Asynchronous wreath product decompositions for trace languages

Pascal Weil

LaBRI, CNRS, Université de Bordeaux, and ReLaX

Joint work with Bharat Adsul (IIT Bombay), Paul Gastin (LMF), Saptarshi Sarkar (IIT Bombay)

IRIF, DeLTA Meeting, June 2021
Distributed alphabet, trace monoid

- Distributed behaviors. \( \mathcal{P} \), set of processes
Distributed alphabet, trace monoid

- Distributed behaviors. $\mathcal{P}$, set of processes
- $\tilde{\Sigma}$ distributed alphabet: $\text{loc}: \Sigma \rightarrow 2^{\mathcal{P}}$; letters $a, b$ are \textit{independent} if $\text{loc}(a) \cap \text{loc}(b) = \emptyset$
Distributed alphabet, trace monoid

- Distributed behaviors. \( \mathcal{P} \), set of processes
- \( \tilde{\Sigma} \) distributed alphabet: \( \text{loc}: \Sigma \rightarrow 2^\mathcal{P} \); letters \( a, b \) are independent if \( \text{loc}(a) \cap \text{loc}(b) = \emptyset \)
- \( \text{Tr}(\tilde{\Sigma}) \): quotient of \( \Sigma^* \) by the relation \( ab = ba \) if \( a, b \) are independent. Projection morphism: \( \pi: \Sigma^* \rightarrow \text{Tr}(\tilde{\Sigma}) \)
Distributed alphabet, trace monoid

- Distributed behaviors. $\mathcal{P}$, set of processes
- $\tilde{\Sigma}$ distributed alphabet: $\text{loc}: \Sigma \rightarrow 2^\mathcal{P}$; letters $a, b$ are independent if $\text{loc}(a) \cap \text{loc}(b) = \emptyset$
- $\text{Tr}(\tilde{\Sigma})$: quotient of $\Sigma^*$ by the relation $ab = ba$ if $a, b$ are independent. Projection morphism: $\pi: \Sigma^* \rightarrow \text{Tr}(\tilde{\Sigma})$
- Poset representation of a trace: $t = (E, \leq, \lambda)$; $E = \text{events}$

Let $\text{loc}(a) = \{1, 2\}$, $\text{loc}(b) = \{3, 4\}$, $\text{loc}(c) = \{2, 3\}$, $t = abcba$
Defn: \( L \subseteq Tr(\tilde{\Sigma}) \) is recognizable if \( \pi^{-1}(L) \subseteq \Sigma^* \) (the language of all linearizations of the traces in \( L \)) is recognizable.
Recognizable trace languages

- Defn: \( L \subseteq \text{Tr}(\tilde{\Sigma}) \) is recognizable if \( \pi^{-1}(L) \subseteq \Sigma^* \) (the language of all linearizations of the traces in \( L \)) is recognizable.

- Equivalently: there exists a morphism \( \varphi \) from \( \text{Tr}(\tilde{\Sigma}) \) to a finite transformation monoid \((X, M)\) such that \( L = \varphi^{-1}(M_s, F) \) (\( s \in X \) initial state, \( F \subseteq X \) accepting states).
Recognizable trace languages

- Defn: $L \subseteq Tr(\tilde{\Sigma})$ is recognizable if $\pi^{-1}(L) \subseteq \Sigma^*$ (the language of all linearizations of the traces in $L$) is recognizable
- Equivalently: there exists a morphism $\varphi$ from $Tr(\tilde{\Sigma})$ to a finite transformation monoid $(X, M)$ such that $L = \varphi^{-1}(M_s, F)$ ($s \in X$ initial state, $F \subseteq X$ accepting states)
- Characterization: diamond automata

\[ \begin{array}{c}
\text{a} \\
\circ \\
\text{b} \\
\circ \\
\text{b} \\
\circ \\
\text{a} \\
\circ \\
\end{array} \]
Recognizable trace languages

- Defn: $L \subseteq Tr(\tilde{\Sigma})$ is recognizable if $\pi^{-1}(L) \subseteq \Sigma^*$ (the language of all linearizations of the traces in $L$) is recognizable.

- Equivalently: there exists a morphism $\varphi$ from $Tr(\tilde{\Sigma})$ to a finite transformation monoid $(X, M)$ such that $L = \varphi^{-1}(M_s, F)$ ($s \in X$ initial state, $F \subseteq X$ accepting states).

