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. $\mathscr{P}$, set of processes

Distributed alphabet, trace monoid

- Distributed behaviors. $\mathscr{P}$, set of processes
- $\widetilde{\Sigma}$ distributed alphabet: $\mathrm{loc}\colon \Sigma \longrightarrow 2^{\mathscr{P}}$; letters a, b are *independent* if $\mathrm{loc}(a) \cap \mathrm{loc}(b) = \emptyset$

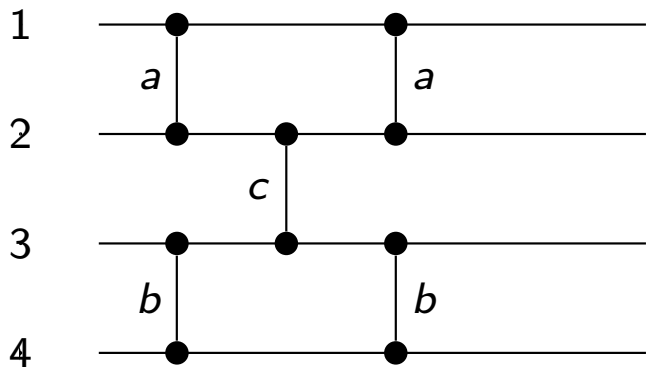
Distributed alphabet, trace monoid

- Distributed behaviors. $\mathscr{P}$, set of processes
- $\widetilde{\Sigma}$ distributed alphabet: $\mathrm{loc}\colon \Sigma \longrightarrow 2^{\mathscr{P}}$; letters a, b are *independent* if $\mathrm{loc}(a) \cap \mathrm{loc}(b) = \emptyset$
- $Tr(\widetilde{\Sigma})$: quotient of Σ^* by the relation $ab = ba$ if a, b are independent. Projection morphism: $\pi\colon \Sigma^* \longrightarrow Tr(\widetilde{\Sigma})$

Distributed alphabet, trace monoid

- Distributed behaviors. $\mathscr{P}$, set of processes
- $\widetilde{\Sigma}$ distributed alphabet: $\mathrm{loc}\colon \Sigma \longrightarrow 2^{\mathscr{P}}$; letters a, b are *independent* if $\mathrm{loc}(a) \cap \mathrm{loc}(b) = \emptyset$
- $Tr(\widetilde{\Sigma})$: quotient of Σ^* by the relation $ab = ba$ if a, b are independent. Projection morphism: $\pi\colon \Sigma^* \longrightarrow Tr(\widetilde{\Sigma})$
- poset representation of a trace: $t = (E, \leq, \lambda)$; $E = \textit{events}$

Let $\mathrm{loc}(a) = \{1, 2\}$, $\mathrm{loc}(b) = \{3, 4\}$, $\mathrm{loc}(c) = \{2, 3\}$, $t = abcba$



Recognizable trace languages

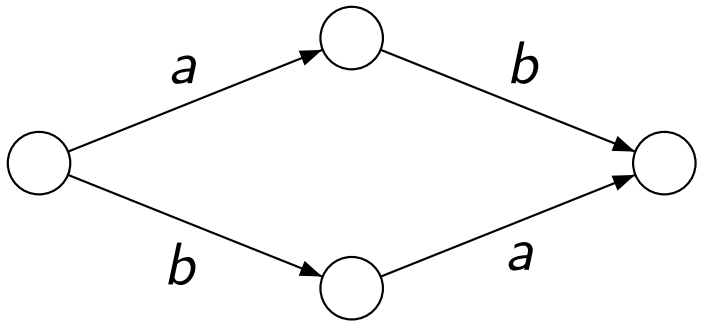
- Defn: $L \subseteq Tr(\widetilde{\Sigma})$ is recognizable if $\pi^{-1}(L) \subseteq \Sigma^*$ (the language of all linearizations of the traces in L) is recognizable

Recognizable trace languages

- Defn: $L \subseteq Tr(\widetilde{\Sigma})$ 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 φ from $Tr(\widetilde{\Sigma})$ 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)

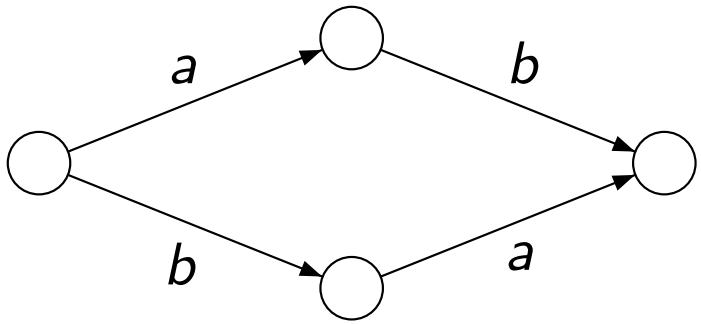
Recognizable trace languages

- Defn: $L \subseteq Tr(\widetilde{\Sigma})$ 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 φ from $Tr(\widetilde{\Sigma})$ 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



