Wreath/Cascade Products and Related Decomposition Results for the Concurrent Setting of Mazurkiewicz Traces

Bharat Adsul
IIT Bombay, India
adsul@cse.iitb.ac.in

Paul Gastin
LSV, ENS Paris-Saclay, CNRS, Université Paris-Saclay, France
paul.gastin@ens-paris-saclay.fr

Saptarshi Sarkar
IIT Bombay, India
sapta@cse.iitb.ac.in

Pascal Weil
Université Bordeaux, LaBRI, CNRS UMR 5800, Talence, France
CNRS, ReLaX, IRL 2000, Siruseri, India
pascal.weil@labri.fr

Abstract

We develop a new algebraic framework to reason about languages of Mazurkiewicz traces. This framework supports true concurrency and provides a non-trivial generalization of the wreath product operation to the trace setting. A novel local wreath product principle has been established. The new framework is crucially used to propose a decomposition result for recognizable trace languages, which is an analogue of the Krohn-Rhodes theorem. We prove this decomposition result in the special case of acyclic architectures and apply it to extend Kamp's theorem to this setting. We also introduce and analyze distributed automata-theoretic operations called local and global cascade products. Finally, we show that aperiodic trace languages can be characterized using global cascade products of localized and distributed two-state reset automata.

2012 ACM Subject Classification Theory of computation → Distributed computing models; Theory of computation → Algebraic language theory

Keywords and phrases Mazurkiewicz traces, asynchronous automata, wreath product, cascade product, Krohn Rhodes decomposition theorem, local temporal logic over traces

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2020.19

Related Version An extended version is available at `https://arxiv.org/abs/2007.07940`.

Funding *Paul Gastin*: Supported by IRL ReLaX.
Pascal Weil: Partially supported by the DeLTA project (ANR-16-CE40- 0007)

1 Introduction

Transformation monoids provide an abstraction of transition systems. One of the key tools in their analysis is the notion of wreath product [7, 20, 18] which, when translated to the language of finite state automata, corresponds to the cascade product. In the cascade product of automata A and B, with A "followed by" B, the automaton A runs on the input sequence, while the automaton B runs on the input sequence as well as the state sequence produced by the automaton A. The wreath product principle (see [20, 18, 17]) is a key result which relates a language accepted by a cascade/wreath product to languages accepted by the individual automata.

In this work, we are interested in generalizing the wreath product operation from the sequential setting to the concurrent setting involving multiple processes. Towards this, we work with Mazurkiewicz traces (or simply traces) [12, 6] which are well established as models of true concurrency, and asynchronous automata [21] which are natural distributed finite state devices working on traces. A trace represents a concurrent behaviour as a labelled partial order which faithfully captures the distribution of events across processes, and causality and concurrency between them. An asynchronous automaton runs on the input trace in a distributed fashion and respects the underlying causality and concurrency between events. During the run, when working on an event, only the local states of the processes participating in that event are updated; the rest of the processes remain oblivious to the occurrence of the event at this point.

A natural generalization of the above mentioned sequential cascade product to asynchronous automata A and B is as follows: the asynchronous automaton A runs on the input trace, thus assigning, for each event, a local state for every process participating in that event. Now the asynchronous automaton B runs on the input trace with the *same* set of events which are *additionally* labelled by the previous local states of the participating processes in A. It is easy to capture this operational semantics by another asynchronous automaton which we call the *local* cascade product of A and B. Such a construction is used in [2] to provide an asynchronous automata-theoretic characterization of aperiodic trace languages.

Here we propose a new algebraic framework to deal with the issues posed by the concurrent setting. More precisely, we introduce a new class of transformation monoids called *asynchronous transformation monoids* (in short, *atm's*). These monoids make a clear distinction between local and global "states" and allow us to reason about whether a global transformation is essentially induced by a particular subset of processes. Recall that, from a purely algebraic viewpoint, the set of all traces forms a free partially commutative monoid in which *independent* actions commute [6]. In order to recognize a trace language via an atm, we introduce the notion of an *asynchronous morphism* which exploits the locality of the underlying atm. It is rather easy to see that asynchronous morphisms are the algebraic counterparts of asynchronous automata.

One of the central results here is a wreath product principle in the new algebraic framework. It turns out that the *standard* wreath product operation yields an operation on asynchronous transformation monoids. Let T_1 and T_2 be atm's and $T_1 \wr T_2$ be the wreath product atm. Our *local* wreath product principle describes a trace language recognized by $T_1 \wr T_2$ in terms of a *local asynchronous transducer* which is a natural *causality and concurrency preserving* map from traces to traces (over an appropriately extended alphabet), and trace languages recognized by T_1 and T_2. It is a novel generalization of the standard wreath product principle. The work [8] presents a wreath product principle for traces in the setting of transformation monoids but it seems less significant since it uses *non-trace* structures.

The importance of the standard wreath product operation is clearly highlighted by the fundamental Krohn-Rhodes decomposition theorem [11] which, broadly speaking, says that any finite transformation monoid can be simulated by wreath products of "simple" transformation monoids. The wreath product principle along with the Krohn-Rhodes theorem can be used to provide alternate and conceptually simpler proofs (see [14, 3]) of several interesting classical results about formal languages of words such as the theorems due to Schützenberger [19], McNaughton-Papert [13] and Kamp [9] which together show the equivalence between star-free, aperiodic, first-order-definable and linear-temporal-logic definable word languages. Motivated by these applications, we investigate an analogue of the fundamental Krohn-Rhodes decomposition theorem over traces. We use the new algebraic

framework to propose a simultaneous generalization of the Krohn-Rhodes theorem (for word languages) and Zielonka's theorem (for trace languages). The proof of this generalization for the special case of acyclic architectures is another significant result. As an application, we extend Kamp's theorem: we show a natural local temporal logic to be expressively complete.

It turns out that asynchronous morphisms into wreath products correspond to the aforementioned distributed automata-theoretic local cascade products. We also introduce the *global* cascade product operation and show that it can be realized as the local cascade product with the help of the ubiquitous gossip automaton from [16].

Our final major contribution concerns aperiodic trace languages and is in the spirit of the Krohn-Rhodes theorem for the aperiodic case. We establish that aperiodic trace languages can be characterized using global cascade products of localized and distributed two-state reset automata. The proof of this characterization crucially uses an expressively complete process-based local temporal logic over traces from [4].

The rest of the paper is organized as follows. After setting up the preliminaries in Section 2, we develop the new algebraic framework in Section 3 and establish the local wreath product principle. In Section 4, we postulate a new decomposition result, and we prove it for acyclic architectures. We introduce and analyze local and global cascade products in Section 5. The global cascade product based characterization of aperiodic trace languages appears in Section 6. Finally, we conclude in Section 7. Due to space constraints some proofs and examples are omitted and can be found in the extended version [1].

2 Preliminaries

2.1 Basic Notions in Trace Theory

Let $\mathcal{P}$ be a finite set of agents/processes. A *distributed alphabet* over $\mathcal{P}$ is a family $\widetilde{\Sigma} = \{\Sigma_i\}_{i\in\mathcal{P}}$. Let $\Sigma = \bigcup_{i\in\mathcal{P}} \Sigma_i$. For $a \in \Sigma$, we set $\mathrm{loc}(a) = \{i \in \mathcal{P} \mid a \in \Sigma_i\}$. By (Σ, I) we denote the corresponding trace alphabet, i.e., I is the *independence relation* $\{(a,b) \in \Sigma^2 \mid \mathrm{loc}(a) \cap \mathrm{loc}(b) = \emptyset\}$ induced by $\widetilde{\Sigma}$. The corresponding *dependence relation* $\Sigma^2 \setminus I$ is denoted by D.

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 $\downarrow e$.

A *trace* over $\widetilde{\Sigma}$ 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 $TR(\widetilde{\Sigma})$ denote the set of all traces over $\widetilde{\Sigma}$. Henceforth a trace means a trace over $\widetilde{\Sigma}$ unless specified otherwise. Let $t = (E, \leq, \lambda) \in TR(\widetilde{\Sigma})$. 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 $i \in \mathcal{P}$. 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 $\leq$.