- Characterization: diamond automata

- Characterization: MSO-definable languages (letter predicates & binary predicate $\leq$)

Pascal Weil
Wreath product decompositions for trace languages
Recognizable trace languages

- Defn: \( L \subseteq Tr(\tilde{\Sigma}) \) is recognizable if \( \pi^{-1}(L) \subseteq \Sigma^* \) (the language of all linearizations of the traces in \( L \)) is recognizable.
- Equivalently: there exists a morphism \( \varphi \) from \( Tr(\tilde{\Sigma}) \) to a finite transformation monoid \((X, M)\) such that \( L = \varphi^{-1}(M_{s,F}) \) (\( s \in X \) initial state, \( F \subseteq X \) accepting states).
- Characterization: diamond automata

![Diamond Automaton Diagram]

- Characterization: MSO-definable languages (letter predicates & binary predicate \( \leq \))
- Characterization: not so simple... Restricted rat’l expressions
An algebraic automata theory for trace languages? 1/3

- Recognizable $=$ finite monoid $=$ MSO-definable
Recognizable = finite monoid = MSO-definable

Star-free = Aperiodic = FO-definable (Guaiana et al. 1992, Ebinger and Muscholl 1993)
Recognizable = finite monoid = MSO-definable

Star-free = Aperiodic = FO-definable (Guaiiana et al. 1992, Ebinger and Muscholl 1993)

A good start, but then... nothing else in the literature
An algebraic automata theory for trace languages? 1/3

- Recognizable = finite monoid = MSO-definable
- Star-free = Aperiodic = FO-definable (Guaiana et al. 1992, Ebinger and Muscholl 1993)
- A good start, but then... nothing else in the literature
- No composition / decomposition results, as for word languages — using wreath products of transformation monoids / cascade products of DFAs
Composition / decomposition for word languages: wreath product of transformation monoids / cascade product of DFAs
Composition / decomposition for word languages: wreath product of transformation monoids / cascade product of DFAs

Krohn-Rhodes theorem: every finite tm \((X, M)\) is simulated by a wreath product of copies of the 2-state reset tm and of transformation monoids of the form \((G, G)\), where \(G\) is a simple group dividing \(M\)
Composition / decomposition for word languages: wreath product of transformation monoids / cascade product of DFAs

Krohn-Rhodes theorem: every finite tm \((X, M)\) is simulated by a wreath product of copies of the 2-state reset tm and of transformation monoids of the form \((G, G)\), where \(G\) is a simple group dividing \(M\)

2-state reset tm: \(U_2 = 1 \xrightarrow{r_1} 2 \xrightarrow{r_2} 1 \xleftarrow{r_1} 2 \xleftarrow{r_2}\)
Composition / decomposition for word languages: wreath product of transformation monoids / cascade product of DFAs

Krohn-Rhodes theorem: every finite tm \((X, M)\) is simulated by a wreath product of copies of the 2-state reset tm and of transformation monoids of the form \((G, G)\), where \(G\) is a simple group dividing \(M\).

2-state reset tm: \(U_2 =\)

Special case for aperiodic transformation monoids: wreath product of copies of \(U_2\)
Composition / decomposition for word languages: wreath product of transformation monoids / cascade product of DFAs

Krohn-Rhodes theorem: every finite tm \((X, M)\) is simulated by a wreath product of copies of the 2-state reset tm and of transformation monoids of the form \((G, G)\), where \(G\) is a simple group dividing \(M\)

2-state reset tm: \(U_2 = \begin{array}{c}
\text{1} \quad \text{r}_1 \quad \text{r}_2 \\
\text{2} \quad \text{r}_1 \quad \text{r}_2
\end{array}\)

Special case for aperiodic transformation monoids: wreath product of copies of \(U_2\)

Useful for inductive proofs, see eg Straubing’s proofs of the equivalence of aperiodic with FO or with LTL
Imitate these results for recognizable trace languages
An algebraic automata theory for trace languages? 3/3

- Imitate these results for recognizable trace languages
- Start with \( L \subseteq Tr(\tilde{\Sigma}) \), recognized by \( \varphi: Tr(\tilde{\Sigma}) \rightarrow (X, M) \) or, equivalently, by a diamond automaton.
Imitate these results for recognizable trace languages

Start with $L \subseteq \text{Tr}(\tilde{\Sigma})$, recognized by $\varphi: \text{Tr}(\tilde{\Sigma}) \rightarrow (X, M)$ or, equivalently, by a diamond automaton

Proofs collapse immediately
Imitate these results for recognizable trace languages

Start with $L \subseteq Tr(\tilde{\Sigma})$, recognized by $\varphi: Tr(\tilde{\Sigma}) \longrightarrow (X, M)$ or, equivalently, by a diamond automaton

Proofs collapse immediately

if $(X, M)$ is simulated by $(Y, N)$, it does not follow that $(Y, N)$ recognizes $L$
Imitate these results for recognizable trace languages

Start with \( L \subseteq \operatorname{Tr}(\tilde{\Sigma}) \), recognized by \( \varphi : \operatorname{Tr}(\tilde{\Sigma}) \rightarrow (X, M) \) or, equivalently, by a diamond automaton.