Recognizable trace languages

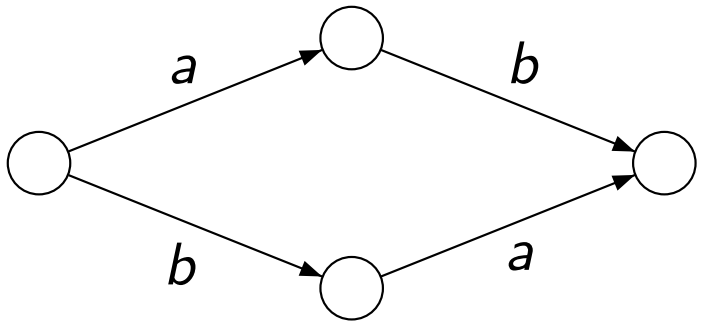
- Defn: $L \subseteq Tr(\widetilde{\Sigma})$ is recognizable if $\pi^{-1}(L) \subseteq \Sigma^*$ (the language of all linearizations of the traces in L) is recognizable
- Equivalently: there exists a morphism φ from $Tr(\widetilde{\Sigma})$ to a finite transformation monoid (X, M) such that $L = \varphi^{-1}(M_{s,F})$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



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

Recognizable trace languages

- Defn: $L \subseteq Tr(\widetilde{\Sigma})$ 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 φ from $Tr(\widetilde{\Sigma})$ 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 $\leq$)
- Characterization: not so simple... Restricted rat'l expressions

An algebraic automata theory for trace languages? 1/3

- Recognizable = finite monoid = MSO-definable

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)

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

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

An algebraic automata theory for trace languages? 2/3

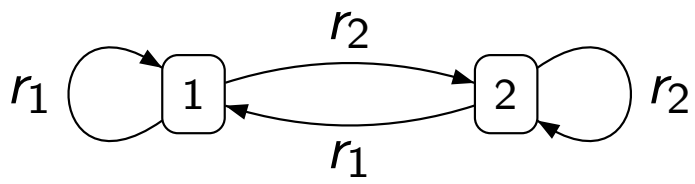
- Composition / decomposition for word languages: wreath product of transformation monoids / cascade product of DFAs

An algebraic automata theory for trace languages? 2/3

- 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

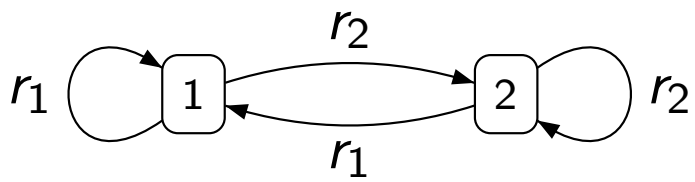
An algebraic automata theory for trace languages? 2/3

- 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: $U_2 =$



An algebraic automata theory for trace languages? 2/3

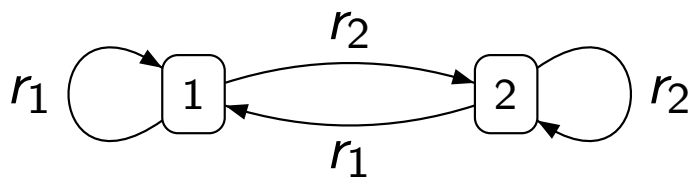
- Composition / decomposition for word languages: wreath product of transformation monoids / cascade product of DFAs
- Krohn-Rhodes theorem: every finite tm (X, M) is simulated by a wreath product of copies of the 2-state reset tm and of transformation monoids of the form (G, G), where G is a simple group dividing M
- 2-state reset tm: $U_2 =$



- Special case for aperiodic transformation monoids: wreath product of copies of U_2

An algebraic automata theory for trace languages? 2/3

- 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
- Useful for inductive proofs, see eg Straubing's proofs of the equivalence of aperiodic with FO or with LTL

An algebraic automata theory for trace languages? 3/3

- 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 $L \subseteq Tr(\widetilde{\Sigma})$, recognized by $\varphi\colon Tr(\widetilde{\Sigma}) \longrightarrow (X, M)$ or, equivalently, by a diamond automaton

An algebraic automata theory for trace languages? 3/3

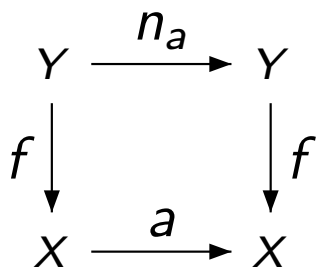
- Imitate these results for recognizable trace languages
- Start with $L \subseteq Tr(\widetilde{\Sigma})$, recognized by $\varphi\colon Tr(\widetilde{\Sigma}) \longrightarrow (X, M)$ or, equivalently, by a diamond automaton
- Proofs collapse immediately