A subset $c \subseteq E$ is a *configuration* of t if and only if $\downarrow c = c$. We let $\mathcal{C}_t$ denote the set of all configurations of t. Notice that $\emptyset$, the empty set, and E are configurations. More importantly, $\downarrow e$ is a configuration for every $e \in E$. There are two natural transition relations that one may associate with the configurations of t. The event based transition relation $\Rightarrow_t \subseteq \mathcal{C}_t \times E \times \mathcal{C}_t$ is defined by $c \overset{e}{\Rightarrow}_t c'$ if and only if $e \notin c$ and $c \cup \{e\} = c'$. The action based transition relation $\to_t \subseteq \mathcal{C}_t \times \Sigma \times \mathcal{C}_t$ is defined by $c \xrightarrow{a}_t c'$ if and only if there exists $e \in E$ such that $\lambda(e) = a$ and $c \overset{e}{\Rightarrow}_t c'$.

Now we turn our attention to the important operation of concatenation of traces. Let $t = (E, \leq, \lambda) \in TR(\widetilde{\Sigma})$ and $t' = (E', \leq', \lambda') \in TR(\widetilde{\Sigma})$. Without loss of generality, we can assume E and E' to be disjoint. We define $tt' \in TR(\widetilde{\Sigma})$ 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 $TR(\widetilde{\Sigma})$ a monoid structure. Observe that, with a (resp. b) denoting the singleton trace with the only event labelled a (resp. b), if $(a, b) \in I$ then $ab = ba$ in $TR(\widetilde{\Sigma})$.

A basic result in trace theory gives a presentation of the trace monoid as a quotient of the *free* word monoid Σ^*. See [6] for more details. Let $\sim_I \subseteq \Sigma^* \times \Sigma^*$ be the congruence generated by $ab \sim_I ba$ for $(a, b) \in I$.

▶ **Proposition 1.** *The canonical morphism from $\Sigma^* \to TR(\widetilde{\Sigma})$, 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 $TR(\widetilde{\Sigma})$.*

2.2 Transformation Monoids and Trace Languages

A map from a set X to itself is called a *transformation* of X. Under function composition, the set of all such transformations forms a monoid; let us denote this monoid by $\mathcal{F}(X)$. The function composition $f_1 f_2$ applies from left-to-right, that is, $(f_1 f_2)(\cdot) = f_2(f_1(\cdot))$.

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.** Consider $X = \{1, 2\}$ with the monoid $M = \{\mathrm{id}_X, r_1, r_2\}$ where id_X is the identity transformation and r_i maps every element in X to element i. Note that $r_1 r_2 = r_2$ and $r_2 r_1 = r_1$. Then (X, M) is a tm. We will refer to it as U_2. ⌟

Let $T = (X, M)$ be a tm. By a morphism φ from $TR(\widetilde{\Sigma})$ to T, we mean a (monoid) morphism $\varphi \colon TR(\widetilde{\Sigma}) \to M$. We abuse the notation and also write this as $\varphi \colon TR(\widetilde{\Sigma}) \to T$. Observe that, if $(a, b) \in I$, then as $ab = ba$ in $TR(\widetilde{\Sigma})$, $\varphi(a)$ and $\varphi(b)$ must commute in M. In fact, in view of Proposition 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 $TR(\widetilde{\Sigma})$ to M.

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

3 New Algebraic Framework

3.1 Asynchronous Transformation Monoids

Recall that we have a fixed finite set $\mathcal{P}$ of processes. If $\mathcal{P}$ is clear from the context, we use the simpler notation $\{X_i\}$ to denote the $\mathcal{P}$-indexed family $\{X_i\}_{i \in \mathcal{P}}$. The elements of the sets in a $\mathcal{P}$-indexed family will be typically called *states*.

We begin with some notation involving local and global states. Suppose that each process $i \in \mathcal{P}$ is equipped with a finite non-empty set of *local i-states*, denoted S_i. We set $S = \bigcup_{i\in\mathcal{P}} S_i$ and call S the set of *local states*. We let P range over non-empty subsets of $\mathcal{P}$ and let i, j range over $\mathcal{P}$. A *P-state* is a map $s\colon P \to S$ such that $s(j) \in S_j$ for every $j \in P$. We let S_P denote the set of all P-states. We call $S_{\mathcal{P}}$ the set of all *global states*.

If $P' \subseteq P$ and $s \in S_P$ then $s_{P'}$ is s restricted to P'. We use the shorthand $-P$ to indicate the complement of P in $\mathcal{P}$. We sometimes split a global state $s \in S_{\mathcal{P}}$ as $(s_P, s_{-P}) \in S_P \times S_{-P}$. We let S_a denote the set of all $\mathrm{loc}(a)$-states which we also call a-states for simplicity. Thus if $\mathrm{loc}(a) \subseteq P$ and s is a P-state we shall write s_a to mean $s_{\mathrm{loc}(a)}$.

Now we are ready to introduce a new class of transformation monoids.

▶ **Definition 3.** *An* asynchronous transformation monoid *(in short, atm) T (over $\mathcal{P}$) is a pair $(\{S_i\}, M)$ where*

- *S_i is a finite non-empty set for each process $i \in \mathcal{P}$.*
- *M is a submonoid of $\mathcal{F}(S_{\mathcal{P}})$, the monoid of all transformations from $S_{\mathcal{P}}$ to itself.*

Note that this definition is dependent on $\mathcal{P}$ and an atm $T = (\{S_i\}, M)$ naturally induces the tm $(S_{\mathcal{P}}, M)$. We abuse the notation and write T also for this tm.

A crucial feature of the definition of an atm is that it makes a clear distinction between local and global states. Observe that the underlying transformations operate on global states. It will be useful to know whether a global transformation is essentially induced by a particular subset of processes. We develop some notions to make this precise.

Fix an atm $(\{S_i\}, M)$ and $P \subseteq \mathcal{P}$. Let $f\colon S_P \to S_P$ be a map. We define $g\colon S_{\mathcal{P}} \to S_{\mathcal{P}}$ as: for $s \in S_{\mathcal{P}}$,

$$g(s) = s' \text{ iff } f(s_P) = s'_P \text{ and } s_{-P} = s'_{-P}$$

We refer to g as the extension of f. More generally, $h\colon S_{\mathcal{P}} \to S_{\mathcal{P}}$ is said to be a P-map if it is the extension of some $f\colon S_P \to S_P$. Note that, in this case, for all $s = (s_P, s_{-P}) \in S_{\mathcal{P}}$, $h((s_P, s_{-P})) = (f(s_P), s_{-P})$ and f is uniquely determined by h. It is worth pointing out that a map $h\colon S_{\mathcal{P}} \to S_{\mathcal{P}}$ with the property that for every $s \in S_{\mathcal{P}}$ there exists $s'_P \in S_P$ such that $h((s_P, s_{-P})) = (s'_P, s_{-P})$ is *not* necessarily a P-map. This condition merely says that the $(-P)$-component of a global state is not updated by h. The update of the P-component may still depend on the $(-P)$-component.

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

▶ **Lemma 4.** *Let $h\colon S_{\mathcal{P}} \to S_{\mathcal{P}}$. Then h is a P-map if and only if for every s in $S_{\mathcal{P}}$, $[h(s)]_{-P} = s_{-P}$ and for every s, s' in $S_{\mathcal{P}}$, $s_P = s'_P$ implies that $[h(s)]_P = [h(s')]_P$.*

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

▶ **Lemma 5.** *Let $f, g\colon S_{\mathcal{P}} \to S_{\mathcal{P}}$ be such that f is a P-map and g is a P'-map. If $P \cap P' = \emptyset$, then $fg = gf$.*

▶ **Example 6.** Fix a process $\ell \in \mathcal{P}$. We define the atm $U_2[\ell] = (\{S_i\}, M)$ where, $S_\ell = \{1, 2\}$ and for each $i \neq \ell$, S_i has exactly one element. Observe that $S_{\mathcal{P}}$ has only two global states which are completely determined by their ℓ-components. We will identify a global state with its ℓ-component. The monoid M is $\{\mathrm{id}_{S_{\mathcal{P}}}, r_1, r_2\}$ where $\mathrm{id}_{S_{\mathcal{P}}}$ is the identity transformation and r_i maps every global state to the global state i. Note that r_1 and r_2 are $\{\ell\}$-maps. ⌟

3.2 Asynchronous Morphisms

Now we fix a distributed alphabet $\widetilde{\Sigma} = \{\Sigma_i\}_{i\in\mathcal{P}}$ over $\mathcal{P}$ and introduce special morphisms from the trace monoid $TR(\widetilde{\Sigma})$ to atm's.