Proofs collapse immediately

if \((X, M)\) is simulated by \((Y, N)\), it does not follow that \((Y, N)\) recognizes \(L\)

For each \(a \in \Sigma\), one can pick \(n_a \in N\) which simulates \(\varphi(a)\), but the map \(a \mapsto n_a\) may not extend to a morphism from \(\operatorname{Tr}(\tilde{\Sigma})\) to \(N\).
Imitate these results for recognizable trace languages

Start with $L \subseteq \text{Tr}(\tilde{\Sigma})$, recognized by $\varphi : \text{Tr}(\tilde{\Sigma}) \rightarrow (X, M)$ or, equivalently, by a diamond automaton

Proofs collapse immediately

if $(X, M)$ is simulated by $(Y, N)$, it does not follow that $(Y, N)$ recognizes $L$

For each $a \in \Sigma$, one can pick $n_a \in N$ which simulates $\varphi(a)$, but the map $a \mapsto n_a$ may not extend to a morphism from $\text{Tr}(\tilde{\Sigma})$ to $N$

A dead-end...
Another track: asynchronous automata

- A model of automata that implements the distributed structure of $\tilde{\Sigma}$
A model of automata that implements the distributed structure of $\tilde{\Sigma}$

For each $i \in \mathcal{P}$, $Q_i = i$-states. Global states: $Q_\mathcal{P} = \prod_i Q_i$. For $a \in \Sigma$, a-states: $Q_a = \prod_{i \in \text{loc}(a)} Q_i$
Another track: asynchronous automata

- A model of automata that implements the distributed structure of $\tilde{\Sigma}$
- For each $i \in \mathcal{P}$, $Q_i = i$-states. Global states: $Q_\mathcal{P} = \prod_i Q_i$.
  For $a \in \Sigma$, $a$-states: $Q_a = \prod_{i \in \text{loc}(a)} Q_i$
- For each letter $a \in \Sigma$, the action of $a$ depends only on, and modifies only the $a$-states: $\delta_a: Q_a \rightarrow Q_a$, extended to a transformation $\Delta_a$ of $Q_\mathcal{P}$
Another track: asynchronous automata

- A model of automata that implements the distributed structure of $\tilde{\Sigma}$
- For each $i \in \mathcal{P}$, $Q_i = i$-states. Global states: $Q_{\mathcal{P}} = \prod_i Q_i$.
  For $a \in \Sigma$, $a$-states: $Q_a = \prod_{i \in \text{loc}(a)} Q_i$
- For each letter $a \in \Sigma$, the action of $a$ depends only on, and modifies only the $a$-states: $\delta_a : Q_a \rightarrow Q_a$, extended to a transformation $\Delta_a$ of $Q_{\mathcal{P}}$
- Global initial and accepting states
Another track: asynchronous automata

- A model of automata that implements the distributed structure of $\tilde{\Sigma}$
- For each $i \in \mathcal{P}$, $Q_i = i$-states. **Global states**: $Q_{\mathcal{P}} = \prod_i Q_i$. For $a \in \Sigma$, $a$-states: $Q_a = \prod_{i \in \text{loc}(a)} Q_i$
- For each letter $a \in \Sigma$, the action of $a$ depends only on, and modifies only the $a$-states: $\delta_a: Q_a \rightarrow Q_a$, extended to a transformation $\Delta_a$ of $Q_{\mathcal{P}}$
- Global initial and accepting states
- **Theorem** (Zielonka, 1987). Every recognizable language $L$ in $Tr(\tilde{\Sigma})$ is accepted by an asynchronous automaton
Another track: asynchronous automata

- A model of automata that implements the distributed structure of $\tilde{\Sigma}$
- For each $i \in \mathcal{P}$, $Q_i = i$-states. **Global states**: $Q_\mathcal{P} = \prod_i Q_i$.
- For $a \in \Sigma$, $a$-states: $Q_a = \prod_{i \in \text{loc}(a)} Q_i$
- For each letter $a \in \Sigma$, the action of $a$ depends only on, and modifies only the $a$-states: $\delta_a : Q_a \rightarrow Q_a$, extended to a transformation $\Delta_a$ of $Q_\mathcal{P}$
- **Global initial and accepting states**
- **Theorem** (Zielonka, 1987). Every recognizable language $L$ in $Tr(\tilde{\Sigma})$ is accepted by an asynchronous automaton
- And if $L$ is FO-definable? Can the asynchronous automaton be chosen to be aperiodic as well?
An algebraic theory of asynchronous automata

- Adapt algebra to the distributed setting: take into account the structure of $\tilde{\Sigma}$ — instead of ignoring it
An algebraic theory of asynchronous automata

- Adapt algebra to the distributed setting: take into account the structure of \( \tilde{\Sigma} \) — instead of ignoring it

- Asynchronous transformation monoid (atm):

  \[
  T = (\{ Q_i \}_{i \in \mathcal{P}}, M)
  \]
An algebraic theory of asynchronous automata

- Adapt algebra to the distributed setting: take into account the structure of $\tilde{\Sigma}$ — instead of ignoring it

- Asynchronous transformation monoid (atm):
  $T = (\{Q_i\}_{i \in \mathcal{P}}, M)$

- Asynchronous morphism: $\varphi: Tr(\tilde{\Sigma}) \to T$ such that $\varphi(a)$ is an $a$-map, for every letter $a \in \Sigma$ — that is: $\varphi(a)$ is the extension to $Q_\mathcal{P}$ of a transformation $Q_a \to Q_a$
An algebraic theory of asynchronous automata

- Adapt algebra to the distributed setting: take into account the structure of $\tilde{\Sigma}$ — instead of ignoring it

- **Asynchronous transformation monoid (atm):**
  \[ T = (\{Q_i\}_{i \in \mathcal{P}}, M) \]

