used to “read” e on the unique run ρ_t .

Let $\Gamma = \{\top, \perp\}$. We now construct an instance of a Γ -transducer $\widehat{U_{XF_i}^r[i]}$ over $\Sigma \times \Gamma$. This transducer, called $\widehat{U_{XF_i}}$ below, implements the localized modality XF_i .



■ **Figure 3** $\widehat{U_{XF_i}^r[i]}$ transducer which implements XF_i modality.

Consider a $\text{LocTL}[XF_i, YP_i]$ formula $\varphi = XF_i \varphi'$. We have the Γ -relabelling functions $\theta^{\varphi'}$ and θ^φ which keep track of truth values of φ' and φ respectively. Let $t = (E, \leq, \lambda)$ be a trace over Σ and $\theta^{\varphi'}(t) = (E, \leq, (\lambda, \mu'))$ and $\theta^\varphi(t) = (E, \leq, (\lambda, \mu))$ be traces over $\Sigma \times \Gamma$. For any event e of t , $\mu'(e) = \top$ iff $t, e \models \varphi'$ and $\mu(e) = \top$ iff $t, e \models \varphi$. Fix an event $e \in E$. By semantics of XF_i , $\mu(e) = \top$ iff e is an i -event and there exists a *strict future* i -event f (that is, $e < f$) such that $\mu'(f) = \top$. For the Γ -transducer $\widehat{U_{XF_i}^r[i]}$ over $\Sigma \times \Gamma$, depicted in Figure 3, it is easy to verify that, for every trace t over Σ , $\widehat{U_{XF_i}^r[i]}((\theta^{\varphi'}(t))) = \theta^\varphi(t)$.

In view of the above discussion, it is not surprising that the following theorem holds. We skip the technical details.

► **Theorem 25.** *Let $L \subseteq TR(\tilde{\Sigma})$. Then the following are equivalent.*

1. L can be expressed by a sentence in $\text{LocTL}[XF, YP]$.
2. L can be accepted by a cascade product of $U_1[i]$ and $U_1^r[i]$ automata.

Even over words, the above result appears to be new. Further, we note that it has been shown in [17] (also see [11], [10]) that, over trace languages, $\text{LocTL}[XF, YP]$ definability coincides with recognizability by monoids from the class DA.

The following theorem is a dual of our main theorem.

► **Theorem 26.** *Let $L \subseteq TR(\tilde{\Sigma})$. The following are equivalent.*

1. L is a \mathcal{L} -trivial trace language.
2. L can be expressed by a sentence in $\text{LocTL}[XF]$.
3. L can be accepted by a cascade product of $U_1^r[i]$ automata.

Recall that a trace language is \mathcal{J} -trivial iff it is \mathcal{R} -trivial as well as \mathcal{L} -trivial. This observation leads to the following result.

► **Theorem 27.** *Let $L \subseteq TR(\tilde{\Sigma})$. The following are equivalent.*

1. L is a \mathcal{J} -trivial trace language.
2. L can be expressed by a sentence in $\text{LocTL}[XF]$ as well as a sentence in $\text{LocTL}[YP]$.
3. L can be accepted by a cascade product of $U_1^r[i]$ automata as well as another cascade product of $U_1[i]$ automata.