▶ **Definition 7.** *Let $T = (\{S_i\}, M)$ be an atm. An* asynchronous morphism *from $TR(\widetilde{\Sigma})$ to T is a (monoid) morphism $\varphi\colon TR(\widetilde{\Sigma}) \to M$ such that, for $a \in \Sigma$, $\varphi(a)$ is a $\mathrm{loc}(a)$-map (or simply, an a-map). We also write this as $\varphi\colon TR(\widetilde{\Sigma}) \to T$.*

It is important to observe that, contrary to the sequential case, a morphism from $TR(\widetilde{\Sigma})$ to M is not necessarily an asynchronous morphism from $TR(\widetilde{\Sigma})$ to the atm $T = (\{S_i\}, M)$. In a morphism $\psi\colon TR(\widetilde{\Sigma}) \to M$, for $(a, b) \in I$, $\psi(a)$ and $\psi(b)$ must commute; however for some $a \in \Sigma$, $\psi(a)$ may not be an a-map.

▶ **Example 8.** Consider $\mathcal{P} = \{p_1, p_2, p_3\}$, $\widetilde{\Sigma} = \{\Sigma_{p_1} = \{a, b\}, \Sigma_{p_2} = \{b, c\}, \Sigma_{p_3} = \{c\}\}$ and the atm $U_2[p_1] = (\{S_i\}, M)$ where $M = \{\mathrm{id}, r_1, r_2\}$. Recall that r_1 and r_2 are $\{p_1\}$-maps. The function $\psi(a) = \mathrm{id}$, $\psi(b) = r_2$ and $\psi(c) = r_1$ extends to a morphism from $TR(\widetilde{\Sigma})$ to the monoid M. However, it is not an asynchronous morphism to the atm $U_2[p_1]$ as $\psi(c)$ (being r_1) is not a $\{p_2, p_3\}$-map. ⌟

A fundamental result about asynchronous morphisms is stated in the following lemma. Its proof follows from Proposition 1 and Lemma 5, and can be found in [1].

▶ **Lemma 9.** *Let $T = (\{S_i\}, M)$ be an atm. Further, let $\varphi\colon \Sigma \to M$ be such that, for $a \in \Sigma$, $\varphi(a)$ is an a-map. Then φ can be uniquely extended to an asynchronous morphism from $TR(\widetilde{\Sigma})$ to T.*

▶ **Example 10.** Consider the setup of Example 8. The function $\varphi(a) = r_1$, $\varphi(b) = r_2$ and $\varphi(c) = \mathrm{id}$ extends to an asynchronous morphism from $TR(\widetilde{\Sigma})$ to $U_2[p_1]$. ⌟

Now we extend the notion of trace-language recognition from tm's to atm's via asynchronous morphisms. Let $L \subseteq TR(\widetilde{\Sigma})$ be a trace language. We say that L is recognized by an atm $T = (\{S_i\}, M)$ if there exists an *asynchronous* morphism $\varphi\colon TR(\widetilde{\Sigma}) \to T$, an *initial* element $s_{\mathrm{in}} \in S_{\mathcal{P}}$ and a *final* subset $S_{\mathrm{fin}} \subseteq S_{\mathcal{P}}$ such that

$$L = \{t \in TR(\widetilde{\Sigma}) \mid \varphi(t)(s_{\mathrm{in}}) \in S_{\mathrm{fin}}\}.$$

In the rest of this subsection, we bring out the intimate relationship between asynchronous morphisms and asynchronous automata. We begin with the description of an asynchronous automaton – a model introduced by Zielonka for concurrent computation on traces.

An *asynchronous automaton* A over $\widetilde{\Sigma}$ is a structure $(\{S_i\}_{i\in\mathcal{P}}, \{\delta_a\}_{a\in\Sigma}, s_{\mathrm{in}})$ 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 transition function on a-states
- $s_{\mathrm{in}} \in S_{\mathcal{P}}$ is an initial global state

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

Now we describe the notion of a run of A on an input trace. A trace run is easiest to define using configurations. Towards this, fix a trace $t = (E, \leq, \lambda) \in TR(\widetilde{\Sigma})$. Recall that (see Section 2.1) $\mathcal{C}_t$ is the set of all configurations of t and $\rightarrow_t \subseteq \mathcal{C}_t \times \Sigma \times \mathcal{C}_t$ is the natural

action based transition relation on configurations. A *trace run* of A over $t \in TR(\widetilde{\Sigma})$ is a map $\rho\colon \mathcal{C}_t \to S_{\mathcal{P}}$ such that $\rho(\emptyset) = s_{\text{in}}$, and for every (c, a, c') in $\to_t$, we have $\Delta_a(\rho(c)) = \rho(c')$. As A is deterministic, t admits a unique trace run; it will be denoted by ρ_t.

Let $L \subseteq TR(\widetilde{\Sigma})$ be a trace language. We say that L is *accepted* by A if there exists a subset $S_{\text{fin}} \subseteq S_{\mathcal{P}}$ of final global states such that $L = \{t = (E, \leq, \lambda) \in TR(\widetilde{\Sigma}) \mid \rho_t(E) \in S_{\text{fin}}\}$.

Our aim is to associate with A, a natural atm T_A and an asynchronous morphism φ_A such that languages accepted by A are precisely the languages recognized via φ_A.

We first describe the *transition* monoid M_A associated to A. It is possible to extend the global transition functions $\{\Delta_a\}$ to arbitrary traces using Proposition 1. For $t \in TR(\widetilde{\Sigma})$, we denote this extended global transition function by $\Delta_t\colon S_{\mathcal{P}} \to S_{\mathcal{P}}$. It is easy to check that, for $t = (E, \leq, \lambda) \in TR(\widetilde{\Sigma})$, $\Delta_t(s_{\text{in}}) = \rho_t(E)$. Further, as expected, for $t, t' \in TR(\widetilde{\Sigma})$, the function composition $\Delta_t\Delta_{t'}$ is identical to $\Delta_{tt'}$. We let M_A be the finite set of functions $\{\Delta_t \mid t \in TR(\widetilde{\Sigma})\}$. Clearly, it is a monoid under the usual composition of functions.

Next, we define the *transition* atm of A to be $T_A = (\{S_i\}, M_A)$ and the natural map $\varphi_A\colon TR(\widetilde{\Sigma}) \to M_A$ sending t to Δ_t. It is clear that φ_A is a morphism of monoids. Furthermore, it is an asynchronous morphism from $TR(\widetilde{\Sigma})$ to T_A; this is because, for $a \in \Sigma$, $\varphi_A(a) = \Delta_a$ is in fact an a-map of the atm T_A. The map φ_A is called the *transition* (asynchronous) morphism of A. Note that, for $t = (E, \leq, \lambda) \in TR(\widetilde{\Sigma})$,

$$\varphi_A(t)(s_{\text{in}}) = \Delta_t(s_{\text{in}}) = \rho_t(E)$$

We refer to the above statement as the *duality* between a run of A and an evaluation of φ_A.

The following lemma summarizes the above discussion for later reference and its proof is immediate.

▶ **Lemma 11.** *Given an asynchronous automaton $A = (\{S_i\}, \{\delta_a\}, s_{in})$ over $\widetilde{\Sigma}$, the transition atm $T_A = (\{S_i\}, M_A)$ and the transition asynchronous morphism $\varphi_A\colon TR(\widetilde{\Sigma}) \to T_A$ are effectively constructible. Moreover, if L is a trace language, then L 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 now provide a form of converse to Lemma 11. Towards this, we fix an atm $T = (\{S_i\}, M)$, a state $s_{\text{in}} \in S_{\mathcal{P}}$ and an asynchronous morphism $\varphi\colon TR(\widetilde{\Sigma}) \to T$. Since φ is an asynchronous morphism, $\varphi(a)$ is an a-map, and is an extension of some $\delta_a\colon S_a \to S_a$ over a-states. We set $A_\varphi = (\{S_i\}, \{\delta_a\}, s_{\text{in}})$ over $\widetilde{\Sigma}$. It turns out that the transition monoid of A_φ is the image of φ, a submonoid of M and the transition morphism of A_φ is the appropriate restriction of φ to this submonoid. The next lemma is easy to prove and we skip its proof.