- **Asynchronous morphism:** $\varphi : Tr(\tilde{\Sigma}) \longrightarrow T$ such that $\varphi(a)$ is an $a$-map, for every letter $a \in \Sigma$ — that is: $\varphi(a)$ is the extension to $Q_{\mathcal{P}}$ of a transformation $Q_a \longrightarrow Q_a$

- **Lemma** Let $T = (\{Q_i\}, M)$ be an atm. For each $a \in \Sigma$, let $\psi(a) \in M$ be an $a$-map. Then $\psi$ extends to a morphism $\psi : Tr(\tilde{\Sigma}) \longrightarrow T$
An algebraic theory of asynchronous automata

- Adapt algebra to the distributed setting: take into account the structure of $\tilde{\Sigma}$ — instead of ignoring it

- **Asynchronous transformation monoid (atm):**
  $$T = (\{Q_i\}_{i \in P}, M)$$

- **Asynchronous morphism:** $\varphi: Tr(\tilde{\Sigma}) \rightarrow T$ such that $\varphi(a)$ is an $a$-map, for every letter $a \in \Sigma$ — that is: $\varphi(a)$ is the extension to $Q_P$ of a transformation $Q_a \rightarrow Q_a$

- **Lemma** Let $T = (\{Q_i\}, M)$ be an atm. For each $a \in \Sigma$, let $\psi(a) \in M$ be an $a$-map. Then $\psi$ extends to a morphism $\psi: Tr(\tilde{\Sigma}) \rightarrow T$

- **Crucial examples:** the transition morphism $\varphi_A$ of an asynchronous automaton $A$ is an asynchronous morphism, and every asynchronous morphism $\varphi$ from $Tr(\tilde{\Sigma})$ to an atm $(\{Q_i\}, M)$, defines an asynchronous automaton $A_\varphi$ (for which $\varphi$ is the transition morphism)
Consider a DFA $\mathcal{A} = (Q, \delta, s_{\text{in}})$, with transition morphism $\varphi$. 
Consider a DFA $\mathcal{A} = (Q, \delta, s_{in})$, with transition morphism $\varphi$.

The associated *sequential transducer* $\sigma_{\mathcal{A}} : \Sigma^* \rightarrow (\Sigma \times Q)^*$ maps 1 to 1, and $ua$ to $\sigma_{\mathcal{A}}(ua) = \sigma_{\mathcal{A}}(u)(a, q)$, where $q = s_{in} \cdot u$. In other words, $q$ is what $\mathcal{A}$ knows before this $a$. 

Pascal Weil

Wreath product decompositions for trace languages
Consider a DFA $A = (Q, \delta, s_{in})$, with transition morphism $\varphi$.

The associated *sequential transducer* $\sigma_A : \Sigma^* \rightarrow (\Sigma \times Q)^*$ maps 1 to 1, and $ua$ to $\sigma_A(ua) = \sigma_A(u)(a, q)$, where $q = s_{in} \cdot u$. In other words, $q$ is what $A$ knows before this $a$.

$\sigma_A$ turns $A$ from an *accepting* to a *computing* device. It is “the most general function” computed by $A$. 

Consider a DFA $A = (Q, \delta, s_{in})$, with transition morphism $\varphi$.

The associated sequential transducer $\sigma_A: \Sigma^* \rightarrow (\Sigma \times Q)^*$ maps 1 to 1, and $ua$ to $\sigma_A(u)(a, q)$, where $q = s_{in} \cdot u$. In other words, $q$ is what $A$ knows before this $a$.

$\sigma_A$ turns $A$ from an accepting to a computing device. It is “the most general function” computed by $A$.

If $B$ is a DFA on alphabet $\Sigma \times Q$, the cascade product $A \circ B$ processes an input word $w$ through $A$, with output $\sigma_A(w)$, and then processes this output through $B$. 
Consider a DFA $A = (Q, \delta, s_{in})$, with transition morphism $\varphi$.

The associated sequential transducer $\sigma_A : \Sigma^* \rightarrow (\Sigma \times Q)^*$ maps 1 to 1, and $ua$ to $\sigma_A(ua) = \sigma_A(u)(a, q)$, where $q = s_{in} \cdot u$. In other words, $q$ is what $A$ knows before this $a$.

$\sigma_A$ turns $A$ from an accepting to a computing device. It is “the most general function” computed by $A$.

If $B$ is a DFA on alphabet $\Sigma \times Q$, the cascade product $A \circ B$ processes an input word $w$ through $A$, with output $\sigma_A(w)$, and then processes this output through $B$.