References

- 1 Bharat Adsul, Paul Gastin, Shantanu Kulkarni, and Pascal Weil. An expressively complete local past propositional dynamic logic over Mazurkiewicz traces and its applications. In *Proceedings of the 39th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS'24*, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3661814.3662110.
- 2 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. doi:10.4230/LIPICs.CONCUR.2020.19.
- 3 Bharat Adsul, Paul Gastin, Saptarshi Sarkar, and Pascal Weil. Asynchronous wreath product and cascade decompositions for concurrent behaviours. *Log. Methods Comput. Sci.*, 18, 2022. doi:10.46298/LMCS-18(2:22)2022.
- 4 Bharat Adsul and Milind A. Sohoni. Asynchronous automata-theoretic characterization of aperiodic trace languages. In Kamal Lodaya and Meena Mahajan, editors, *FSTTCS 2004: Foundations of Software Technology and Theoretical Computer Science, 24th International Conference, Chennai, India, December 16-18, 2004, Proceedings*, volume 3328 of *Lecture Notes in Computer Science*, pages 84–96. Springer, 2004. doi:10.1007/978-3-540-30538-5_8.
- 5 Mikołaj Bojańczyk. Languages recognised by finite semigroups, and their generalisations to objects such as trees and graphs, with an emphasis on definability in monadic second-order logic, 2020. arXiv:2008.11635.
- 6 J.A. Brzozowski and Faith E. Fich. Languages of R-trivial monoids. *Journal of Computer and System Sciences*, 20(1):32–49, 1980. doi:10.1016/0022-0000(80)90003-3.
- 7 Joëlle Cohen, Dominique Perrin, and Jean-Eric Pin. On the expressive power of temporal logic. *Journal of Computer and System Sciences*, 46(3):271–294, 1993. doi:10.1016/0022-0000(93)90005-H.
- 8 Volker Diekert and Paul Gastin. Pure future local temporal logics are expressively complete for Mazurkiewicz traces. *Information and Computation*, 204(11):1597–1619, November 2006. doi:10.1016/J.IC.2006.07.002.
- 9 Volker Diekert, Paul Gastin, and Manfred Kufleitner. A survey on small fragments of first-order logic over finite words. *Int. J. Found. Comput. Sci.*, 19(3):513–548, 2008. doi:10.1142/S0129054108005802.
- 10 Volker Diekert, Martin Horsch, and Manfred Kufleitner. On first-order fragments for Mazurkiewicz traces. *Fundamenta Informaticae*, 80(1-3):1–29, 2007. URL: <http://content.iiospress.com/articles/fundamenta-informaticae/fi80-1-3-02>.
- 11 Volker Diekert and Manfred Kufleitner. On first-order fragments for words and Mazurkiewicz traces: a survey. In *Proceedings of the 11th international conference on Developments in language theory*, pages 1–19, 2007.
- 12 Volker Diekert and Grzegorz Rozenberg, editors. *The Book of Traces*. World Scientific, 1995. doi:10.1142/2563.
- 13 Werner Ebinger and Anca Muscholl. Logical definability on infinite traces. *Theor. Comput. Sci.*, 154(1):67–84, 1996. doi:10.1016/0304-3975(95)00130-1.
- 14 Kousha Etessami and Thomas Wilke. An until hierarchy and other applications of an Ehrenfeucht-Fraïssé game for temporal logic. *Information and Computation*, 160(1):88–108, 2000. doi:10.1006/inco.1999.2846.
- 15 Giovanna Guaiana, Antonio Restivo, and Sergio Salemi. On aperiodic trace languages. In Christian Choffrut and Matthias Jantzen, editors, *STACS 91, 8th Annual Symposium on Theoretical Aspects of Computer Science, Hamburg, Germany, February 14-16, 1991, Proceedings*, volume 480 of *Lecture Notes in Computer Science*, pages 76–88. Springer, 1991. doi:10.1007/BFB0020789.

- 16 Johan Anthony Willem Kamp. *Tense Logic and the Theory of Linear Order*. University of California, Los Angeles, CA, USA, 1968.
- 17 Manfred Kufleitner. Polynomials, fragments of temporal logic and the variety DA over traces. *Theoretical Computer Science*, 376:89–100, 2007. Special issue DLT 2006. doi:10.1016/J.TCS.2007.01.014.
- 18 Robert McNaughton and Seymour A Papert. *Counter-Free Automata (MIT research monograph no. 65)*. The MIT Press, 1971.
- 19 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.
- 20 M.P. Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8(2):190–194, 1965. doi:10.1016/S0019-9958(65)90108-7.
- 21 Howard Straubing. *Finite automata, formal logic, and circuit complexity*. Birkhäuser Verlag, Basel, Switzerland, 1994.
- 22 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.
- 23 Wiesław Zielonka. Notes on finite asynchronous automata. *RAIRO Theor. Informatics Appl.*, 21(2):99–135, 1987. doi:10.1051/ita/1987210200991.