ASYNCHRONOUS WREATH PRODUCT AND CASCADE DECOMPOSITIONS FOR CONCURRENT BEHAVIOURS

BHARAT ADSUL [a], PAUL GASTIN [b,d], SAPTARSHI SARKAR [a], AND PASCAL WEIL [c,d]

[a] Indian Institute of Technology Bombay, Mumbai, India
    e-mail address: adsul@cse.iitb.ac.in, sapta@cse.iitb.ac.in

[b] Université Paris-Saclay, ENS Paris-Saclay, CNRS, LMF, 91190, Gif-sur-Yvette, France
    e-mail address: paul.gastin@ens-paris-saclay.fr

[c] Univ. Bordeaux, LaBRI, CNRS UMR 5800, F-33400 Talence, France
    e-mail address: pascal.weil@labri.fr

[d] CNRS, ReLaX, IRL 2000, Siruseri, India

ABSTRACT. We develop new algebraic tools to reason about concurrent behaviours modelled as languages of Mazurkiewicz traces and asynchronous automata. These tools reflect the distributed nature of traces and the underlying causality and concurrency between events, and can be said to support true concurrency. They generalize the tools that have been so efficient in understanding, classifying and reasoning about word languages. In particular, we introduce an asynchronous version of the wreath product operation and we describe the trace languages recognized by such products (the so-called asynchronous wreath product principle). We then propose a decomposition result for recognizable trace languages, analogous to the Krohn-Rhodes theorem, and we prove this decomposition result in the special case of acyclic architectures. Finally, we introduce and analyze two distributed automata-theoretic operations. One, the local cascade product, is a direct implementation of the asynchronous wreath product operation. The other, global cascade sequences, although conceptually and operationally similar to the local cascade product, translates to a more complex asynchronous implementation which uses the gossip automaton of Mukund and Sohoni. This leads to interesting applications to the characterization of trace languages definable in first-order logic: they are accepted by a restricted local cascade product of the gossip automaton and 2-state asynchronous reset automata, and also by a global cascade sequence of 2-state asynchronous reset automata. Over distributed alphabets for which the asynchronous Krohn-Rhodes theorem holds, a local cascade product of such automata is sufficient and this, in turn, leads to the identification of a simple temporal logic which is expressively complete for such alphabets.

1. Introduction

Algebraic automata theory, that is, the use of algebraic tools and notions such as monoids, morphisms and varieties, has been very successful in classifying recognizable word languages, from both the theoretical and the algorithmic points of view, offering structural descriptions of, say, logically defined classes of languages and decision algorithms for membership in these classes [Eil76, Str94, Pin19]. The purpose of this paper is to extend some of this approach to the concurrent setting. We are particularly interested in decomposition results, in the spirit of the Krohn-Rhodes theorem, and their applications to the study of first-order definable trace languages.

Let us first specify our model of concurrency. Words represent sequential behaviours: a sequence of letters models a sequence of events, occurring on a single process. In a concurrent setting involving multiple processes, we work with the well established (Mazurkiewicz) traces [Maz77, DR95]: a trace represents a concurrent behaviour as a labelled partial order which captures the distribution of events across processes, as well as causality and concurrency between them.

The notion of a recognizable trace language is also very well established: a set of traces is recognizable if the set of all the words representing these traces is a regular language. A key contribution, due to Zielonka, is the description of an automata-theoretic model for the acceptance of recognizable trace languages, namely asynchronous automata [Zie87]. These automata, with their local state sets (one for each process), are natural distributed devices, which run on input traces in a distributed fashion, respecting the underlying causality and concurrency between events. More precisely, when working on an event during a run on an input trace, an asynchronous automaton updates only the local states of the processes participating in that event; the other processes remain oblivious to the occurrence of this event. Zielonka's theorem states that every recognizable trace language is accepted by an asynchronous automaton.

Early results seemed to indicate that the algebraic approach could be neatly transferred to recognizable trace languages, with the monoid-theoretic definition of recognizability matching the operational model of asynchronous automata (this is Zielonka's theorem mentioned above [Zie87]) and the characterization of star-free and first-order definable trace languages (Guaiana *et al.* [GRS92], Ebinger and Muscholl [EM96]) in terms of aperiodic monoids. Very few significant results in this direction emerged since, and especially no strong Krohn-Rhodes like decomposition results[1].

There are indeed deep, technical obstacles to this approach, which are discussed in some detail in [Sar22, AGSW21]. Our first contribution is the introduction of better suited notions of *asynchronous transformation monoids*, *asynchronous morphisms* and *asynchronous wreath products*. Because these notions closely adhere to the distributed nature of traces, we obtain important results, such as a so-called *asynchronous wreath product principle*, describing the languages recognized by an asynchronous wreath product of asynchronous transformation monoids. Moreover, just as (ordinary) transformation monoids model DFAs and their wreath product models the cascade product of DFAs, our asynchronous wreath product of asynchronous transformation monoids can be implemented as a *local cascade product* of asynchronous automata. The local cascade product, in its purely automata-theoretic form,

---

[1]An exception may be Guaiana et al.'s [GMPW98], which gives a wreath product principle for traces. This work however uses non-trace structures (structures that ignore the distributed nature of the alphabet) to circumvent technical difficulties, thus limiting its relevance.

appeared in Adsul and Sohoni [AS04, Ads04] in a characterization of first-order definable trace languages in terms of asynchronous automata.

The next question is that of the possibility of a Krohn-Rhodes theorem in this setting. The classical (sequential) Krohn-Rhodes theorem states that every morphism from the free monoid Σ^* to a transformation monoid is simulated (see Definition 3.1 for a formal definition of simulation) by a morphism to the wreath product of particular simple transformation monoids: copies of the 2-state reset transformation monoids and transformation groups based on the simple groups dividing the given monoid of transformations. This has many applications, in particular to the description of simple automata (cascade products of reset automata) accepting all first-order definable languages, see [Str94].

The asynchronous analogue would state that any morphism from a trace monoid to a transformation monoid is simulated by an asynchronous morphism to an asynchronous wreath product of localized reset automata and transformation groups (where localization means that all non-trivial actions take place on a single designated process). We cannot conclude, at this stage, that every distributed alphabet has this property, which we call the asynchronous Krohn-Rhodes property (aKR), but we identify a large class of alphabets for which it holds, namely those where the communication graph between processes is acyclic. The question is raised of which distributed alphabets have the aKR property.

Then we focus on first-order definable trace languages, which we know are the trace languages recognized by aperiodic transformation monoids [EM96]. Thus, over aKR distributed alphabets (*e.g.*, acyclic architectures), these languages are exactly those that are recognized by an asynchronous wreath product of localized reset asynchronous transformation monoids (or, equivalently, by a local cascade product of localized reset asynchronous automata). We show that, over arbitrary distributed alphabets, the trace languages recognized by such a product are exactly those which are definable in a natural local temporal logic using 'process-based strict since' operators as its only modalities. This logic, $\mathsf{LocTL}[\mathsf{S}_i]$, is closely related to an expressively complete logic introduced by Diekert and Gastin [DG06]. Our result implies that $\mathsf{LocTL}[\mathsf{S}_i]$ has the same expressive power as first-order logic over aKR distributed alphabets.

Over arbitrary alphabets again, we show that all first-order definable languages are accepted by a local cascade product of the gossip automaton (introduced by Mukund and Sohoni [MS97]), followed by localized reset asynchronous automata.

We know from Adsul and Sohoni [AS04] that gossip asynchronous automata exhibit non-aperiodic behaviour, which is contrary to what we expect when discussing first-order definable languages. In order to avoid this situation, we introduce two new operations. One is the *restricted local cascade product with the gossip automaton*, which exploits the constants of an expressively complete logic called $\mathsf{LocTL}[\mathsf{Y}_i \leq \mathsf{Y}_j, \mathsf{S}_i]$, and strongly encapsulates the non-aperiodic behaviour of the gossip automaton. We show that the first order definable trace languages are exactly the trace languages accepted by a restricted cascade product of the gossip automaton with a local cascade product of localized reset asynchronous automata, thus adding a new characterization for this important class.

The second of these operations is the *global cascade sequence* of asynchronous automata. Such a device is not properly an asynchronous automaton itself, but it composes, in a novel way, the runs of the automata in the sequence to produce an acceptance mechanism. We exploit another expressively complete extension of $\mathsf{LocTL}[\mathsf{S}_i]$ called $\mathsf{LocTL}[\mathsf{Y}_i, \mathsf{S}_i]$ to show that the first order definable trace languages are exactly the trace languages accepted by global

cascade sequences of localized reset asynchronous automata, yet another characterization of this language class.

Independently of this characterization, we discuss how global cascade sequences can be implemented by an asynchronous automaton, in the form of a restricted local cascade product of the gossip automaton with the *global-state detectors* for the factors of the sequence.

Overview. Before we dive into the body of the paper, and for the benefit of readers already fluent in the vocabulary of traces and asynchronous automata, it is worth discussing in a little more detail some of the inner workings of these results, especially around the notions of accepting vs. computing (or relabelling) devices, and of sequential or asynchronous transducer.

The classical *sequential transducer* σ_A associated with a DFA A (or with a morphism from a free monoid to a transformation monoid) maps each word w to a word of equal length, obtained by replacing letter a in position i by the pair (a, q), where q is the state reached after reading the length $(i - 1)$ prefix of w (starting at the initial state of A). If we see the automaton A not as an acceptor but as a computing device, outputting on every transition the full information it has, *i.e.*, the name of that transition (the pair composed of the label and the source state of the transition), then $\sigma_A(w)$ is the output of A on input w. It is reasonable to view σ_A as the most general function computed by A. Moreover, the cascade product of DFAs A and B can be described as the operation of A on an input word w, followed by the operation of B on input $\sigma_A(w)$. In particular, the sequential transducer of the cascade product of A and B is the composition $\sigma_B \circ \sigma_A$.

This idea of decorating each position in a word with information computed by an automaton carries over nicely to the distributed setting: we can decorate every event in a trace t with information computed by an asynchronous automaton A, thus viewing A as an asynchronous computing device. However, the distinction between local and global states in asynchronous automata leads to two distinct approaches, both discussed in this paper.

One relies on local information. More precisely, we define the *asynchronous transducer* χ_A, which maps a trace t to a trace with the same underlying poset, where the label of an event e, say a, is replaced with the pair (a, s_a) as follows. If t_e is the prefix of t which consists of all events strictly less than e (that is, the strict causal past of e), s_a is the collection of local states reached after reading t_e at all the processes which participate in a. Again, one can see χ_A as the most general function computed by A when considering only the local states. With this notion in hand, stating and proving the asynchronous wreath product principle, which describes the languages recognized by an asynchronous wreath product of asynchronous transformation monoids, while technically more demanding, proceeds along the same lines as in the sequential case. Also, the local cascade product of asynchronous automata $A \circ_\ell B$ can be viewed as the operation of A, running on trace t and outputting trace $\chi_A(t)$, followed by automaton B running on $\chi_A(t)$. Here again, the asynchronous transducer of the local cascade product $A \circ_\ell B$ is $\chi_B \circ \chi_A$.

Global cascade sequences are defined in the same spirit, using global rather than local state information. With the same notation as above, the *global-state labelling function* ζ_A replaces the label a of event e with the pair (a, s) where s is the global state reached after reading t_e. This is a different function that can be said to be globally computed by A, and the definition of global cascade sequences corresponds exactly to the composition of these global-state labelling functions.

Both the asynchronous transducer and the global-state labelling function are asynchronous computing devices but the latter carries more information. This is reflected in the different ways these functions can be implemented. The composition of the asynchronous transducers of (appropriately defined) asynchronous automata, as we explained, translates directly to the local cascade product of these automata. In contrast, an additional ingredient is required to implement a global cascade sequence by means of an asynchronous automaton, that is, to make the global information carried by ζ_A known to local states. This ingredient is the restricted cascade product with the gossip automaton.

Organization. We now briefly describe the organization of the paper. Section 2 summarizes the necessary notions on distributed alphabets, traces, transformation monoids, recognizable trace languages and asynchronous automata — including the statement of Zielonka's theorem.

We begin Section 3 with a brief account of the wreath product operation on transformation monoids, the notion of simulation and the Krohn-Rhodes theorem. We introduce our notions of asynchronous transformation monoids (Section 3.2) — directly in the spirit of Zielonka's asynchronous automata —, asynchronous morphisms (Section 3.3) and asynchronous wreath products (Section 3.4). Asynchronous transducers are used in Section 3.5 to state and prove the asynchronous wreath product principle (Theorem 3.24). The question of the implementation of the asynchronous wreath product, in the form of the local cascade product of asynchronous automata, is discussed in Section 3.6. Finally, in Section 3.7, we formulate and briefly discuss the asynchronous Krohn-Rhodes property of distributed alphabets.

Section 4 is entirely dedicated to the proof that the asynchronous Krohn-Rhodes property holds for distributed alphabets over an acyclic architecture (Theorem 4.2). The next section explores the applications of the notions of asynchronous wreath product and local cascade product to the special case of first-order definable languages: the characterization of the trace languages recognized by asynchronous wreath products of localized resets by $\mathsf{LocTL}[\mathsf{S}_i]$ (Theorem 5.6), which therefore is expressively complete over aKR distributed alphabets (Theorem 5.7); and the characterization of the first-order definable trace languages by means of the restricted local cascade product of the gossip automaton with local cascade product of localized asynchronous reset automata (Theorem 5.14). This characterization is obtained using a new local temporal logic $\mathsf{LocTL}[\mathsf{Y}_i \leq \mathsf{Y}_j, \mathsf{S}_i]$ which is proved to be expressively complete in Theorem 5.5.

The notions of global-state labelling function associated with an asynchronous automaton, and of global cascade sequences as computing and accepting devices are introduced in Section 6, where we prove the characterization of first-order definable languages in terms of global cascade sequences of asynchronous localized reset automata (Theorem 6.12).

Section 7 completes the operational point of view on the latter results by presenting an asynchronous implementation of global cascade sequences (Theorems 7.3 and 7.7). Finally, Section 8 outlines an intriguing question left open by this work.

The results in this paper are an elaboration and an extension of those presented at CONCUR 2020 [AGSW20]. Proofs of several key results e.g. Theorem 4.2, Theorem 5.6 and Theorem 6.12 are unavailable in the conference proceedings and are included here. The complete Section 7 discussing implementation of a global cascade sequence as an asynchronous automaton is a technical addition that is briefly alluded to in our conference paper without details. Finally, Section 5 includes significant new technical contributions that

originate from introducing temporal logic constants $\mathsf{Y}_i \leq \mathsf{Y}_j$ in the logic under consideration. This leads to a new characterization of first-order definable trace languages in Theorem 5.14.

2. Preliminaries

2.1. **Basic notions in trace theory.** Let $\mathscr{P}$ be a finite set of agents/processes. If $\mathscr{P}$ is clear from the context, we use the simpler notation $\{X_i\}$ to denote a $\mathscr{P}$-indexed family $\{X_i\}_{i \in \mathscr{P}}$.

A *distributed alphabet* over $\mathscr{P}$ is a family $\widetilde{\Sigma} = \{\Sigma_i\}$, where the Σ_i are non-empty finite sets that may overlap one another. Let $\Sigma = \bigcup_{i \in \mathscr{P}} \Sigma_i$. The *location function* $\mathrm{loc} \colon \Sigma \to 2^{\mathscr{P}}$ is defined by setting $\mathrm{loc}(a) = \{i \in \mathscr{P} \mid a \in \Sigma_i\}$. Note that from Σ and loc, one can reconstruct the distributed alphabet, and hence we also use the notation (Σ, loc) for distributed alphabet. The corresponding *trace alphabet* is the pair (Σ, I), where I is the *independence relation* $I = \{(a,b) \in \Sigma^2 \mid \mathrm{loc}(a) \cap \mathrm{loc}(b) = \emptyset\}$ induced by (Σ, loc). The corresponding *dependence relation* is $D = \Sigma^2 \setminus I$.

A Σ-labelled poset is a structure $t = (E, \leq, \lambda)$ where E is a set, $\leq$ is a partial order on E and $\lambda \colon E \to \Sigma$ is a labelling function. For $e, e' \in E$, define $e \lessdot e'$ if and only if $e < e'$ and for each e'' with $e \leq e'' \leq e'$ either $e = e''$ or $e' = e''$. For $X \subseteq E$, let $\downarrow X = \{y \in E \mid y \leq x \text{ for some } x \in X\}$. For $e \in E$, we abbreviate $\downarrow\{e\}$ by simply $\downarrow e$. We also write $\Downarrow e = \downarrow e \setminus \{e\}$ for the *strict past* of e.

A *trace* over (Σ, loc) is a finite Σ-labelled poset $t = (E, \leq, \lambda)$ such that

- If $e, e' \in E$ with $e \lessdot e'$ then $(\lambda(e), \lambda(e')) \in D$
- If $e, e' \in E$ with $(\lambda(e), \lambda(e')) \in D$, then $e \leq e'$ or $e' \leq e$

Let $\mathrm{Tr}(\Sigma, \mathrm{loc})$ denote the set of all traces over (Σ, loc); if loc is clear from the context, we simply use the notation $\mathrm{Tr}(\Sigma)$. Henceforth a trace means a trace over (Σ, loc) unless specified otherwise. Now we turn our attention to the important operation of concatenation of traces. Let $t = (E, \leq, \lambda) \in \mathrm{Tr}(\Sigma)$ and $t' = (E', \leq', \lambda') \in \mathrm{Tr}(\Sigma)$. Without loss of generality, we can assume E and E' to be disjoint. We define $tt' \in \mathrm{Tr}(\Sigma)$ to be the trace $(E'', \leq'', \lambda'')$ where

- $E'' = E \cup E'$,
- $\leq''$ is the transitive closure of $\leq \cup \leq' \cup \{(e, e') \in E \times E' \mid (\lambda(e), \lambda'(e')) \in D\}$,
- $\lambda'' \colon E'' \to \Sigma$ where $\lambda''(e) = \lambda(e)$ if $e \in E$; otherwise, $\lambda''(e) = \lambda'(e)$.

This operation, henceforth referred to as *trace concatenation*, gives $\mathrm{Tr}(\Sigma)$ a monoid structure. A trace t' is said to be a *prefix* of a trace t if there exists t'' such that $t = t't''$.

Observe that, with a (resp. b) denoting the singleton trace whose only event is labelled a (resp. b), if $(a, b) \in I$ then $ab = ba$ in $\mathrm{Tr}(\Sigma)$. A basic result in trace theory gives a presentation of the trace monoid as a quotient of the *free* word monoid Σ^* by the congruence $\sim_I \subseteq \Sigma^* \times \Sigma^*$ generated by $ab \sim_I ba$ for $(a, b) \in I$. See [DR95] for more details.

Proposition 2.1. *The canonical morphism from $\Sigma^* \to \mathrm{Tr}(\Sigma)$, sending a letter $a \in \Sigma$ to the trace a, factors through the quotient monoid $\Sigma^*/\!\!\sim_I$ and induces an isomorphism from $\Sigma^*/\!\!\sim_I$ to the trace monoid $\mathrm{Tr}(\Sigma)$.*

Let $t = (E, \leq, \lambda) \in \mathrm{Tr}(\Sigma)$. The elements of E are referred to as *events* in t and for an event e in t, $\mathrm{loc}(e)$ abbreviates $\mathrm{loc}(\lambda(e))$. Further, let $i \in \mathscr{P}$. The set of i-events in t is $E_i = \{e \in E \mid i \in \mathrm{loc}(e)\}$. This is the set of events in which process i participates. It is clear that E_i is totally ordered by $\leq$.

Note that, if we restrict the trace t to a downward-closed subset of events $c = {\downarrow}c$, we get another trace $(c, \leq, \lambda)$ which is a prefix of t. In fact, every prefix of t arises this way, and we often identify prefixes with downward-closed sets of events. Examples of prefixes are defined by the empty set, E itself and, more importantly, ${\downarrow}e$ or ${\Downarrow}e$, for every event $e \in E$.

2.2. Recognizable trace languages.
A map from a set X to itself is called a *transformation* of X. The set $\mathcal{F}(X)$ of all such transformations forms a monoid under function composition: $fg = g \circ f$. A *transformation monoid* (or simply *tm*) is a pair $T = (X, M)$ where M is a submonoid of $\mathcal{F}(X)$. The tm (X, M) is called *finite* if X is finite.

Example 2.2. Let $X = \{1, 2\}$ and let r_1, r_2 be the constant maps on X, mapping each element to 1 and 2, respectively. Then $M = \{\mathrm{id}_X, r_1, r_2\}$ is a monoid. We denote the tm (X, M) by U_2.

Example 2.3. Let M be a monoid. For each $m \in M$, let t_m be the transformation of M defined by $t_m(x) = xm$ for all $x \in M$. The map $m \mapsto t_m$ is an injective morphism from M to $\mathcal{F}(M)$, allowing us to view M as a submonoid of $\mathcal{F}(M)$. We denote by (M, M) the resulting tm.

Let N be a monoid and let $T = (X, M)$ be a tm. By a morphism φ from N to T, we mean a (monoid) morphism $\varphi \colon N \to M$. We abuse the notation and also write this as $\varphi \colon N \to T$.

Remark 2.4. Morphisms from free word monoids to transformation monoids almost correspond to deterministic and complete automata, the only difference being that an automaton also includes an initial state and a set of final states. More precisely, let Σ be a finite alphabet, $T = (X, M)$ be a tm and $\varphi \colon \Sigma^* \to T$ be a morphism. In the corresponding automaton, X is the set of states and for each $a \in \Sigma$, $\varphi(a) \in M \subseteq \mathcal{F}(X)$ defines the deterministic and complete transition function for letter a. Conversely, a deterministic and complete automaton A over Σ, defines a tm $T = (X, M)$ where X is the set of states of A, M is the *transition monoid* of A and a surjective morphism $\varphi \colon \Sigma^* \to M$ as follows. For each $a \in \Sigma$, let $\varphi(a) \in \mathcal{F}(X)$ be the transformation on X induced by the transition function for letter a in A. We obtain a morphism $\varphi \colon \Sigma^* \to \mathcal{F}(X)$ and we let $M = \varphi(\Sigma^*)$ be the submonoid of $\mathcal{F}(X)$ generated by $\{\varphi(a)\}_{a \in \Sigma}$. For instance, the tm U_2 defined in Example 2.2 corresponds to the DFA in Figure 1 below, and the induced morphism is given by $\varphi(a) = r_1$, $\varphi(b) = r_2$ and $\varphi(c) = \mathrm{id}_X$.
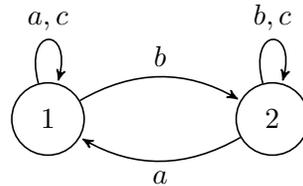

FIGURE 1. Automaton corresponding to the tm U_2

We now fix a distributed alphabet (Σ, loc). Let $\varphi \colon \text{Tr}(\Sigma) \to M$ be a morphism to a monoid M. We note that, if $(a, b) \in I$, then $ab = ba$ in $\text{Tr}(\Sigma)$ and hence $\varphi(a)$ and $\varphi(b)$ commute in M. In fact, in view of Proposition 2.1, any function $\varphi \colon \Sigma \to M$ which has the property that $\varphi(a)$ and $\varphi(b)$ commute for every $(a, b) \in I$, can be uniquely extended to a morphism from $\text{Tr}(\Sigma)$ to M.

Transformation monoids can be naturally used to recognize trace languages. We say that a trace language $L \subseteq \text{Tr}(\Sigma)$ is *recognized by* a tm $T = (X, M)$ if there exists a morphism $\varphi \colon \text{Tr}(\Sigma) \to T$, an *initial* element $x_{\text{in}} \in X$ and a *final* subset $X_{\text{fin}} \subseteq X$ such that $L = \{t \in \text{Tr}(\Sigma) \mid \varphi(t)(x_{\text{in}}) \in X_{\text{fin}}\}$. A trace language is said to be *recognizable* if it is recognized by a finite tm (see [DR95, Die90]).

Remark 2.5. A morphism $\varphi \colon \text{Tr}(\Sigma) \to (X, M)$ from the trace monoid into a tm can also be viewed, by Proposition 2.1, as a morphism from the free monoid Σ^* with the additional property that $\varphi(a)$ and $\varphi(b)$ commute for every $(a, b) \in I$. The automaton corresponding to $\varphi \colon \Sigma^* \to (X, M)$ (cf. Remark 2.4) thus has the property that ab and ba have the same state transitions for all $(a, b) \in I$. An automaton with this property is called a *diamond-automaton*. It is a classical sequential automata model for recognizing trace languages since all linearizations of a trace reach the same destination state starting from any given state. For instance, the DFA in Figure 1 is a diamond-automaton for the distributed alphabet $\{\Sigma_1 = \{a, c\}, \Sigma_2 = \{a, b\}\}$. Here $(b, c) \in I$ and indeed, the words bc and cb have identical state transitions in the DFA. Note however that the runs of the DFA on the two linearizations are different, namely, $1 \to 1 \to 2$ for cb and $1 \to 2 \to 2$ for bc.