The wreath product is the translation on transformation monoids of the cascade product, which really is just the composition of the corresponding sequential transducers.
Cascade product of DFAs — Illustration

$A$ $\rightarrow_t$ $\sigma_A(t)$ $\rightarrow B$

$t$ $\rightarrow s$ $\rightarrow q$
Back to asynchronous automata, atm

- Same idea: turn an asynchronous automaton to a computing device. Consider an asynchronous automaton $\mathcal{A} = (\{Q_i\}, \{\delta_a\}, s_{in})$, with transition morphism $\varphi$
Back to asynchronous automata, atm

- Same idea: turn an asynchronous automaton to a computing device. Consider an asynch. automaton $\mathcal{A} = (\{Q_i\}, \{\delta_a\}, s_{in})$, with transition morphism $\varphi$

- The *asynchronous transducer* $\chi_A$ maps each trace $t = (E, \leq, \lambda)$ to a trace $(E, \leq, (\lambda, \mu))$ as follows: if $e \in E$ and $\lambda(e) = a$, if $t_e = \downarrow e$ is the causal past of $e$ (all events $e' < e$), then $\mu(e) = (s_{in} \cdot \varphi(t_e))_a$. That is: $\mu(e)$ is the local view (a-view) of the causal past of $e$ — exactly as in the seq’l case.
Same idea: turn an asynchronous automaton to a computing device. Consider an asynch. automaton $A = (\{Q_i\}, \{\delta_a\}, s_{in})$, with transition morphism $\varphi$

The\textit{ asynchronous transducer} $\chi_A$ maps each trace $t = (E, \leq, \lambda)$ to a trace $(E, \leq, (\lambda, \mu))$ as follows: if $e \in E$ and $\lambda(e) = a$, if $t_e = \downarrow e$ is the causal past of $e$ (all events $e' < e$), then $\mu(e) = (s_{in} \cdot \varphi(t_e))_a$. That is: $\mu(e)$ is the\textit{ local view (a-view) of the causal past of $e$} — exactly as in the seq’l case.

Again, the function $\chi_A$ is “the most general function” computed by $A$ using only local state information.
Local cascade product, asynchronous wreath product

- *Local cascade product* $A \circ_\ell B$: process input trace $t$ through $A$, output $\chi_A(t)$, and then process it through $B$
Local cascade product, asynchronous wreath product

- **Local cascade product** $A \circ_{\ell} B$: process input trace $t$ through $A$, output $\chi_A(t)$, and then process it through $B$

\begin{tikzpicture}
  
  
  
  \node (A) at (0,0) {$A$};
  \node (B) at (1,0) {$B$};

  \draw[->] (A) -- node[midway, above] {$\chi_A(t)$} (B);
  \draw[->] (A) -- node[midway, left] {$s$} (B);
  \draw[->] (A) -- node[midway, left] {$t$} (0,0);
  \draw[->] (B) -- node[midway, right] {$q$} (1,0);

\end{tikzpicture}

- **Asynchronous wreath product** of atm’s: algebraic translation of the local cascade product, that is, the composition of the corresponding asynchronous transducers
Local cascade product, asynchronous wreath product

- Local cascade product $A \circ_{ℓ} B$: process input trace $t$ through $A$, output $\chi_A(t)$, and then process it through $B$

$$
\begin{array}{c}
\xrightarrow{t} \quad A \quad \xrightarrow{\chi_A(t)} \quad B \\
\downarrow s \\
\downarrow q
\end{array}
$$

- Asynchronous wreath product of atm’s: algebraic translation of the local cascade product, that is, the composition of the corresponding asynchronous transducers

- Asynchronous wreath product principle: $L$ is accepted by $A \circ_{ℓ} B$ iff $L$ is a Boolean combination of languages accepted by $A$ and languages of the form $\chi_A^{-1}(K)$, where $K$ is accepted by $B$ (with structurally the same proof as in the seq’l case)
The theorem we would like is an imitation of the (sequential) Krohn-Rhodes theorem.
What about a Krohn-Rhodes theorem? 1/2

- The theorem we would like is an imitation of the (sequential) Krohn-Rhodes theorem.
- It would say this: *Every morphism from* $\text{Tr}(\tilde{\Sigma})$ *to a tm* $(X, M)$ *is simulated by an asynchronous wreath product of localized reset automata* $U_2[p]$ *and permutation automata* $G[p]$ *(p ∈ $\mathcal{P}$), G a simple group dividing* $M$. 
What about a Krohn-Rhodes theorem? 1/2

- The theorem we would like is an imitation of the (sequential) Krohn-Rhodes theorem
- It would say this: Every morphism from $Tr(\tilde{\Sigma})$ to a tm $(X, M)$ is simulated by an asynchronous wreath product of localized reset automata $U_2[p]$ and permutation automata $G[p]$ ($p \in \mathcal{P}$), $G$ a simple group dividing $M$
- Simulation. A morphism $\varphi: Tr(\tilde{\Sigma}) \rightarrow (X, M)$ is simulated by $\psi: Tr(\tilde{\Sigma}) \rightarrow (Y, N)$ if there exists a surjective map $f: Y \rightarrow X$ such that, for each $a \in \Sigma$, $y \in Y$, $f(y) \cdot \varphi(a) = f(y \cdot \psi(a))$
What about a Krohn-Rhodes theorem? 1/2