An algebraic automata theory for trace languages? 3/3

- Imitate these results for recognizable trace languages
- Start with $L \subseteq Tr(\widetilde{\Sigma})$, recognized by $\varphi\colon Tr(\widetilde{\Sigma}) \longrightarrow (X, M)$ 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

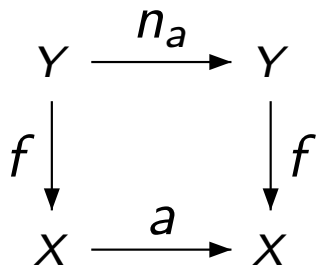
An algebraic automata theory for trace languages? 3/3

- Imitate these results for recognizable trace languages
- Start with $L \subseteq Tr(\widetilde{\Sigma})$, recognized by $\varphi\colon Tr(\widetilde{\Sigma}) \longrightarrow (X, M)$ 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
- 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 $Tr(\widetilde{\Sigma})$ to N



An algebraic automata theory for trace languages? 3/3

- Imitate these results for recognizable trace languages
- Start with $L \subseteq Tr(\widetilde{\Sigma})$, recognized by $\varphi\colon Tr(\widetilde{\Sigma}) \longrightarrow (X, M)$ 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
- 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 $Tr(\widetilde{\Sigma})$ to N



- A dead-end...

Another track: asynchronous automata

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

Another track: asynchronous automata

- A model of automata that implements the distributed structure of $\widetilde{\Sigma}$
- For each $i \in \mathscr{P}$, $Q_i = i$-*states*. *Global states*: $Q_{\mathscr{P}} = \prod_i Q_i$. For $a \in \Sigma$, *a-states*: $Q_a = \prod_{i \in \mathrm{loc}(a)} Q_i$

Another track: asynchronous automata

- A model of automata that implements the distributed structure of $\widetilde{\Sigma}$
- For each $i \in \mathscr{P}$, $Q_i = i$-*states*. *Global states*: $Q_{\mathscr{P}} = \prod_i Q_i$. For $a \in \Sigma$, *a-states*: $Q_a = \prod_{i \in \mathrm{loc}(a)} Q_i$
- For each letter $a \in \Sigma$, the action of a depends only on, and modifies only the a-states: $\delta_a : Q_a \longrightarrow Q_a$, extended to a transformation Δ_a of $Q_{\mathscr{P}}$

Another track: asynchronous automata

- A model of automata that implements the distributed structure of $\widetilde{\Sigma}$
- For each $i \in \mathscr{P}$, $Q_i = i$-*states*. *Global states*: $Q_{\mathscr{P}} = \prod_i Q_i$. For $a \in \Sigma$, *a-states*: $Q_a = \prod_{i \in \mathrm{loc}(a)} Q_i$
- For each letter $a \in \Sigma$, the action of a depends only on, and modifies only the a-states: $\delta_a : Q_a \longrightarrow Q_a$, extended to a transformation Δ_a of $Q_{\mathscr{P}}$
- Global initial and accepting states

Another track: asynchronous automata

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

Another track: asynchronous automata

- A model of automata that implements the distributed structure of $\widetilde{\Sigma}$
- For each $i \in \mathscr{P}$, $Q_i = i$-*states*. *Global states*: $Q_{\mathscr{P}} = \prod_i Q_i$. For $a \in \Sigma$, *a-states*: $Q_a = \prod_{i \in \mathrm{loc}(a)} Q_i$
- For each letter $a \in \Sigma$, the action of a depends only on, and modifies only the a-states: $\delta_a : Q_a \longrightarrow Q_a$, extended to a transformation Δ_a of $Q_{\mathscr{P}}$
- Global initial and accepting states
- **Theorem** (Zielonka, 1987). Every recognizable language L in $Tr(\widetilde{\Sigma})$ 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 $\widetilde{\Sigma}$ — instead of ignoring it

An algebraic theory of asynchronous automata

- Adapt algebra to the distributed setting: take into account the structure of $\widetilde{\Sigma}$ — instead of ignoring it
- *Asynchronous transformation monoid* (atm): $T = (\{Q_i\}_{i \in \mathscr{P}}, M)$

An algebraic theory of asynchronous automata

- Adapt algebra to the distributed setting: take into account the structure of $\widetilde{\Sigma}$ — instead of ignoring it
- *Asynchronous transformation monoid* (atm): $T = (\{Q_i\}_{i \in \mathscr{P}}, M)$
- *Asynchronous morphism*: $\varphi : Tr(\widetilde{\Sigma}) \longrightarrow T$ such that $\varphi(a)$ is an *a-map*, for every letter $a \in \Sigma$ — that is: $\varphi(a)$ is the extension to $Q_{\mathscr{P}}$ of a transformation $Q_a \longrightarrow Q_a$