▶ **Lemma 12.** *Given $T = (\{S_i\}, M)$, $\varphi\colon TR(\widetilde{\Sigma}) \to T$ and $s_{in} \in S_{\mathcal{P}}$, the asynchronous automaton A_φ over $\widetilde{\Sigma}$ is effectively constructible. Moreover, a trace language $L \subseteq TR(\widetilde{\Sigma})$ is recognized by T via φ (with initial state s_{in}) if and only if it is accepted by A_φ.*

3.3 Asynchronous Wreath Product

We begin with the crucial definition of a wreath product of transformation monoids. For sets U and V, we denote the set of all functions from U to V by $\mathcal{F}(U, V)$.

▶ **Definition 13** (Wreath Product). *Let $T_1 = (X, M)$ and $T_2 = (Y, N)$ be two tm's. We define $T = T_1 \wr T_2$ to be the tm $(X \times Y, M \times \mathcal{F}(X, N))$ where, for $m \in M$ and $f \in \mathcal{F}(X, N)$, (m, f) represents the following transformation on $X \times Y$:*

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

The tm T is called the wreath product *of T_1 and T_2. It turns out that, for $(m_1, f_1), (m_2, f_2)$ in $M \times \mathcal{F}(X, N)$, the composition law $(m_1, f_1)(m_2, f_2) = (m, f)$ is such that $m = m_1 m_2$ and for $x \in X$, $f(x) = f_1(x) + f_2(m_1(x))$. Here $+$ denotes the composition operation of N.*

It is a standard fact that the wreath product operation is associative [7]. We now adapt this operation to asynchronous transformation monoids.

▶ **Definition 14.** *Let $T_1 = (\{S_i\}, M)$ and $T_2 = (\{Q_i\}, N)$ be two atm's. We define their asynchronous wreath product, also denoted by $T_1 \wr T_2$, to be the atm $(\{S_i \times Q_i\}, M \times \mathcal{F}(S_\mathcal{P}, N))$. An element $(m, f) \in M \times \mathcal{F}(S_\mathcal{P}, N)$ represents the following global*[1] *transformation on $S_\mathcal{P} \times Q_\mathcal{P}$:*

$$\text{for } (s, q) \in S_\mathcal{P} \times Q_\mathcal{P}, \quad (m, f)((s, q)) = (m(s), f(s)(q))$$

The composition law on $M \times \mathcal{F}(S_\mathcal{P}, N)$ is the same as in Definition 13.

An important observation is that the tm associated with $T_1 \wr T_2$ is the wreath product of the tm's $(S_\mathcal{P}, M)$ and $(Q_\mathcal{P}, N)$ associated with T_1 and T_2 respectively. Sometimes, we will refer to the asynchronous wreath product simply as wreath product. The associativity of the asynchronous wreath product operation follows immediately.

We now present an important combinatorial lemma regarding the "support" of a global transformation in the wreath product. It plays a crucial role later.

▶ **Lemma 15.** *Fix atm's $T_1 = (\{S_i\}, M)$ and $T_2 = (\{Q_i\}, N)$. Let $(m, f) \in M \times \mathcal{F}(S_\mathcal{P}, N)$ represent a P-map in $T_1 \wr T_2$ for some subset $P \subseteq \mathcal{P}$. Then*

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

Proof. Fix $x_0 \in S_{-P}$ and $y_0 \in Q_{-P}$. We define $g_1 \colon S_P \to S_P$ and $g_2 \colon S_P \times Q_P \to Q_P$ by $g_1(x) = [m((x, x_0))]_P$ and $g_2(x, y) = [f((x, x_0))(y, y_0)]_P$. We first show that for all $s \in S_\mathcal{P}, q \in Q_\mathcal{P}$, $(m, f)((s, q)) = ((g_1(s_P), s_{-P}), (g_2(s_P, q_P), q_{-P}))$. Take an arbitrary $(s, q) \in S_\mathcal{P} \times Q_\mathcal{P}$. Then consider the global state $((s_P, x_0), (q_P, y_0))$ sharing the same P-component as (s, q) and the fixed $-P$-component (x_0, y_0). By the wreath product action (see Definition 13), $(m, f)\,(((s_P, x_0), (q_P, y_0))) = (m((s_P, x_0)), f((s_P, x_0))((q_P, y_0)))$. Being a P-map, (m, f) does not change the $-P$-component of any global state. So we have $m((s_P, x_0)) = ([m((s_P, x_0))]_P, x_0)$ and $f((s_P, x_0))((q_P, y_0)) = ([f((s_P, x_0))((q_P, y_0))]_P, y_0)$.

Let $(m, f)((s, q)) = (s', q')$. Since (m, f) is a P-map and the two global states (s, q) and $((s_P, x_0), (q_P, y_0))$ share the same P-component, by Lemma 4, $s'_P = [m((s_P, x_0))]_P$ and $q'_P = [f((s_P, x_0))((q_P, y_0))]_P$. Further, $s'_{-P} = s_{-P}$ and $q'_{-P} = q_{-P}$. Using the definitions of g_1 and g_2, we immediately see that $(m, f)((s, q)) = ((g_1(s_P), s_{-P}), (g_2(s_P, q_P), q_{-P}))$. However, by the wreath product action, $(m, f)((s, q)) = (m(s), f(s)(q))$. Comparing this with the previous expression, we have $m(s) = (g_1(s_P), s_{-P})$ and $f(s)(q) = (g_2(s_P, q_P), q_{-P})$. The result now follows from Lemma 4. ◀

[1] a global state (resp. P-state) of $T_1 \wr T_2$ is canonically identified with an element of $S_\mathcal{P} \times Q_\mathcal{P}$ (resp. $S_P \times Q_P$)

3.4 Local Wreath Product Principle

Let $A = (\{S_i\}, \{\delta_a\}, s_{in})$ be an asynchronous automaton over $\widetilde{\Sigma}$. Based on A and $\widetilde{\Sigma}$, we define the alphabet $\Sigma^{\|_S} = \{(a, s_a) \mid a \in \Sigma, s \in S_{\mathcal{P}}\}$ where a letter a in Σ is extended with local a-state information of A. This can naturally be viewed as a distributed alphabet $\widetilde{\Sigma^{\|_S}}$ where $\forall a \in \Sigma, \forall s \in S_{\mathcal{P}}$, $(a, s_a) \in \Sigma_i^{\|_S}$ if and only if $a \in \Sigma_i$. Then A induces the following transducer over traces.

▶ **Definition 16** (Local Asynchronous Transducer). *Let $\chi_A \colon TR(\widetilde{\Sigma}) \to TR(\widetilde{\Sigma^{\|_S}})$ be defined as follows. If $t = (E, \leq, \lambda) \in TR(\widetilde{\Sigma})$, then $\chi_A(t) = t'$ where $t' = (E, \leq, \mu) \in TR(\widetilde{\Sigma^{\|_S}})$ with the labelling $\mu \colon E \to \Sigma^{\|_S}$ defined by:*

$$\forall e \in E, \mu(e) = (a, s_a) \text{ where } a = \lambda(e) \text{ and } s = \rho_t(\downarrow e \setminus \{e\})$$

(recall that ρ_t is the unique trace run of A over t). We call χ_A the local asynchronous transducer of A.

▶ **Example 17.** Let χ be the local asynchronous transducer associated to A_φ where φ is as in Example 10. Figure 1 shows the run of A_φ on a trace $t \in TR(\widetilde{\Sigma})$ (by showing local process states before and after each event), and the resulting trace $\chi(t) \in TR(\widetilde{\Sigma^{\|_S}})$. ⌟

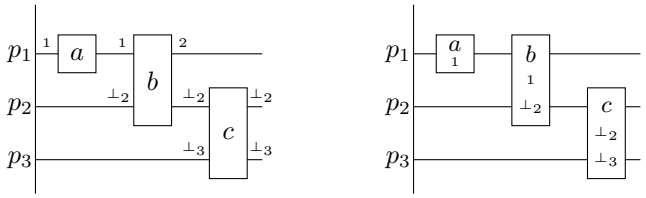

Figure 1 Local asynchronous transducer output on a trace; $S_{p_2} = \{\perp_2\}, S_{p_3} = \{\perp_3\}$.

Note that, in general, χ_A is not a morphism of monoids. The following lemma is a straightforward consequence of the definition of χ_A and the duality between trace runs of A and evaluations of the asynchronous asynchronous morphism φ_A.