2.3. **Asynchronous automata.** Recognizability of trace languages can also be seen as an automata-theoretic notion that is concurrent in nature. In the upcoming definition of an asynchronous automaton, the set of states is structured as a $\mathscr{P}$-indexed family of finite non-empty sets $\{S_i\}_{i \in \mathscr{P}}$. The elements of S_i are called the *local i-states*, or the *local states* of process i. If P is a non-empty subset of $\mathscr{P}$, a *P-state* is a map $s \colon P \to \bigcup_{i \in \mathscr{P}} S_i$ such that $s(j) \in S_j$ for every $j \in P$. We denote by S_P the set of all P-states and we call $S = S_{\mathscr{P}}$ the set of *global states*.[2]

If $P' \subseteq P$ and $s \in S_P$ then $s_{P'}$ denotes the restriction of s to P'. We use the shorthand $-P$ to indicate the complement of P in $\mathscr{P}$. We sometimes split a global state $s \in S$ as $(s_P, s_{-P}) \in S_P \times S_{-P}$. If $a \in \Sigma$, we talk about *a-states* to mean $\text{loc}(a)$-states and we write S_a for $S_{\text{loc}(a)}$. If $a \in \Sigma$, $\text{loc}(a) \subseteq P$ and s is a P-state, we write s_a for $s_{\text{loc}(a)}$.

Finally, we use the following notion of extension. If $P \subseteq \mathscr{P}$ and f is a transformation of S_P, the *extension* of f to S is the transformation $g \in \mathcal{F}(S)$ such that $(g(s))_P = f(s_P)$ and $(g(s))_{-P} = s_{-P}$. In other words, if $s = (s_P, s_{-P}) \in S$, then $g((s_P, s_{-P})) = (f(s_P), s_{-P})$. We observe that f is entirely determined by g and P. Extensions of transformations of S_P are called *P-maps* (see Section 3.2 for more details on P-maps).

Asynchronous automata were introduced by Zielonka for concurrent computation on traces [Zie87]. An *asynchronous automaton* A over (Σ, loc) is a tuple $(\{S_i\}_{i \in \mathscr{P}}, \{\delta_a\}_{a \in \Sigma}, s_{\text{in}})$ where

- S_i is a finite non-empty set of local i-states for each process i;
- For $a \in \Sigma$, $\delta_a \colon S_a \to S_a$ is a (complete) transition function on a-states;
- $s_{\text{in}} \in S$ is an initial global state.

---

[2]Note that we can naturally identify S_P with $\prod_{i \in P} S_i$ and S with $\prod_{i \in \mathscr{P}} S_i$.

Similar to $\mathscr{P}$-indexed families, we will follow the convention of writing $\{Y_a\}$ to denote the Σ-indexed family $\{Y_a\}_{a \in \Sigma}$.

Observe that an a-transition of A reads and updates only the local states of the agents which participate in a. As a result, actions which involve disjoint sets of agents are processed concurrently by A. For $a \in \Sigma$, let $\Delta_a \colon S \to S$ be the extension of $\delta_a \colon S_a \to S_a$. Clearly, if $(a, b) \in I$ then Δ_a and Δ_b commute. Hence, the (global) transition functions $\{\Delta_a\}$ induce a trace morphism $t \mapsto \Delta_t$ from $\mathrm{Tr}(\Sigma)$ to $\mathcal{F}(S)$. We denote by $A(t)$ the global state reached when running A on t, $A(t) = \Delta_t(s_{\mathrm{in}})$.

Let $L \subseteq \mathrm{Tr}(\Sigma)$ be a trace language. We say that L is *accepted* by A if there exists a subset $S_{\mathrm{fin}} \subseteq S$ of final global states such that $L = \{t \in \mathrm{Tr}(\Sigma) \mid A(t) \in S_{\mathrm{fin}}\}$.

The fundamental theorem of Zielonka [Zie87] states that a trace language is recognizable if and only if it is accepted by some asynchronous automaton (see [Muk12] for another proof of the theorem).

Asynchronous labelling functions. We will also use asynchronous automata to decorate events of a trace with extra information computed by the automaton. This is similar to the notion of *locally computable functions* defined in [MS97]. It extends to traces the notion of sequential letter-to-letter word transducers. When dealing with a trace $t = (E, \leq, \lambda)$, we wish to preserve the underlying poset $(E, \leq)$ and enrich the labels with extra information.

Formally, let (Σ, loc) be a distributed alphabet and Γ be a finite set. The alphabet $\Sigma \times \Gamma$ can be equipped with a distributed structure over $\mathscr{P}$ by letting $(\Sigma \times \Gamma)_i = \Sigma_i \times \Gamma$. As a result, the location of $(a, \gamma) \in \Sigma \times \Gamma$ is simply $\mathrm{loc}(a)$; thus we unambiguously reuse the notation loc for the location function of $\Sigma \times \Gamma$, and use the notation $\mathrm{Tr}(\Sigma \times \Gamma)$ to denote the set of traces over this distributed alphabet. A Γ-labelling function is a map $\theta \colon \mathrm{Tr}(\Sigma) \to \mathrm{Tr}(\Sigma \times \Gamma)$ such that, for $t = (E, \leq, \lambda) \in \mathrm{Tr}(\Sigma)$, we have $\theta(t) = (E, \leq, (\lambda, \mu))$, i.e., the map θ adds a new label $\mu(e) \in \Gamma$ to each event e in t.

For instance, given $i \in \mathscr{P}$, we may consider the $\{0, 1\}$-labelling function θ_i which decorates each event e of a trace t with $\mu_i(e) = 1$ if the strict causal past of e contains some event on process i, i.e., if $E_i \cap \Downarrow e \neq \emptyset$, and $\mu_i(e) = 0$ otherwise.

An *asynchronous (letter-to-letter) Γ-transducer* over (Σ, loc) is a tuple $\hat{A} = (A, \{\mu_a\})$ where $A = (\{S_i\}, \{\delta_a\}, s_{\mathrm{in}})$ is an asynchronous automaton and each μ_a $(a \in \Sigma)$ is a map $\mu_a \colon S_a \to \Gamma$. We associate with $\hat{A}$, a Γ-labelling function, also denoted by $\hat{A}$, $\hat{A} \colon \mathrm{Tr}(\Sigma) \to \mathrm{Tr}(\Sigma \times \Gamma)$ as follows: for $t = (E, \leq, \lambda) \in \mathrm{Tr}(\Sigma)$, we let $\hat{A}(t) = (E, \leq, (\lambda, \mu))$ where for all events $e \in E$ with $\lambda(e) = a$ and $s = A(\Downarrow e)$, we have $\mu(e) = \mu_a(s_a)$.

Given a Γ-labelling function θ, we say that $\hat{A}$ *computes* (or *implements*) θ if for every $t \in \mathrm{Tr}(\Sigma)$, $\hat{A}(t) = \theta(t)$. We also say that an asynchronous automaton $A = (\{S_i\}, \{\delta_a\}, s_{\mathrm{in}})$ *computes* θ if there are maps $\mu_a \colon S_a \to \Gamma$ such that $\hat{A} = (\{S_i\}, \{\delta_a\}, s_{\mathrm{in}}, \{\mu_a\})$ computes θ.

Notice that a Γ-labelling function θ is defined on all input traces, hence an asynchronous automaton which computes θ must be complete in the sense that it admits a run on all traces from $\mathrm{Tr}(\Sigma)$ and it does not use an acceptance condition: when considering an asynchronous automaton A which computes a Γ-labelling function, we always assume that all global states of A are accepting.

For instance, the above $\{0, 1\}$-labelling function θ_i can be computed by an asynchronous $\{0, 1\}$-transducer. We need two states $\{0, 1\}$ for each process. Initially, all processes start in state 0. When the first event e occurs on process i, all processes in $\mathrm{loc}(e)$ switch to state 1: for all $a \in \Sigma_i$, δ_a is the constant map sending all states in S_a to $(1, \ldots, 1)$. Then, the information is propagated via synchronizing events: for all $b \in \Sigma \setminus \Sigma_i$, the map δ_b sends

$(0, \ldots, 0)$ to itself and all other states to $(1, \ldots, 1)$. It is easy to add output functions $\{\mu_a\}$ in order to compute θ_i.

There exists a canonical (most general) function computed by an asynchronous automaton $A = (\{S_i\}, \{\delta_a\}, s_{\text{in}})$, called the *asynchronous transducer of A* and denoted χ_A. This function, which was already defined in [AS04], simply adds to an event e the local state information of A before executing e. Formally, letting $\Gamma_A = \bigcup_a S_a$, it is implemented by taking each output function μ_a to be the identity function from S_a to Γ_A. Notice that, traces in $\chi_A(\text{Tr}(\Sigma))$ have labels from $\{(a, s_a) \mid a \in \Sigma, s_a \in S_a\} \subseteq \Sigma \times \Gamma_A$. We denote by $\Sigma \times_\ell S$ the set $\{(a, s_a) \mid a \in \Sigma, s_a \in S_a\}$, and consider it a distributed alphabet as it naturally inherits the location function of $\Sigma \times \Gamma_A$, that is, $\text{loc}((a, s_a)) = \text{loc}(a)$. Clearly, all Γ-labelling functions computed by A are abstractions of χ_A.

3. Asynchronous structures and decomposition problems

This section is devoted to the development of new algebraic asynchronous structures. Our main interest is in transferring from the algebraic theory of word languages to trace languages the results and methods which rely on the wreath product operation: the wreath product principle (see [Str94]) and the Krohn-Rhodes theorem (see [Eil76]). We present a new algebraic framework for the setting of traces, and introduce an appropriate asynchronous wreath product operation; an asynchronous wreath product principle is also established that works in the realm of traces. This allows posing the question of a meaningful distributed analogue of the Krohn-Rhodes theorem which we then partially resolve in the remainder of this article.

3.1. Wreath product in sequential setting.
We first recall the definitions of division, simulation and wreath products in the context of transformation monoids, see [Eil76].

Figure 2. Conditions $\pi(n)(f(y)) = f(n(y))$ and $\varphi(a)(f(y)) = f(\psi(a)(y))$ in Definition 3.1

Definition 3.1. We say that a monoid M *divides* a monoid N (written $M \prec N$) if there exists a surjective morphism $\varphi \colon N' \to M$, defined on a submonoid N' of N.

We say that a tm (X, M) *divides* a tm (Y, N) (written $(X, M) \prec (Y, N)$) if there exists a surjective map $f \colon Y \to X$ and a surjective morphism $\pi \colon N' \to M$ defined on a submonoid N' of N, such that $\pi(n)(f(y)) = f(n(y))$ for all $n \in N'$ and all $y \in Y$, see Figure 2 (left).

Finally, given morphisms $\varphi \colon \Sigma^* \to T = (X, M)$ and $\psi \colon \Sigma^* \to T' = (Y, N)$, we say that ψ *simulates* φ if there exists a surjective map $f \colon Y \to X$ such that $\varphi(a)(f(y)) = f(\psi(a)(y))$ for all $a \in \Sigma$ and all $y \in Y$, see Figure 2 (right).
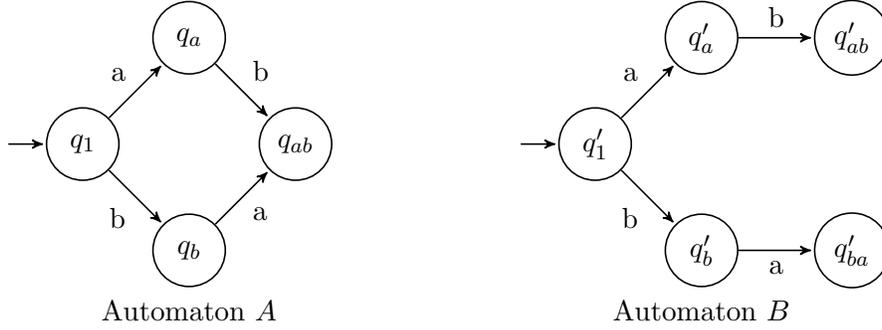
FIGURE 3. Automata A and B on alphabet $\Sigma = \{a, b\}$

Example 3.2. Automata A and B in Figure 3 define, respectively, morphisms $\varphi\colon \Sigma^* \to T$ and $\psi\colon \Sigma^* \to T'$, to transformation monoids $T = (X, M)$ and $T' = (Y, N)$ (cf. Remark 2.4). Note that to keep the pictures simple, the automata are not complete, all missing transitions go to an implicit sink state (state s for automaton A and s' for automaton B). In particular, $X = \{q_1, q_a, q_b, q_{ab}, s\}$ and M is generated by the transition functions $\varphi(a)$ and $\varphi(b)$ defined by A. We can check that ψ simulates φ using the surjective map $f\colon Y \to X$ defined by $q'_1 \mapsto q_1, q'_a \mapsto q_a, q'_b \mapsto q_b, q'_{ab} \mapsto q_{ab}, q'_{ba} \mapsto q_{ab}, s' \mapsto s$. The same map f, along with $\pi\colon N \to M$ as $\pi(\psi(w)) = \varphi(w)$, shows that T divides T'.

Remark 3.3. It is an elementary verification that, if a morphism $\psi\colon \Sigma^* \to T' = (Y, N)$ simulates a morphism $\varphi\colon \Sigma^* \to T = (X, M)$, then any language recognized by φ is also recognized by ψ. Moreover, if φ is onto M, then $(X, M) \prec (Y, N)$: this can be checked using $N' = \psi(\Sigma^*)$ and $\pi(\psi(w)) = \varphi(w)$.

Similarly, if $(X, M) \prec (Y, N)$ and $\varphi\colon \Sigma^* \to T = (X, M)$ is a morphism, then there exists a morphism $\psi\colon \Sigma^* \to T' = (Y, N)$ which simulates φ: it suffices to choose an arbitrary element $\psi(a)$ in $\pi^{-1}(\varphi(a))$ for every $a \in \Sigma$.

For sets U and V, we denote the set of all functions from U to V by $\mathcal{F}(U, V)$.

Definition 3.4 (Wreath Product). Let $T_1 = (X, M)$ and $T_2 = (Y, N)$ be two tm's. We define the *wreath product* of T_1 and T_2 to be the tm $T = T_1 \wr T_2 = (X \times Y, M \times \mathcal{F}(X, N))$ where, for $m \in M$ and $f \in \mathcal{F}(X, N)$, (m, f) represents the following transformation on $X \times Y$:

$$\text{for } (x, y) \in X \times Y, \quad (m, f)((x, y)) = (m(x), f(x)(y)).$$

One verifies that the product of $(m_1, f_1), (m_2, f_2) \in M \times \mathcal{F}(X, N)$ is $(m_1, f_1)(m_2, f_2) = (m_1 m_2, f)$, where, for each $x \in X$, $f(x) = f_1(x) \odot f_2(m_1(x))$ (here, $\odot$ denotes the operation in N).

It is well known [Eil76] that the wreath product operation is associative on transformation monoids. The celebrated Krohn-Rhodes theorem [KR65] (see [Str94, DKS12] for different proofs), in its division and its simulation formulations, is as follows.

Theorem 3.5 (Krohn-Rhodes Theorem). *Let $\varphi\colon \Sigma^* \to T = (X, M)$ be a morphism into a finite tm. Then φ is simulated by a morphism $\psi\colon \Sigma^* \to T'$, where the tm T' is the wreath product of finitely many transformation monoids which are either copies of U_2 or of the form (G, G) for some non-trivial simple group G dividing M.*

In particular, every finite transformation monoid (X, M) divides a wreath product of the form above.

Directly interpreting these results in terms of morphisms from trace monoids to ordinary tm's leads to major technical difficulties (see [Sar22, AGSW21]). For instance, division of transformation monoids does not imply simulation of morphisms from trace monoid to the tm's due to the trace monoid not being a free monoid. Also the crucial wreath product principle, that describes word languages recognized by a wreath product of two transformation monoids in terms of word languages recognized by the individual tm's, breaks down when working with trace languages and morphisms from trace monoid; this is primarily due to the fact that the principle uses a sequential transducer that takes into account the runs of an automaton on words. But in a diamond automaton different linearizations of a trace may produce different runs, albeit ending in the same final state (see Remark 2.5).

3.2. **Asynchronous transformation monoids.** We now introduce a new algebraic framework to discuss recognizability for trace languages, which is more consistent with the distributed nature of the alphabet (Σ, loc) and with the notion of asynchronous automata. This point of view resolves the issues mentioned in the last subsection.

Asynchronous transformation monoids are defined as follows.

Definition 3.6. An *asynchronous transformation monoid* (in short, atm) T (over $\mathscr{P}$) is a pair $(\{S_i\}, M)$ where
- $\{S_i\}$ is a $\mathscr{P}$-indexed family of finite non-empty sets.
- M is a submonoid of $\mathcal{F}(S)$.

Note that an atm $T = (\{S_i\}, M)$ naturally induces the tm (S, M), and that one can view T as the tm (S, M), equipped with an additional structure which depends on $\mathscr{P}$. We abuse notation and write T also for this tm.

More precisely, a crucial feature of the definition of an atm is that it makes a clear distinction between local and global states. While the underlying transformations operate on global states, we will be interested in global transformations that are essentially "induced" by a particular subset P of processes, that is, P-maps in the sense of Section 2.2.

It is worth pointing out at this stage, that a transformation $g \in \mathcal{F}(S)$ such that $g(s)$ is of the form (s'_P, s_{-P}) for every $s \in S$, is *not* necessarily a P-map. This condition merely says that the $(-P)$-component of a global state is not updated by g. The update of the P-component might still depend on the $(-P)$-component.

The following lemma provides a characterization of P-maps. We skip the easy proof.

Lemma 3.7. *Let $h \colon S \to S$. Then h is a P-map if and only if for every s in S, $[h(s)]_{-P} = s_{-P}$ and for every s, s' in S, $s_P = s'_P$ implies that $[h(s)]_P = [h(s')]_P$.*

Example 3.8. Fix a process $p \in \mathscr{P}$. We define the atm $U_2[p] = (\{S_i\}, M)$ by letting $S_p = \{1, 2\}$ and, for each $i \neq p$, S_i is a singleton set. Observe that S has only two global states which are completely determined by their p-components. We therefore identify a global state with its p-component. Then we let $M = \{\mathrm{id}_S, r_1, r_2\}$, where r_i maps every global state to the global state i. Note that r_1 and r_2 are $\{p\}$-maps.

Similarly, if $T = (X, M)$ is a tm, we let $T[p]$ be the atm $T[p] = (\{S_i\}, M)$ where $S_p = X$ and all other S_i are singletons. The extensions of the elements of M (transformations of $X = S_p$) are p-maps and form a monoid in natural bijection with M, which we again write M.

A simple but crucial observation regarding P-maps is recorded in the following lemma.

Lemma 3.9. *Let $f, g \in \mathcal{F}(S)$ and let $P, P' \subseteq \mathscr{P}$. If f is a P-map, g is a P'-map and $P \cap P' = \emptyset$, then $fg = gf$.*

Proof. Suppose that f and g are the extensions of some $f' \in \mathcal{F}(S_P)$ and $g' \in \mathcal{F}(S_{P'})$, respectively. Let $Q = \mathscr{P} \setminus (P \cup P')$. We can denote, unambiguously, a global state $s \in S$ as $s = (s_P, s_{P'}, s_Q)$. We then have

$$fg\,((s_P, s_{P'}, s_Q)) = g\,((f'(s_P), s_{P'}, s_Q)) = (f'(s_P), g'(s_{P'}), s_Q)$$
$$gf\,((s_P, s_{P'}, s_Q)) = f\,((s_P, g'(s_{P'}), s_Q)) = (f'(s_P), g'(s_{P'}), s_Q).$$

This shows that f and g commute. $\qquad\qquad\square$

3.3. Asynchronous morphisms.

We now introduce particular morphisms from the trace monoid $\mathrm{Tr}(\Sigma)$ to asynchronous transformation monoids.

Definition 3.10. Let $T = (\{S_i\}, M)$ be an atm. An *asynchronous morphism* from $\mathrm{Tr}(\Sigma)$ to T is a (monoid) morphism $\varphi \colon \mathrm{Tr}(\Sigma) \to M$ such that $\varphi(a)$ is an a-map for each $a \in \Sigma$.

It is important to observe that not every morphism $\varphi \colon \mathrm{Tr}(\Sigma) \to M$ defines an asynchronous morphism: indeed $\varphi(a)$ and $\varphi(b)$ may commute (say, if $(a, b) \in I$) even if $\varphi(a)$ (resp. $\varphi(b)$) is not an a-map (resp. a b-map).

An elementary yet fundamental result about asynchronous morphisms is stated in Lemma 3.11 below.

Lemma 3.11. *Let $T = (\{S_i\}, M)$ be an atm. Further, let $\varphi \colon \Sigma \to M$ be such that $\varphi(a)$ is an a-map for every $a \in \Sigma$. Then φ can be uniquely extended to an asynchronous morphism from $\mathrm{Tr}(\Sigma)$ to T.*

Proof. The map φ uniquely extends to a morphism from the free monoid Σ^* to M. By Proposition 2.1, $\mathrm{Tr}(\Sigma)$ is the quotient of Σ^* by the relations of the form $ab = ba$ where $(a, b) \in I$, so we only need to show that $\varphi(a)$ and $\varphi(b)$ commute. Indeed, $(a, b) \in I$ means that $\mathrm{loc}(a) \cap \mathrm{loc}(b) = \emptyset$. As $\varphi(a)$ and $\varphi(b)$ are an a-map and a b-map, respectively, the result follows from Lemma 3.9. $\qquad\qquad\square$

Example 3.12. Let (Σ, loc) over $\mathscr{P} = \{p_1, p_2, p_3\}$ be given by $\Sigma = \{a, b, c\}$ and $\mathrm{loc}(a) = \{p_1\}$, $\mathrm{loc}(b) = \{p_1, p_2\}$, $\mathrm{loc}(c) = \{p_2, p_3\}$. Letting $\varphi(a) = r_1$, $\varphi(b) = r_2$ and $\varphi(c) = \mathrm{id}$, determines an asynchronous morphism from $\mathrm{Tr}(\Sigma)$ to $U_2[p_1]$.

If instead we let $\varphi(c) = r_1$, φ determines a morphism from $\mathrm{Tr}(\Sigma)$ to $U_2[p_1]$, which is not asynchronous.

A very important example of an asynchronous morphism is given by the transition morphism of an asynchronous automaton. Let $A = (\{S_i\}, \{\delta_a\}, s_{\mathrm{in}})$ be an asynchronous automaton over (Σ, loc). For each $a \in \Sigma$, let $\Delta_a \in \mathcal{F}(S)$ be the extension of the local transition function δ_a, an a-map by definition. Let also M_A be the submonoid of $\mathcal{F}(S)$ generated by the Δ_a ($a \in \Sigma$). By Lemma 3.11 the map $a \mapsto \Delta_a$ extends to an asynchronous morphism φ_A, from $\mathrm{Tr}(\Sigma)$ to the atm $T_A = (\{S_i\}, M_A)$. We say that φ_A is the *transition* (asynchronous) *morphism* of A and $T_A = (\{S_i\}, M_A)$ is the *transition atm* of A.

We record the following lemma, whose proof is immediate.

Lemma 3.13. *Given an asynchronous automaton $A = (\{S_i\}, \{\delta_a\}, s_{\text{in}})$ over (Σ, loc), its transition atm T_A and its transition asynchronous morphism $\varphi_A \colon \text{Tr}(\Sigma) \to T_A$ are effectively constructible.*

If $t \in \text{Tr}(\Sigma)$, then $\varphi_A(t)(s_{\text{in}}) = A(t)$.

A trace language is accepted by A if and only if it is recognized by T_A via φ_A, with s_{in} as the initial state.

We also record the converse construction. Let $T = (\{S_i\}, M)$ be an atm, let $s_{\text{in}} \in S$ be a global state and let $\varphi \colon \text{Tr}(\Sigma) \to T$ be an asynchronous morphism. For each $a \in \Sigma$, $\varphi(a)$ is an a-map: let δ_a be the (uniquely determined) transformation of S_a of which $\varphi(a)$ is an extension. Finally let $A_\varphi = (\{S_i\}, \{\delta_a\}, s_{\text{in}})$. Then A_φ is an asynchronous automaton (over (Σ, loc)) and the following lemma is easily verified.

Lemma 3.14. *Given an atm $T = (\{S_i\}, M)$, a global state $s_{\text{in}} \in S$ and an asynchronous morphism $\varphi \colon \text{Tr}(\Sigma) \to T$, the asynchronous automaton A_φ is effectively constructible. Moreover, φ is the asynchronous transition morphism of A_φ.*

A trace language $L \subseteq \text{Tr}(\Sigma)$ is recognized by T via φ (with initial state s_{in}) if and only if it is accepted by A_φ.

Thus Zielonka's theorem can be rephrased to state that a trace language is recognizable if and only if it is recognized by an asynchronous morphism to an atm. We will see a more precise rephrasing in Section 3.7 below (Theorem 3.33).

3.4. Asynchronous wreath product.
We adapt the definition of wreath product (see Section 3.1) to the setting of asynchronous transformation monoids.

Definition 3.15. Let $T_1 = (\{S_i\}, M)$ and $T_2 = (\{Q_i\}, N)$ be two asynchronous transformation monoids. Their *asynchronous wreath product*, written $T_1 \wr_{\text{as}} T_2$, is defined to be the atm $(\{S_i \times Q_i\}, M \times \mathcal{F}(S, N))$. An element $(m, f) \in M \times \mathcal{F}(S, N)$ represents the following global[3] transformation on $S \times Q$:

$$\text{for } (s, q) \in S \times Q, \quad (m, f)((s, q)) = (m(s), f(s)(q)).$$

An important observation is that the tm associated with the atm $T_1 \wr_{\text{as}} T_2$ is the wreath product of the transformation monoids (S, M) and (Q, N) associated with T_1 and T_2 respectively. In particular, the composition law on $M \times \mathcal{F}(S, N)$ is the same as in Definition 3.4. The associativity of the asynchronous wreath product operation follows immediately.