An algebraic theory of asynchronous automata

- Adapt algebra to the distributed setting: take into account the structure of $\widetilde{\Sigma}$ — instead of ignoring it
- *Asynchronous transformation monoid* (atm): $T = (\{Q_i\}_{i \in \mathscr{P}}, M)$
- *Asynchronous morphism*: $\varphi \colon Tr(\widetilde{\Sigma}) \longrightarrow T$ such that $\varphi(a)$ is an *a-map*, for every letter $a \in \Sigma$ — that is: $\varphi(a)$ is the extension to $Q_{\mathscr{P}}$ 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 ψ extends to a morphism $\psi \colon Tr(\widetilde{\Sigma}) \longrightarrow T$

An algebraic theory of asynchronous automata

- Adapt algebra to the distributed setting: take into account the structure of $\widetilde{\Sigma}$ — instead of ignoring it
- *Asynchronous transformation monoid* (atm): $T = (\{Q_i\}_{i \in \mathscr{P}}, M)$
- *Asynchronous morphism*: $\varphi \colon Tr(\widetilde{\Sigma}) \longrightarrow T$ such that $\varphi(a)$ is an *a-map*, for every letter $a \in \Sigma$ — that is: $\varphi(a)$ is the extension to $Q_{\mathscr{P}}$ 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 ψ extends to a morphism $\psi \colon Tr(\widetilde{\Sigma}) \longrightarrow T$
- Crucial examples: the transition morphism $\varphi_{\mathcal{A}}$ of an asynchronous automaton $\mathcal{A}$ is an asynchronous morphism, and every asynchronous morphism φ from $Tr(\widetilde{\Sigma})$ to an atm $(\{Q_i\}, M)$, defines an asynchronous automaton $\mathcal{A}_\varphi$ (for which φ is the transition morphism)

Cascade product of DFAs — Digression

- Consider a DFA $\mathcal{A} = (Q, \delta, s_{\text{in}})$, with transition morphism φ

Cascade product of DFAs — Digression

- Consider a DFA $\mathcal{A} = (Q, \delta, s_{\text{in}})$, with transition morphism φ
- The associated *sequential transducer* $\sigma_{\mathcal{A}} \colon \Sigma^* \longrightarrow (\Sigma \times Q)^*$ maps 1 to 1, and ua to $\sigma_{\mathcal{A}}(ua) = \sigma_{\mathcal{A}}(u)(a, q)$, where $q = s_{\text{in}} \cdot u$. In other words, q is what $\mathcal{A}$ knows *before* this a

Cascade product of DFAs — Digression

- Consider a DFA $\mathcal{A} = (Q, \delta, s_{\text{in}})$, with transition morphism φ
- The associated *sequential transducer* $\sigma_{\mathcal{A}} \colon \Sigma^* \longrightarrow (\Sigma \times Q)^*$ maps 1 to 1, and ua to $\sigma_{\mathcal{A}}(ua) = \sigma_{\mathcal{A}}(u)(a, q)$, where $q = s_{\text{in}} \cdot u$. In other words, q is what $\mathcal{A}$ knows *before* this a
- $\sigma_{\mathcal{A}}$ turns $\mathcal{A}$ from an *accepting* to a *computing* device. It is "the most general function" computed by $\mathcal{A}$

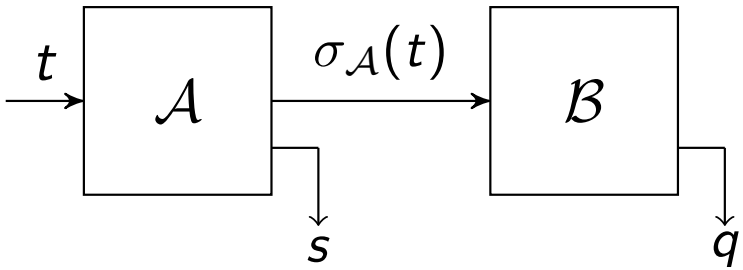
Cascade product of DFAs — Digression

- Consider a DFA $\mathcal{A} = (Q, \delta, s_{\text{in}})$, with transition morphism φ
- The associated *sequential transducer* $\sigma_{\mathcal{A}} \colon \Sigma^* \longrightarrow (\Sigma \times Q)^*$ maps 1 to 1, and ua to $\sigma_{\mathcal{A}}(ua) = \sigma_{\mathcal{A}}(u)(a, q)$, where $q = s_{\text{in}} \cdot u$. In other words, q is what $\mathcal{A}$ knows *before* this a
- $\sigma_{\mathcal{A}}$ turns $\mathcal{A}$ from an *accepting* to a *computing* device. It is "the most general function" computed by $\mathcal{A}$
- If $\mathcal{B}$ is a DFA on alphabet $\Sigma \times Q$, the *cascade product* $\mathcal{A} \circ \mathcal{B}$ processes an input word w through $\mathcal{A}$, with output $\sigma_{\mathcal{A}}(w)$, and then processes this output through $\mathcal{B}$

