# String Diagrammatic Trace Theory

Matt Earnshaw[1]

j.w.w. Paweł Sobociński[1]

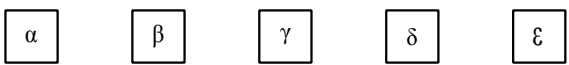
[1]Tallinn University of Technology, Estonia

MFCS, Bordeaux
September 1st 2023

# Mazurkiewicz traces (in pictures)

Mazurkiewicz traces are a non-interleaving semantics for behaviour of systems with concurrent atomic actions.

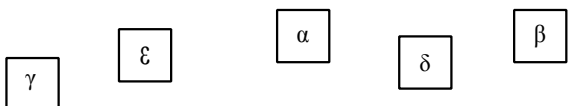
*Words* are the behaviour of sequential machines with atomic actions. Trace generalize words by allowing specified pairs of actions to commute.



# Mazurkiewicz traces (in pictures)

Mazurkiewicz traces are a non-interleaving semantics for behaviour of systems with concurrent atomic actions.

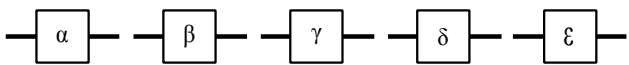
*Words* are the behaviour of sequential machines with atomic actions. Trace generalize words by allowing specified pairs of actions to commute.



# Mazurkiewicz traces (in pictures)

Mazurkiewicz traces are a non-interleaving semantics for behaviour of systems with concurrent atomic actions.

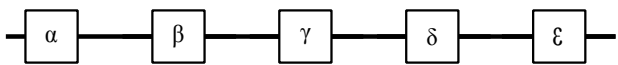
*Words* are the behaviour of sequential machines with atomic actions. Trace generalize words by allowing specified pairs of actions to commute.



# Mazurkiewicz traces (in pictures)

Mazurkiewicz traces are a non-interleaving semantics for behaviour of systems with concurrent atomic actions.

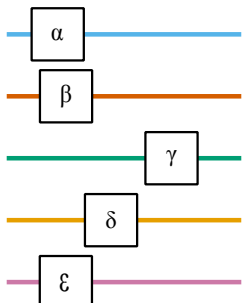
*Words* are the behaviour of sequential machines with atomic actions. Trace generalize words by allowing specified pairs of actions to commute.



# Mazurkiewicz traces (in pictures)

Mazurkiewicz traces are a non-interleaving semantics for behaviour of systems with concurrent atomic actions.

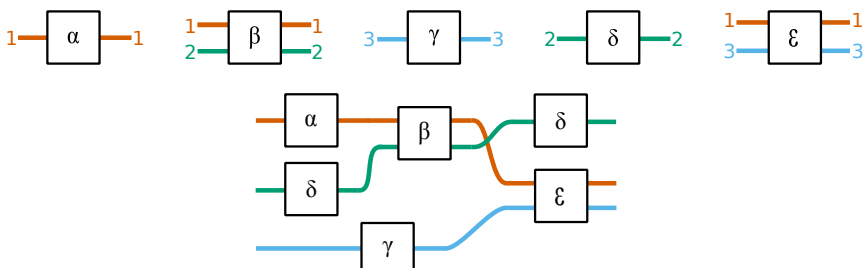
*Words* are the behaviour of sequential machines with atomic actions. Trace generalize words by allowing specified pairs of actions to commute.



# Mazurkiewicz traces (in pictures)

Mazurkiewicz traces are a non-interleaving semantics for behaviour of systems with concurrent atomic actions.

*Words* are the behaviour of sequential machines with atomic actions. Trace generalize words by allowing specified pairs of actions to commute.




$$\langle \alpha, \beta, \gamma, \delta, \varepsilon \mid \alpha\delta = \delta\alpha, \beta\gamma = \gamma\beta, \delta\varepsilon = \varepsilon\delta, \alpha\gamma = \gamma\alpha, \gamma\delta = \delta\gamma \rangle$$

# Mazurkiewicz traces in pictures

Such pictures often appear in the traces literature.