Lemma 3.16. *Let $T_1 = (\{S_i\}, M)$ and $T_2 = (\{Q_i\}, N)$ be asynchronous transformation monoids and let $P \subseteq \mathscr{P}$. If $(m, f) \in M \times \mathcal{F}(S, N)$ is a P-map in $T_1 \wr_{as} T_2$, then*

- *m is a P-map in T_1.*
- *For every $s \in S$, $f(s)$ is a P-map in T_2. Further, if $s, s' \in S$ are such that $s_P = s'_P$, then $f(s) = f(s')$.*

Proof. Recall that by Lemma 3.7, for a $\mathscr{P}$-indexed family $\{X_i\}$, a transformation $h \in \mathcal{F}(X)$ is a P-map if and only if (a) $(h(x))_{-P} = x_{-P}$ for every $x \in X$ and (b) $x_P = x'_P$ implies $(h(x))_P = (h(x'))_P$ for all $x, x' \in X$.

Let $s \in S$ and $q \in Q$. Since (m, f) is a P-map, we have $[(m, f)((s, q))]_{-P} = (s_{-P}, q_{-P})$, that is, $(m(s))_{-P} = s_{-P}$ and $(f(s)(q))_{-P} = q_{-P}$.

---

[3]a global state (resp. P-state) of $T_1 \wr T_2$ is canonically identified with an element of $S \times Q$ (resp. $S_P \times Q_P$)

In addition, if $s' \in S$ and $q' \in Q$ are such that $s'_P = s_P$ and $q'_P = q_P$, we have $[(m, f)((s, q))]_P = [(m, f)((s', q'))]_P$, that is, $m(s)_P = m(s')_P$ and $(f(s)(q))_P = (f(s')(q'))_P$. This already establishes that m is a P-map in T_1.

Moreover, if we choose $s = s'$, we conclude that $f(s)$ is a P-map in T_2. If instead we choose $q = q'$, we get

$$f(s)(q) = ((f(s)(q))_P, q_{-P}) = ((f(s')(q))_P, q_{-P}) = f(s')(q),$$

that is, $f(s) = f(s')$, which concludes the proof.   □

3.5. **Asynchronous wreath product principle.** The classical wreath product principle [Str94] is a critical result that defines the importance and utility of wreath product structures in formal language theory. We give here analogous results which exploit the distributed structure of asynchronous automata and asynchronous transformation monoids. In fact we recover the classical principle as a special case when there is only one process.

Let $T = (\{S_i\}, M)$ be an atm and $\varphi \colon \mathrm{Tr}(\Sigma) \to T$ be an asynchronous morphism. We associate with T the alphabet $\Sigma \times_\ell S = \{(a, s_a) \mid a \in \Sigma, s \in S\}$ where each letter a is decorated with local a-state information of T. Recall that this alphabet can be viewed as a distributed alphabet by letting $(\Sigma \times_\ell S)_i$ $(i \in \mathscr{P})$ be the set of letters $(a, s_a) \in \Sigma \times_\ell S$ such that $a \in \Sigma_i$. In other words, $\mathrm{loc}((a, s_a)) = \mathrm{loc}(a)$. The choice of an initial global state $s_{\mathrm{in}} \in S$ induces the following transducer over traces.

Definition 3.17. The *asynchronous transducer* associated with φ and s_{in} is the map $\chi^{s_{\mathrm{in}}}_\varphi \colon \mathrm{Tr}(\Sigma) \to \mathrm{Tr}(\Sigma \times_\ell S)$ defined as follows. If $t = (E, \leq, \lambda) \in \mathrm{Tr}(\Sigma)$, then $\chi^{s_{\mathrm{in}}}_\varphi(t)$ is the trace $(E, \leq, (\lambda, \mu)) \in \mathrm{Tr}(\Sigma \times_\ell S)$ where the labelling function μ is defined as follows. For each event e of t, if $\lambda(e) = a$ and $s = \varphi(\Downarrow e)(s_{\mathrm{in}})$, then $\mu(e) = s_a$.

It is immediately verified that, if $A = (\{S_i\}, \{\delta_a\}, s_{\mathrm{in}})$ is an asynchronous automaton and φ is its transition morphism, then $\chi^{s_{\mathrm{in}}}_\varphi$ coincides with χ_A, the asynchronous transducer of A defined in Section 2.3.

Run of A_φ on trace t Trace $\chi(t)$
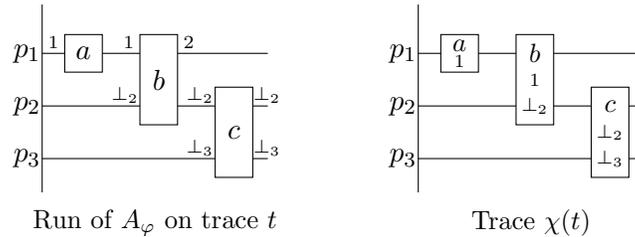
FIGURE 4. Asynchronous transducer output on a trace.

Example 3.18. Let φ be the first asynchronous morphism in Example 3.12, with $S_{p_1} = \{1, 2\}$, $S_{p_2} = \{\perp_2\}$ and $S_{p_3} = \{\perp_3\}$. Let $s_{\mathrm{in}} = (1, \perp_2, \perp_3)$ be the global initial state and let χ be the corresponding asynchronous transducer. Figure 4 shows (automata-style) the computation of the asynchronous morphism φ on the trace $t = abc \in \mathrm{Tr}(\Sigma)$ (by showing local process states before and after each event), and the resulting trace $\chi(t) \in \mathrm{Tr}(\Sigma \times_\ell S)$.

The following lemma is a straightforward consequence of the definition of the asynchronous transducer.

Lemma 3.19. *Let $\varphi \colon \mathrm{Tr}(\Sigma) \to T$ be an asynchronous morphism to an atm $T = (\{S_i\}, M)$, let $s_{\mathrm{in}} \in S$ and let χ be the associated asynchronous transducer. Let $t \in \mathrm{Tr}(\Sigma)$, $a \in \Sigma$ and $s = \varphi(t)(s_{\mathrm{in}})$. Then the trace $\chi(ta) \in \mathrm{Tr}(\Sigma \times_\ell S)$ factors as $\chi(ta) = \chi(t)(a, s_a)$.*

We now define a notion of asynchronous wreath product of asynchronous morphisms defined on trace monoids.

Definition 3.20. Let $\varphi \colon \mathrm{Tr}(\Sigma) \to T$ and $\psi \colon \mathrm{Tr}(\Sigma \times_\ell S) \to T'$ be asynchronous morphisms to asynchronous transformation monoids $T = (\{S_i\}, M)$ and $T' = (\{Q_i\}, N)$. For each $a \in \Sigma$ and $s \in S$, let $g_a(s) = \psi(a, s_a)$. The map $\varphi \wr_{\mathrm{as}} \psi$ is defined on Σ by letting $(\varphi \wr_{\mathrm{as}} \psi)(a) = (\varphi(a), g_a)$.

Lemma 3.21. *If φ and ψ are as in the above definition, then $\varphi \wr_{as} \psi$ induces an asynchronous morphism to the atm $T \wr_{as} T'$.*

Proof. For each $a \in \Sigma$, $(\varphi \wr_{\mathrm{as}} \psi)(a)$ is an a-map since $\mathrm{loc}(a) = \mathrm{loc}((a, s_a))$ and φ and ψ are asynchronous. As a result (Lemma 3.11), $\varphi \wr_{\mathrm{as}} \psi$ extends to an asynchronous morphism $\varphi \wr_{\mathrm{as}} \psi \colon \mathrm{Tr}(\Sigma) \to T \wr_{\mathrm{as}} T'$. $\square$

The following technical result establishes an important connection between asynchronous transducers and asynchronous wreath product of asynchronous morphisms. This is crucially utilized in the proof of the asynchronous wreath product principle.

Proposition 3.22. *Let $\varphi \colon \mathrm{Tr}(\Sigma) \to T$ be an asynchronous morphism to an atm $T = (\{S_i\}, M)$, let $s_{\mathrm{in}} \in S$ and let χ be the associated asynchronous transducer. Let $\psi \colon \mathrm{Tr}(\Sigma \times_\ell S) \to T'$ be an asynchronous morphism to an atm $T' = (\{Q_i\}, N)$.*

For each $t \in \mathrm{Tr}(\Sigma)$, let $\pi_1(t)$ and $\pi_2(t)$ be the first and second component projections of $(\varphi \wr_{as} \psi)(t) \in M \times \mathcal{F}(S, N)$. Then $\pi_1(t) = \varphi(t)$ and $\pi_2(t)(s_{\mathrm{in}}) = \psi(\chi(t))$.

Proof. Let $t = (E, \leq, \lambda) \in \mathrm{Tr}(\Sigma)$. The fact that $\pi_1(t) = \varphi(t)$ follows directly from the definition of wreath products. The second equality is verified by induction on the cardinality of E. It is trivial if $|E| = 0$ and we now suppose $t = t'a$, so that

$$(\varphi \wr_{\mathrm{as}} \psi)(t) = (\varphi \wr_{\mathrm{as}} \psi)(t') \cdot (\varphi \wr_{\mathrm{as}} \psi)(a).$$

As in Definition 3.4, to visually distinguish the operations in the different monoids, we write $\odot$ for the operation of N. We have $(\pi_1(t), \pi_2(t)) = (\pi_1(t'), \pi_2(t'))(\pi_1(a), \pi_2(a))$ and hence, for $x \in S$, $\pi_2(t)(x) = \pi_2(t')(x) \odot \pi_2(a)(\pi_1(t')(x))$. This holds in particular for $x = s_{\mathrm{in}}$. Let also $s = \pi_1(t')(s_{\mathrm{in}}) = \varphi(t')(s_{\mathrm{in}})$. Then we have

$$
\begin{aligned}
\pi_2(t)(s_{\mathrm{in}}) &= \pi_2(t')(s_{\mathrm{in}}) \odot \pi_2(a)(\pi_1(t')(s_{\mathrm{in}})) \\
&= \psi(\chi(t')) \odot \pi_2(a)(s) \quad \text{by induction} \\
&= \psi(\chi(t')) \odot \psi((a, s_a)) \\
&= \psi(\chi(t')(a, s_a)) \\
&= \psi(\chi(t)) \quad \text{by Lemma 3.19,}
\end{aligned}
$$

and this concludes the proof. $\square$

We can now state and prove both directions of what we term the *asynchronous wreath product principle*.

Theorem 3.23. *Let $\varphi \colon \mathrm{Tr}(\Sigma) \to T$ be an asynchronous morphism to an atm $T = (\{S_i\}, M)$ and let $s_{\mathrm{in}} \in S$. Let $\chi \colon \mathrm{Tr}(\Sigma) \to \mathrm{Tr}(\Sigma \times_\ell S)$ be the associated asynchronous transducer. If $L \subseteq \mathrm{Tr}(\Sigma \times_\ell S)$ is recognized by an atm T', then $\chi^{-1}(L)$ is recognized by the atm $T \wr_{as} T'$.*

Proof. Let $\psi\colon \mathrm{Tr}(\Sigma \times_\ell S) \to T' = (\{Q_i\}, N)$ be an asynchronous morphism recognizing L, with $q_{\mathrm{in}} \in Q$ as the initial global state, and $Q_{\mathrm{fin}} \subseteq Q$ as the set of final global states. Then a trace $t \in \mathrm{Tr}(\Sigma \times_\ell S)$ is in L if and only if $\psi(t)(q_{\mathrm{in}}) \in Q_{\mathrm{fin}}$.

By Proposition 3.22, for $t \in \mathrm{Tr}(\Sigma)$, we have $(\varphi \wr_{\mathrm{as}} \psi)(t)(s_{\mathrm{in}}, q_{\mathrm{in}}) = (\varphi(t)(s_{\mathrm{in}}), \psi(\chi(t))(q_{\mathrm{in}}))$. Therefore $t \in \chi^{-1}(L)$ if and only if $(\varphi \wr_{\mathrm{as}} \psi)(t)(s_{\mathrm{in}}, q_{\mathrm{in}}) \in S \times Q_{\mathrm{fin}}$, which concludes the proof that $\varphi \wr_{\mathrm{as}} \psi$ recognizes $\chi^{-1}(L)$. $\qquad\square$

Theorem 3.24. *Let $T_1 = (\{S_i\}, M)$ and $T_2 = (\{Q_i\}, N)$ be atms and let $L \subseteq \mathrm{Tr}(\Sigma)$ be a trace language recognized by an asynchronous morphism $\eta\colon \mathrm{Tr}(\Sigma) \to T_1 \wr_{as} T_2$, with initial global state $(s_{\mathrm{in}}, q_{\mathrm{in}})$. For each $a \in \Sigma$, let $\eta(a) = (m_a, f_a)$. Mapping each $a \in \Sigma$ to m_a defines an asynchronous morphism $\varphi\colon \mathrm{Tr}(\Sigma) \to T_1$. Let χ be the local asynchronous transducer associated to φ and s_{in}. Then L is a finite union of languages of the form $U \cap \chi^{-1}(V)$, where $U \subseteq \mathrm{Tr}(\Sigma)$ is recognized by T_1 and $V \subseteq \mathrm{Tr}(\Sigma \times_\ell S)$ is recognized by T_2.*

Proof. For $a \in \Sigma$, $\eta(a) = (m_a, f_a) \in M \times \mathcal{F}(S, N)$ is an a-map and, by Lemma 3.16, $m_a \in M$ is an a-map (of T_1) and $f_a\colon S \to N$ is such that, for each $s \in S$, $f_a(s) \in N$ is an a-map (of T_2) which depends only on s_a. In particular, $f_a\colon S \to N$ may be viewed as a map $f_a\colon S_a \to N$. Below we will use f_a in this sense.

It follows, by Lemma 3.11, that the map $a \mapsto m_a$ extends to an asynchronous morphism $\varphi\colon \mathrm{Tr}(\Sigma) \to T_1$. Similarly, mapping $(a, s_a) \in \Sigma \times_\ell S$ to $f_a(s_a)$ defines an asynchronous morphism $\psi\colon \mathrm{Tr}(\Sigma \times_\ell S) \to T_2$ and we have $\eta = \varphi \wr_{\mathrm{as}} \psi$.

By definition of recognizability, L is the union of a finite family of languages recognized by η with initial global state $(s_{\mathrm{in}}, q_{\mathrm{in}})$ and a single final global state and we can, without loss of generality, assume that L is recognized with a single final global state, say, $(s_{\mathrm{fin}}, q_{\mathrm{fin}})$.

Let $\pi_1(t)$ and $\pi_2(t)$ be the first and second component projections of $\eta(t)$, for each $t \in \mathrm{Tr}(\Sigma)$. Thus $t \in L$ if and only if $\eta(t)((s_{\mathrm{in}}, q_{\mathrm{in}})) = (s_{\mathrm{fin}}, q_{\mathrm{fin}})$, that is,

$$(\pi_1(t)(s_{\mathrm{in}}), \pi_2(t)(s_{\mathrm{in}})(q_{\mathrm{in}})) = (s_{\mathrm{fin}}, q_{\mathrm{fin}}).$$

Proposition 3.22, applied to $\eta = \varphi \wr_{\mathrm{as}} \psi$, shows that this is equivalent to $\varphi(t)(s_{\mathrm{in}}) = s_{\mathrm{fin}}$ and $\psi(\chi(t))(q_{\mathrm{in}}) = q_{\mathrm{fin}}$.

Let now $U \subseteq \mathrm{Tr}(\Sigma)$ be recognized by the asynchronous morphism φ with initial and final states s_{in} and s_{fin}, and let $V \subseteq \mathrm{Tr}(\Sigma \times_\ell S)$ be recognized by ψ with initial and final states q_{in} and q_{fin}. Then $t \in L$ if and only if $t \in U$ and $\chi(t) \in V$, that is, $L = U \cap \chi^{-1}(V)$, which completes the proof. $\qquad\square$

Remark 3.25. Note that the asynchronous wreath product principle, when restricted to a single process, corresponds exactly to the sequential wreath product principle.

Example 3.26. Consider the distributed alphabet $\widetilde{\Sigma} = (\{a, b\}, \{b, c\}, \{c\})$ over $\mathscr{P} = \{p_1, p_2, p_3\}$ from Example 3.12. We define an asynchronous morphism η from $\mathrm{Tr}(\Sigma)$ into the asynchronous wreath product $(\{S_i\}, M) \wr_{\mathrm{as}} (\{Q_i\}, N)$ where $(\{S_i\}, M) = U_2[p_1]$ and $(\{Q_i\}, N) = U_2[p_3]$. Denoting the (isomorphic) monoids of $U_2[p_1]$ and $U_2[p_3]$ by U_2, by Definition 3.15, we know that $\eta(a) = (m_a, f_a) \in U_2 \times \mathcal{F}(S, U_2)$. Further, by Lemma 3.16, f_a can be described as a function in $\mathcal{F}(S_a, U_2)$. Let the local states of the first tm be $S_{p_1} = \{1, 2\}, S_{p_2} = \{\perp_2\}, S_{p_3} = \{\perp_3\}$, and those of the second tm be $Q_{p_1} = \{\perp_1'\}, Q_{p_2} = \{\perp_2'\}, Q_{p_3} = \{1', 2'\}$. It is clear from the description of η below that $\eta(a)$ (resp. $\eta(b)$ and $\eta(c)$)

is an a-map (resp. b-map and c-map). Therefore η extends to an asynchronous morphism.

$$\eta(a) = (r_2, \{1 \mapsto \mathrm{id}, 2 \mapsto \mathrm{id}\})$$
$$\eta(b) = (\mathrm{id}, \{1\bot_2 \mapsto \mathrm{id}, 2\bot_2 \mapsto \mathrm{id}\})$$
$$\eta(c) = (\mathrm{id}, \{\bot_2\bot_3 \mapsto r_{2'}\})$$

The naturally derived asynchronous morphisms $\varphi\colon \mathrm{Tr}(\Sigma) \to U_2[p_1]$ and $\psi\colon \mathrm{Tr}(\Sigma \times_\ell S) \to U_2[p_3]$ to the individual atm's, as in the proof of Theorem 3.24, are $\varphi = \{a \mapsto r_2, b \mapsto \mathrm{id}\}$, and $\psi = \{c\bot_2\bot_3 \mapsto r_{2'}\}$; since $U_2[p_1]$ is localized, we only need to describe φ on Σ_{p_1} letters (the other letters being mapped to id), similarly for ψ. Note that since there is no letter on which processes p_1 and p_3 synchronize, and the information passed by $U_2[p_1]$ via re-labelling of events is 'local', it cannot be utilised by $U_2[p_3]$, and hence the asynchronous morphism η to the asynchronous wreath product structure $U_2[p_1] \wr_{\mathrm{as}} U_2[p_3]$ essentially reduces to an asynchronous morphism into the direct product $U_2[p_1] \times U_2[p_3]$.

3.6. **Local cascade product.** Now we present an automata-theoretic view of the asynchronous wreath product.

Definition 3.27. Let $A_1 = (\{S_i\}, \{\delta_a\}, s_{\mathrm{in}})$ and $A_2 = (\{Q_i\}, \{\delta_{(a,s_a)}\}, q_{\mathrm{in}})$ be asynchronous automata over the distributed alphabets (Σ, loc) and $(\Sigma \times_\ell S, \mathrm{loc})$, respectively. The *local cascade product* of A_1 and A_2, written $A_1 \circ_\ell A_2$, is the asynchronous automaton $(\{R_i\}, \{\Delta_a\}, (s_{\mathrm{in}}, q_{\mathrm{in}}))$ over (Σ, loc) where, for $i \in \mathscr{P}$, $R_i = S_i \times Q_i$ and where, for $a \in \Sigma$ and $(s_a, q_a) \in R_a{}^4$, $\Delta_a((s_a, q_a)) = (\delta_a(s_a), \delta_{(a,s_a)}(q_a))$.

This operation on asynchronous automata corresponds exactly to the asynchronous wreath product defined in Section 3.4, in the sense of the following statement.

Proposition 3.28. *Let* $\varphi\colon \mathrm{Tr}(\Sigma) \to (\{S_i\}, M)$ *and* $\psi\colon \mathrm{Tr}(\Sigma \times_\ell S) \to (\{Q_i\}, N)$ *be the asynchronous transition morphisms of* A_1 *and* A_2, *respectively. Then the asynchronous transition morphism of* $A_1 \circ_\ell A_2$ *is* $\varphi \wr_{as} \psi\colon \mathrm{Tr}(\Sigma) \to (\{S_i\}, M) \wr_{as} (\{Q_i\}, N)$.

Proof. Let $\bar\delta_a, \bar\delta_{(a,s_a)}$ and $\bar\Delta_a$ denote the extensions to global states (S, Q and R, respectively) of the maps $\delta_a \in \mathcal{F}(S_a)$, $\delta_{(a,s_a)} \in \mathcal{F}(Q_a)$ and $\Delta_a \in \mathcal{F}(R_a)$. By definition of transition morphisms (Section 3.3), for each $a \in \Sigma$ and $s \in S$, we have $\varphi(a) = \bar\delta_a$ and $\psi(a, s_a) = \bar\delta_{(a,s_a)}$, while the transition morphism of $A_1 \circ_\ell A_2$ maps a to $\bar\Delta_a$.

Let $a \in \Sigma$, let $P = \mathrm{loc}(a) = \mathrm{loc}((a, s_a))$, and let $r \in R$, say, $r = (s, q)$. Then $\bar\Delta_a(r) = (\Delta_a(r_P), r_{-P})$. Now $\Delta_a(r_P) = (\delta_a(s_P), \delta_{(a,s_a)}(q_P))$, while $r_{-P} = (s_{-P}, q_{-P})$, so $\bar\Delta_a(r) = (\bar\delta_a(s), \bar\delta_{(a,s_a)}(q))$.

Now compare with Definition 3.20: the map $a \mapsto \bar\Delta_a$ coincides with the restriction of $\varphi \wr_{\mathrm{as}} \psi$ to Σ, which concludes the proof. $\square$

The correspondence between the asynchronous wreath product of asynchronous transformation monoids and the local cascade product of asynchronous automata established in Proposition 3.28, induces an automata-theoretic version of the asynchronous wreath product principle, using the asynchronous transducers of the asynchronous automata involved.

More precisely, a run of the local cascade product $A_1 \circ_\ell A_2$ on a trace t can be understood as follows. One first views A_1 as an asynchronous computing device, namely the asynchronous

---

[4]We identify $R_a = \prod_{i \in \mathrm{loc}(a)} S_i \times Q_i$ with $S_a \times Q_a = \left(\left(\prod_{i \in \mathrm{loc}(a)} S_i\right) \times \left(\prod_{i \in \mathrm{loc}(a)} Q_i\right)\right)$.

transducer $\hat{A}_1$ (see Section 2.3) which computes χ_{A_1}. Its run on t outputs the trace $\chi_{A_1}(t) \in \mathrm{Tr}(\Sigma \times_\ell S)$ (see Figure 4) and one then runs A_2 on that trace, leading to state $q \in Q$. Note that, due to the asynchronous nature of the computing device, as soon as A_1 has finished working on some event (of t), A_2 can start working on the 'same' event (of $\chi_{A_1}(t)$). Further, A_1 and A_2 work asynchronously and can 'simultaneously' process concurrent events. Finally, the run of $A_1 \circ_\ell A_2$ on t takes the initial state $(s_{\mathrm{in}}, q_{\mathrm{in}})$ to the pair (s, q), as in
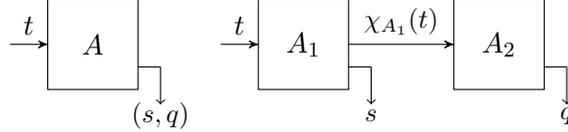

FIGURE 5. Operational view of local cascade product

Figure 5. A different take on this operational point of view will be developed in Section 6.

Finally, note that the local cascade product is associative. The following *local cascade product principle*, which relies on associativity, is the announced rephrasing of the asynchronous wreath product principle in automata-theoretic terms.

Theorem 3.29. *Let $A = A_1 \circ_\ell \ldots \circ_\ell A_n$ and $B = B_1 \circ_\ell \ldots \circ_\ell B_m$ be local cascade products such that $C = A \circ_\ell B$ is defined in the sense that the distributed alphabets of A and B (say, (Σ, loc) and (Π, loc) respectively) match as in Definition 3.27. Further, let $\chi_A : \mathrm{Tr}(\Sigma) \to \mathrm{Tr}(\Pi)$ be the asynchronous transducer of A. Then any language $L \subseteq \mathrm{Tr}(\Sigma)$ accepted by C is a finite union of languages of the form $U \cap \chi_A^{-1}(V)$ where $U \subseteq \mathrm{Tr}(\Sigma)$ is accepted by A, and $V \subseteq \mathrm{Tr}(\Pi)$ is accepted by B.*

Example 3.30. Consider a distributed alphabet $\widetilde{\Sigma} = (\{a, b\}, \{b, c\}, \{c, d\})$ over processes $\mathscr{P} = \{1, 2, 3\}$. We define a local cascade product $A \circ_\ell B$ where $A = A_1 \circ_\ell A_2 \circ_\ell A_3$ itself is also a local cascade product. A_1, A_2, A_3 and B are all localized asynchronous automata, having two local states in processes $1, 2, 3$, and 3 respectively, and a single local state in each of the remaining processes. The automata are described in Figure 6; note that due to its localized structure, A_1 is completely described by transitions induced by the letters of Σ_1 on the local states of process 1. A similar statement holds for A_2, A_3, and B; also for simplicity, in the extended alphabet letters for A_2, A_3 and B, we have only displayed non-trivial state information.