Cascade product of DFAs — Digression

- Consider a DFA $\mathcal{A} = (Q, \delta, s_{\text{in}})$, with transition morphism φ
- The associated *sequential transducer* $\sigma_{\mathcal{A}} \colon \Sigma^* \longrightarrow (\Sigma \times Q)^*$ maps 1 to 1, and ua to $\sigma_{\mathcal{A}}(ua) = \sigma_{\mathcal{A}}(u)(a, q)$, where $q = s_{\text{in}} \cdot u$. In other words, q is what $\mathcal{A}$ knows *before* this a
- $\sigma_{\mathcal{A}}$ turns $\mathcal{A}$ from an *accepting* to a *computing* device. It is "the most general function" computed by $\mathcal{A}$
- If $\mathcal{B}$ is a DFA on alphabet $\Sigma \times Q$, the *cascade product* $\mathcal{A} \circ \mathcal{B}$ processes an input word w through $\mathcal{A}$, with output $\sigma_{\mathcal{A}}(w)$, and then processes this output through $\mathcal{B}$
- 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



Back to asynchronous automata, atm

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

Back to asynchronous automata, atm

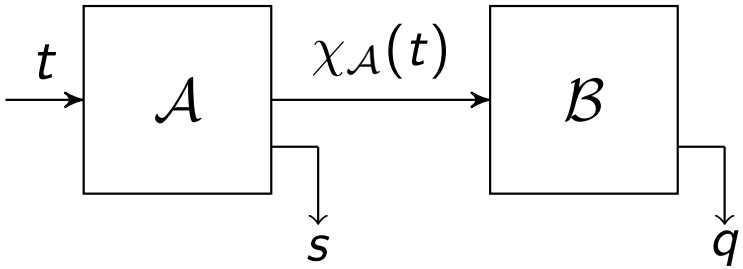
- Same idea: turn an asynchronous automaton to a computing device. Consider an asynch. automaton $\mathcal{A} = (\{Q_i\}, \{\delta_a\}, s_{\text{in}})$, with transition morphism φ
- The *asynchronous transducer* $\chi_{\mathcal{A}}$ maps each trace $t = (E, \leq, \lambda)$ to a trace $(E, \leq, (\lambda,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

Back to asynchronous automata, atm

- Same idea: turn an asynchronous automaton to a computing device. Consider an asynch. automaton $\mathcal{A} = (\{Q_i\}, \{\delta_a\}, s_{\text{in}})$, with transition morphism φ
- The *asynchronous transducer* $\chi_{\mathcal{A}}$ 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 $\mu(e) = (s_{\text{in}} \cdot \varphi(t_e))_a$. That is: $\mu(e)$ *is the local view (a-view) of the causal past of* e — exactly as in the seq'l case
- Again, the function $\chi_{\mathcal{A}}$ is "the most general function" computed by $\mathcal{A}$ using only local state information

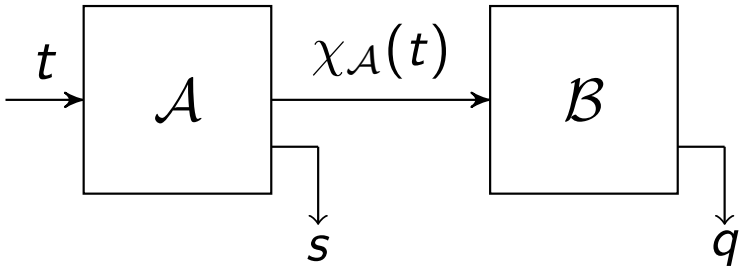
Local cascade product, asynchronous wreath product

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



Local cascade product, asynchronous wreath product

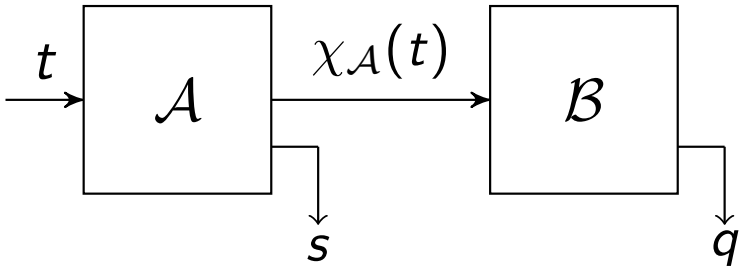
- *Local cascade product* $\mathcal{A} \circ_\ell \mathcal{B}$: process input trace t through $\mathcal{A}$, output $\chi_{\mathcal{A}}(t)$, and then process it through $\mathcal{B}$