- The theorem we would like is an imitation of the (sequential) Krohn-Rhodes theorem
- It would say this: *Every morphism from* \( Tr(\tilde{\Sigma}) \) *to a tm* \((X, M)\) *is simulated by an asynchronous wreath product of localized reset automata* \( U_2[p] \) *and permutation automata* \( G[p] \) *\((p \in \mathcal{P})\),* \( G \) *a simple group dividing* \( M \)
- Simulation. A morphism \( \varphi : Tr(\tilde{\Sigma}) \rightarrow (X, M) \) is simulated by \( \psi : Tr(\tilde{\Sigma}) \rightarrow (Y, N) \) if there exists a surjective map \( f : Y \rightarrow X \) such that, for each \( a \in \Sigma \), \( y \in Y \),
  \[
  f(y) \cdot \varphi(a) = f(y \cdot \psi(a))
  \]
- \( U_2[p] \) is the distributed automaton \( (\{S_i\}, R) \) where \( S_p = \{1, 2\} \) and \( |S_i| = 1 \) if \( i \neq p \), and \( R \) consists of the identity, and the constant maps to global states 1 and 2
What about a Krohn-Rhodes theorem? 1/2

- The theorem we would like is an imitation of the (sequential) Krohn-Rhodes theorem.
- It would say this: Every morphism from $\text{Tr}(\tilde{\Sigma})$ to a tm $(X, M)$ is simulated by an asynchronous wreath product of localized reset automata $U_2[p]$ and permutation automata $G[p]$ ($p \in \mathcal{P}$), $G$ a simple group dividing $M$.
- Simulation. A morphism $\varphi : \text{Tr}(\tilde{\Sigma}) \rightarrow (X, M)$ is simulated by $\psi : \text{Tr}(\tilde{\Sigma}) \rightarrow (Y, N)$ if there exists a surjective map $f : Y \rightarrow X$ such that, for each $a \in \Sigma$, $y \in Y$, $f(y) \cdot \varphi(a) = f(y \cdot \psi(a))$.
- $U_2[p]$ is the distributed automaton $(\{S_i\}, R)$ where $S_p = \{1, 2\}$ and $|S_i| = 1$ if $i \neq p$, and $R$ consists of the identity, and the constant maps to global states 1 and 2.
- If $G$ is a group, $(G, G)$ is a tm, and $G[p]$ is its localization at process $p$: the set of $p$-states is $G$, the other sets of $i$-states are singletons.
Say that a distributed architecture \( \tilde{\Sigma} \) has the *asynchronous Krohn-Rhodes property* (aKR) if this ideal theorem holds for \( \tilde{\Sigma} \).
Say that a distributed architecture $\tilde{\Sigma}$ has the *asynchronous Krohn-Rhodes property* (aKR) if this ideal theorem holds for $\tilde{\Sigma}$

We do not know which distributed architectures are aKR (maybe all?... no known counterexample)
What about a Krohn-Rhodes theorem? 2/2

- Say that a distributed architecture $\tilde{\Sigma}$ has the *asynchronous Krohn-Rhodes property* (aKR) if this ideal theorem holds for $\tilde{\Sigma}$
- We do not know which distributed architectures are aKR (maybe all?... no known counterexample)
- **Theorem.** Acyclic architectures are aKR
Say that a distributed architecture $\tilde{\Sigma}$ has the *asynchronous Krohn-Rhodes property* (aKR) if this ideal theorem holds for $\tilde{\Sigma}$.

We do not know which distributed architectures are aKR (maybe all?... no known counterexample).

**Theorem.** Acyclic architectures are aKR.

Given $\tilde{\Sigma}$, let $\Gamma$ be the *communication graph*: vertices $= \mathcal{P}$; there is an edge from process $i$ to process $j$ if $\Sigma_i \cap \Sigma_j \neq \emptyset$. An *acyclic architecture* is one for which $\Gamma$ is acyclic (see Krishna & Muscholl 2013).
Weaker versions of the aKR property (for a distributed alphabet): *aperiodic aKR* or *aperiodic Zielonka* (i.e. every FO-definable language is accepted by an aperiodic asynchronous automaton). No counter-example known there either.
Weaker versions of the aKR property (for a distributed alphabet): aperiodic aKR or aperiodic Zielonka (i.e. every FO-definable language is accepted by an aperiodic asynchronous automaton). No counter-example known there either.

To work with FO-definable languages, over arbitrary distributed alphabets: $\text{LocTL}[Y_i \leq Y_j, Y_i, S_i]$. 
Weaker versions of the aKR property (for a distributed alphabet): *aperiodic aKR* or *aperiodic Zielonka* (i.e. every FO-definable language is accepted by an aperiodic asynchronous automaton). No counter-example known there either.

To work with FO-definable languages, over arbitrary distributed alphabets: $\text{LocTL}[Y_i \leq Y_j, Y_i, S_i]$.