It is not difficult to see that the asynchronous transducer $\hat{A}$ labels process 1 events (resp. process 2 events, and process 3 events) by p_2 (resp. q_2 and s_2) if and only if that event has an a-labelled event in its past. Since B detects the existence of letter (d, s_2) by changing its state, $A \circ_\ell B$ can recognize for instance the language "there exists a d that has an a in its past".

To conclude this section, we relate local cascade products with the Γ-labelling functions introduced in Section 2.3.

Proposition 3.31. *Let $A = (\{S_i\}, \{\delta_a\}, s_{\mathrm{in}})$ be an asynchronous automaton on the distributed alphabet (Σ, loc), and let $\theta \colon \mathrm{Tr}(\Sigma) \to \mathrm{Tr}(\Sigma \times \Gamma)$ be the Γ-labelling function computed by an asynchronous Γ-transducer of the form $\hat{A} = (\{S_i\}, \{\delta_a\}, s_{\mathrm{in}}, \{\mu_a\})$. Let B be an asynchronous automaton on the distributed alphabet $(\Sigma \times \Gamma, \mathrm{loc})$ accepting a language $L \subseteq \mathrm{Tr}(\Sigma \times \Gamma)$. Then $\theta^{-1}(L)$ is accepted by $A \circ_\ell B$.*
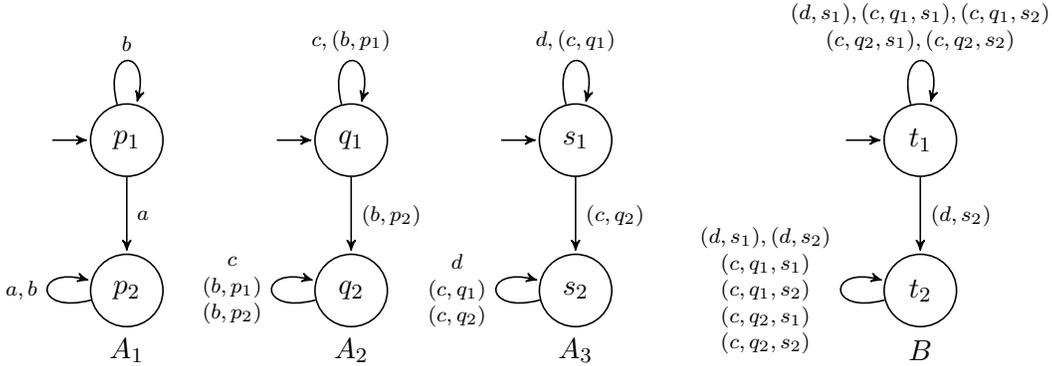
Figure 6. Cascade product of localized reset asynchronous automata

Proof. Let $\gamma\colon \Sigma \times_\ell S \to \Sigma \times \Gamma$ be the map given by $\gamma(a, s_a) = (a, \mu_a(s_a))$. We also denote by γ the extension of this map to a morphism from $\mathrm{Tr}(\Sigma \times_\ell S)$ to $\mathrm{Tr}(\Sigma \times \Gamma)$. By definition, $\theta = \gamma \circ \chi_A$. In particular $\theta^{-1}(L) = \chi_A^{-1}(\gamma^{-1}(L))$.

Let B' be the automaton on alphabet $\Sigma \times_\ell S$ obtained from B by keeping the same state sets and global initial and accepting states, and letting the transition $\delta_{(a,s_a)}$ be equal to the transition $\delta_{(a,\mu_a(s_a))}$ of B. It is clear that B' accepts $\gamma^{-1}(L)$ and, by Theorem 3.23, $A \circ_\ell B$ accepts $\theta^{-1}(L) = \chi_A^{-1}(\gamma^{-1}(L))$. $\qquad\square$

3.7. **Questions of decomposition.** Most of the known characterizations of interesting classes of recognizable trace languages (*e.g.*, star-free languages, languages definable in first-order logic or in a global or local temporal logic, see [GRS92, EM96, DG02, DG06]) are in terms of morphisms into ordinary transformation monoids or of syntactic monoids (equivalently, of the transition monoids of certain canonical minimal diamond-automata). In contrast, Zielonka's theorem simulates a morphism into a tm by an asynchronous morphism into an atm. We seek an asynchronous version of the Krohn-Rhodes theorem that would be similar in spirit: starting with a morphism into a tm, we ask whether it can be simulated by an asynchronous morphism into an asynchronous wreath product of 'simpler' asynchronous transformation monoids. A positive resolution in this form would help us lift the sequential or diamond automata-theoretic chacterizations of first-order definable trace languages mentioned above to asynchronous automata-theoretic characterizations (cf. Theorem 5.14 below).

We would also like the statement of the proposed asynchronous version of the Krohn-Rhodes theorem to coincide with the classical sequential Krohn-Rhodes Theorem (Theorem 3.5) when the underlying distributed alphabet involves only one process.

We first extend the notion of simulation (Definition 3.1) to trace morphisms as follows.

Definition 3.32. Let $\varphi\colon \mathrm{Tr}(\Sigma) \to T = (X, M)$ and $\psi\colon \mathrm{Tr}(\Sigma) \to T' = (Y, N)$ be morphisms to transformation monoids. We say that ψ *simulates* φ if there exists a surjective function $f\colon Y \to X$ such that, for all $a \in \Sigma$ and all $y \in Y$, $f(\psi(a)(y)) = \varphi(a)(f(y))$.

If ψ happens to be an asynchronous morphism to an atm, we say that ψ simulates φ if it is the case when ψ is viewed as a morphism to the tm underlying T'.

We note that Zielonka's fundamental theorem [Zie87], already mentioned in Sections 2.2 and 3.3, can actually be rephrased as follows (see [Muk12]).

Theorem 3.33 (Zielonka Theorem). *Every morphism $\varphi\colon \mathrm{Tr}(\Sigma) \to T$ to a finite tm is simulated by an asynchronous morphism $\psi\colon \mathrm{Tr}(\Sigma) \to T'$ to an atm.*[5]

Recall the atm $U_2[p]$ defined in Example 3.8, an asynchronous analogue at process $p \in \mathscr{P}$ of the tm U_2. If G is a group, recall that (G, G) is a tm and let $G[p]$ be its asynchronous analogue at process p (again, see Example 3.8).

Definition 3.34. We say that a distributed alphabet (Σ, loc) has the *asynchronous Krohn-Rhodes property* (($\Sigma, \mathrm{loc})$ is *aKR*, for short) if every morphism $\varphi\colon \mathrm{Tr}(\Sigma) \to T$ to a tm $T = (X, M)$ is simulated by an asynchronous morphism to an atm T' which is the asynchronous wreath product of asynchronous transformation monoids of the form $U_2[p]$ or $G[p]$, where $p \in \mathscr{P}$ and G is a simple group dividing M.

It is an interesting question to characterize which distributed alphabets are aKR, or even whether all are. While we are not able to answer this question, we show in Section 4 that acyclic architectures are aKR. In Section 5, we show that a weaker property holds when we restrict our attention to morphisms from $\mathrm{Tr}(\Sigma)$ to aperiodic transformation monoids. See also the discussion in Section 8.

In view of our discussion so far, it is clear that establishing that a distributed alphabet is aKR amounts to a simultaneous generalization, for this particular distributed alphabet, of the Krohn-Rhodes theorem (Theorem 3.5) and of Zielonka's theorem (Theorem 3.33).

4. The case of acyclic architectures

Definition 4.1. The *communication graph* of a distributed alphabet $\{\Sigma_i\}_{i \in \mathscr{P}}$ is the undirected graph $G = (\mathscr{P}, E)$ where $E = \{(i, j) \in \mathscr{P} \times \mathscr{P} \mid i \neq j \text{ and } \Sigma_i \cap \Sigma_j \neq \emptyset\}$. An *acyclic architecture* is a distributed alphabet whose communication graph is acyclic.

Observe that if (Σ, loc) is an acyclic architecture, then no action is shared by more than two processes. We note that Zielonka's theorem admits a simpler proof in this case [KM13]. Our objective in this section is to establish the following result.

Theorem 4.2. *If (Σ, loc) is an acyclic architecture, then (Σ, loc) has the asynchronous Krohn-Rhodes property.*

To prove Theorem 4.2, we need several technical lemmas. We first define a notion of wreath product of trace morphisms into tm's.

Definition 4.3. Let $\varphi\colon \mathrm{Tr}(\Sigma) \to (X, M)$ and $\psi\colon \mathrm{Tr}(\Sigma \times X) \to (Y, N)$ be morphisms to transformation monoids. For each $a \in \Sigma$ and $x \in X$, let $f_a(x) = \psi(a, x)$. The map $\varphi \wr \psi$ is defined on Σ by letting $(\varphi \wr \psi)(a) = (\varphi(a), f_a)$ for each $a \in \Sigma$.

In general, the above map $\varphi \wr \psi$ may not extend to a morphism from the trace monoid (see [Sar22, AGSW21] for an example). However, it does so under a technical condition.

Lemma 4.4. *Let φ and ψ be as in Definition 4.3. If, for all independent letters a, b and for all $x \in X$, we have $\psi(b, \varphi(a)(x)) = \psi(b, x)$, then $\varphi \wr \psi$ induces a morphism from $\mathrm{Tr}(\Sigma)$ to the tm $(X, M) \wr (Y, N)$.*

---

[5]The proof in [Muk12] provides an automata-theoretic simulation of any diamond-automaton by an asynchronous automaton. The morphism version stated here is an easy consequence of this result and the correspondence between morphisms (resp. asynchronous morphisms) defined on trace monoids and diamond-automata (resp. asynchronous automata).

Proof. We verify that, if a and b are independent letters, then $(\varphi \wr \psi)(ab) = (\varphi \wr \psi)(ba)$. More precisely, let $(\varphi \wr \psi)(ab) = (\eta_1, \eta_2)$. By definition, we have $\eta_1 = \varphi(a)\varphi(b)$ and for each $x \in X$, we have $\eta_2(x) = \psi(a, x) + \psi(b, \varphi(a)(x))$. Using our hypothesis, we get $\eta_2(x) = \psi(a, x) + \psi(b, x)$. Since φ and ψ are trace morphisms, we deduce that $(\varphi \wr \psi)(ab) = (\varphi \wr \psi)(ba)$. $\qquad\square$

We now show how to simulate a morphism from a trace monoid to a tm, by the wreath product of morphisms to transformation monoids that are simpler.

Lemma 4.5. *For a process $p \in \mathscr{P}$, let us define the set $\Sigma_0 = \{a \in \Sigma \mid \mathrm{loc}(a) = \{p\}\}$. Let $\varphi \colon \mathrm{Tr}(\Sigma) \to (X, M)$ be a morphism to a tm. There exist morphisms $\varphi_1 \colon \mathrm{Tr}(\Sigma) \to (X_1, M_1)$ and $\varphi_2 \colon \mathrm{Tr}(\Sigma \times X_1) \to (X, M)$ such that*

- *φ_1 ignores all non-process-p letters (that is, φ_1 maps letters outside Σ_p to the identity transformation of X_1). Moreover, any group dividing M_1 also divides M.*
- *φ_2 ignores all letters local only to process p (that is, φ_2 maps letters in $\Sigma_0 \times X_1$ to the identity transformation of X). Also, for any $a \notin \Sigma_p$, $\varphi_2(a, x) = \varphi(a)$.*
- *The morphism $\varphi_1 \wr \varphi_2 \colon \mathrm{Tr}(\Sigma) \to (X_1, M_1) \wr (X, M)$ simulates φ.*

Proof. Observe that $\Sigma_0 \subseteq \Sigma_p$. The letters in Σ_0 are mutually dependent, that is, Σ_0^* is in fact a submonoid of $\mathrm{Tr}(\Sigma)$. Let N be the submonoid of M generated by $\varphi(\Sigma_0)$; we can view N as $N = \varphi(\Sigma_0^*)$. For each $n \in N$, let $\bar{n}$ be the constant transformation of N which maps every element to n. The set $\bar{N} = N \cup \{\bar{n} \mid n \in N\}$ is easily verified to be a submonoid of $\mathcal{F}(N)$.

Let $\varphi_1 \colon \Sigma \to (N, \bar{N})$ be defined by letting

$$\varphi_1(a) = \begin{cases} \varphi(a) & \text{if } a \in \Sigma_0, \\ \overline{\mathrm{id}} & \text{if } a \in \Sigma_p \setminus \Sigma_0, \\ \mathrm{id} & \text{if } a \notin \Sigma_p. \end{cases}$$

If a and b are independent letters, then they cannot be both in Σ_p, one of $\varphi_1(a)$ and $\varphi_1(b)$ at least is the identity, and hence $\varphi_1(a)$ and $\varphi_1(b)$ commute. It follows that φ_1 naturally extends to a morphism $\varphi_1 \colon \mathrm{Tr}(\Sigma) \to (N, \bar{N})$, which is the identity on the submonoid generated by $\Sigma \setminus \Sigma_p$.

We note that if a group G divides $\bar{N}$, then it also divides N. Indeed, suppose that τ is a morphism defined on a submonoid $N' \subseteq \bar{N}$ onto G. Each element of the form $\bar{n}$ ($n \in N$) in N' is idempotent, and hence $\tau(\bar{n})$ is the identity 1_G of G. In particular, the restriction of τ to the submonoid $N'' = N' \cap N$ has the same range as τ, that is, G is a quotient of N'' and hence $G \prec N''$.

Observe that, if $w = a_1 \ldots a_r$, i is maximal such that $a_i \in \Sigma_p \setminus \Sigma_0$ ($i = 0$ if w has no letter in $\Sigma_p \setminus \Sigma_0$) and w' is the projection of $a_{i+1} \cdots a_r$ onto Σ_0^*, then $\varphi_1(w) = \varphi(w')$. In other words, $\varphi_1(w)$ is the evaluation under φ of the word read by process p since its last joint action with a neighbour.

Now let us make $\Sigma \times N$ a distributed alphabet (over $\mathscr{P}$) by letting $\mathrm{loc}((a, n)) = \mathrm{loc}(a)$ for each $(a, n) \in \Sigma \times N$. Let also

$$\varphi_2(a, n) = \begin{cases} \mathrm{id} & \text{if } a \in \Sigma_0, \\ n\varphi(a) & \text{if } a \in \Sigma_p \setminus \Sigma_0, \\ \varphi(a) & \text{if } a \notin \Sigma_p. \end{cases}$$

If (a, n) and (b, n') are independent letters, that is, if a and b are independent in Σ, then $\varphi(a)$ and $\varphi(b)$ commute because φ is defined on $\mathrm{Tr}(\Sigma)$. Moreover, either $a, b \notin \Sigma_p$, or $a \in \Sigma_p$

and $b \notin \Sigma_p$ (or vice versa). In the first case, $\varphi_2(a,n) = \varphi(a)$ and $\varphi_2(b,n') = \varphi(b)$ commute. In the second case, $\varphi(b)$ commutes with n since the latter is (by definition of N) the φ-image of a word in Σ_0^*. It follows that $\varphi_2(a,n)$ and $\varphi_2(b,n')$ commute.

Let $\eta \colon \Sigma \to \bar{N} \times \mathcal{F}(N,M)$ be the map $\eta = \varphi_1 \wr \varphi_2$ in Definition 4.3. Assume that $a, b \in \Sigma$ are independent letters. Let $n \in N$. If $a \notin \Sigma_p$ then $\varphi_1(a)(n) = n$ and if $a \in \Sigma_p$, then $b \notin \Sigma_p$ and $\varphi_2(b, \varphi_1(a)(n)) = \varphi(b) = \varphi_2(b,n)$. In both cases, we get $\varphi_2(b, \varphi_1(a)(n)) = \varphi_2(b,n)$. By Lemma 4.4, it follows that η extends to a morphism $\eta \colon \mathrm{Tr}(\Sigma) \to (N, \bar{N}) \wr (X, M)$.

Finally, for each $(n, x) \in N \times X$, let $f(n,x) = n(x) \in X$ (recall that $N \subseteq M \subseteq \mathcal{F}(X)$). For any $a \in \Sigma$, $n \in N$ and $x \in X$, we note that

$$f(\eta(a)(n,x)) = f(\varphi_1(a)(n), \varphi_2(a,n)(x)).$$

A case analysis proves that η simulates φ via f.

  i) $a \in \Sigma_0$. In this case, $\varphi_2(a,n) = \mathrm{id}$ and the above becomes $f(\varphi(a)(n), x) = f(n\varphi(a), x) = (n\varphi(a))(x) = \varphi(a)(n(x)) = \varphi(a)(f(n,x))$. The first equality is because the transformation by $\varphi(a) \in N$ is the right multiplication on N.
  ii) $a \in \Sigma_p \setminus \Sigma_0$. In this case, $\varphi_1(a)(n) = \mathrm{id}$ and the above becomes $f(\mathrm{id}, (n\varphi(a))(x)) = (n\varphi(a))(x) = \varphi(a)(f(n,x))$.
  iii) $a \notin \Sigma_p$. In this case,

$$f(n, \varphi(a)(x)) = n(\varphi(a)(x)) = (\varphi(a)n)(x) = (n\varphi(a))(x) = \varphi(a)(f(n,x)).$$

  The penultimate equality follows from the fact that the generators of N commute with $\varphi(a)$ since $p \notin \mathrm{loc}(a)$.

In all cases, $f(\eta(a)(n,x)) = \varphi(a)(f(n,x))$. This completes the proof.   □

Remark 4.6. If, in Lemma 4.5, Σ_0 is empty, then M_1 is the trivial monoid (so $\varphi_1(a) = \mathrm{id}$ for every letter a) and $\varphi_2(a,x) = \varphi(a)$.

Our next lemma shows how to combine simulations by asynchronous morphisms, under certain technical assumptions.

Lemma 4.7. *Let* $\varphi_1 \colon \mathrm{Tr}(\Sigma) \to (X,M)$ *and* $\varphi_2 \colon \mathrm{Tr}(\Sigma \times X) \to (Y,N)$ *be morphisms to transformation monoids. Suppose that* φ_1 *is simulated by an asynchronous morphism* $\psi_1 \colon \mathrm{Tr}(\Sigma) \to (\{S_i\}, P)$ *via a map* $f_1 \colon S \to X$. *Suppose also that, for all* $a \in \Sigma$, *if* $s, s' \in S$ *and* $s_a = s'_a$, *then* $\varphi_2(a, f_1(s)) = \varphi_2(a, f_1(s'))$. *Then*

- *The map* $\varphi_1 \wr \varphi_2$ *extends to a morphism from* $\mathrm{Tr}(\Sigma)$ *to* $(X,M) \wr (Y,N)$.
- *The map* φ_2', *defined on* $\Sigma \times_\ell S$ *by letting* $\varphi_2'(a, s_a) = \varphi_2(a, f_1(s))$ *for all* $a \in \Sigma$ *and* $s \in S$, *extends to a morphism from* $\mathrm{Tr}(\Sigma \times_\ell S)$ *to* (Y,N).
- *If* φ_2' *is simulated by an asynchronous morphism* $\psi_2 \colon \mathrm{Tr}(\Sigma \times_\ell S) \to (\{Q_i\}, R)$, *then* $\varphi_1 \wr \varphi_2 \colon \mathrm{Tr}(\Sigma) \to (X,M) \wr (Y,N)$ *is simulated by the asynchronous morphism* $\psi_1 \wr_{as} \psi_2 \colon \mathrm{Tr}(\Sigma) \to (\{S_i\}, P) \wr_{as} (\{Q_i\}, R)$.

Proof. We rely on Lemma 4.4 to prove the first statement, that is, we want to show that, if $a, b \in \Sigma$ are independent letters and $x \in X$, then $\varphi_2(a, \varphi_1(b)(x)) = \varphi_2(a, x)$. Since f_1 is surjective, we can choose $s \in S$ such that $f_1(s) = x$. Then $\varphi_2(a, x) = \varphi_2(a, f_1(s))$. Also, $\varphi_2(a, \varphi_1(b)(x)) = \varphi_2(a, \varphi_1(b)(f_1(s))) = \varphi_2(a, f_1(\psi_1(b)(s)))$. Since a and b are independent and ψ_1 is an asynchronous morphism, $s_a = (\psi_1(b)(s))_a$ and our assumption on f_1 and φ_2 shows that $\varphi_2(a, x) = \varphi_2(a, \varphi_1(b)(x))$. This concludes the proof of the first statement.

The fact that φ_2' extends to a morphism defined on $\mathrm{Tr}(\Sigma \times_\ell S)$ follows directly from the fact that φ_2 is a morphism from the trace monoid $\mathrm{Tr}(\Sigma \times X)$.

Suppose φ_2' is simulated by ψ_2 via the map $f_2\colon Q \to Y$. For each $s \in S$, $q \in Q$, we let $f(s, q) = (f_1(s), f_2(q))$. For any $a \in \Sigma$, we have

$$
\begin{aligned}
f((\psi_1 \wr_{\mathrm{as}} \psi_2)(a)(s, q)) &= f(\psi_1(a)(s), \psi_2(a, s_a)(q)) \\
&= (f_1(\psi_1(a)(s)), f_2(\psi_2(a, s_a)(q))) \\
&= (\varphi_1(a)(f_1(s)), \varphi_2'(a, s_a)(f_2(q))) \\
&= (\varphi_1(a)(f_1(s)), \varphi_2(a, f_1(s))(f_2(q))) \\
&= (\varphi_1 \wr \varphi_2)(a)(f_1(s), f_2(q)) \\
&= (\varphi_1 \wr \varphi_2)(a)(f(s, q))
\end{aligned}
$$

This completes the proof. $\qquad\square$

Proof (of Theorem 4.2). The proof proceeds via induction on the number of processes. The base case, with only one process (and hence no distributed structure on the alphabet), is the Krohn-Rhodes theorem (Theorem 3.5).

Let now $\mathscr{P} = \{1, 2, \ldots, k\}$, with $k \geq 2$, and assume that the theorem holds for acyclic architectures with less than k processes. Since the communication graph is acyclic, there exists a 'leaf' process which communicates with at most one other process. Without loss of generality, let this leaf process be 1, and its only neighbouring process be 2 (if process 1 has no neighbour, then process 2 can be any other process). We let Σ_0 be the set of letters a such that $\mathrm{loc}(a) = \{1\}$. By our assumptions, $\mathrm{loc}(a) = \{1, 2\}$ for every $a \in \Sigma_1 \setminus \Sigma_0$.

Let $\varphi\colon \mathrm{Tr}(\Sigma) \to (X, M)$ be a morphism into a tm. Let $\varphi_1\colon \mathrm{Tr}(\Sigma) \to (X_1, M_1)$ and $\varphi_2\colon \mathrm{Tr}(\Sigma \times X_1) \to (X, M)$ be the morphisms into transformation monoids given by Lemma 4.5 for $p = 1$.

Note that no two letters of Σ_1 are independent, so that the submonoid of $\mathrm{Tr}(\Sigma)$ generated by Σ_1 is freely generated by Σ_1: we write it Σ_1^*. Let $\varphi_1'\colon \Sigma_1^* \to (X_1, M_1)$ be the restriction of φ_1 to Σ_1^*. The Krohn-Rhodes theorem shows that φ_1' is simulated by a morphism $\psi_1'\colon \Sigma_1^* \to T$, where T is a wreath product of transformation monoids of the form U_2 and (G, G), where G is a nontrivial simple group G dividing M_1 — and hence, by Lemma 4.5, dividing M.

Next, we extend ψ_1' to a morphism $\psi_1\colon \mathrm{Tr}(\Sigma) \to T$ by letting $\psi_1(a) = \mathrm{id}$ for each letter $a \notin \Sigma_1$. Consider the atm $T[1]$ defined in Example 3.8. It is easily verified that $\psi_1\colon \mathrm{Tr}(\Sigma) \to T[1]$ induces an asynchronous morphism (Lemma 3.11), which simulates φ_1, and that $T[1]$ is an asynchronous wreath product of asynchronous transformation monoids of the required form (that is, of the form $U_2[1]$ or $G[1]$ for a group G dividing M).

Suppose S is the global state space of $T[1]$ and ψ_1 simulates φ_1 via $f_1\colon S \to X_1$. Let $s, s' \in S$ be global states such that $s_a = s_a'$. If $1 \notin \mathrm{loc}(a)$, then $\varphi_2(a, f_1(s)) = \varphi(a) = \varphi_2(a, f_1(s'))$ by Lemma 4.5. If $1 \in \mathrm{loc}(a)$, then $s_a = s_a'$ implies $s = s'$ by the structure of $T[1]$. Hence $\varphi_2(a, f_1(s)) = \varphi_2(a, f_1(s'))$.