▶ **Lemma 18.** *Let $t \in TR(\widetilde{\Sigma})$ with factorization $t = t'a$ (where $a \in \widetilde{\Sigma}$), and $s = \varphi_A(t')(s_{in})$. Then the trace $\chi_A(t) \in TR(\widetilde{\Sigma^{\|_S}})$ factors as $\chi_A(t) = \chi_A(t')(a, s_a)$.*

▶ **Theorem 19.** *Let A be an asynchronous automaton over $\widetilde{\Sigma}$ and χ_A be the corresponding local asynchronous transducer. If $L \subseteq TR(\widetilde{\Sigma^{\|_S}})$ is recognized by an atm T, then $\chi_A^{-1}(L)$ is recognized by the atm $T_A \wr T$.*

Proof. Let $\psi \colon TR(\widetilde{\Sigma^{\|_S}}) \to T = (\{Q_i\}, N)$ be an asynchronous morphism, which recognizes L with $q_{\text{in}} \in Q_{\mathcal{P}}$ as the initial global state, and $Q_{\text{fin}} \subseteq Q_{\mathcal{P}}$ as the set of final global states. Then $L = \{t \in TR(\widetilde{\Sigma^{\|_S}}) \mid \psi(t)(q_{\text{in}}) \in Q_{\text{fin}}\}$. Note that, for $(a, s_a) \in \Sigma^{\|_S}$, $\psi((a, s_a))$ is an a-map (that is, an extension of a map from Q_a to Q_a; recall that $\text{loc}((a, s_a)) = \text{loc}(a)$).

For $a \in \Sigma$, we set $\eta(a) = (\varphi_A(a), f_a)$ where $f_a \colon S_{\mathcal{P}} \to N$ is defined by $f_a(s) = \psi((a, s_a))$. It is easy to check that $\eta(a)$ is an a-map (that is, an extension of a map from $S_a \times Q_a$ to $S_a \times Q_a$). By Lemma 9, this uniquely defines an asynchronous morphism $\eta \colon TR(\widetilde{\Sigma}) \to T_A \wr T$.

Let $t = (E, \leq, \lambda) \in TR(\widetilde{\Sigma})$. We write $\eta(t) = (\pi_1(t), \pi_2(t))$. It follows from the definition of wreath product that $\pi_1(t) = \varphi_A(t)$. Now we claim that $\pi_2(t)(s_{\text{in}}) = \psi(\chi_A(t))$. We prove this by induction on the cardinality of E. Suppose $t = t'a$. Then $\eta(t) = \eta(t')\eta(a)$.

As a result, we have $(\pi_1(t), \pi_2(t)) = (\pi_1(t'), \pi_2(t'))(\pi_1(a), \pi_2(a))$. Therefore, for $s \in S_{\mathcal{P}}$, $\pi_2(t)(s) = \pi_2(t')(s) + \pi_2(a)(\pi_1(t')(s))$. In particular, it holds with $s = s_{\text{in}}$. Recall that $\pi_1(t') = \varphi_A(t')$. Also, by induction, $\pi_2(t')(s_{\text{in}}) = \psi(\chi_A(t'))$. Hence, with $s = \varphi_A(t')(s_{\text{in}})$,

$$\begin{aligned}\pi_2(t)(s_{\text{in}}) &= \pi_2(t')(s_{\text{in}}) + \pi_2(a)(\pi_1(t')(s_{\text{in}}))\\ &= \psi(\chi_A(t')) + \pi_2(a)(s)\\ &= \psi(\chi_A(t')) + \psi((a, s_a))\\ &= \psi(\chi_A(t')(a, s_a))\\ &= \psi(\chi_A(t))\end{aligned}$$

The last equality follows from Lemma 18. So, $t \in \chi_A^{-1}(L)$ if and only if $\chi_A(t) \in L$ if and only if $\psi(\chi_A(t))(q_{\text{in}}) \in Q_{\text{fin}}$ if and only if $\pi_2(t)(s_{\text{in}})(q_{\text{in}}) \in Q_{\text{fin}}$ if and only if $\eta(t)(s_{\text{in}}, q_{\text{in}}) \in S_{\mathcal{P}} \times Q_{\text{fin}}$. This shows that η recognizes $\chi_A^{-1}(L)$ with $(s_{\text{in}}, q_{\text{in}}) \in S_{\mathcal{P}} \times Q_{\mathcal{P}}$ as the initial global state, and $S_{\mathcal{P}} \times Q_{\text{fin}} \subseteq S_{\mathcal{P}} \times Q_{\mathcal{P}}$ as the set of final global states. ◀

Now we focus our attention on what is usually termed as the wreath product principle.

▶ **Theorem 20** (Local Wreath Product Principle). *Let T_1 and T_2 be atm's and let $L \subseteq TR(\widetilde{\Sigma})$ be a trace language recognized by an asynchronous morphism $\eta\colon TR(\widetilde{\Sigma}) \to T_1 \wr T_2$, with initial global state (s_{in}, q_{in}). For each $a \in \Sigma$, let $\eta(a) = (m_a, f_a)$. Then $\varphi\colon TR(\widetilde{\Sigma}) \to T_1$, defined by $\varphi(a) = m_a$, is an asynchronous morphism. Finally, let $A = A_\varphi$ be the asynchronous automaton associated to φ and s_{in}, and let χ_A be the corresponding local asynchronous transducer. Then L is a finite union of languages of the form $U \cap \chi_A^{-1}(V)$, where $U \subseteq TR(\tilde{\Sigma})$ is recognized by T_1, and $V \subseteq TR(\widetilde{\Sigma^{\|s}})$ is recognized by T_2.*

Proof. We write $T_1 = (\{S_i\}, M)$ and $T_2 = (\{Q_i\}, N)$. Consider $a \in \Sigma$ and the a-map $\eta(a) = (m_a, f_a) \in M \times \mathcal{F}(S_{\mathcal{P}}, N)$. This means that $\eta(a)$ is an extension of a map from $S_a \times Q_a$ to $S_a \times Q_a$. By Lemma 15, $m_a \in M$ is an a-map (of T_1) and $f_a\colon S_{\mathcal{P}} \to N$ is such that, for $s \in S_{\mathcal{P}}$, $f_a(s) \in N$ is an a-map (of T_2) and it depends only on s_a. In particular, $f_a\colon S_{\mathcal{P}} \to N$ may be viewed as $f_a\colon S_a \to N$. Below we will use f_a in this sense.

Now we define an asynchronous morphism $\psi\colon TR(\widetilde{\Sigma^{\|s}}) \to T_2$ as follows: $\psi((a, s_a)) = f_a(s_a)$. Note that, by Lemma 9, ψ is indeed an asynchronous morphism as $f_a(s_a)$ is an a-map. Further, as m_a is an a-map, $\varphi\colon TR(\widetilde{\Sigma}) \to T_1$, defined by $\varphi(a) = m_a$, also extends to an asynchronous morphism.

Our aim is to express L in terms of languages recognized by T_1 and T_2. It suffices to show the result when L is recognized with a single final global state, say $(s_{\text{fin}}, q_{\text{fin}})$. Then $L = \{t \in TR(\widetilde{\Sigma}) \mid \eta(t)((s_{\text{in}}, q_{\text{in}})) = (s_{\text{fin}}, q_{\text{fin}})\}$.

For $t \in TR(\widetilde{\Sigma})$, we write $\eta(t) = (\pi_1(t), \pi_2(t))$. It follows from the definition of φ that $\varphi(t) = \pi_1(t)$. Hence, we can alternatively write L as

$$L = \{t \in TR(\widetilde{\Sigma}) \mid \varphi(t)(s_{\text{in}}) = s_{\text{fin}} \text{ and } \pi_2(t)(s_{\text{in}})(q_{\text{in}}) = q_{\text{fin}}\}$$

Let $U = \{t \in TR(\widetilde{\Sigma}) \mid \varphi(t)(s_{\text{in}}) = s_{\text{fin}}\}$. Then, with $W = \{t \in TR(\widetilde{\Sigma}) \mid \pi_2(t)(s_{\text{in}})(q_{\text{in}}) = q_{\text{fin}}\}$, $L = U \cap W$. By using essentially the same ideas as in the proof of Theorem 19, we can show that $\pi_2(t)(s_{\text{in}}) = \psi(\chi_A(t))$. Therefore, $W = \{t \in TR(\widetilde{\Sigma}) \mid \psi(\chi_A(t))(q_{\text{in}}) = q_{\text{fin}}\}$.