- *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* $\mathcal{A} \circ_\ell \mathcal{B}$: process input trace t through $\mathcal{A}$, output $\chi_{\mathcal{A}}(t)$, and then process it through $\mathcal{B}$



- *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 $\mathcal{A} \circ_\ell \mathcal{B}$ iff L is a Boolean combination of languages accepted by $\mathcal{A}$ and languages of the form $\chi_{\mathcal{A}}^{-1}(K)$, where K is accepted by $\mathcal{B}$ (with structurally the same proof as in the seq'l case)

What about a Krohn-Rhodes theorem? 1/2

- 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* $Tr(\widetilde{\Sigma})$ *to a tm* (X, M) *is simulated by an asynchronous wreath product of localized reset automata* $U_2[p]$ *and permutation automata* $G[p]$ $(p \in \mathscr{P})$, 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* $Tr(\widetilde{\Sigma})$ *to a tm* (X, M) *is simulated by an asynchronous wreath product of localized reset automata* $U_2[p]$ *and permutation automata* $G[p]$ $(p \in \mathscr{P})$, G *a simple group dividing* M
- Simulation. A morphism $\varphi\colon Tr(\widetilde{\Sigma}) \longrightarrow (X, M)$ is simulated by $\psi\colon Tr(\widetilde{\Sigma}) \longrightarrow (Y, N)$ if there exists a surjective map $f\colon Y \longrightarrow X$ 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* $Tr(\widetilde{\Sigma})$ *to a tm* (X, M) *is simulated by an asynchronous wreath product of localized reset automata* $U_2[p]$ *and permutation automata* $G[p]$ $(p \in \mathscr{P})$, G *a simple group dividing* M
- Simulation. A morphism $\varphi\colon Tr(\widetilde{\Sigma}) \longrightarrow (X, M)$ is simulated by $\psi\colon Tr(\widetilde{\Sigma}) \longrightarrow (Y, N)$ if there exists a surjective map $f\colon Y \longrightarrow X$ 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* $Tr(\widetilde{\Sigma})$ *to a tm* (X, M) *is simulated by an asynchronous wreath product of localized reset automata* $U_2[p]$ *and permutation automata* $G[p]$ $(p \in \mathscr{P})$, G *a simple group dividing* M
- Simulation. A morphism $\varphi\colon Tr(\widetilde{\Sigma}) \longrightarrow (X, M)$ is simulated by $\psi\colon Tr(\widetilde{\Sigma}) \longrightarrow (Y, N)$ if there exists a surjective map $f\colon Y \longrightarrow X$ 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

What about a Krohn-Rhodes theorem? 2/2

- Say that a distributed architecture $\widetilde{\Sigma}$ has the *asynchronous Krohn-Rhodes property* (aKR) if this ideal theorem holds for $\widetilde{\Sigma}$

What about a Krohn-Rhodes theorem? 2/2

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

What about a Krohn-Rhodes theorem? 2/2

- Say that a distributed architecture $\widetilde{\Sigma}$ has the *asynchronous Krohn-Rhodes property* (aKR) if this ideal theorem holds for $\widetilde{\Sigma}$
- We do not know which distributed architectures are aKR (maybe all?... no known counterexample)
- **Theorem**. Acyclic architectures are aKR
- Given $\widetilde{\Sigma}$, let Γ be the *communication graph*: vertices $= \mathscr{P}$; 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 Γ is acyclic (see Krishna & Muscholl 2013)

First-order definable trace languages 1/4

- 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

First-order definable trace languages 1/4

- 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: LocTL$[\mathsf{Y}_i \leq \mathsf{Y}_j, \mathsf{Y}_i, \mathsf{S}_i]$.

First-order definable trace languages 1/4

- 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: LocTL$[\mathsf{Y}_i \leq \mathsf{Y}_j, \mathsf{Y}_i, \mathsf{S}_i]$.
- S_i and Y_i are localized versions of Since and Yesterday, and $\mathsf{Y}_i \leq \mathsf{Y}_j$ compares the latest i- and j-events in the strict past of an event. More precisely...