Let $\varphi_2'\colon \mathrm{Tr}(\Sigma \times_\ell S) \to (X, M)$ be the morphism given in Lemma 4.7 (where $\varphi_2'(a, s_a) = \varphi_2(a, f_1(s))$). Consider the distributed alphabet $\widetilde{\Sigma}'$ over $\mathscr{P} \setminus \{1\}$ where $\Sigma_i' = (\Sigma \times_\ell S)_i$ for every $i \in \mathscr{P}$, $i \neq 1$, where the location of a letter (a, s_a) is $\mathrm{loc}(a) \setminus \{1\}$.

Suppose that letters (a, s_a) and (b, s_b') are independent in $\widetilde{\Sigma}'$. We first verify that they are independent in $\widetilde{\Sigma}$ as well. If this is not the case, then $\mathrm{loc}(a) \cap \mathrm{loc}(b) = \{1\}$. However, as we noticed before, letters whose location contains 1 and is not reduced to $\{1\}$ (as is the case for all letters a such that some $(a, s_a) \in \Sigma'$) have location $\{1, 2\}$, and this means that $\mathrm{loc}(a) \cap \mathrm{loc}(b) = \{1, 2\}$, a contradiction.

It follows that $\mathrm{Tr}(\Sigma')$ is a submonoid of $\mathrm{Tr}(\Sigma \times_\ell S)$, and we now consider the restriction φ_2'' of φ_2' to $\mathrm{Tr}(\Sigma')$.

By induction hypothesis, φ_2'' is simulated by an asynchronous morphism $\psi_2 \colon \mathrm{Tr}(\Sigma') \to T'$, where T' is an asynchronous wreath product of asynchronous transformation monoids of the form $U_2[p]$ or $G[p]$ for some simple group G dividing M, and some $p \in \mathscr{P} \setminus \{1\}$. These asynchronous transformation monoids can again be trivially extended over $\mathscr{P}$ (by adding a singleton state set for process 1). The corresponding extension of T', denoted $T'[\uparrow\!\mathscr{P}]$ is an asynchronous wreath product of the desired form. Moreover ψ_2 can also be extended over $\Sigma \times_\ell S$ by mapping any letter in $(\Sigma \times_\ell S) \setminus \Sigma'$ to the identity. It is easily verified that ψ_2 is an asynchronous morphism, which simulates φ_2'.

We can now apply Lemma 4.7 to conclude the proof. $\square$

5. Temporal logics, first-order trace languages & local cascade products

Recall that star-free trace languages, aperiodic trace languages (those recognized by an aperiodic monoid) and first-order definable trace languages coincide, by the combined results of Guaiana, Restivo and Salemi [GRS92] and Ebinger and Muscholl [EM96].

A consequence of Theorem 4.2 is that any *aperiodic* trace language over an acyclic architecture, and indeed over any aKR distributed alphabet, is recognized by an asynchronous wreath product of 2-state asynchronous transformation monoids of the form $U_2[p]$ (see Example 3.8). Proposition 3.28 then shows that it is accepted by a local cascade product of localized two-state reset asynchronous automata. For convenience, we denote these automata by $U_2[p]$ as well: $U_2[p]$ has state set $\{S_i\}$, where $S_p = \{1, 2\}$ and each S_i $(i \neq p)$ is a singleton, and transitions as follows. The alphabet Σ_p contains two disjoint subsets R_1 and R_2 that reset the states in S_p to 1 and 2, respectively. All remaining letters act as the identity, in particular letters in $\Sigma_p \setminus (R_1 \cup R_2)$. In this section, we aim at generalizing this result to any distributed alphabet. Our route towards this utilizes yet another formalism used to classify trace languages, that of temporal logics.

5.1. **Local temporal logics.** We first introduce a process-based past-oriented local temporal logic over traces, called $\mathsf{LocTL}[\mathsf{Y}_i \leq \mathsf{Y}_j, \mathsf{Y}_i, \mathsf{S}_i]$. This logic, as well as some of its fragments, turns out be as expressive as first-order logic over traces, see Theorem 5.5 below. Furthermore, we have chosen the logic in a way that facilitates showing correspondence with local cascade products (see the proofs of Theorem 5.6 and Corollary 5.12). The syntax of $\mathsf{LocTL}[\mathsf{Y}_i \leq \mathsf{Y}_j, \mathsf{Y}_i, \mathsf{S}_i]$ is as follows.

Event formula $\qquad \alpha ::= a \mid \neg\alpha \mid \alpha \vee \alpha \mid \mathsf{Y}_i \leq \mathsf{Y}_j \mid \mathsf{Y}_i\,\alpha \mid \alpha\,\mathsf{S}_i\,\alpha \qquad a \in \Sigma, i, j \in \mathscr{P}$

Trace formula $\qquad \beta ::= \exists_i \alpha \mid \neg\beta \mid \beta \vee \beta$

Turning to the semantics of $\mathsf{LocTL}[\mathsf{Y}_i \leq \mathsf{Y}_j, \mathsf{Y}_i, \mathsf{S}_i]$, each event formula is evaluated at an event of a trace $t \in \mathrm{Tr}(\Sigma)$, say, $t = (E, \leq, \lambda)$. For any event $e \in E$ and process $i \in \mathscr{P}$, we

denote by e_i the unique maximal event of $\Downarrow e \cap E_i$, if it exists, i.e., if $\Downarrow e \cap E_i \neq \emptyset$. Then

$$
\begin{aligned}
&t, e \models a && \text{if } \lambda(e) = a \\
&t, e \models \neg\alpha && \text{if } t, e \not\models \alpha \\
&t, e \models \alpha \vee \alpha' && \text{if } t, e \models \alpha \text{ or } t, e \models \alpha' \\
&t, e \models \mathsf{Y}_i \leq \mathsf{Y}_j && \text{if } e_i, e_j \text{ exist, and } e_i \leq e_j \\
&t, e \models \mathsf{Y}_i \, \alpha && \text{if } e_i \text{ exists, and } t, e_i \models \alpha \\
&t, e \models \alpha \, \mathsf{S}_i \, \alpha' && \text{if } e \in E_i, \text{ there exists } f \in E_i \text{ such that } f < e \text{ and } t, f \models \alpha' \\
& && \text{and, for all } g \in E_i, \ f < g < e \text{ implies } t, g \models \alpha.
\end{aligned}
$$

Note that, Y_i is a modality expecting an argument α, whereas a and $\mathsf{Y}_i \leq \mathsf{Y}_j$ are basic constant formulas. Also, the since operators S_i are strict. $\mathsf{LocTL}[\mathsf{Y}_i \leq \mathsf{Y}_j, \mathsf{Y}_i, \mathsf{S}_i]$ trace formulas are evaluated on traces by interpreting the Boolean connectives in the natural way and by letting

$$t \models \exists_i \alpha \qquad \text{if there exists a maximal } i\text{-event } e \text{ in } t \text{ such that } t, e \models \alpha.$$

We will also consider various fragments of $\mathsf{LocTL}[\mathsf{Y}_i \leq \mathsf{Y}_j, \mathsf{Y}_i, \mathsf{S}_i]$ with inherited semantics. $\mathsf{LocTL}[\mathsf{Y}_i, \mathsf{S}_i]$ is the fragment where the constants $\mathsf{Y}_i \leq \mathsf{Y}_j$ are not allowed. Similarly, $\mathsf{LocTL}[\mathsf{Y}_i \leq \mathsf{Y}_j, \mathsf{S}_i]$ is the fragment without the modalities Y_i, and $\mathsf{LocTL}[\mathsf{S}_i]$ is the fragment where both $\mathsf{Y}_i \leq \mathsf{Y}_j$ and Y_i are disallowed.

We say that a trace language $L \subseteq \mathrm{Tr}(\Sigma)$ is definable in $\mathsf{LocTL}[\mathsf{Y}_i \leq \mathsf{Y}_j, \mathsf{Y}_i, \mathsf{S}_i]$ (resp. one of its fragments) if there is a trace formula β in $\mathsf{LocTL}[\mathsf{Y}_i \leq \mathsf{Y}_j, \mathsf{Y}_i, \mathsf{S}_i]$ (resp. in the corresponding fragment) such that $L = \{t \in \mathrm{Tr}(\Sigma) \mid t \models \beta\}$.

In Theorem 5.5, the three local temporal logics $\mathsf{LocTL}[\mathsf{Y}_i \leq \mathsf{Y}_j, \mathsf{Y}_i, \mathsf{S}_i]$, $\mathsf{LocTL}[\mathsf{Y}_i \leq \mathsf{Y}_j, \mathsf{S}_i]$ and $\mathsf{LocTL}[\mathsf{Y}_i, \mathsf{S}_i]$ are shown to have equal expressive power. We first prove in Lemma 5.4 that $\mathsf{LocTL}[\mathsf{Y}_i \leq \mathsf{Y}_j, \mathsf{S}_i]$ is at least as expressive as $\mathsf{LocTL}[\mathsf{Y}_i, \mathsf{S}_i]$; we only need to show that the modality Y_i of $\mathsf{LocTL}[\mathsf{Y}_i, \mathsf{S}_i]$ can be expressed in $\mathsf{LocTL}[\mathsf{Y}_i \leq \mathsf{Y}_j, \mathsf{S}_i]$. To this end, we will use the following derived constants $(\mathsf{Y}_i = \mathsf{Y}_j) = (\mathsf{Y}_i \leq \mathsf{Y}_j) \wedge (\mathsf{Y}_j \leq \mathsf{Y}_i)$ and $(\mathsf{Y}_i < \mathsf{Y}_j) = (\mathsf{Y}_i \leq \mathsf{Y}_j) \wedge \neg(\mathsf{Y}_j \leq \mathsf{Y}_i)$. Notice that $(\mathsf{Y}_i = \mathsf{Y}_i)$ is equivalent to $\mathsf{Y}_i \top$ and simply means that there are i events in the strict past of the current event.

For $i \in \mathscr{P}$ and an event formula α, we define

$$\mathsf{Y}_i^1 \, \alpha = \bigvee_j (\mathsf{Y}_i = \mathsf{Y}_j) \wedge (\bot \, \mathsf{S}_j \, \alpha)$$

$$\mathsf{Y}_i^{m+1} \, \alpha = \bigvee_j (\mathsf{Y}_i < \mathsf{Y}_j) \wedge [(\mathsf{Y}_i < \mathsf{Y}_j) \, \mathsf{S}_j \, (\mathsf{Y}_i^m \, \alpha)]$$

Remark 5.1. Notice that an event satisfying $(\mathsf{Y}_i = \mathsf{Y}_j) \wedge (\bot \, \mathsf{S}_j \, \alpha)$ or $(\mathsf{Y}_i < \mathsf{Y}_j) \wedge [(\mathsf{Y}_i < \mathsf{Y}_j) \, \mathsf{S}_j \, (\mathsf{Y}_i^m \, \alpha)]$ must be a j-event. Hence, we may restrict the use of $(\mathsf{Y}_i = \mathsf{Y}_j)$ and $(\mathsf{Y}_i < \mathsf{Y}_j)$ to be at j-events only and still get an expressively complete logic.

Lemma 5.2. *Consider a trace $t \in \mathrm{Tr}(\Sigma)$. For any event e in the trace, any process i, and any natural number m, if $t, e \models \mathsf{Y}_i^m \, \alpha$ then $t, e \models \mathsf{Y}_i \alpha$.*

Proof. The proof is by induction on m. The base case is when $m = 1$. Suppose that $t, e \models (\mathsf{Y}_i = \mathsf{Y}_j) \wedge (\bot \, \mathsf{S}_j \, \alpha)$ for some $j \in \mathscr{P}$. Due to $(\mathsf{Y}_i = \mathsf{Y}_j)$, we know that e_i, e_j exist and $e_i = e_j$. Now, from $\bot \, \mathsf{S}_j \, \alpha$, we get that $e \in E_j$ and $t, e_j \models \alpha$. We deduce that $t, e_i \models \alpha$ and $t, e \models \mathsf{Y}_i \alpha$.

For the inductive step, let $m \geq 1$ and suppose that $t, e \models (\mathsf{Y}_i < \mathsf{Y}_j) \wedge [(\mathsf{Y}_i < \mathsf{Y}_j)\mathsf{S}_j(\mathsf{Y}_i^m \alpha)]$ for some $j \in \mathscr{P}$. Due to $(\mathsf{Y}_i < \mathsf{Y}_j)$, we know that e_i, e_j exist and $e_i < e_j$. Now, from $t, e \models (\mathsf{Y}_i < \mathsf{Y}_j) \mathsf{S}_j (\mathsf{Y}_i^m \alpha)$, we deduce that $e \in E_j$ and there exists an event $f \in E_j$ in the strict past of e such that $t, f \models \mathsf{Y}_i^m \alpha$ and at all j-events between e and f, the formula $(\mathsf{Y}_i < \mathsf{Y}_j)$ is true. By induction, we get $t, f \models \mathsf{Y}_i \alpha$. Let $f = f^k < f^{k-1} < \cdots < f^1 < f^0 = e$ be the sequence of j-events between f and e. For all $0 \leq \ell < k$, we have $f_j^\ell = f^{\ell+1}$ and $t, f^\ell \models (\mathsf{Y}_i < \mathsf{Y}_j)$. We deduce by induction on ℓ that $e_i = f_i^\ell$ for all $0 \leq \ell \leq k$. This is clear when $\ell = 0$ since $e = f^0$ and for the inductive step it follows from $e_i = f_i^\ell < f_j^\ell = f^{\ell+1}$. Finally, $e_i = f_i$ and we get $t, e_i \models \alpha$ as desired. $\qquad\square$

Lemma 5.3. *Consider a trace $t \in \mathrm{Tr}(\Sigma)$. For any event e in the trace, any process i, if $t, e \models \mathsf{Y}_i \alpha$ then $t, e \models \mathsf{Y}_i^m \alpha$ for some $m \leq |\mathscr{P}|$.*

Proof. Assume that $t, e \models \mathsf{Y}_i \alpha$, i.e., e_i exists and $t, e_i \models \alpha$. Since e_i is in the strict past of e, there exists f such that $e_i \lessdot f \leq e$. It is well-known that there is $m \geq 1$ and a sequence of events $f = f^1 < f^2 < \cdots < f^m = e$ and processes j_ℓ such that $j_\ell \in \mathrm{loc}(f^{\ell-1}) \cap \mathrm{loc}(f^\ell)$ for all $1 < \ell \leq m$. Moreover, we may assume that the processes $j_2, \ldots, j_m$ are pairwise distinct, and also different from i since $e_i \lessdot f$. We get $m \leq |\mathscr{P}|$.

We show by induction on ℓ that $e_i = f_i^\ell$ and $t, f^\ell \models \mathsf{Y}_i^\ell \alpha$. For the base case $\ell = 1$, since $e_i \lessdot f^1$ we have $e_i = f_i^1$ and we find $j_1 \in \mathrm{loc}(e_i) \cap \mathrm{loc}(f^1)$. We get $t, f^1 \models (\mathsf{Y}_i = \mathsf{Y}_{j_1}) \wedge (\bot \mathsf{S}_{j_1} \alpha)$. Therefore, $t, f^1 \models \mathsf{Y}_i^1 \alpha$ and we are done. Consider now $1 \leq \ell < m$ and assume that $e_i = f_i^\ell$ and $t, f^\ell \models \mathsf{Y}_i^\ell \alpha$. Since $e_i < f^\ell < f^{\ell+1} \leq e$, we deduce that $f_i^{\ell+1} = e_i$ and all $j_{\ell+1}$ events g with $f^\ell < g \leq f^{\ell+1}$ satisfy $(\mathsf{Y}_i < \mathsf{Y}_{j_{\ell+1}})$. Therefore,

$$t, f^{\ell+1} \models (\mathsf{Y}_i < \mathsf{Y}_{j_{\ell+1}}) \wedge [(\mathsf{Y}_i < \mathsf{Y}_{j_{\ell+1}}) \, \mathsf{S}_{j_{\ell+1}} \, (\mathsf{Y}_i^\ell \alpha)].$$

We obtain $t, f^{\ell+1} \models \mathsf{Y}_i^{\ell+1} \alpha$ which concludes this proof. $\qquad\square$

Lemma 5.4. $\mathsf{LocTL}[\mathsf{Y}_i \leq \mathsf{Y}_j, \mathsf{S}_i]$ *is at least as expressive as* $\mathsf{LocTL}[\mathsf{Y}_i, \mathsf{S}_i]$.

Proof. From Lemmas 5.2 and 5.3 we see that $\mathsf{Y}_i \alpha$ is equivalent to the $\mathsf{LocTL}[\mathsf{Y}_i \leq \mathsf{Y}_j, \mathsf{S}_i]$ formula $\bigvee_{m \leq |\mathscr{P}|} \mathsf{Y}_i^m \alpha$. This concludes the proof. $\qquad\square$

We now establish the expressive completeness of our past-oriented local temporal logics $\mathsf{LocTL}[\mathsf{Y}_i, \mathsf{S}_i]$, $\mathsf{LocTL}[\mathsf{Y}_i \leq \mathsf{Y}_j, \mathsf{S}_i]$ and $\mathsf{LocTL}[\mathsf{Y}_i \leq \mathsf{Y}_j, \mathsf{Y}_i, \mathsf{S}_i]$. This crucially depends on the expressive completeness of a process-based pure future local temporal logic proved by Diekert and Gastin in [DG06]. It is unknown whether the fragment $\mathsf{LocTL}[\mathsf{S}_i]$ is as expressive as $\mathsf{LocTL}[\mathsf{Y}_i, \mathsf{S}_i]$ in general, see Theorem 5.7 for a partial result.

Theorem 5.5. *Let (Σ, loc) be a distributed alphabet over $\mathscr{P}$. Over $\mathrm{Tr}(\Sigma)$, first-order logic, $\mathsf{LocTL}[\mathsf{Y}_i \leq \mathsf{Y}_j, \mathsf{Y}_i, \mathsf{S}_i]$, $\mathsf{LocTL}[\mathsf{Y}_i, \mathsf{S}_i]$ and $\mathsf{LocTL}[\mathsf{Y}_i \leq \mathsf{Y}_j, \mathsf{S}_i]$ have the same expressive power, i.e., a language $L \subseteq \mathrm{Tr}(\Sigma)$ is definable by a first-order sentence if and only if it is definable by a trace formula in $\mathsf{LocTL}[\mathsf{Y}_i, \mathsf{S}_i]$ or in $\mathsf{LocTL}[\mathsf{Y}_i \leq \mathsf{Y}_j, \mathsf{S}_i]$.*

Proof. First, from the semantics of $\mathsf{LocTL}[\mathsf{Y}_i \leq \mathsf{Y}_j, \mathsf{Y}_i, \mathsf{S}_i]$, we clearly see that each language definable in $\mathsf{LocTL}[\mathsf{Y}_i \leq \mathsf{Y}_j, \mathsf{Y}_i, \mathsf{S}_i]$ is also first-order definable.

Conversely, we first show that $\mathsf{LocTL}[\mathsf{Y}_i, \mathsf{S}_i]$ is expressively complete. In [DG06], Diekert and Gastin give a process-based pure future local temporal logic which they show is expressively equivalent to first order logic over traces with a unique minimal event. The

event formulas of its past dual have the following syntax and semantics.

$$\text{Syntax:}\quad \alpha = \top \mid a \mid \neg\alpha \mid \alpha \vee \alpha \mid \mathsf{Y}_i\,\alpha \mid \alpha\,\mathbb{S}_i\,\alpha \qquad (a \in \Sigma, i \in \mathscr{P})$$

$$\text{Semantics:}\quad t, e \models \alpha\,\mathbb{S}_i\,\alpha' \text{ if there exists } f \in E_i \text{ such that } f \leq e \text{ and } t, f \models \alpha'$$

$$\text{and, for all } g \in E_i, \;\; f < g \leq e \text{ implies } t, g \models \alpha.$$

We show that $\mathsf{LocTL}[\mathsf{Y}_i, \mathsf{S}_i]$ is expressive enough to define the $\mathbb{S}_i$ operator. Consider the $\mathsf{LocTL}[\mathsf{Y}_i, \mathsf{S}_i]$ event formulas $\gamma = \alpha' \vee (\alpha \wedge \alpha\,\mathsf{S}_i\,\alpha')$ and $\underline{i} = \bigvee_{a \in \Sigma_i} a$. Then $\alpha\,\mathbb{S}_i\,\alpha'$ is equivalent to the $\mathsf{LocTL}[\mathsf{Y}_i, \mathsf{S}_i]$ event formula $(\underline{i} \wedge \gamma) \vee (\neg\underline{i} \wedge \mathsf{Y}_i\,\gamma)$. This shows $\mathsf{LocTL}[\mathsf{Y}_i, \mathsf{S}_i]$ is expressively complete over *prime traces* (traces with a unique maximal event). More precisely, for each first order sentence φ, there is an event formula $\overline{\varphi}$ in $\mathsf{LocTL}[\mathsf{Y}_i, \mathsf{S}_i]$ such that for all prime traces t we have $t \models \varphi$ iff $t, \max(t) \models \overline{\varphi}$.

Adsul and Sohoni [AS02, Ads04] showed that any first order sentence over traces can be equivalently expressed in a *first normal form* which is a boolean combination of first order sentences evaluated only in the process views of traces. More precisely, for a trace $t = (E, \leq, \lambda)$, let t_i denote the trace induced by the restriction to $\downarrow E_i$. Given any first order sentence φ, by [AS02] there exists a natural number n and sentences $\varphi_{i,m}$ ($i \in \mathscr{P}$, $1 \leq m \leq n$) such that $t \models \varphi$ iff for some m, for each $i \in \mathscr{P}$, we have $t_i \models \varphi_{i,m}$. Consider the equivalent event formulas $\overline{\varphi_{i,m}}$ in $\mathsf{LocTL}[\mathsf{Y}_i, \mathsf{S}_i]$. Notice that each t_i is a prime trace or is the empty trace ε. If $\varepsilon \not\models \varphi_{i,m}$ then we let $\beta_{i,m} = \exists_i \overline{\varphi_{i,m}}$, otherwise we let $\beta_{i,m} = \exists_i \overline{\varphi_{i,m}} \vee \neg\exists_i \top$. We deduce that $t_i \models \varphi_{i,m}$ iff $t \models \beta_{i,m}$. The $\mathsf{LocTL}[\mathsf{Y}_i, \mathsf{S}_i]$ sentence $\bigvee_m \bigwedge_i \beta_{i,m}$ is equivalent to φ. Therefore, $\mathsf{LocTL}[\mathsf{Y}_i, \mathsf{S}_i]$ is expressively complete.

By Lemma 5.4, $\mathsf{LocTL}[\mathsf{Y}_i \leq \mathsf{Y}_j, \mathsf{S}_i]$ is also expressively complete and the proof is complete. $\qquad\square$

5.2. Cascade decomposition for $\mathsf{LocTL}[\mathsf{S}_i]$.
We now relate $\mathsf{LocTL}[\mathsf{S}_i]$ with local cascade products of localized reset automata $U_2[p]$.

Theorem 5.6. *A trace language is defined by a $\mathsf{LocTL}[\mathsf{S}_i]$ formula if and only if it is accepted by a local cascade product of asynchronous reset automata of the form $U_2[p]$.*

Proof. First consider a local cascade product $A = U_2[p] \circ_\ell B$ ($p \in \mathscr{P}$) and suppose that the languages accepted by B are $\mathsf{LocTL}[\mathsf{S}_i]$-definable. Let $\{S_i\}$ be the state sets of $U_2[p]$ and let χ be the asynchronous transducer associated with $U_2[p]$ and its initial state, say 1. By the local cascade product principle (Theorem 3.29), any language accepted by A is a union of languages of the form $L_1 \cap \chi^{-1}(L_2)$ where $L_1 \subseteq \mathrm{Tr}(\Sigma)$ is accepted by $U_2[p]$ and $L_2 \subseteq \mathrm{Tr}(\Sigma \times_\ell S)$ is accepted by B.

The languages accepted by $U_2[p]$ are defined by the $\mathsf{LocTL}[\mathsf{S}_i]$-formulas

$$\text{With global accepting state 2:}\qquad \exists_p (R_2 \vee (\neg R_1 \wedge ((\neg R_1)\,\mathsf{S}_p\,R_2)));$$
$$\text{With global accepting state 1:}\qquad \neg\exists_p (R_2 \vee (\neg R_1 \wedge ((\neg R_1)\,\mathsf{S}_p\,R_2))).$$

To conclude that the languages accepted by A are $\mathsf{LocTL}[\mathsf{S}_i]$-definable, we only need to show that if L_2 is $\mathsf{LocTL}[\mathsf{S}_i]$-definable over alphabet $\Sigma \times_\ell S$, then $\chi^{-1}(L_2)$ is $\mathsf{LocTL}[\mathsf{S}_i]$-definable over (Σ, loc). This is done by structural induction on $\mathsf{LocTL}[\mathsf{S}_i]$-formulas over $\Sigma \times_\ell S$. For an event formula α of $\mathsf{LocTL}[\mathsf{S}_i]$ over $\Sigma \times_\ell S$, we provide an event formula $\hat{\alpha}$ over (Σ, loc) such that for any trace $t \in \mathrm{Tr}(\Sigma)$, and any event e in t, we have $t, e \models \hat{\alpha}$ if and

only if $\chi(t), e \models \alpha$. The non-trivial case here is the base case of letter formula $\alpha = (a, s_a)$. If $p \notin \mathrm{loc}(a)$, then we let $\hat{\alpha} = a$. If instead $p \in \mathrm{loc}(a)$, we let

$$\hat{\alpha} = a \wedge (\neg R_1) \, \mathsf{S}_p \, R_2 \qquad\qquad\qquad \text{if } [s_a]_p = 2$$
$$\hat{\alpha} = a \wedge \neg((\neg R_1) \, \mathsf{S}_p \, R_2) \qquad\qquad \text{if } [s_a]_p = 1$$

We now establish the converse implication. For any $\mathsf{LocTL}[\mathsf{S}_i]$ event formula α, We construct an asynchronous automaton A_α, which is a local cascade product of copies of $U_2[p]$, and which is such that for any trace t and event e of t, *each* local state $[A_\alpha(\downarrow e)]_i$ $(i \in \mathrm{loc}(e))$ completely determines whether $t, e \models \alpha$. A_α is constructed by structural induction on the $\mathsf{LocTL}[\mathsf{S}_i]$ event formula α.