It follows that, with $V = \{t' \in TR(\widetilde{\Sigma^{\|s}}) \mid \psi(t')(q_{\text{in}}) = q_{\text{fin}}\}$, $W = \chi_A^{-1}(V)$. Clearly, U is recognized by the atm T_1 (via φ), V is recognized by the atm T_2 (via ψ) and $L = U \cap \chi_A^{-1}(V)$. This completes the proof. ◀

4 Towards a Decomposition Result

In this section, we use the algebraic framework developed so far to propose an analogue of the fundamental Krohn-Rhodes decomposition theorem over traces. We first recall the Krohn-Rhodes theorem in the purely algebraic setting of transformation monoids. We briefly explain how it is used to analyze/decompose morphisms from the free monoid and point out some difficulties that arise when we consider morphisms from the trace monoid.

Let M and N be monoids. We say that M divides N (in notation, $M \prec N$) if M is a homomorphic image of some submonoid of N. This notion can be extended to transformation monoids. Let (X, M) and (Y, N) be two tm's. We say that (X, M) *divides* (Y, N), denoted $(X, M) \prec (Y, N)$, if there exists a pair of mappings (f, φ) where $f\colon Y \to X$ is a surjective function and $\varphi\colon N' \to M$ is a surjective morphism from a submonoid N' of N, such that $\varphi(n)(f(y)) = f(n(y))$ for all $n \in N'$ and all $y \in Y$.

Recall that $U_2 = (\{1, 2\}, \{\mathrm{id}, r_1, r_2\})$ denotes the *reset* transformation monoid on two elements. Along with it, the following class of transformation monoids plays an important role in the Krohn Rhodes theorem.

▶ **Example 21.** Let G be a group. Then (G, G) is a tm where the monoid element g *represents* the transformation $m_g\colon G \to G$ of the set G, which is the right multiplication by g. In other words, for $h \in G, m_g(h) = hg$. ⌟

We are now in a position to state the Krohn-Rhodes theorem [11]. See [20] for a classical proof of the theorem, and [5] for a modern proof.

▶ **Theorem 22** (Krohn-Rhodes Theorem). *Every finite transformation monoid $T = (X, M)$ divides a wreath product of the form $T_1 \wr \ldots \wr T_n$ where each factor T_i is either U_2 or is of the form (G, G) for some non-trivial simple group G dividing M.*

Henceforth, we will be dealing with only finite tm's and sometimes we will omit the qualifier "finite". Now we turn our attention to the use of this decomposition theorem for analysing word languages recognized by morphisms from the free monoid.

▶ **Definition 23.** *Let $\varphi\colon \Sigma^* \to T = (X, M)$ be a morphism. Further, let $\psi\colon \Sigma^* \to T' = (Y, N)$ be another morphism. 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 $y \in Y$, $f(\psi(a)(y)) = \varphi(a)(f(y))$.*

$$\begin{array}{ccc} Y & \xrightarrow{\psi(a)} & Y \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle f} \\ X & \xrightarrow[\varphi(a)]{} & X \end{array}$$

Figure 2 Visual illustration of condition $f(\psi(a)(y)) = \varphi(a)(f(y))$ in Definition 23.

Observe that if ψ simulates φ then a language recognized by φ is also recognized by ψ.

▶ **Proposition 24.** *Let $\varphi\colon \Sigma^* \to T = (X, M)$ be a morphism. Then there exists a morphism $\psi\colon \Sigma^* \to T'$ which simulates φ such that the tm T' is of the form $T_1 \wr \ldots \wr T_n$ where each factor T_i is either U_2 or (G, G) for some non-trivial simple group G dividing M.*

Proof. Given T, we get $T' = T_1 \wr \ldots \wr T_n = (Y, N)$ by the Krohn-Rhodes theorem. Since $T \prec T'$, there exists a pair of mappings (f, θ) where $f\colon Y \to X$ is a surjective function and $\theta\colon N' \to M$ is a surjective morphism from a submonoid N' of N, such that $\theta(n)(f(y)) = f(n(y))$ for

all $n \in N'$ and all $y \in Y$. Construct $\psi\colon \Sigma \to N$ by mapping $\psi(a)$, for each a in Σ, to an arbitrary element in $\theta^{-1}(\varphi(a))$. Thanks to the fact that Σ^* is a free monoid, ψ uniquely extends to a morphism $\psi\colon \Sigma^* \to T'$. It is easily checked that ψ simulates φ. ◀

Combined with the wreath product principle, the above proposition provides a powerful inductive tool to prove many non-trivial results in the theory of finite words. See [14, 3].

Motivated by these applications, we look for an analogue of the above proposition for the setting of traces. We now point to some problems that arise if one tries to naively lift the Krohn-Rhodes theorem to the setting of traces. The first problem is that, unlike in the word scenario, division does not imply simulation of morphisms from the *trace monoid*. By simulation of morphisms from the trace monoid, we simply mean an obvious adaptation of the Definition 23 to the morphisms from the trace monoid. See the extended version [1] for an example of the problem of the first kind. The second problem is that even if there is a morphism from $TR(\widetilde{\Sigma})$ to a wreath product of tm's, in general it does not induce morphisms from trace monoids to the individual tm's beyond the first one. This is primarily because the output of the *sequential* transducer associated with the first tm is *not* a trace.

▶ **Example 25.** Assume the DFA in Figure 3 represents the induced morphism to the first tm in a wreath product chain. The figure below shows the outputs of the sequential transducer associated with this DFA on three different linearizations of a single input trace. These outputs have different sets of letters and can not constitute a single trace. ⌟

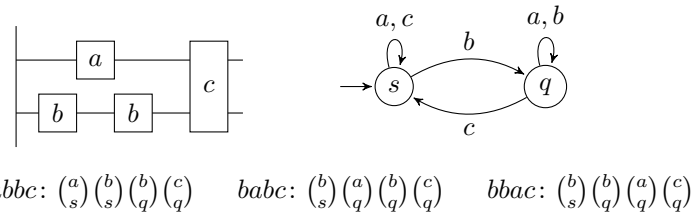

Figure 3 Sequential transducer outputs for all linearizations of a trace.

Prior work in [8] devised a wreath product principle for traces, but it uses non-trace structures to circumvent the second problem, thus limiting its applicability.

As seen in the previous section, the new algebraic framework of asynchronous structures supports true concurrency and is well suited to reason about trace languages. Most importantly, an asynchronous morphism to a wreath product chain gives rise to asynchronous morphisms to individual atm's of the chain (see the proof of Theorem 20 for an illustration). This can be seen as a resolution of the second problem mentioned above.

Going ahead, we extend the notion of simulation to the case when the "simulating" morphism is an asynchronous morphism to an atm.

▶ **Definition 26.** *Let $\varphi\colon TR(\widetilde{\Sigma}) \to T = (X, M)$ be a morphism to a tm. Further, let $T' = (\{S_i\}, N)$ be an atm and $\psi\colon TR(\widetilde{\Sigma}) \to T'$ be an asynchronous morphism. We say that ψ is an asynchronous simulation of φ (or simply ψ simulates φ) if there exists a surjective function $f\colon S_{\mathcal{P}} \to X$ such that, for all $a \in \Sigma$ and all $s \in S_{\mathcal{P}}$, $f(\psi(a)(s)) = \varphi(a)(f(s))$.*

The fundamental theorem of Zielonka [21] states that every recognizable language is accepted by some asynchronous automaton. See [15] for another proof of the theorem. From the viewpoint of our algebraic setup and the previous definition, it guarantees the existence of a simulating asynchronous morphism.

▶ **Theorem 27** (Zielonka's Theorem). *Let $\varphi\colon TR(\widetilde{\Sigma}) \to T$ be a morphism to a finite tm. There exists an asynchronous morphism $\psi\colon TR(\widetilde{\Sigma}) \to T'$, to a finite atm, which simulates φ.*

Recall that the atm $U_2[\ell]$, defined in Example 6, is a natural extension of the tm U_2 to the process ℓ. In a similar vein, for a group G, the atm $G[\ell]$ denotes the natural extension of the tm (G, G) from Example 21 to the process ℓ. We will use a similar notation to extend a tm to an atm localized to a particular process.