$S_i$ and $Y_i$ are localized versions of Since and Yesterday, and $Y_i \leq Y_j$ compares the latest $i$- and $j$-events in the strict past of an event. More precisely...
First-order definable trace languages 2/4

\[(a \in \Sigma, i, j \in \mathcal{P})\]

Event formulas \[\alpha := a \mid \neg \alpha \mid \alpha \lor \alpha \mid Y_i \leq Y_j \mid Y_i \alpha \mid \alpha S_i \alpha\]

Trace formulas \[\beta := \exists i \alpha \mid \neg \beta \mid \beta \lor \beta\]
First-order definable trace languages 2/4

◮ \((a \in \Sigma, \; i, j \in \mathcal{P})\)

Event formulas \(\alpha := a \mid \neg \alpha \mid \alpha \lor \alpha \mid Y_i \leq Y_j \mid Y_i \alpha \mid \alpha S_i \alpha\)

Trace formulas \(\beta := \exists_i \alpha \mid \neg \beta \mid \beta \lor \beta\)

◮ \((e_i = \text{latest } i\text{-event in the strict past of } e, \text{ if it exists})\)

\[ t, e \models a \quad \text{if } \lambda(e) = a \]

\[ t, e \models Y_i \leq Y_j \quad \text{if } e_i, e_j \text{ exist, and } e_i \leq e_j \]

\[ t, e \models Y_i \alpha \quad \text{if } e_i \text{ exists, and } t, e_i \models \alpha \]

\[ t, e \models \alpha S_i \alpha' \quad \text{if } e \in E_i, \exists f \in E_i \text{ s.t. } f < e \text{ and } t, f \models \alpha' \]

and, \(\forall g \in E_i, \; f < g < e \implies t, g \models \alpha\)

\[ t \models \exists_i \alpha \quad \text{if } e = \max E_i \text{ exists, and } t, e \models \alpha \]
Proposition. \( \text{LocTL}[Y_i \leq Y_j, Y_i, S_i] \) is expressively complete (uses Diekert & Gastin 2006)
Proposition. \( \text{LocTL}[Y_i \leq Y_j, Y_i, S_i] \) is expressively complete (uses Diekert & Gastin 2006)

Theorem. \( L \) is \( \text{LocTL}[S_i] \)-definable iff \( L \) is accepted by an asynchronous wreath product of localized resets
Proposition. \( \text{LocTL}[Y_i \leq Y_j, Y_i, S_i] \) is expressively complete (uses Diekert & Gastin 2006)

Theorem. \( L \) is \( \text{LocTL}[S_i] \)-definable iff \( L \) is accepted by an asynchronous wreath product of localized resets

Corollary. Over an aKR distributed alphabet, \( \text{LocTL}[S_i] \) is expressively complete
For general distributed alphabets, what we need beside \( \text{LocTL}[S_i] \), is the truth values of the constants \( Y_i \leq Y_j \). Let \( \theta^Y \) be the function computing these truth values.
For general distributed alphabets, what we need beside LocTL[S_i], is the truth values of the constants Y_i ≤ Y_j. Let θ^Y be the function computing these truth values.

**Theorem.** If $\mathcal{A}$ is an asynchronous automaton computing $\theta^Y$, then FO-definable languages are accepted by a local cascade product $\mathcal{A} \circ_\ell \mathcal{B}$, where $\mathcal{B}$ accepts a language that is LocTL[S_i]-definable.
For general distributed alphabets, what we need beside \( \text{LocTL}[S_i] \), is the truth values of the constants \( Y_i \leq Y_j \). Let \( \theta^Y \) be the function computing these truth values.

**Theorem.** If \( \mathcal{A} \) is an asynchronous automaton computing \( \theta^Y \), then FO-definable languages are accepted by a local cascade product \( \mathcal{A} \circ_{\ell} \mathcal{B} \), where \( \mathcal{B} \) accepts a language that is LocTL\([S_i]\)-definable.

**Corollary.** FO-definable trace languages are accepted by a local cascade product of \( \mathcal{A} \) with localized resets.
For general distributed alphabets, what we need beside LocTL[S_i], is the truth values of the constants Y_i ≤ Y_j. Let \( \theta^Y \) be the function computing these truth values.

**Theorem.** If \( \mathcal{A} \) is an asynchronous automaton computing \( \theta^Y \), then FO-definable languages are accepted by a local cascade product \( \mathcal{A} \circ_\ell \mathcal{B} \), where \( \mathcal{B} \) accepts a language that is LocTL[S_i]-definable.

**Corollary.** FO-definable trace languages are accepted by a local cascade product of \( \mathcal{A} \) with localized resets.

Are we closer to an aperiodic Zielonka theorem?
Asynchronous automata computing $\theta^Y$

- The *gossip automaton* $\mathcal{G}$ (Mukund, Sohoni, 1997) is an asynchronous automaton which computes $\theta^Y$
The gossip automaton $\mathcal{G}$ (Mukund, Sohoni, 1997) is an asynchronous automaton which computes $\theta^Y$

But $\mathcal{G}$ is not aperiodic in general! Let $\mathcal{P} = \{1, 2, 3\}$, $\text{loc}(a) = \{1, 2\}$, $\text{loc}(b) = \{2, 3\}$, $\text{loc}(c) = \{1, 3\}$. So $\text{Tr}(\tilde{\Sigma}) = \Sigma^*$, impl’d on 3 processes. Yet no aperiodic asynchronous automaton computes the truth value of $Y_1 \leq Y_3$
The *gossip automaton* $G$ (Mukund, Sohoni, 1997) is an asynchronous automaton which computes $\theta^Y$.