Base case: Suppose that $\alpha = a \in \Sigma$. We let $A_\alpha = (\{S_i\}, \{\delta_a\}, s_{\mathrm{in}})$ where $S_i = \{\bot\}$ for all $i \notin \mathrm{loc}(a)$, and $S_i = \{\top, \bot\}$ for all $i \in \mathrm{loc}(a)$. For any P-state s, if for all $i \in P$ we have $s_i = \bot$ (resp. $\top$), then we write $s = \bot$ (resp. $s = \top$). We let $s_{\mathrm{in}} = \bot$. The local transition δ_a is a reset to $\top$ and the transitions δ_b $(b \neq a)$ are resets to $\bot$. This construction ensures that, for all $i \in \mathrm{loc}(a)$ we have $[A_\alpha(\downarrow e)]_i = \top$ if and only if $t, e \models \alpha$. It is also easy to see that A_α is a local cascade product of $U_2[p]$ for $p \in \mathrm{loc}(a)$.

Inductive case: The non-trivial case is $\alpha = \beta \, \mathsf{S}_j \, \gamma$. By inductive hypothesis, we have constructed automata A_β and A_γ as local cascade products of copies of $U_2[p]$. Let $A = (\{S_i\}, \{\delta_a\}, s_{\mathrm{in}}) = U_2[j] \circ_\ell \hat{A}_\beta \circ_\ell \hat{A}_\gamma$ where the first $U_2[j]$ with initial state $\bot$ is such that all letters from Σ_j reset the state to $\top$; and $\hat{A}_\beta$ (resp. $\hat{A}_\gamma$) simply lifts A_β (resp. A_γ) to appropriate input alphabet by ignoring the local state information provided in the local cascade product. Hence, A is a local cascade product of copies of $U_2[p]$ which simultaneously provides the truth values of β and γ at any event and remembers whether some j-event already occured. Let χ be the associated local asynchronous transducer.

We construct $B = (\{Q_i\}, \{\delta_{(a,s_a)}\}, q_{\mathrm{in}})$ over $\Sigma \times_\ell S$ such that $A \circ_\ell B$ is the required asynchronous automaton. Let $Q_i = \{\top, \bot\}$ for all $i \in \mathscr{P}$. Again, we denote a P-state q as $\bot$ (resp. $\top$) if $q_i = \bot$ (resp. $q_i = \top$) for all $i \in P$. We let the initial state be $q_{\mathrm{in}} = \bot$. For any $a \notin \Sigma_j$, we let the local transition $\delta_{(a,s_a)}$ be the reset to $\bot$. By assumption, if a j-event e of the trace $\chi(t)$ is labelled (a, s_a), then $[s_a]_j$ determines whether some j-event occured in the past of e, and if this is the case, the truth values of β and γ at the previous j-event e_j. When ξ is a boolean combination of β, γ, then we write $[s_a]_j \vdash \mathsf{Y}_j \xi$ if according to the j state of s_a, there is a previous j-event e_j and ξ is true at e_j. Then the transition for $a \in \Sigma_j$ is given by

$$\delta_{(a,s_a)} = \text{reset to } \bot \qquad\qquad \text{if } [s_a]_j \nvdash \mathsf{Y}_j \top \text{ or } [s_a]_j \vdash \mathsf{Y}_j(\neg\beta \wedge \neg\gamma)$$
$$\delta_{(a,s_a)} = \text{reset to } \top \qquad\qquad \text{if } [s_a]_j \vdash \mathsf{Y}_j \gamma$$
$$\delta_{(a,s_a)}(q_a) = \top \qquad\qquad \text{if } [s_a]_j \vdash \mathsf{Y}_j(\beta \wedge \neg\gamma) \text{ and } [q_a]_j = \top$$
$$\delta_{(a,s_a)}(q_a) = \bot \qquad\qquad \text{if } [s_a]_j \vdash \mathsf{Y}_j(\beta \wedge \neg\gamma) \text{ and } [q_a]_j = \bot$$

The transitions make sense if we recall the identity $\beta \mathsf{S}_j \gamma \equiv \bot \mathsf{S}_j (\gamma \vee (\beta \wedge (\beta \mathsf{S}_j \gamma)))$. Note that in the last two cases above, $\delta_{(a,s_a)}$ is the identity transformation on process j states. Hence, process j update is realised by some $U_2[j]$. The other processes of $\mathrm{loc}(a)$ can update their states mimicking process j state update, once they also have the truth value of $\alpha = \beta \mathsf{S}_j \gamma$ at the previous j-event e_j, which is being made available at event e by the above $U_2[j]$'s

state. In view of this, it is easy to verify that B is a local cascade product of $U_2[j]$ followed by $U_2[p]$ for $p \neq j$.

To conclude the proof, we have to handle trace formulas of $\mathsf{LocTL}[\mathsf{S}_i]$, which are the sentences defining trace languages. So consider a trace formula $\exists_j \alpha$ where α is an event formula in $\mathsf{LocTL}[\mathsf{S}_i]$. Let A_α be the local cascade product of copies of $U_2[p]$ constructed above. As in the inductive case above, we also use a copy of $U_2[j]$ which remembers whether some j-event already occurred in the past. Let $t = (E, \leq, \lambda) \in \mathrm{Tr}(\Sigma)$ and let $s = (U_2[j] \circ_\ell A_\alpha)(t)$. The local state s_j allows to determine whether $E_j \neq \emptyset$ thanks to $U_2[j]$, and in this case whether the maximal event of E_j satisfies α thanks to A_α. $\qquad\square$

We are now ready to give new characterizations of first-order definable trace languages over aKR distributed alphabets.

Theorem 5.7. *Let (Σ, loc) be a distributed alphabet and $L \subseteq \mathrm{Tr}(\Sigma)$ be a trace language. If (Σ, loc) is aKR (e.g., if (Σ, loc) is an acyclic architecture), then the following statements are equivalent.*

(1) *L is definable in first-order logic.*
(2) *L is accepted by a counter-free diamond-automaton (or, recognized by an aperiodic monoid).*
(3) *L is accepted by a local cascade product of copies of $U_2[p]$ (or, recognized by an asynchronous wreath product of atms of the form $U_2[p]$).*
(4) *L is accepted by a counter-free asynchronous automaton (or, recognized by an aperiodic asynchronous transformation monoid).*
(5) *L is definable in $\mathsf{LocTL}[\mathsf{S}_i]$.*

Proof. By [EM96], first-order definability coincides with recognizability by an aperiodic monoid. By Theorem 5.6, $\mathsf{LocTL}[\mathsf{S}_i]$-definability coincides with acceptability by local cascade product of asynchronous reset automata of the form $U_2[p]$, or equivalently asynchronous wreath product of atm's of the form $U_2[p]$.

If (Σ, loc) is aKR, recognizability by an aperiodic monoid implies recognizability by an asynchronous wreath product of asynchronous transformation monoids of the form $U_2[p]$. The converse implication follows from the easy verification that a wreath product of asynchronous transformation monoids of the form $U_2[p]$ is an aperiodic atm (that is, the associated global tm is aperiodic). Putting these equivalences together and keeping in mind the correspondences between (asynchronous) automata and (asynchronous) morphisms into (asynchronous) transformation monoids, we get the desired result. $\qquad\square$

5.3. **Cascade decomposition for $\mathsf{LocTL}[\mathsf{Y}_i \leq \mathsf{Y}_j, \mathsf{S}_i]$.** We turn now to the logic $\mathsf{LocTL}[\mathsf{Y}_i \leq \mathsf{Y}_j, \mathsf{S}_i]$ and its relation with local cascade products. Here, we seek a decomposition result which is valid for all distributed alphabets, and not only for those that are known to be aKR. Unfortunately, we do not know whether all aperiodic trace languages can be accepted by local cascade products of copies of $U_2[p]$. It is interesting to notice first that the argument in the proof of Theorem 5.6 cannot be lifted to the logic $\mathsf{LocTL}[\mathsf{Y}_i \leq \mathsf{Y}_j, \mathsf{S}_i]$. More precisely, when $\alpha = \mathsf{Y}_i \leq \mathsf{Y}_j$, the automaton A_α specified in this proof cannot, in general, be obtained as a local cascade product of copies of $U_2[p]$. To explain this formally, we use the notion of Γ-labelling functions computed by asynchronous automata, defined in Section 2.3.

Given an event formula $\alpha \in \mathsf{LocTL}[\mathsf{Y}_i \leq \mathsf{Y}_j, \mathsf{Y}_i, \mathsf{S}_i]$, we let θ_α be the $\{0, 1\}$-labelling function which decorates each event of a trace $t = (E, \leq, \lambda) \in \mathrm{Tr}(\Sigma)$ with the truth value of

α, i.e., $\theta_\alpha(t) = (E, \leq, (\lambda, \mu))$ where for all $e \in E$, we have $\mu_\alpha(e) = 1$ if $t, e \models \alpha$ and $\mu_\alpha(e) = 0$ otherwise.

In the proof of Theorem 5.6, we have constructed for each formula $\alpha \in \mathsf{LocTL}[\mathsf{S}_i]$ an asynchronous automaton A_α which computes θ_α, and which is a local cascade product of copies of $U_2[p]$. Lemma 5.8 below shows that this cannot be extended to $\mathsf{LocTL}[\mathsf{Y}_i \leq \mathsf{Y}_j, \mathsf{S}_i]$. This is a slight modification of an example from Adsul and Sohoni [AS04].

Lemma 5.8. *Let $\mathscr{P} = \{1, 2, 3\}$, $\Sigma = \{a, b, c\}$ with the distribution $\Sigma_1 = \{a, c\}$, $\Sigma_2 = \{a, b\}$, $\Sigma_3 = \{b, c\}$ and let $\alpha = \mathsf{Y}_1 \leq \mathsf{Y}_3$. As no two letters of Σ are independent, $\mathrm{Tr}(\Sigma)$ is isomorphic to the free monoid Σ^*.*

Nevertheless, there is no aperiodic asynchronous automaton over (Σ, loc) which computes θ_α. In particular, θ_α cannot be computed by a local cascade product of copies of $U_2[p]$.

Proof. Suppose, for the sake of contradiction, that there exists an aperiodic asynchronous automaton $A = (\{S_i\}, \{\delta_a\}, s_{\mathrm{in}})$ and a $\{0, 1\}$-transducer $\hat{A} = (\{S_i\}, \{\delta_a\}, s_{\mathrm{in}}, \{\mu_a\})$ which computes θ_α. The output function $\mu_c \colon S_c \to \{0, 1\}$ computes the truth value of $\alpha = \mathsf{Y}_1 \leq \mathsf{Y}_3$ at c-events.

As A is aperiodic, there exists n such that, starting at the initial global state s_{in}, traces $(ab)^n$ and $(ab)^{n+1}$ reach the same global state, say $s = (s_1, s_2, s_3) = A((ab)^n) = A((ab)^{n+1})$. It follows that the trace ab fixes s. As the transition function of a (resp. b) does not change the local state of process 3 (resp. process 1), it must be that the a-transition at s leads to a global state of the form $s' = (s_1, s'_2, s_3)$. In particular, s' is the global state reached on input $(ab)^n a$. Now, consider the traces $t = (ab)^n c$ and $t' = (ab)^n ac$. The c-event in t satisfies $\alpha = \mathsf{Y}_1 \leq \mathsf{Y}_3$ whereas the c-event in t' does not. Since $s_c = (s_1, s_3) = s'_c$, this contradicts the fact that μ_c computes the truth value of α at c-events. $\square$

On the other hand, the *gossip automaton*, one of the most important tools in the theory of asynchronous automata, due to Mukund and Sohoni [MS97], computes all the constants of $\mathsf{LocTL}[\mathsf{Y}_i \leq \mathsf{Y}_j, \mathsf{S}_i]$. Let $\Gamma = \{0, 1\}^{\mathscr{P} \times \mathscr{P}}$ and θ^{Y} be the Γ-labelling function which decorates each event e of a trace t with the truth values of all constants $\mathsf{Y}_i \leq \mathsf{Y}_j$, i.e., for all $i, j \in \mathscr{P}$, $\mu^{\mathsf{Y}}_{i,j}(e) = 1$ if $t, e \models \mathsf{Y}_i \leq \mathsf{Y}_j$ and $\mu^{\mathsf{Y}}_{i,j}(e) = 0$ otherwise. Since the events referred to by $\{\mathsf{Y}_i\}$ are called primary events, we call θ^{Y} the *primary order labelling function*.

Theorem 5.9 (Gossip Automaton [MS97])**.** *There exists an asynchronous automaton $\mathcal{G} = (\{\Upsilon_i\}, \{\nabla_a\}, v_{in})$, called the* gossip automaton*, which computes θ^{Y}.*

Note that, as a consequence of Lemma 5.8, the gossip automaton is not aperiodic in general, a fact that was already established in [AS04].

In view of Theorems 5.6 and 5.9, the following lemma will help relate $\mathsf{LocTL}[\mathsf{Y}_i \leq \mathsf{Y}_j, \mathsf{S}_i]$ languages and local cascade products on arbitrary distributed alphabets.

Lemma 5.10. *Let (Σ, loc) be a distributed alphabet.*
(1) *If $L \subseteq \mathrm{Tr}(\Sigma)$ is $\mathsf{LocTL}[\mathsf{Y}_i \leq \mathsf{Y}_j, \mathsf{S}_i]$-definable over Σ then $L' = \theta^{\mathsf{Y}}(L)$ is $\mathsf{LocTL}[\mathsf{S}_i]$-definable over $\Sigma \times \Gamma$. Notice that $L = (\theta^{\mathsf{Y}})^{-1}(L')$.*
(2) *If $L' \subseteq \mathrm{Tr}(\Sigma \times \Gamma)$ is $\mathsf{LocTL}[\mathsf{Y}_i \leq \mathsf{Y}_j, \mathsf{S}_i]$-definable over $\Sigma \times \Gamma$ then $L = (\theta^{\mathsf{Y}})^{-1}(L')$ is $\mathsf{LocTL}[\mathsf{Y}_i \leq \mathsf{Y}_j, \mathsf{S}_i]$-definable over Σ.*

Proof. We simply write θ for the primary order labelling function θ^{Y}.
(1) For each event formula $\alpha \in \mathsf{LocTL}[\mathsf{Y}_i \leq \mathsf{Y}_j, \mathsf{S}_i]$ over Σ, we construct a formula $\widetilde{\alpha} \in \mathsf{LocTL}[\mathsf{S}_i]$ over $\Sigma \times \Gamma$ such that, for all traces $t \in \mathrm{Tr}(\Sigma)$ and all events e in t, we have $t, e \models \alpha$

if and only if $\theta(t), e \models \widetilde{\alpha}$. The construction is by structural induction on α. The constants are the interesting cases. For $a \in \Sigma$, we define

$$\tilde{a} = \bigvee_{\gamma \in \Gamma} (a, \gamma) \,.$$

Now, for $i, j \in \mathscr{P}$, we let $\Gamma_{i,j} = \{\gamma \in \Gamma \mid \gamma_{i,j} = 1\}$ and we define

$$\widetilde{\mathsf{Y}_i \leq \mathsf{Y}_j} = \bigvee_{a \in \Sigma, \gamma \in \Gamma_{i,j}} (a, \gamma) \,.$$

The inductive cases are trivial, for instance $\widetilde{\alpha \, \mathsf{S}_i \, \beta} = \widetilde{\alpha} \, \mathsf{S}_i \, \widetilde{\beta}$. We deduce that, if L is $\mathsf{LocTL}[\mathsf{Y}_i \leq \mathsf{Y}_j, \mathsf{S}_i]$-definable over Σ then $\theta(L)$ is $\mathsf{LocTL}[\mathsf{S}_i]$-definable over $\Sigma \times \Gamma$.

(2) Given an event formula $\alpha \in \mathsf{LocTL}[\mathsf{Y}_i \leq \mathsf{Y}_j, \mathsf{S}_i]$, we construct an event formula $\widehat{\alpha} \in \mathsf{LocTL}[\mathsf{Y}_i \leq \mathsf{Y}_j, \mathsf{S}_i]$ such that, for all traces $t \in \mathrm{Tr}(\Sigma)$ and all events e in t, we have $\theta(t), e \models \alpha$ if and only if $t, e \models \widehat{\alpha}$. Again, the construction is by structural induction and the interesting cases are the constants. For $(a, \gamma) \in \Sigma \times \Gamma$, we define

$$\widehat{(a, \gamma)} = a \;\wedge\; \bigwedge_{i,j \in \mathscr{P} \mid \gamma_{i,j}=1} \mathsf{Y}_i \leq \mathsf{Y}_j \;\wedge\; \bigwedge_{i,j \in \mathscr{P} \mid \gamma_{i,j}=0} \neg \, \mathsf{Y}_i \leq \mathsf{Y}_j \,.$$

The other cases are trivial, e.g., $\widehat{\mathsf{Y}_i \leq \mathsf{Y}_j} = \mathsf{Y}_i \leq \mathsf{Y}_j$ and $\widehat{\alpha \, \mathsf{S}_i \, \beta} = \widehat{\alpha} \, \mathsf{S}_i \, \widehat{\beta}$.   □

Since the gossip automaton $\mathcal{G}$ computes θ^{Y} and all $\mathsf{LocTL}[\mathsf{S}_i]$-definable languages over $(\Sigma \times \Gamma, \mathrm{loc})$ can be accepted by a local cascade product of copies of asynchronous reset automata of the form $U_2[p]$ (Theorem 5.6), we deduce from Lemma 5.10 (1) and Proposition 3.31 that all $\mathsf{LocTL}[\mathsf{Y}_i \leq \mathsf{Y}_j, \mathsf{S}_i]$-definable languages over (Σ, loc) can be accepted by a local cascade product of the gossip automaton $\mathcal{G}$ followed by copies of asynchronous reset automata.

Now, as we saw, the gossip automaton exhibits a non-aperiodic behaviour in general. In order to get a converse of the above statement, we introduce a restricted version of the local cascade product.

Let A be an asynchronous automaton over $(\Sigma \times \Gamma, \mathrm{loc})$. The θ^{Y}-*restricted* local cascade product $\mathcal{G} \circ_\ell^r A$ is an asynchronous automaton which runs $\mathcal{G}$ on an input trace $t \in \mathrm{Tr}(\Sigma)$, and runs A over $\theta^{\mathsf{Y}}(t)$. Formally, if $\widehat{\mathcal{G}} = (\{\Upsilon_i\}, \{\nabla_a\}, v_{\mathrm{in}}, \{\mu_a\})$ computes θ^{Y} and $A = (\{S_i\}, \{\delta_{(a,\gamma)}\}, s_{\mathrm{in}})$ then $\mathcal{G} \circ_\ell^r A$ is the asynchronous automaton $\mathcal{G} \circ_\ell^r A = (\{R_i\}, \{\Delta_a\}, (v_{\mathrm{in}}, s_{\mathrm{in}}))$ over (Σ, loc), where $R_i = \Upsilon_i \times S_i$ for $i \in \mathscr{P}$ and, for $a \in \Sigma$ and $(v_a, s_a) \in R_a$, $\Delta_a((v_a, s_a)) = (\nabla_a(v_a), \delta_{(a, \mu_a(v_a))}(s_a))$. Notice that, in the definition of the transition relation of this *restricted* cascade product, the a-state v_a of $\mathcal{G}$ has been abstracted to $\mu_a(v_a) \in \Gamma$. Notice that languages accepted by $\mathcal{G} \circ_\ell^r A$ are also accepted by $\mathcal{G} \circ_\ell A'$ where A' is the extension of A to the suitable alphabet $\Sigma \times_\ell S$.

Lemma 5.11. *A trace language $L \subseteq \mathrm{Tr}(\Sigma)$ is accepted by $\mathcal{G} \circ_\ell^r A$ (with all global states of the gossip automaton accepting) if and only if $L = (\theta^{\mathsf{Y}})^{-1}(L')$, where L' is a trace language over $(\Sigma \times \Gamma, \mathrm{loc})$ accepted by A.*

Proof. The first component of $\mathcal{G} \circ_\ell^r A$ runs $\mathcal{G}$ on the input trace $t \in \mathrm{Tr}(\Sigma)$ and accepts since all global states of $\mathcal{G}$ are accepting. Now, the second components runs A on $\theta^{\mathsf{Y}}(t)$. Therefore, t is accepted by $\mathcal{G} \circ_\ell^r A$ if and only if $\theta^{\mathsf{Y}}(t)$ is accepted by A.   □

Corollary 5.12. *Let* (Σ, loc) *be a distributed alphabet and let* $L \subseteq \mathrm{Tr}(\Sigma)$ *be a trace language. Then* L *is* $\mathsf{LocTL}[\mathsf{Y}_i \leq \mathsf{Y}_j, \mathsf{S}_i]$*-definable if and only if* L *is accepted by a* θ^{Y}*-restricted* local cascade product $\mathcal{G} \circ_\ell^r A$, *where* $\mathcal{G}$ *is the gossip automaton and* A *is a local cascade product of asynchronous reset automata of the form* $U_2[p]$.

Proof. If L is $\mathsf{LocTL}[\mathsf{Y}_i \leq \mathsf{Y}_j, \mathsf{S}_i]$-definable then $L' = \theta^{\mathsf{Y}}(L)$ is $\mathsf{LocTL}[\mathsf{S}_i]$-definable by Lemma 5.10 (1). By Theorem 5.6, L' is accepted by an asynchronous automaton A which is a local cascade product of asynchronous reset automata of the form $U_2[p]$. Since $L = \left(\theta^{\mathsf{Y}}\right)^{-1}(L')$, the left-to-right implication follows from Lemma 5.11.

Conversely, assume that L is accepted by $\mathcal{G} \circ_\ell^r A$ where A is a local cascade product of asynchronous reset automata of the form $U_2[p]$. By Lemma 5.11, we have $L = \left(\theta^{\mathsf{Y}}\right)^{-1}(L')$ where L' is a trace language over $(\Sigma \times \Gamma, \mathrm{loc})$ accepted by A. By Theorem 5.6, L' is $\mathsf{LocTL}[\mathsf{S}_i]$-definable over $(\Sigma \times \Gamma, \mathrm{loc})$. Finally, Lemma 5.10 (2) implies that L is $\mathsf{LocTL}[\mathsf{Y}_i \leq \mathsf{Y}_j, \mathsf{S}_i]$-definable. $\qquad\square$

Remark 5.13. In Lemma 5.11 and Corollary 5.12, as well as in Theorem 5.14 below, we may replace the gossip automaton $\mathcal{G}$ by any asynchronous automaton computing the primary order labelling function θ^{Y}.

We close this section with the following theorem, summarizing our characterization of first-order definable trace languages using local cascade products.

Theorem 5.14. *Let* (Σ, loc) *be a distributed alphabet and let* $L \subseteq \mathrm{Tr}(\Sigma)$ *be a trace language. The following are equivalent:*

(1) L *is recognized by an aperiodic monoid.*
(2) L *is star-free.*
(3) L *is definable in first-order logic.*
(4) L *is definable in* $\mathsf{LocTL}[\mathsf{Y}_i, \mathsf{S}_i]$ *or in* $\mathsf{LocTL}[\mathsf{Y}_i \leq \mathsf{Y}_j, \mathsf{S}_i]$ *or in* $\mathsf{LocTL}[\mathsf{Y}_i \leq \mathsf{Y}_j, \mathsf{Y}_i, \mathsf{S}_i]$.
(5) L *is accepted by a* θ^{Y}*-restricted* local cascade product $\mathcal{G} \circ_\ell^r A$ *where* $\mathcal{G}$ *is the gossip automaton and* A *is a local cascade product of copies of asynchronous reset automata of the form* $U_2[p]$.
(6) L *is accepted by a* θ^{Y}*-restricted* local cascade product $\mathcal{G} \circ_\ell^r A$ *where* $\mathcal{G}$ *is the gossip automaton and* A *is an aperiodic asynchronous automaton.*

Proof. As mentioned before, Guaiana, Restivo and Salemi [GRS92] established the equivalence of (1) and (2), and Ebinger and Muscholl [EM96] proved that these are equivalent to (3). The equivalence between (3) and (4) is Theorem 5.5 and Corollary 5.12 gives the equivalence with (5).