Now we formulate the following decomposition question:

▶ **Question 1.** *Let $\varphi\colon TR(\widetilde{\Sigma}) \to (X, M)$ be a morphism to a finite tm. Does there exist an asynchronous morphism $\psi\colon TR(\widetilde{\Sigma}) \to T'$ to a finite atm, such that ψ simulates φ, and the atm T' is of the form $T_1 \wr \ldots \wr T_n$ where each factor T_i is, for some $\ell \in \mathcal{P}$, either the atm $U_2[\ell]$ or is of the form $G[\ell]$ for some non-trivial simple group G dividing M?*

In view of our discussion so far, it is clear that the above question asks for a simultaneous generalization of the Krohn-Rhodes theorem for the setting of words (that is, Proposition 24), and Zielonka's theorem for the setting of traces (that is, Theorem 27). Question 1 in general remains open. However we answer it positively in a particular case, namely that of acyclic architectures, which is general enough to include the common client-server settings.

▶ **Definition 28.** *Let $\widetilde{\Sigma} = \{\Sigma_i\}_{i\in\mathcal{P}}$ be a distributed alphabet. Then its communication graph is $G = (\mathcal{P}, E)$ where $E = \{(i, j) \in \mathcal{P} \times \mathcal{P} \mid i \neq j \text{ and } \Sigma_i \cap \Sigma_j \neq \emptyset\}$. If the communication graph is acyclic, then the distributed alphabet is called an acyclic architecture.*

Observe that if $\widetilde{\Sigma}$ is an acyclic architecture, then no action is shared by more than two processes. The work [10] provides a simpler proof of Zielonka's theorem in this case.

▶ **Theorem 29.** *If $\widetilde{\Sigma}$ is an acyclic architecture, then Question 1 admits a positive answer.*

5 Local and Global Cascade Products

In this section, we introduce distributed automata-theoretic operations called local and global cascade products.

5.1 Local Cascade Product

As seen before, asynchronous morphisms are the algebraic counterparts of asynchronous automata. It turns out that an asynchronous morphism into a wreath product of atm's corresponds to the "local cascade product" of asynchronous automata. See [1] for details. Here we simply define the local cascade product of two asynchronous automata.

▶ **Definition 30.** *Let $A_1 = (\{S_i\}, \{\delta_a\}, s_{in})$ over $\widetilde{\Sigma}$, and $A_2 = (\{Q_i\}, \{\delta_{(a,s_a)}\}, q_{in})$ over $\widetilde{\Sigma^{\|s}}$. We define the local cascade product of A_1 and A_2 to be the asynchronous automaton $A_1 \circ_\ell A_2 = (\{S_i \times Q_i\}, \{\Delta_a\}, (s_{in}, q_{in}))$ over $\widetilde{\Sigma}$, where, for $a \in \Sigma$ and $(s_a, q_a) \in S_a \times Q_a$, $\Delta_a((s_a, q_a)) = (\delta_a(s_a), \delta_{(a,s_a)}(q_a))$.*

The operational working of $A = A_1 \circ_\ell A_2$ can be understood in terms of A_1 and A_2 using the local asynchronous transducer $\chi_{A_1}\colon TR(\widetilde{\Sigma}) \to TR(\widetilde{\Sigma^{\|s}})$ (associated with A_1) as follows: for an input trace $t \in TR(\widetilde{\Sigma})$, the run of A on t ends in global state (s, q) if and only if the run of A_1 on t ends in global state s and the run of A_2 on $\chi_{A_1}(t)$ ends in global state q. This *operational cascade* of A_1 followed by A_2 is summarized in the right part of the Figure 4 and is the essence of the local wreath product principle. Further, it is not difficult to check that the local cascade product is associative and $\chi_{A_1 \circ_\ell A_2}(t) = \chi_{A_2}(\chi_{A_1}(t))$ for all $t \in TR(\widetilde{\Sigma})$.

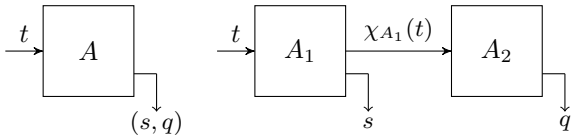

Figure 4 Operational view of local cascade product.

5.2 Global Asynchronous Transducer and its Local Implementation

Let $A = (\{S_i\}, \{\delta_a\}, s_{\text{in}})$ be an asynchronous automaton over $\widetilde{\Sigma}$. Recall that the local asynchronous transducer χ_A preserves the underlying set of events and, at an event, simply records the previous local states of the processes *participating* in that event.

Now we introduce a natural variant of χ_A which is called the *global asynchronous transducer*. In this variant, at an event, we record the *maximal/best global state* that causally precedes the current event. This is the best global state that the processes participating in the current event are (collectively) aware of. It is important to note that the global and local asynchronous transducers coincide in the sequential setting.

We first define the alphabet $\Sigma^{S_{\mathcal{P}}} = \Sigma \times S_{\mathcal{P}}$ where each letter in Σ is extended with global state information of A. This can naturally be viewed as a distributed alphabet $\widetilde{\Sigma^{S_{\mathcal{P}}}}$ where for all $a \in \Sigma$ and $s \in S_{\mathcal{P}}$, we have $(a, s) \in \Sigma_i^{S_{\mathcal{P}}}$ if and only if $a \in \Sigma_i$.

▶ **Definition 31** (Global Asynchronous Transducer). *Let A be an asynchronous automaton over $\widetilde{\Sigma}$. The global asynchronous transducer of A is the map $\theta_A \colon TR(\widetilde{\Sigma}) \to TR(\widetilde{\Sigma^{S_{\mathcal{P}}}})$ defined as follows. If $t = (E, \leq, \lambda) \in TR(\widetilde{\Sigma})$, then $\theta_A(t) = (E, \leq, \mu) \in TR(\widetilde{\Sigma^{S_{\mathcal{P}}}})$ with the labelling $\mu \colon E \to \Sigma \times S_{\mathcal{P}}$ defined by:*

$$\forall e \in E,\ \mu(e) = (a, s) \text{ where } a = \lambda(e) \text{ and } s = \rho_t(\downarrow e \setminus \{e\})$$

▶ **Example 32.** For t and A_φ from Example 17, Figure 5 shows its global asynchronous transducer output $\theta(t)$. Note the difference from Figure 1. For example, here the only 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 processes p_1 and p_3 never interact directly. ⌟

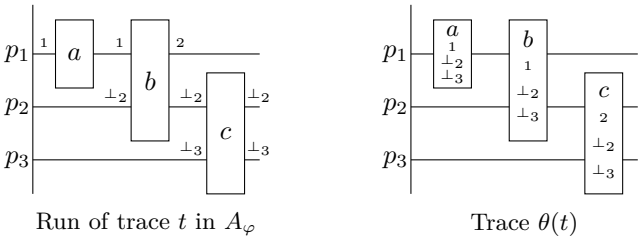

Figure 5 Global asynchronous transducer output on a trace.

It is possible, albeit non-trivial, to give a uniform translation from the automaton A to another automaton $\mathcal{G}(A)$ such that the global asynchronous transducer of A is realized by the local asynchronous transducer of $\mathcal{G}(A)$. It turns out that one must make crucial use of the latest information that the agents have about each other when defining the automaton $\mathcal{G}(A)$. It has been shown in [16] that this information can be kept track of by a deterministic asynchronous automaton. See [1] for more details.

5.3 Global Cascade Product

Now we are ready to define a cascade model which uses the global asynchronous transducer.

▶ **Definition 33** (Operational Global Cascade Product). *Let $A_1 = (\{S_i\}, \{\delta_a\}, s_{in})$ be an asynchronous automaton over $\widetilde{\Sigma}$, and $A_2 = (\{Q_i\}, \{\delta_{(a,s)}\}, q_{in})$ be an asynchronous automaton over $\widetilde{\Sigma^{S_\mathcal{P}}}$. Then their* operational *global cascade product, denoted by $A_1 \circ_g A_2$, is a cascade model where, for any input trace $t \in TR(\widetilde{\Sigma})$, A_1 runs on t (and "outputs" $\theta_{A_1}(t)$) and A_2 runs on $\theta_{A_1}(t)$. See Figure 6.*

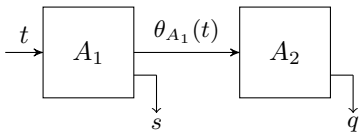

Figure 6 Operational view of global cascade product.