First-order definable trace languages 2/4

- $(a \in \Sigma,\ i, j \in \mathscr{P})$

Event formulas $\quad \alpha := a \mid \neg\alpha \mid \alpha \vee \alpha \mid \mathsf{Y}_i \leq \mathsf{Y}_j \mid \mathsf{Y}_i\,\alpha \mid \alpha\,\mathsf{S}_i\,\alpha$

Trace formulas $\quad \beta := \exists_i \alpha \mid \neg\beta \mid \beta \vee \beta$

First-order definable trace languages 2/4

- $(a \in \Sigma,\ i, j \in \mathscr{P})$

Event formulas $\quad \alpha := a \mid \neg\alpha \mid \alpha \vee \alpha \mid \mathsf{Y}_i \leq \mathsf{Y}_j \mid \mathsf{Y}_i\,\alpha \mid \alpha\,\mathsf{S}_i\,\alpha$

Trace formulas $\quad \beta := \exists_i \alpha \mid \neg\beta \mid \beta \vee \beta$

- (e_i = latest i-event in the strict past of e, if it exists)

$t, e \models a$ if $\lambda(e) = a$

$t, e \models \mathsf{Y}_i \leq \mathsf{Y}_j$ if e_i, e_j exist, and $e_i \leq e_j$

$t, e \models \mathsf{Y}_i\,\alpha$ if e_i exists, and $t, e_i \models \alpha$

$t, e \models \alpha\,\mathsf{S}_i\,\alpha'$ if $e \in E_i$, $\exists f \in E_i$ s.t. $f < e$ and $t, f \models \alpha'$
and, $\forall g \in E_i$, $f < g < e \Longrightarrow t, g \models \alpha$

$t \models \exists_i \alpha$ if $e = \max E_i$ exists, and $t, e \models \alpha$

First-order definable trace languages 3/4

- **Proposition**. LocTL$[Y_i \leq Y_j, Y_i, S_i]$ is expressively complete (uses Diekert & Gastin 2006)

First-order definable trace languages 3/4

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

First-order definable trace languages 3/4

- **Proposition**. LocTL$[Y_i \leq Y_j, Y_i, S_i]$ is expressively complete (uses Diekert & Gastin 2006)
- **Theorem**. L is LocTL$[S_i]$-definable iff L is accepted by an asynchronous wreath product of localized resets
- **Corollary**. Over an aKR distributed alphabet, LocTL$[S_i]$ is expressively complete

First-order definable trace languages 4/4

- For general distributed alphabets, what we need beside $\mathrm{LocTL}[\mathrm{S}_i]$, is the truth values of the constants $\mathrm{Y}_i \leq \mathrm{Y}_j$. Let θ^Y be the function computing these truth values

First-order definable trace languages 4/4

- For general distributed alphabets, what we need beside LocTL[S_i], is the truth values of the constants $Y_i \leq Y_j$. Let θ^Y be the function computing these truth values
- **Theorem**. If $\mathcal{A}$ is an asynchronous automaton computing θ^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

First-order definable trace languages 4/4

- For general distributed alphabets, what we need beside $\mathrm{LocTL}[\mathsf{S}_i]$, is the truth values of the constants $\mathsf{Y}_i \leq \mathsf{Y}_j$. Let θ^Y be the function computing these truth values
- **Theorem**. If $\mathcal{A}$ is an asynchronous automaton computing θ^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 $\mathrm{LocTL}[\mathsf{S}_i]$-definable
- **Corollary**. FO-definable trace languages are accepted by a local cascade product of $\mathcal{A}$ with localized resets

First-order definable trace languages 4/4

- For general distributed alphabets, what we need beside LocTL[S_i], is the truth values of the constants $Y_i \leq Y_j$. Let θ^Y be the function computing these truth values
- **Theorem**. If $\mathcal{A}$ is an asynchronous automaton computing θ^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 θ^Y

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

Asynchronous automata computing θ^Y

- The *gossip automaton* $\mathcal{G}$ (Mukund, Sohoni, 1997) is an asynchronous automaton which computes θ^Y
- But $\mathcal{G}$ is not aperiodic in general! Let $\mathscr{P} = \{1, 2, 3\}$, $\mathrm{loc}(a) = \{1, 2\}$, $\mathrm{loc}(b) = \{2, 3\}$, $\mathrm{loc}(c) = \{1, 3\}$. So $Tr(\widetilde{\Sigma}) = \Sigma^*$, impl'd on 3 processes. Yet *no aperiodic asynchronous automaton computes the truth value of* $\mathsf{Y}_1 \leq \mathsf{Y}_3$