The equivalence with (6) was first proved by Adsul and Sohoni [AS04]. We can obtain it as follows. First, (5) implies (6) since a local cascade product of aperiodic asynchronous automata is again aperiodic. Next, we show that (6) implies (4) as in the proof of Corollary 5.12. Assume that L is accepted by $\mathcal{G} \circ_\ell^r A$ where A is an aperiodic asynchronous automaton. By Lemma 5.11, we have $L = \left(\theta^{\mathsf{Y}}\right)^{-1}(L')$ where L' is a trace language over $(\Sigma \times \Gamma, \mathrm{loc})$ accepted by A. Since (1) implies (4), the language L' accepted by A is $\mathsf{LocTL}[\mathsf{Y}_i \leq \mathsf{Y}_j, \mathsf{S}_i]$-definable over $(\Sigma \times \Gamma, \mathrm{loc})$. Finally, Lemma 5.10 (2) implies that L is $\mathsf{LocTL}[\mathsf{Y}_i \leq \mathsf{Y}_j, \mathsf{S}_i]$-definable over (Σ, loc). $\qquad\square$

6. GLOBAL CASCADE SEQUENCES, THE RELATED PRINCIPLE AND TEMPORAL LOGICS

We have established a direct correspondence between asynchronous wreath products of asynchronous transformation monoids and local cascade products of asynchronous automata (Proposition 3.28 and Theorem 3.29). In particular, over aKR distributed alphabets, any asynchronous automaton is simulated by a local cascade product of our proposed localised two-state reset and localised permutation asynchronous automata. This yields the same benefits as in the theory of word languages (see Section 5, in particular Theorem 5.7, for a concrete example). However, we do not know which distributed alphabets with non-acyclic architecture are aKR. Despite this, we have been able to get decomposition results (see Theorem 5.14) for first order definable trace languages over general architectures. This has been possible thanks to the expressive completeness of $\mathsf{LocTL}[\mathsf{Y}_i \leq \mathsf{Y}_j, \mathsf{S}_i]$ (Theorem 5.5) and the primary order labelling function θ^Y which allows to reason about $\mathsf{LocTL}[\mathsf{Y}_i \leq \mathsf{Y}_j, \mathsf{S}_i]$-definability over (Σ, loc) in terms of $\mathsf{LocTL}[\mathsf{S}_i]$-definability over $(\Sigma \times \Gamma, \mathrm{loc})$ (see Lemma 5.10). We have also ruled out, in Lemma 5.8, the possibility of an aperiodic asynchronous automaton that computes θ^Y. This finally leads to a restricted version of the local cascade product characterization in Theorem 5.14, which circumvents the non-aperiodic behaviour of the gossip automaton which computes θ^Y.

In this section, we propose *global cascade sequences* as a new model for accepting trace languages. This model is built using asynchronous automata and lets us pose automata-theoretic and language-theoretic decomposition questions in the same spirit as with the local cascade product. Its definition and acceptance condition are inspired by the operational point of view of the local cascade product, and are natural from an automata-theoretic viewpoint. Further, it supports a global cascade principle in the same vein as the local cascade principle supported by local cascade products.

Later in the section, we show that global cascade sequences of localised two-state reset asynchronous automata accept exactly the first order definable trace languages. In fact, we establish that $\mathsf{LocTL}[\mathsf{Y}_i, \mathsf{S}_i]$-definability matches acceptability by a global cascade sequence of copies of $U_2[p]$ and use the expressive completeness of $\mathsf{LocTL}[\mathsf{Y}_i, \mathsf{S}_i]$ from Theorem 5.5. This allows a characterization of first order logic purely in terms of an asynchronous cascade of copies of $U_2[p]$ albeit using *global cascade sequences*. The new characterization remains in the realm of aperiodic asynchronous devices and can be considered *intrinsic* in the spirit of the 'first-order = aperiodic' slogan. Of course, all this comes at a cost! What we lose, bringing in global cascade sequences instead of local cascade products in decomposition results, is the nice correspondence to an algebraic operation. In Section 7, we show how to construct an asynchronous automaton that realizes a global cascade sequence, making use, again, of the gossip automaton.

Let $A = (\{S_i\}, \{\delta_a\}, s_{\mathrm{in}})$ be an asynchronous automaton over (Σ, loc) and χ_A be the asynchronous transducer computed by A. Recall that $\chi_A(t)$ preserves the underlying poset of events of t and, at each event, records the previous local states of the processes *participating* in that event. We now introduce a natural variant of χ_A called the *global-state labelling function*, where we record at each event e the *best global state* that causally precedes e. This is the best global state that the processes participating in the current event are (collectively) aware of.

To be more precise, we first set additional notation. Recall that the alphabet $\Sigma \times S$ can be equipped with a distributed structure (over $\mathscr{P}$) by letting $\mathrm{loc}((a, s)) = \mathrm{loc}(a)$, that is,

$(\Sigma \times S)_i = \Sigma_i \times S$. We refer to (Σ, loc) as the *input alphabet* of A and $(\Sigma \times S, \mathrm{loc})$ as its *output alphabet*.

Definition 6.1 (Global-state Labelling Function)**.** Let $A = (\{S_i\}, \{\delta_a\}, s_{\mathrm{in}})$ be an asynchronous automaton over (Σ, loc). The global-state labelling function of A is the S-labelling function $\zeta_A \colon \mathrm{Tr}(\Sigma) \to \mathrm{Tr}(\Sigma \times S)$ defined for $t = (E, \leq, \lambda) \in \mathrm{Tr}(\Sigma)$ by $\zeta_A(t) = (E, \leq, (\lambda, \mu)) \in \mathrm{Tr}(\Sigma \times S)$ with the labelling $\mu \colon E \to S$ given by $\mu(e) = s$ where $s = A(\Downarrow e)$

Remark 6.2. The global-state labelling function and asynchronous transducer associated with an automaton coincide in the sequential (single process) setting.

Remark 6.3. Letting $f((a, s)) = (a, s_a)$ for each $(a, s) \in \Sigma \times S$ defines a morphism $f \colon \mathrm{Tr}(\Sigma \times S) \to \mathrm{Tr}(\Sigma \times_\ell S)$. We then have $\chi_A(t) = f(\zeta_A(t))$ for each $t \in \mathrm{Tr}(\Sigma)$.

Example 6.4. Figure 7 shows the image by the global-state labelling function ζ, for the same asynchronous automaton A (or asynchronous morphism φ) and trace t as in Example 3.18. Note the difference from Figure 4. For example, here the p_3-event has process p_1 state 2 in its label (which is the best process p_1 state in its causal past) even though process p_1 and process p_3 never interact directly.

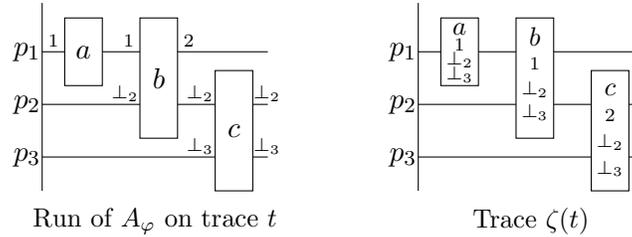

FIGURE 7. Global-state labelling function output on a trace

As $\zeta_A(t)$ carries more information than $\chi_A(t)$ (Remark 6.3), one can view ζ_A as an information-theoretic generalization of χ_A. However, unlike χ_A, it is not clear a priori whether it can be computed by an asychronous automaton. We will return to this important issue in Section 7. At the moment, we simply extend the operational point of view of local cascade products (see Figure 5) using global-state labelling functions instead of asynchronous transducers.

Definition 6.5. A *global cascade sequence* (in short, gcs) A_{seq} is a sequence $(A_1, A_2, \ldots, A_n)$ of asynchronous automata such that, for $1 \leq i < n$, the input alphabet of A_{i+1} is the output alphabet of A_i. The input alphabet of A_1 is called the input alphabet of A_{seq} and the output alphabet of A_n is called the output alphabet of A_{seq}.

We associate a global-state labelling function $\zeta_{A_{\mathrm{seq}}}$ from traces over the input alphabet of A_{seq} to traces over the output alphabet of A_{seq}, namely the composition

$$\zeta_{A_{\mathrm{seq}}} = \zeta_{A_1} \zeta_{A_2} \cdots \zeta_{A_n}$$

of the global-state labelling functions of the A_i. For instance, if $A_{\mathrm{seq}} = (A_1, A_2)$ then $\zeta_{A_{\mathrm{seq}}}(t) = \zeta_{A_2}(\zeta_{A_1}(t))$.

It is important to observe that a gcs A_{seq} is *not* an asychronous automaton. A gcs is simply a cascade of a sequence of *compatible* automata which are *connected* via global-state labelling mechanisms. The following lemmas are immediate.
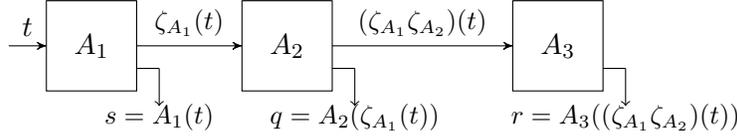
FIGURE 8. Global cascade product

Lemma 6.6. *Let $A_{\mathrm{seq}} = (A_1, A_2, \ldots, A_n)$ and $B_{\mathrm{seq}} = (B_1, B_2, \ldots, B_m)$ be two global cascade sequences such that the input alphabet of B_{seq} is the output alphabet of A_{seq}. Then $C_{\mathrm{seq}} = (A_1, \ldots, A_n, B_1, \ldots, B_m)$ is a valid global cascade sequence. Moreover, $\zeta_{C_{\mathrm{seq}}} = \zeta_{A_{\mathrm{seq}}} \zeta_{B_{\mathrm{seq}}}$.*

The concatenation of global cascade sequences, as in Lemma 6.6, is denoted by $C_{\mathrm{seq}} = A_{\mathrm{seq}} \cdot B_{\mathrm{seq}}$. This is an associative operation, as verified in the following lemma.

Lemma 6.7. *Let A_{seq}, B_{seq}, C_{seq} be global cascade sequences such that the input alphabet of B_{seq} is the output alphabet of A_{seq}, and the input alphabet of C_{seq} is the output alphabet of B_{seq}. Then $(A_{\mathrm{seq}} \cdot B_{\mathrm{seq}}) \cdot C_{\mathrm{seq}} = A_{\mathrm{seq}} \cdot (B_{\mathrm{seq}} \cdot C_{\mathrm{seq}})$ and $\zeta_{(A_{\mathrm{seq}} \cdot B_{\mathrm{seq}})} \zeta_{C_{\mathrm{seq}}} = \zeta_{A_{\mathrm{seq}}} \zeta_{(B_{\mathrm{seq}} \cdot C_{\mathrm{seq}})}$.*

We now show that, one can view a gcs as an acceptor of trace languages in a natural way. We begin with some notation.

For an automaton A, the set of global states of A is denoted by $\mathrm{gs}(A)$. Given a global cascade sequence $A_{\mathrm{seq}} = (A_1, \ldots, A_n)$, we refer to the set $\mathrm{gs}(A_1) \times \ldots \times \mathrm{gs}(A_n)$ as the *global states* of A_{seq} and denote it as $\mathrm{gs}(A_{\mathrm{seq}})$. Similarly the set of P-states of A_{seq} is the cartesian product of the sets of P-states of its constituent asynchronous automata. Given a P-state $s = (s_1, \ldots, s_n)$ of A_{seq}, and $P' \subseteq P$, we let $s_{P'} = ((s_1)_{P'}, \ldots, (s_n)_{P'})$ be the natural restriction of s to P'.

Definition 6.8 (Language accepted by a global cascade sequence). Let $A_{\mathrm{seq}} = (A_1, \ldots, A_n)$ be a global cascade sequence with input alphabet (Σ, loc). Given a trace $t \in \mathrm{Tr}(\Sigma)$, we let $A_{\mathrm{seq}}(t) \in \mathrm{gs}(A_{\mathrm{seq}})$ be the global state of A_{seq} reached after reading t:

$$A_{\mathrm{seq}}(t) = \big(A_1(t), A_2(\zeta_{A_1}(t)), \ldots, A_n([\zeta_{A_1} \cdots \zeta_{A_{n-1}}](t))\big) .$$

Given $F \subseteq \mathrm{gs}(A_{\mathrm{seq}})$, we define the language $L(A_{\mathrm{seq}}, F)$, a language accepted by A_{seq}, by

$$L(A_{\mathrm{seq}}, F) = \{t \in \mathrm{Tr}(\Sigma) \mid A_{\mathrm{seq}}(t) \in F\} .$$

A language $L \subseteq \mathrm{Tr}(\Sigma)$ is said to be *accepted by* A_{seq} if there exists a subset $F \subseteq \mathrm{gs}(A_{\mathrm{seq}})$ such that $L = L(A_{\mathrm{seq}}, F)$. See the Figure 8.

The following *global cascade sequence principle* is an easy consequence of the definitions.

Theorem 6.9. *Let A_{seq} and B_{seq} be global cascade sequences, let (Σ, loc) be the input alphabet of A_{seq}, and suppose that the output alphabet of A_{seq} is the input alphabet of B_{seq}, say (Π, loc). Let $C_{\mathrm{seq}} = A_{\mathrm{seq}} \cdot B_{\mathrm{seq}}$. Then any language $L \subseteq \mathrm{Tr}(\Sigma)$ accepted by C_{seq} is a finite union of languages of the form $U \cap \zeta_{A_{\mathrm{seq}}}^{-1}(V)$ where $U \subseteq \mathrm{Tr}(\Sigma)$ is accepted by A_{seq}, and $V \subseteq \mathrm{Tr}(\Pi)$ is accepted by B_{seq}.*

Building on the simple observation in Remark 6.3 that global-state labelling functions are information-theoretic generalizations of local asynchronous transducers, we now show that a local cascade product can be realized by an appropriate global cascade sequence.

An asynchronous automaton B over $(\Sigma \times_\ell S, \mathrm{loc})$ naturally gives rise to another asynchronous automaton $\widehat{B}$ (with the same state sets, etc.) operating over $(\Sigma \times S, \mathrm{loc})$ by defining

the transition of $\widehat{B}$ on letter $(a, s) \in \Sigma \times S$ to be the transition of B on letter (a, s_a). We abuse the notation slightly in the following lemma and denote $\widehat{B}$ also by B.

Lemma 6.10. *The action of a local cascade product $A = A_1 \circ_\ell \ldots \circ_\ell A_n$ can be simulated by the gcs $A_{\mathrm{seq}} = (A_1, \ldots, A_n)$ in the following sense: for any trace t, and any process i, $[A(t)]_i = [A_{\mathrm{seq}}(t)]_i$.*

Proof. To be completely rigorous, the gcs in question is $A_{\mathrm{seq}} = (A_1, \hat{A}_2, \ldots, \hat{A}_n)$. We prove the lemma by induction on n. If $n = 1$, the statement is trivially true.

For the inductive step, we assume that the lemma holds for $A' = A_1 \circ_\ell \ldots \circ_\ell A_{n-1}$, and $A'_{\mathrm{seq}} = (A_1, \hat{A}_2, \ldots, \hat{A}_{n-1})$. Fix a trace t. For any event e in the trace and any process i, by induction hypothesis, $[A'(\Downarrow e)]_i = [A'_{\mathrm{seq}}(\Downarrow e)]_i$. Hence if the label of the event e in $\zeta_{A'_{\mathrm{seq}}}(t)$ is (a, s) (for some $s \in \mathrm{gs}(A'_{\mathrm{seq}})$), then the label of the *same* event in $\chi_{A'}(t)$ corresponds to (a, s_a) (see Remark 6.3). By construction, transition on (a, s_a) in the local cascade product component A_n is same as that by (a, s) in the corresponding global cascade sequence component $\hat{A}_n$. Hence there is a natural correspondence between the run of A_n (in A) over $\chi_{A'}(t)$ and the run of $\hat{A}_n$ (in A_{seq}) over $\zeta_{A'_{\mathrm{seq}}}(t)$, and we can conclude that $[A(t)]_i = [A_{\mathrm{seq}}(t)]_i$. $\square$

An important language-theoretic consequence of Lemma 6.10 is that every language accepted by $A = A_1 \circ_\ell \ldots \circ_\ell A_n$ is also accepted by the gcs $A_{\mathrm{seq}} = (A_1, \ldots, A_n)$.

We now come to the main result of this section, that relates the logic $\mathsf{LocTL}[\mathsf{Y}_i, \mathsf{S}_i]$ with global cascade sequences of localized reset automata $U_2[p]$. Recall that Theorem 5.7 gives an exact correspondence between first order definable trace languages and local cascade products of $U_2[p]$ asynchronous automata for aKR distributed alphabets. We generalize this language-theoretic decomposition result to any distributed alphabet, using a global cascade sequence of the same distributed resets instead of local cascade product.

Theorem 6.11. *A trace language is defined by a $\mathsf{LocTL}[\mathsf{Y}_i, \mathsf{S}_i]$ formula if and only if it is accepted by a global cascade sequence of asynchronous reset automata of the form $U_2[p]$.*

The proof of Theorem 6.11 follows the same structure and re-uses elements of the proof of Theorem 5.6.

Proof. First consider a global cascade sequence $A = U_2[p] \cdot B$ where B is a global cascade sequence. Recall that $U_2[p]$ has two global states, say $S = \{1, 2\}$, identified with the two local states of its p-component. The input alphabet of the gcs B is $(\Sigma \times S, \mathrm{loc})$. Suppose that the languages accepted by B are $\mathsf{LocTL}[\mathsf{Y}_i, \mathsf{S}_i]$-definable over $(\Sigma \times S, \mathrm{loc})$.

Let $\zeta \colon \mathrm{Tr}(\Sigma) \to \mathrm{Tr}(\Sigma \times S)$ be the global-state labelling function associated with $U_2[p]$ and its initial state, say, 1. By the global cascade principle (Theorem 6.9), any language recognized by A is a union of languages of the form $L_1 \cap \zeta^{-1}(L_2)$ where $L_1 \subseteq \mathrm{Tr}(\Sigma)$ is recognized by $U_2[p]$, and $L_2 \subseteq \mathrm{Tr}(\Sigma \times S)$ is recognized by B. We have seen in the proof of Theorem 5.6 that L_1 is $\mathsf{LocTL}[\mathsf{S}_i]$ definable over alphabet (Σ, loc).

By assumption, we know that L_2 is $\mathsf{LocTL}[\mathsf{Y}_i, \mathsf{S}_i]$ definable over alphabet $(\Sigma \times S, \mathrm{loc})$ and we need to prove that $\zeta^{-1}(L_2)$ is $\mathsf{LocTL}[\mathsf{Y}_i, \mathsf{S}_i]$ definable over (Σ, loc). This is done by structural induction on the $\mathsf{LocTL}[\mathsf{Y}_i, \mathsf{S}_i]$-formula over $(\Sigma \times S, \mathrm{loc})$ defining L_2. For a $\mathsf{LocTL}[\mathsf{Y}_i, \mathsf{S}_i]$ event formula α over $(\Sigma \times S, \mathrm{loc})$, we construct a $\mathsf{LocTL}[\mathsf{Y}_i, \mathsf{S}_i]$ event formula $\hat{\alpha}$ over (Σ, loc) such that for any trace $t \in \mathrm{Tr}(\Sigma)$ and any event e in t, we have $t, e \models \hat{\alpha}$ if and only if $\zeta(t), e \models \alpha$. The non-trivial case here is the base case of a letter formula from

$(\Sigma \times S, \mathrm{loc})$. We let

$$\widehat{(a,2)} = a \wedge \mathsf{Y}_p(R_2 \vee (\neg R_1 \wedge ((\neg R_1)\, \mathsf{S}_p\, R_2)))$$

$$\widehat{(a,1)} = a \wedge \neg \mathsf{Y}_p(R_2 \vee (\neg R_1 \wedge ((\neg R_1)\, \mathsf{S}_p\, R_2)))\,.$$

The inductive cases are trivial, for instance $\widehat{\mathsf{Y}_i\,\alpha} = \mathsf{Y}_i\,\hat\alpha$.

Towards establishing the converse implication, we construct, for any $\mathsf{LocTL}[\mathsf{Y}_i, \mathsf{S}_i]$ event formula α, a gcs A_α from reset asynchronous automata of the form $U_2[p]$, which is such that for any trace t, event e of t and process $i \in \mathrm{loc}(e)$, the local state $[A_\alpha(\downarrow e)]_i$ determines whether $t, e \models \alpha$. Again, this is done by structural induction on the $\mathsf{LocTL}[\mathsf{Y}_i, \mathsf{S}_i]$ event formula α. The case of $\mathsf{LocTL}[\mathsf{S}_i]$-formulas, in view of Lemma 6.10, is already handled in (the proof of) Theorem 5.6 and we only need to deal with the inductive case where α is of the form $\alpha = \mathsf{Y}_j\,\beta$.

By induction, a gcs of reset automata A_β has been constructed, which provides the truth value of β at any event. We construct an asynchronous automaton $B = (\{Q_i\}, \{\delta_{(a,s_a)}\}, q_{\mathrm{in}})$ over $(\Sigma \times S, \mathrm{loc})$ such that $A_\alpha = A_\beta \cdot U_2[j] \cdot B$ as follows. The middle $U_2[j]$ remembers whether some j-event already occured. Concerning B, we let $Q_i = \{\top, \bot\}$ for all $i \in \mathscr{P}$ and, again, we denote a P-state q as $\bot$ (resp. $\top$) if $q_i = \bot$ (resp. $q_i = \top$) for all $i \in P$. We let the initial state be $q_{\mathrm{in}} = \bot$. Let ζ be the global-state labelling function associated with $A_\beta \cdot U_2[j]$. Let $t \in \mathrm{Tr}(\Sigma)$ be a trace, e be an event in t, and (a, s) be the label of e in $\zeta(t)$. Write e_j the last j-event in $\Downarrow e$ if it exists, i.e., if $\Downarrow e \cap E_j \neq \emptyset$. The local state s_j determines whether e_j exists, written $s_j \vdash \mathsf{Y}_j\,\top$, and in this case whether it satisfies β, written $s_j \vdash \mathsf{Y}_j\,\beta$. The transition functions of B are:

$$\delta_{(a,s)} = \text{reset to } \top \qquad\qquad \text{if } s_j \vdash \mathsf{Y}_j\,\beta$$

$$\delta_{(a,s)} = \text{reset to } \bot \qquad\qquad \text{if } s_j \not\vdash \mathsf{Y}_j\,\beta\,.$$

It is easy to see that B is a gcs of copies of $U_2[p]$, one for each process $p \in \mathscr{P}$. These reset automata work independently of each other, each $U_2[p]$ depends only on the global state information from $A_\beta \cdot U_2[j]$ provided by ζ. Hence, B is also a local cascade product of these $U_2[p]$. This completes the proof. $\qquad\square$

Our subsequent result crucially uses the expressive completeness of $\mathsf{LocTL}[\mathsf{Y}_i, \mathsf{S}_i]$ and its proof is immediate from Theorems 5.5 and 6.11. It is best seen as an addition to the several characterizations of first-order definable trace languages presented in Theorem 5.14.

Theorem 6.12. *Let (Σ, loc) be a distributed alphabet and let $L \subseteq \mathrm{Tr}(\Sigma)$ be a trace language. Then L is definable in first-order logic if and only if L is accepted by a global cascade sequence of asynchronous reset automata of the form $U_2[p]$.*

7. Asynchronous implementation of a global cascade sequence

We have already noted in Section 6 that the global-state labelling function ζ_A associated with an asynchronous automaton A is not an abstraction of the asynchronous transducer of A, that is, it cannot be directly computed by A. However, we will show how to construct an asynchronous automaton $A^{\mathcal{G}}$ which computes ζ_A using the *gossip automaton*.

Recall that the gossip automaton $\mathcal{G} = (\{\Upsilon_i\}, \{\nabla_a\}, v_{\mathrm{in}})$ keeps track of the truth values all the constants of $\mathsf{LocTL}[\mathsf{Y}_i \leq \mathsf{Y}_j, \mathsf{S}_i]$: it computes the primary order labelling function

$\theta^{\mathsf{Y}} \colon \mathrm{Tr}(\Sigma) \to \mathrm{Tr}(\Sigma \times \Gamma)$ which decorates each event e of a trace t with the truth values $\gamma \in \Gamma = \{0,1\}^{\mathscr{P} \times \mathscr{P}}$ of all constants $\mathsf{Y}_i \le \mathsf{Y}_j$, i.e., for all $i,j \in \mathscr{P}$, $\gamma_{i,j} = 1$ if $t, e \models \mathsf{Y}_i \le \mathsf{Y}_j$ and $\gamma_{i,j} = 0$ otherwise.

Construction of $A^{\mathcal{G}}$. Roughly speaking, in the automaton $A^{\mathcal{G}}$, each process keeps track of its local gossip state in the gossip automaton and the best global state of A that it is aware of. In fact, $A^{\mathcal{G}}$ is realised as a θ^{Y}-restricted local cascade product of the gossip automaton and an asynchronous automaton A^g – the *global-state detector* – derived from A where each process in A^g keeps track of the best global state of A that it is aware of. When processes synchronize in A^g, they use the Γ-labelling information to correctly update the best global state that they are aware of at the synchronizing event.