But $G$ is not aperiodic in general! Let $\mathcal{P} = \{1, 2, 3\}$, $\text{loc}(a) = \{1, 2\}$, $\text{loc}(b) = \{2, 3\}$, $\text{loc}(c) = \{1, 3\}$. So $Tr(\tilde{\Sigma}) = \Sigma^*$, impl’d on 3 processes. Yet no aperiodic asynchronous automaton computes the truth value of $Y_1 \leq Y_3$.

Thus our description of FO-definable, as accepted by a local cascade product of $G$ with localized resets cannot be a characterization.
Restricted cascade product with the gossip automaton

- No characterization of FO-definable trace languages in terms of local cascade products of localized resets?...
No characterization of FO-definable trace languages in terms of local cascade products of localized resets...

$G$ is an overkill: restrict the definition of the local cascade product to pass on only the information computed by $\theta^Y$
Restricted cascade product with the gossip automaton

- No characterization of FO-definable trace languages in terms of local cascade products of localized resets?

- $\mathcal{G}$ is an overkill: restrict the definition of the local cascade product to pass on only the information computed by $\theta^Y$

- Hence: yet another (!) characterization of FO-definability
No characterization of FO-definable trace languages in terms of local cascade products of localized resets?

\[ G \] is an overkill: restrict the definition of the local cascade product to pass on only the information computed by \( \theta^Y \)

\[ \begin{array}{c}
\text{t} \\
\downarrow \\
G \\
\theta^Y(t) \\
\downarrow \\
A \\
q
\end{array} \]

Hence: yet another (!) characterization of FO-definability

Question (open): For which distributed alphabets do we have an aperiodic atm computing \( \theta^Y \)? For these \( \tilde{\Sigma} \), the aperiodic Zielonka theorem holds
Another device: Define a global state labeling function of $\mathcal{A}$ (instead of local state labeling function as in the asynchronous transducer $\chi_{\mathcal{A}}$). For $t = (E, \leq, \lambda) \in Tr(\tilde{\Sigma})$,

$\zeta_{\mathcal{A}}(t) = (E, \leq, (\lambda, \mu))$, where $\mu$ is as follows. If $e \in E$ and $t_e = \text{strict past of } e$, $\mu(e) = s_{\text{in}} \cdot \varphi(t_e)$: the global state of $\mathcal{A}$ after reading $t_e$. 
Global cascade sequences

- Another device: Define a global state labeling function of $\mathcal{A}$ (instead of local state labeling function as in the asynchronous transducer $\chi_{\mathcal{A}}$). For $t = (E, \leq, \lambda) \in Tr(\tilde{\Sigma})$, $\zeta_{\mathcal{A}}(t) = (E, \leq, (\lambda, \mu))$, where $\mu$ is as follows. If $e \in E$ and $t_e = \text{strict past of } e$, $\mu(e) = s_{in} \cdot \varphi(t_e)$: the global state of $\mathcal{A}$ after reading $t_e$.

- Global cascade sequence $(\mathcal{A}_1, \ldots, \mathcal{A}_n)$: composition of the $\zeta_{\mathcal{A}_i}$, with global accepting condition (a tuple of global states of the $\mathcal{A}_i$).
Another device: Define a *global state labeling function* of $\mathcal{A}$ (instead of *local* state labeling function as in the asynchronous transducer $\chi_\mathcal{A}$). For $t = (E, \leq, \lambda) \in Tr(\tilde{\Sigma})$, $\zeta_\mathcal{A}(t) = (E, \leq, (\lambda, \mu))$, where $\mu$ is as follows. If $e \in E$ and $t_e = \text{strict past of } e$, $\mu(e) = s_{in} \cdot \phi(t_e)$: the global state of $\mathcal{A}$ after reading $t_e$

Global cascade sequence ($\mathcal{A}_1, \ldots, \mathcal{A}_n$): composition of the $\zeta_\mathcal{A}_i$, with global accepting condition (a tuple of global states of the $\mathcal{A}_i$)

**Theorem.** A trace language is FO-definable iff it is accepted by a global cascade sequence of localized resets
Global cascade sequences

- Another device: Define a *global state labeling function* of $A$ (instead of *local* state labeling function as in the asynchronous transducer $\chi_A$). For $t = (E, \leq, \lambda) \in Tr(\tilde{\Sigma})$, $\zeta_A(t) = (E, \leq, (\lambda, \mu))$, where $\mu$ is as follows. If $e \in E$ and $t_e = \text{strict past of } e$, $\mu(e) = s_{\text{in}} \cdot \varphi(t_e)$: the global state of $A$ after reading $t_e$

- Global cascade sequence $(A_1, \ldots, A_n)$: composition of the $\zeta_{A_i}$, with global accepting condition (a tuple of global states of the $A_i$)

- **Theorem.** A trace language is FO-definable iff it is accepted by a global cascade sequence of localized resets

- Implementation of global cascade sequences: possible as an asynchronous automaton, involving a restricted local cascade product of the gossip automaton and the local cascade product of the $A_i$
Thank you for your attention!