Asynchronous automata computing θ^Y

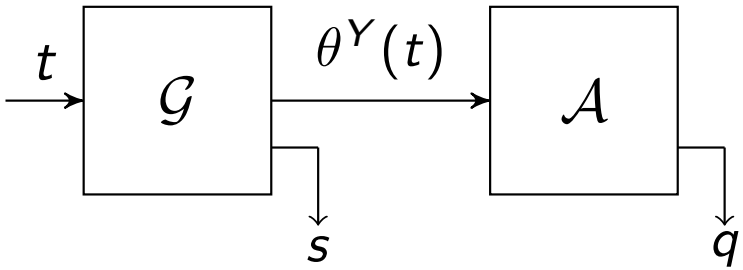
- The *gossip automaton* $\mathcal{G}$ (Mukund, Sohoni, 1997) is an asynchronous automaton which computes θ^Y
- But $\mathcal{G}$ is not aperiodic in general! Let $\mathscr{P} = \{1, 2, 3\}$, $\mathrm{loc}(a) = \{1, 2\}$, $\mathrm{loc}(b) = \{2, 3\}$, $\mathrm{loc}(c) = \{1, 3\}$. So $Tr(\widetilde{\Sigma}) = \Sigma^*$, impl'd on 3 processes. Yet *no aperiodic asynchronous automaton computes the truth value of* $\mathsf{Y}_1 \leq \mathsf{Y}_3$
- Thus our description of FO-definable, as accepted by a local cascade product of $\mathcal{G}$ 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?. . .

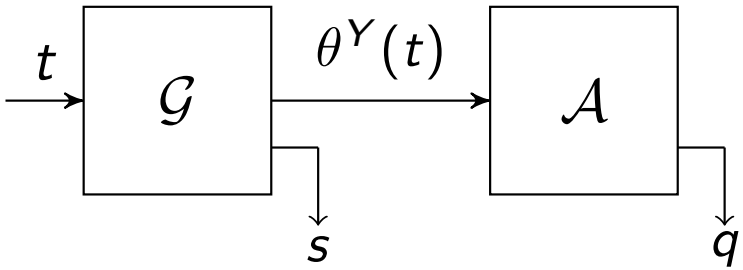
Restricted cascade product with the gossip automaton

- 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 θ^Y



Restricted cascade product with the gossip automaton

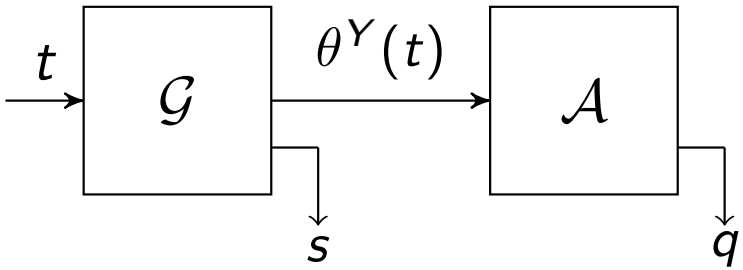
- No characterization of FO-definable trace languages in terms of local cascade products of localized resets?...
- $\mathcal{G}$ is an overkill: restrict the definition of the local cascade product to pass on only the information computed by θ^Y



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

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 θ^Y



- Hence: yet another (!) characterization of FO-definability
- **Question (open)**: For which distributed alphabets do we have an aperiodic atm computing θ^Y? For these $\widetilde{\Sigma}$, the aperiodic Zielonka theorem holds

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 $t = (E, \leq, \lambda) \in Tr(\widetilde{\Sigma})$, $\zeta_{\mathcal{A}}(t) = (E, \leq, (\lambda, \mu))$, where μ is as follows. If $e \in E$ and $t_e =$ strict past of e, $\mu(e) = s_{\mathrm{in}} \cdot \varphi(t_e)$: 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 $t = (E, \leq, \lambda) \in Tr(\widetilde{\Sigma})$, $\zeta_{\mathcal{A}}(t) = (E, \leq, (\lambda, \mu))$, where μ is as follows. If $e \in E$ and $t_e =$ strict past of e, $\mu(e) = s_{\mathrm{in}} \cdot \varphi(t_e)$: 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 $\mathcal{A}_i$)

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 $t = (E, \leq, \lambda) \in Tr(\widetilde{\Sigma})$, $\zeta_{\mathcal{A}}(t) = (E, \leq, (\lambda, \mu))$, where μ is as follows. If $e \in E$ and $t_e =$ strict past of e, $\mu(e) = s_{\mathrm{in}} \cdot \varphi(t_e)$: 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 $\mathcal{A}_i$)
- **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 $\mathcal{A}$ (instead of *local* state labeling function as in the asynchronous transducer $\chi_{\mathcal{A}}$). For $t = (E, \leq, \lambda) \in Tr(\widetilde{\Sigma})$, $\zeta_{\mathcal{A}}(t) = (E, \leq, (\lambda, \mu))$, where μ is as follows. If $e \in E$ and $t_e =$ strict past of e, $\mu(e) = s_{\mathrm{in}} \cdot \varphi(t_e)$: 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 $\mathcal{A}_i$)
- **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 $\mathcal{A}_i$

Thank you for your attention!