So $A^{\mathcal{G}} = \mathcal{G} \circ_\ell^r A^g$ where A^g is an asynchronous automaton over $(\Sigma \times \Gamma, \mathrm{loc})$ derived from A. Therefore, for the construction, we only need to describe A^g. Recall that $A = (\{S_i\}, \{\delta_a\}, s_{\mathrm{in}})$. Then $A^g = (\{Q_i\}, \{\delta_{(a,\gamma)}\}, q_{\mathrm{in}})$ where $Q_i = S$ for all $i \in \mathscr{P}$, and $q_{\mathrm{in}} = (s_{\mathrm{in}}, \dots, s_{\mathrm{in}})$. Before defining the transitions, we define for $(a, \gamma) \in \Sigma \times \Gamma$ the function $\mathsf{globalstate}_{(a,\gamma)} \colon Q_a \to S$ as follows: $\mathsf{globalstate}_{(a,\gamma)}(q_a) = s \in S$ where, for each $i \in \mathscr{P}$,

$$s(i) = \begin{cases} q_a(j)(i) & \text{if there exists } j \in \mathrm{loc}(a) \text{ such that } \gamma_{i,j} = 1 \\ s_{\mathrm{in}}(i) & \text{otherwise.} \end{cases}$$

Note that, when $\gamma_{i,j} = 1$ at some event e of a trace $\theta^{\mathsf{Y}}(t)$, then $t, e \models \mathsf{Y}_i \le \mathsf{Y}_j$ and process j has the latest information about process i. Hence, the function $\mathsf{globalstate}_{(a,\gamma)}$ determines the best global-state that processes in $\mathrm{loc}(a)$ are collectively aware of. We define the local transition functions of A^g by $\delta_{(a,\gamma)}(q_a) = q_a'$ where for all $i \in \mathrm{loc}(a)$ we set

$$q_a'(i) = \Delta_a(\mathsf{globalstate}_{(a,\gamma)}(q_a)).$$

Recall that Δ_a is the extension of the local transition function δ_a of A to global states in S.

In order to prove the correctness of the construction, we first introduce some notation. Let $t = (E, \le, \lambda) \in \mathrm{Tr}(\Sigma)$, and $i \in \mathscr{P}$. Then $\downarrow^i(t)$ is the *i-view* of t and it is defined by $\downarrow^i(t) = \downarrow E_i$. It is easy to see that if $\downarrow^i(t) \ne \emptyset$, then there exists $e \in E_i$ such that $\downarrow^i(t) = \downarrow e$. We note that $\downarrow^i(t)$ is a trace prefix of t and it represents knowledge of the agent i about t.

The next lemma shows that in $A^{\mathcal{G}}$, each process keeps track of the best global state of A that it is aware of.

Lemma 7.1. *Let $t \in \mathrm{Tr}(\Sigma)$ with $A^{\mathcal{G}}(t) = (v, q)$. Then for every $i \in \mathscr{P}$, $q_i = A(\downarrow^i(t))$.*

Proof. Note that as $A^{\mathcal{G}} = \mathcal{G} \circ_\ell^r A^g$, $A^{\mathcal{G}}(t) = (v, q)$ implies that $\mathcal{G}(t) = v$ and $A^g(\theta^{\mathsf{Y}}(t)) = q$. We prove the lemma by induction on the size of $t = (E, \le, \lambda)$, that is, on $|E|$. The base case of the empty trace is easy and skipped.

Consider $t' = ta$, $\theta^{\mathsf{Y}}(t') = \theta^{\mathsf{Y}}(t)(a, \gamma)$ and let $(v, q) = A^{\mathcal{G}}(t)$ and $(v', q') = A^{\mathcal{G}}(t')$. Clearly $A^g(\theta^{\mathsf{Y}}(t)) = q$, $A^g(\theta^{\mathsf{Y}}(t')) = q'$ and $\delta_{(a,\gamma)}(q_a) = q_a'$. By definition of the (a, γ)-transition function of A^g, we have $q_i' = \Delta_a(\mathsf{globalstate}_{(a,\gamma)}(q_a))$ for each $i \in \mathrm{loc}(a)$. Observe that, by the local nature of a-transition functions of asynchronous automata, we have $q_i' = q_i$ for $i \notin \mathrm{loc}(a)$.

By induction, for every $i \in \mathscr{P}$, $q_i = A(\downarrow^i(t))$. If $i \notin \mathrm{loc}(a)$, $\downarrow^i(t') = \downarrow^i(t)$. As, in this case, we also have $q_i' = q_i$, we are done by the induction hypothesis. Now we let $i \in \mathrm{loc}(a)$. Let e correspond to the last occurrence of a in t'. Then process i participates in e. As a result, $\downarrow^i(t') = \downarrow e$.

We first study the global state $s = A(\Downarrow e) \in S$. For $i \in \mathscr{P}$, let e_i be the maximal i-event in $\Downarrow e$ if it exists, i.e., if $\Downarrow e \cap E_i \neq \emptyset$. Fix a process $i \in \mathscr{P}$. If e_i does not exist, then $s(i) = s_{\text{in}}(i)$. Otherwise there exists $j \in \text{loc}(a)$ such that $e_i \leq e_j$. Therefore $s(i) = [A(\Downarrow e)]_i = [A(\downarrow^j(t))]_i = q_j(i)$ where the last equality is obtained using the induction hypothesis, $q_j = A(\downarrow^j(t))$. From Theorem 5.9, we have $\gamma_{i,j} = 1$ (as $\theta^{\mathsf{Y}}(t') = \theta^{\mathsf{Y}}(t)(a,\gamma)$) and, using the definition of the function $\mathsf{globalstate}_{(a,\gamma)}$, it follows that $s = \mathsf{globalstate}_{(a,\gamma)}(q_a)$.

Now, with $s' = A(\downarrow e)$, we see that $s' = \Delta_a(s) = \Delta_a(\mathsf{globalstate}_{(a,\gamma)}(q_a))$. As already observed, $q'_i = \Delta_a(\mathsf{globalstate}_{(a,\gamma)}(q_a))$, for $i \in \text{loc}(a)$. The proof of the inductive step is now complete, as for $i \in \text{loc}(a)$, $q'_i = s' = A(\downarrow e) = A(\downarrow^i(t'))$. $\qquad\square$

We now show that the asynchronous automaton $A^{\mathcal{G}}$ simulates A. More precisely, we define a simulation map $f \colon \Upsilon \times Q \to S$ by $f(v,q) = s$ where, for $i \in \mathscr{P}$, $s(i) = q_i(i)$.

Lemma 7.2. *The action of A can be simulated by $A^{\mathcal{G}}$ as, for $t \in \text{Tr}(\Sigma)$, $A(t) = f(A^{\mathcal{G}}(t))$.*

Proof. Let $t \in \text{Tr}(\Sigma)$, $A(t) = s$ and $A^{\mathcal{G}}(t) = (v,q)$. By Lemma 7.1, for each $i \in \mathscr{P}$, we have $q_i = A(\downarrow^i(t))$ and hence, $q_i(i) = [A(\downarrow^i(t))]_i = s(i)$. This shows that $A(t) = s = f(A^{\mathcal{G}}(t))$. $\quad\square$

An important consequence of Lemma 7.2 is that any language accepted by A is also accepted by $A^{\mathcal{G}}$: if L is accepted by A via $F \subseteq S$ then it is accepted by $A^{\mathcal{G}}$ via $f^{-1}(F)$. Note that, as $f^{-1}(F) = \Upsilon \times F'$ where $F' \subset \text{gs}(A^g)$, the accepting set $f^{-1}(F)$ is solely determined by F' and does *not* depend on the state reached by the gossip automaton.

Finally, we establish that $A^{\mathcal{G}}$ computes the global-state labelling function ζ_A. For this we fix the Γ-transducer $\widehat{\mathcal{G}} = (\{\Upsilon_i\}, \{\nabla_a\}, v_{\text{in}}, \{\nu_a^{\mathsf{Y}} \colon \Upsilon_a \to \Gamma\})$ which computes θ^{Y}. Now we are ready to extend $A^{\mathcal{G}}$ to a S-transducer $\hat{A}^{\mathcal{G}} = (A^{\mathcal{G}}, \{\xi_a\})$. We define $\xi_a \colon \Upsilon_a \times Q_a \to S$ as follows: $\xi_a(v_a, q_a) = \mathsf{globalstate}_{(a, \nu_a^{\mathsf{Y}}(v_a))}(q_a)$.

Theorem 7.3. *The transducer $\hat{A}^{\mathcal{G}}$ computes the global-state labelling function ζ_A.*

Proof. Let $t = (E, \leq, \lambda) \in \text{Tr}(\Sigma)$ and $t' = \theta^{\mathsf{Y}}(t) = (E, \leq, (\lambda, \mu)) \in TR((\Sigma \times \Gamma), \text{loc})$. Fix an event $e \in E$ with $\lambda(e) = a$ and $\mu(e) = \gamma$. Note that, ζ_A decorates the event e with the additional information $s = A(\Downarrow e)$. On the other hand, with $A^{\mathcal{G}}(t) = (v, q)$, the transducer $\hat{A}^{\mathcal{G}}$ decorates the event e with the additional information $\xi_a(v_a, q_a) = \mathsf{globalstate}_{(a, \nu_a^{\mathsf{Y}}(v_a))}(q_a)$.

By Theorem 5.9, $\hat{\mathcal{G}}$ computes θ^{Y} and we have $\nu_a^{\mathsf{Y}}(v_a) = \gamma$. Therefore it remains to show that $s = \mathsf{globalstate}_{(a,\gamma)}(q_a)$ to complete the proof. But we have already seen in the proof of Lemma 7.1 (inductive step applied with $t = \Downarrow e$ and $t' = \downarrow e$) that $s = \mathsf{globalstate}_{(a,\gamma)}(q_a)$. $\quad\square$

Now we are ready to realize a global cascade sequence as an asynchronous automaton. We associate with a gcs $A_{\text{seq}} = (A_1, \ldots, A_n)$ an asychronous automaton $A_{\text{seq}}^{\mathcal{G}}$ and an asynchronous transducer $\hat{A}_{\text{seq}}^{\mathcal{G}}$, whose constructions extend the constructions of $A^{\mathcal{G}}$ and $\hat{A}^{\mathcal{G}}$ from A in a natural fashion. In particular, $A_{\text{seq}}^{\mathcal{G}}$ is a local cascade product $\mathcal{G} \circ_\ell^r (A_1^g \circ_\ell \ldots \circ_\ell A_n^g)$. The first *component* in this product is the gossip automaton which computes θ^{Y} and the first cascade product is θ^{Y}-restricted. In the subsequent components, each process keeps track of the best global state it is aware of for the corresponding automaton in A_{seq}. In other words, we use the earlier construction for each component automaton but keep a single copy of the gossip automaton.

Let (Σ, loc) be the input (distributed) alphabet of A_1 (or that of A_{seq}) with total alphabet Σ. Note that the input alphabet for A_j is the output alphabet of A_{j-1}. We abuse the notation and use the total alphabet instead of the distributed alphabet. The

distribution is anyway induced from the alphabet (Σ, loc). The input alphabet for A_j is $\Sigma \times \text{gs}(A_1) \times \cdots \times \text{gs}(A_{j-1})$.

Remark 7.4. It is important to keep in mind the special case $A_{\text{seq}} = (A)$. Our constructions of $A_{\text{seq}}^{\mathcal{G}}$ and $\hat{A}_{\text{seq}}^{\mathcal{G}}$ in this case are exactly identical to those of $A^{\mathcal{G}}$ and $\hat{A}^{\mathcal{G}}$. The general case of a gcs is only notationally more involved. It merely follows the constructions of $A^{\mathcal{G}}$ and $\hat{A}^{\mathcal{G}}$ in spirit and extends them naturally.

In order to describe the complete construction, we need to define $A_1^g, \ldots, A_n^g$. For brevity, we simply define A_1^g and A_2^g here and skip the remaining details.

Let $A_1 = (\{S_i\}, \{\delta_a\}, s_{\text{in}})$ and $A_2 = (\{S_i'\}, \{\delta_{(a,s)}'\}, s_{\text{in}}')$. The definition of A_1^g is verbatim to that of A^g defined earlier. In particular, A_1^g is defined over $(\Sigma \times \Gamma, \text{loc})$ and $A_1^g = (\{Q_i\}, \{\delta_{(a,\gamma)}\}, q_{\text{in}})$ where $Q_i = S$ for all $i \in \mathscr{P}$, and $q_{\text{in}} = (s_{\text{in}}, \ldots, s_{\text{in}})$. Further, we have defined the function $\text{globalstate}_{(a,\gamma)} : Q_a \to S$ there which is used to construct the local transition functions of A_1^g by $\delta_{(a,\gamma)}(q_a) = q_a'$ where for all $i \in \text{loc}(a)$ we have $q_a'(i) = \Delta_a(\text{globalstate}_{(a,\gamma)}(q_a))$.

Now we set $A_2^g = (\{Q_i'\}, \{\delta_{(a,\gamma,q_a)}'\}, q_{\text{in}}')$ where, for all $i \in \mathscr{P}$, $Q_i' = S'$ is the set of global states of A_2 and $q_{\text{in}}' = (s_{\text{in}}', \ldots, s_{\text{in}}')$. As expected, we use the 'same' function $\text{globalstate}_{(a,\gamma)}' : Q_a' \to S'$ defined by $\text{globalstate}_{(a,\gamma)}'(q_a') = s' \in S'$ where, for each $i \in \mathscr{P}$,

$$s'(i) = \begin{cases} q_a'(j)(i) & \text{if } \exists j \in \text{loc}(a) \text{ such that } \gamma_{i,j} = 1 \\ s_{\text{in}}'(i) & \text{otherwise.} \end{cases}$$

Now we define the local transitions of A_2^g by $\delta_{(a,\gamma,q_a)}(q_a') = q_a''$ where for all $i \in \text{loc}(a)$ we have

$$q_a''(i) = \Delta_{(a,\text{globalstate}_{(a,\gamma)}(q_a))}'(\text{globalstate}_{(a,\gamma)}'(q_a')) \,.$$

Recall that $\Delta_{(a,s=\text{globalstate}_{(a,\gamma)}(q_a))}'$ is the extension of the local transition function $\delta_{(a,s)}'$ of A_2 to global states in S'.

In general, in A_p^g – the global-state detector of A_p, the local state-set of each process is simply the set $\text{gs}(A_p)$ of global states of A_p and each process starts in the initial global-state of A_p. As mentioned earlier, each process in A_p^g simply records the best global state of A_p that it is aware of. At a synchronizing event, the participating processes simply use the Γ-labelling information γ to correctly compute, using the $\text{globalstate}_{(a,\gamma)}$ function, the best *collective* global-state of A_p that they are aware of just prior to the synchronization. These participating processes from the earlier components A_q^g for $q < p$ can also similarly compute best collective global-state of A_q prior to the *current* synchronization that they are aware of and make it available to A_p^g via the local-cascade mechanism. This allows A_p^g to correctly simulate the operational global-cascade mechanism used by the component A_p in A_{seq}.

Now we simply describe the final form of $A_{\text{seq}}^{\mathcal{G}} = \mathcal{G} \circ_{\ell}^r (A_1^g \circ_{\ell} \ldots \circ_{\ell} A_n^g)$. It is easy to see that, $\text{gs}(A_{\text{seq}}^{\mathcal{G}}) = \Upsilon_{\mathscr{P}} \times \text{gs}(A_1^g) \times \ldots \times \text{gs}(A_n^g) = \Upsilon_{\mathscr{P}} \times \text{gs}(A_1)^{\mathscr{P}} \times \ldots \times \text{gs}(A_n)^{\mathscr{P}}$. As a result, we write a global state of $A_{\text{seq}}^{\mathcal{G}}$ as $(\upsilon, q_1, \ldots, q_n)$ where $\upsilon \in \Upsilon_{\mathscr{P}}$ and $q_j \in \text{gs}(A_j)^{\mathscr{P}} = \prod_{i \in \mathscr{P}} \text{gs}(A_j)$.

We next exhibit a simulation of the global cascade sequence A_{seq} by the asynchronous automaton $A_{\text{seq}}^{\mathcal{G}}$. We define $f_{\text{seq}} : \text{gs}(A_{\text{seq}}^{\mathcal{G}}) \to \text{gs}(A_{\text{seq}})$ as follows: $f_{\text{seq}}((\upsilon, q_1, q_2, \ldots, q_n)) = (s_1, s_2, \ldots, s_n)$ where $s_p(i) = q_p(i)(i)$ for each $1 \leq p \leq n$ and each $i \in \mathscr{P}$.

Finally, we enrich $A_{\text{seq}}^{\mathcal{G}}$ to get the asynchronous $\text{gs}(A_{\text{seq}})$-transducer $\hat{A}_{\text{seq}}^{\mathcal{G}} = (A_{\text{seq}}^{\mathcal{G}}, \{\xi_a\})$ over (Σ, loc) which computes the global-state labelling function $\zeta_{A_{\text{seq}}}$. For each $a \in \Sigma$, we

define ξ_a from a-states of $A^{\mathcal{G}}_{\mathrm{seq}}$ as follows: with $\gamma = \nu^{\mathsf{Y}}_a(v_a)$,

$$\xi_a(v_a, q^a_1, \ldots, q^a_n) = (\mathsf{globalstate}_{(a,\gamma)}(q^a_1), \ldots, \mathsf{globalstate}_{(a,\gamma)}(q^a_n))$$

In view of Remark 7.4, our next set of results regarding the correctness of $A^{\mathcal{G}}_{\mathrm{seq}}$ and $\hat{A}^{\mathcal{G}}_{\mathrm{seq}}$ is hardly surprising. Their proofs are almost verbatim duplicates of the proofs of the corresponding results about $A^{\mathcal{G}}$ and $\hat{A}^{\mathcal{G}}$, differing only in the bookkeeping notation needed to talk about a sequence.

Lemma 7.5. *Let $t \in \mathrm{Tr}(\Sigma)$ with $A^{\mathcal{G}}_{\mathrm{seq}}(t) = (v, q_1, \ldots, q_n)$. Then, for every $i \in \mathscr{P}$, $A_{\mathrm{seq}}(\downarrow^i(t)) = (q_1(i), \ldots, q_n(i))$.*

Lemma 7.6. *The action of $A_{\mathrm{seq}} = (A_1, \ldots, A_n)$ is simulated by $A^{\mathcal{G}}_{\mathrm{seq}}$ in the following sense: for $t \in \mathrm{Tr}(\Sigma)$, $A_{\mathrm{seq}}(t) = f_{\mathrm{seq}}(A^{\mathcal{G}}_{\mathrm{seq}}(t))$.*

Lemma 7.6 implies that every language accepted by A_{seq} is also accepted by $A^{\mathcal{G}}_{\mathrm{seq}}$.

Theorem 7.7. *The transducer $\hat{A}^{\mathcal{G}}_{\mathrm{seq}}$ computes the global-state labelling function $\zeta_{A_{\mathrm{seq}}}$.*

8. In lieu of a conclusion

An intriguing question, first asked by Adsul and Sohoni [AS04] and revived by the results in this paper, is the following. Zielonka's theorem [Zie87] states that every trace language that is accepted by a (diamond) automaton, is accepted by an asynchronous automaton. Equivalently, every trace language that is recognized by a morphism to a tm, is recognized by an asynchronous morphism to an atm. Ebinger and Muscholl's theorem [EM96] also states that first-order definable trace languages are exactly those that are recognized by an aperiodic tm. In view of the importance of first-order definability — and of the vast literature concerning that class of languages, it would be interesting to know whether an *aperiodic Zielonka* theorem holds, or for which distributed alphabets it does. Such a theorem would state that the first-order definable trace languages are exactly those that are recognized by an asynchronous morphism to an aperiodic atm.

The asynchronous Krohn-Rhodes property introduced in Section 3.7 is stronger: over aKR distributed alphabets, a trace language L recognized by a tm (X, M) is also recognized by a local cascade product of localized reset automata and localized permutation automata of the form $G[p]$, where G is a simple group dividing M. If L is first-order definable, the tm can be chosen such that M is aperiodic, so L is accepted by a local cascade product of localized reset automata, and hence recognized by an aperiodic atm. It is conceivable that certain distributed alphabets would have the aperiodic Zielonka property without being aKR. It is also conceivable that certain distributed alphabets would fail to be aKR, yet would have that property when restricted to first-order definable languages, that is, to languages recognized by an aperiodic tm (a property that we term *aperiodic aKR*).

We showed in Section 4 that acyclic architectures have the stronger aKR property. Apart from that, we do not know which distributed alphabets have the aperiodic Zielonka or the asynchronous Krohn-Rhodes property, nor even whether all distributed alphabets have these properties. However, the results in Section 5 have the following consequence. For a fixed distributed alphabet (Σ, loc), recall (from Section 5) that θ^{Y} is the primary order labelling function which decorates every event of a trace t with the truth values of the constants $Y_i \leq Y_j$. It follows from Remark 5.13 and Theorem 5.14 that if, for some distributed alphabet

(Σ, loc), the function θ^{Y} can be computed by an aperiodic asynchronous automaton (resp. by an asynchronous wreath product of asynchronous transformation monoids of the form $U_2[p]$), then (Σ, loc) has the aperiodic Zielonka (resp. aperiodic aKR) property. Notice that such an aperiodic asynchronous automaton is easily constructed for acyclic architectures, thus providing an alternate proof of Theorem 4.2 in the aperiodic case.

REFERENCES

[Ads04]   Bharat Adsul. *Complete local logics for reasoning about traces*. PhD thesis, Indian Institute of Technology - Bombay, Mumbai, India, 2004.

[AGSW20]   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)*, volume 171 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 19:1–19:17, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. `doi:10.4230/LIPIcs.CONCUR.2020.19`.

[AGSW21]   Bharat Adsul, Paul Gastin, Saptarshi Sarkar, and Pascal Weil. Asynchronous wreath product and cascade decompositions for concurrent behaviours. `https://arxiv.org/abs/2105.10897v1`, 2021. `arXiv:2105.10897v1`.

[AS02]   Bharat Adsul and Milind A. Sohoni. Local normal forms for logics over traces. In Manindra Agrawal and Anil Seth, editors, *FSTTCS 2002: Foundations of Software Technology and Theoretical Computer Science, 22nd Conference Kanpur, India, December 12-14, 2002, Proceedings*, volume 2556 of *Lecture Notes in Computer Science*, pages 47–58. Springer, 2002. `doi:10.1007/3-540-36206-1\_6`.

[AS04]   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`.

[DG02]   Volker Diekert and Paul Gastin. LTL is expressively complete for Mazurkiewicz traces. *Journal of Computer and System Sciences*, 64(2):396–418, March 2002. URL: `http://www.lsv.ens-cachan.fr/Publis/PAPERS/PS/JCSS02dg.ps`.

[DG06]   Volker Diekert and Paul Gastin. Pure future local temporal logics are expressively complete for mazurkiewicz traces. *Inf. Comput.*, 204(11):1597–1619, 2006. `doi:10.1016/j.ic.2006.07.002`.

[Die90]   Volker Diekert. *Combinatorics on Traces*, volume 454 of *Lecture Notes in Computer Science*. Springer, 1990. `doi:10.1007/3-540-53031-2`.

[DKS12]   Volker Diekert, Manfred Kufleitner, and Benjamin Steinberg. The Krohn-Rhodes theorem and local divisors. *Fundam. Inform.*, 116(1-4):65–77, 2012. `doi:10.3233/FI-2012-669`.

[DR95]   Volker Diekert and Grzegorz Rozenberg. *The Book of Traces*. World Scientific Publishing Co., Inc., USA, 1995.

[Eil76]   Samuel Eilenberg. *Automata, Languages and Machines*, volume B. Academic Press, 1976.

[EM96]   Werner Ebinger and Anca Muscholl. Logical definability on infinite traces. *Theoretical Computer Science*, 154(1):67–84, 1996.

[GMPW98]   Giovanna Guaiana, Raphaël Meyer, Antoine Petit, and Pascal Weil. An extension of the wreath product principle for finite Mazurkiewicz traces. *Information Processing Letters*, 67(6):277 – 282, 1998. `doi:10.1016/S0020-0190(98)00123-9`.

[GRS92]   Giovanna Guaiana, Antonio Restivo, and Sergio Salemi. Star-free trace languages. *Theoretical Computer Science*, 97(2):301–311, 1992.

[KM13]   Siddharth Krishna and Anca Muscholl. A quadratic construction for Zielonka automata with acyclic communication structure. *Theor. Comput. Sci.*, 503(C):109–114, September 2013.

[KR65]   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`.

[Maz77]   Antoni Mazurkiewicz. Concurrent program schemes and their interpretations. *DAIMI Report Series*, 6(78), Jul. 1977. `doi:10.7146/dpb.v6i78.7691`.

[MS97]   Madhavan Mukund and Milind A. Sohoni. Keeping track of the latest gossip in a distributed system. *Distributed Computing*, 10(3):137–148, 1997. `doi:10.1007/s004460050031`.

[Muk12]   Madhavan Mukund. Automata on distributed alphabets. In Deepak D'Souza and Priti Shankar, editors, *Modern applications of automata theory*, pages 257–288. World Scientific, 2012.

[Pin19]   Jean-Éric Pin. Mathematical foundations of automata theory. `www.irif.fr/~jep/PDF/MPRI/MPRI.pdf`, 2019.

[Sar22]   Saptarshi Sarkar. *Algebraic Products and its Applications to Logic : Countable Words and Mazurkiewicz Traces*. PhD thesis, Indian Institute of Technology - Bombay, Mumbai, India, 2022.

[Str94]   Howard Straubing. *Finite automata, formal logic, and circuit complexity*. Birkhaüser Verlag, Basel, Switzerland, 1994.

[Zie87]   Wieslaw Zielonka. Notes on finite asynchronous automata. *RAIRO-Theoretical Informatics and Applications*, 21(2):99–135, 1987.