Note that $A_1 \circ_g A_2$ is not, a priori, an asynchronous automaton, but in view of the discussion in the preceeding subsection, it is simulated by the asynchronous automaton $\mathcal{G}(A_1) \circ_\ell A_2$.

For simplicity, we view $A_1 \circ_g A_2$ as an automaton with $S_\mathcal{P} \times Q_\mathcal{P}$ as its global states, and extend the notions of run, acceptance etc. to it in a natural way (see [1]). Henceforth, we refer to the operational global cascade product as the simply global cascade product. It turns out that the global cascade product is associative in a natural sense. See [1] for more details. Thanks to this, we can also talk about the global cascade product of a *sequence* of asynchronous automata.

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

▶ **Theorem 34** (Global Cascade Product Principle). *Let A (resp. B) be a global cascade product over $\widetilde{\Sigma}$ (resp. $\widetilde{\Sigma^{S_\mathcal{P}}}$), where $S_\mathcal{P}$ is the set of global states of A. Then any language $L \subseteq TR(\widetilde{\Sigma})$ accepted by $A \circ_g B$ is a finite union of languages of the form $U \cap \theta_A^{-1}(V)$ where $U \subseteq TR(\widetilde{\Sigma})$ is accepted by A, and $V \subseteq TR(\widetilde{\Sigma^{S_\mathcal{P}}})$ is accepted by B.*

6 Temporal Logics, Aperiodic Trace Languages & Cascade Products

An automata-theoretic consequence of Theorem 29 is that any aperiodic trace language (that is, a trace language recognized by an aperiodic monoid) over an acyclic architecture is accepted by a local cascade product of localized two-state reset automata. We call these automata $U_2[\ell]$ as well. In this section, we generalize this result to any distributed alphabet, but using global cascade product of $U_2[\ell]$s.

Our proof uses a process-based past local temporal logic (over traces) called $\mathsf{LocTL}[\mathsf{Y}_i, \mathsf{S}_i]$ that exactly defines aperiodic trace languages. This expressive completeness property of $\mathsf{LocTL}[\mathsf{Y}_i, \mathsf{S}_i]$ is an easy consequence of a non-trivial result from [4], where the future version of a similar local temporal logic is shown to coincide with first-order logic definable, equivalently, aperiodic trace languages. The syntax of $\mathsf{LocTL}[\mathsf{Y}_i, \mathsf{S}_i]$ is as follows.

$$\text{Event formula } \alpha = a \mid \neg\alpha \mid \alpha \vee \alpha \mid \mathsf{Y}_i\,\alpha \mid \alpha\,\mathsf{S}_i\,\alpha \qquad a \in \Sigma, i \in \mathcal{P}$$

$$\text{Trace formula } \beta = \exists_i \alpha \mid \neg\beta \mid \beta \vee \beta$$

The semantics of the logic is given below. Each event formula is evaluated at an event of a trace. Let $t = (E, \leq, \lambda) \in TR(\widetilde{\Sigma})$ be a trace with $e \in E$. For any event x in t and $i \in \mathcal{P}$, we denote by x_i the unique maximal event of $(\downarrow x \setminus \{x\}) \cap E_i$, if it exists.

$t, e \models a$	if $\lambda(e) = a$
$t, e \models \neg\alpha$	if $t, e \not\models \alpha$
$t, e \models \alpha \vee \beta$	if $t, e \models \alpha$ or $t, e \models \beta$
$t, e \models \mathsf{Y}_i\, \alpha$	if e_i exists, and $t, e_i \models \alpha$
$t, e \models \alpha\, \mathsf{S}_i\, \alpha'$	if $e \in E_i$ and $\exists f \in E_i$ such that $f < e$ and $t, f \models \alpha'$ and $\forall g \in E_i \;\; f < g < e \Rightarrow t, g \models \alpha$

Note that the since operator is a strict version. $\mathsf{LocTL}[\mathsf{Y}_i, \mathsf{S}_i]$ trace formulas are evaluated for traces, with the following semantics.

$t \models \exists_i \alpha$	if there exists a maximal i-event e in t such that $t, e \models \alpha$

The semantics of the boolean combinations of trace formulas are obvious. Any $\mathsf{LocTL}[\mathsf{Y}_i, \mathsf{S}_i]$ trace formula β over $\widetilde{\Sigma}$ defines the trace language $L_\beta = \{t \in TR(\widetilde{\Sigma}) \mid t \models \beta\}$. The following theorem gives a global cascade product characterization of $\mathsf{LocTL}[\mathsf{Y}_i, \mathsf{S}_i]$ definable languages.

▶ **Theorem 35.** *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 product of* $U_2[\ell]$*.*

By the expressive completeness of $\mathsf{LocTL}[\mathsf{Y}_i, \mathsf{S}_i]$ from [4], this gives a new characterization of aperiodic or equivalently, first-order logic definable trace languages in terms of global cascade products of localized two state asynchronous reset automata. It is in the spirit of the classical Krohn-Rhodes theorem for aperiodic word languages.

We now give a temporal logic characterization of local cascade product of $U_2[\ell]$. The local temporal logic $\mathsf{LocTL}[\mathsf{S}_i]$ is simply the fragment of $\mathsf{LocTL}[\mathsf{Y}_i, \mathsf{S}_i]$ where Y_i is disallowed. The semantics is inherited. It is unknown whether the logic $\mathsf{LocTL}[\mathsf{S}_i]$ is as expressive as $\mathsf{LocTL}[\mathsf{Y}_i, \mathsf{S}_i]$.

▶ **Theorem 36.** *A trace language is defined by a* $\mathsf{LocTL}[\mathsf{S}_i]$ *formula if and only if it is recognized by a local cascade product of* $U_2[\ell]$*.*

Note that if our postulated decomposition (see Question 1) were true, it would imply that $\mathsf{LocTL}[\mathsf{S}_i]$ is expressively complete, which would be a stronger temporal logic characterization for aperiodic trace languages than what is currently known. In particular, by Theorem 29, $\mathsf{LocTL}[\mathsf{S}_i]$ is expressively complete over tree architecture. And this holds true for any distributed alphabet where Question 1 admits a positive answer.

7 Conclusion

We have presented an algebraic framework equipped with wreath products and proved a wreath product principle which is well suited for the analysis of trace languages. Building on this framework, we have postulated a natural decomposition theorem which has been proved for the case of acyclic architectures. This special case already provides an interesting generalization of the Krohn-Rhodes theorem. It simultaneously proves Zielonka's theorem for acyclic architectures.

The wreath product operation in the new framework, when viewed in terms of automata, manifests itself in the form of a local cascade product of asynchronous automata. We have also proposed global cascade products of asynchronous automata and applied them to

arrive at a novel decomposition of aperiodic trace languages. This is a non-trivial and truly concurrent generalization of the cascade decomposition of aperiodic word languages using two-state reset automata.

References

1 Bharat Adsul, Paul Gastin, Saptarshi Sarkar, and Pascal Weil. Wreath/cascade products and related decomposition results for the concurrent setting of mazurkiewicz traces (extended version), 2020. arXiv:2007.07940.

2 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.

3 Joëlle Cohen, Dominique Perrin, and Jean-Eric Pin. On the expressive power of temporal logic. *Journal of Computer and System Sciences*, 46(3):271–294, 1993.

4 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.

5 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.

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

7 Samuel Eilenberg. *Automata, Languages and Machines*, volume B. Academic Press, 1976.

8 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.

9 Hans Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, UCLA, 1968.

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

11 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, April 1965. doi:10.2307/1994127.

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

13 Robert McNaughton and Seymour A. Papert. *Counter-Free Automata*. M.I.T. Research Monograph Nr 65. The MIT Press, 1971.

14 Albert R. Meyer. A note on star-free events. *J. ACM*, 16(2):220–225, April 1969. doi:10.1145/321510.321513.

15 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.

16 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.

17 Dominique Perrin and Jean-Eric Pin. *Infinite words - automata, semigroups, logic and games*, volume 141 of *Pure and applied mathematics series*. Elsevier Morgan Kaufmann, 2004.

18 Jean-Éric Pin. Mathematical foundations of automata theory, 2019. URL: https://www.irif.fr/~jep/PDF/MPRI/MPRI.pdf.

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

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

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