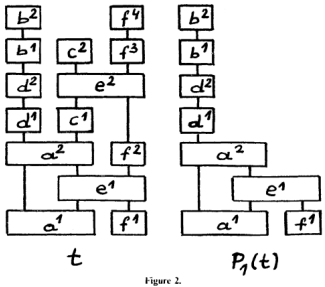
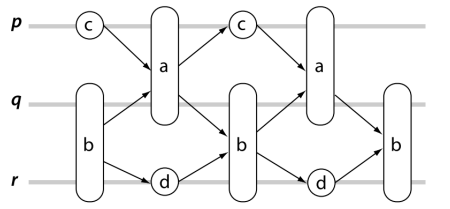



Figure 2.

Zielonka 1987




Krishna, Muscholl 2013

We can make these formal, and in doing so recover asynchronous automata and more.

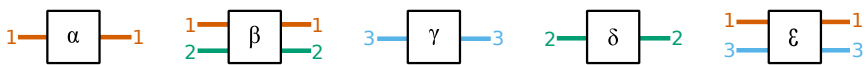
Claim: such pictures can be given semantics as morphisms in *props*, which can be represented as *string diagrams*.

# Distributed alphabets

A *monoidal graph* is a kind of multi-input multi-output graph given by:

A set $S$ of sorts, a set $B$ of boxes, and functions $s, t : B \rightrightarrows S^*$.

We have already seen some examples.

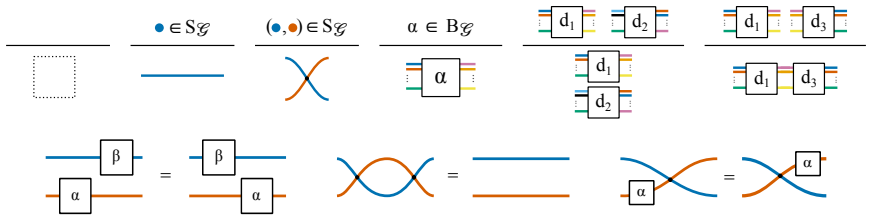



We call a monoidal graph with the following properties a *distributed alphabet*:

- $B$ is finite and $S$ is a finite ordinal (*locations*).
- sorts appear in order in the sources and targets of each box,
- each sort $i \in S$ appears at most once in each source and target,
- for each box $\gamma \in B$ , the sources and targets are non-empty and equal: $s(\gamma) = t(\gamma)$.

# From monoidal graphs to monoidal languages

A monoidal graph $G$ freely generates a *prop* $\mathcal{F}G$: a *category* whose objects are elements of $S_G^*$ and whose morphisms are *string diagrams* built from the graph.




A *symmetric monoidal language* is a set of morphisms in the free prop $\mathcal{F}G$ over a finite monoidal graph $G$.

***Theorem.*** Let $G$ be a distributed alphabet. Then the monoid of string diagrams in $\mathcal{F}G$ from the (ordered) set of locations to itself is the monoid of traces.

Therefore, *Mazurkewicz trace languages are symmetric monoidal languages over distributed alphabets*.

# Why string diagrams?

Traces take many guises. In particular, their topological representation as *dependence graphs* is well understood and widely used.

String diagrams can also be understood as certain (open) acyclic graphs.

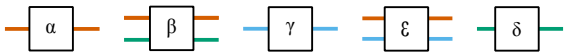
So what are the advantages of string diagrams?

- We can apply our general theory of automata over string diagrams.[a]
  This recovers *asynchronous automata* and their generalizations (e.g. probabilistic).
- Linearization of traces is a diagrammatic operation with algebraic meaning.
- Suggests various generalizations of trace languages, using the powerful algebra of monoidal categories.
- Shift in perspective allows us to apply new tools, and link to new literature.

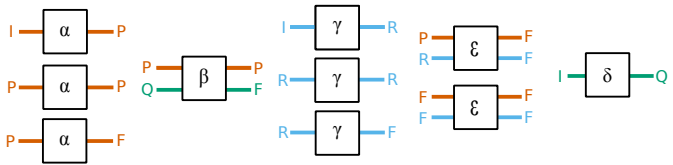
[a] Introduced in E., Sobocinski 2022.

# Regular symmetric monoidal languages

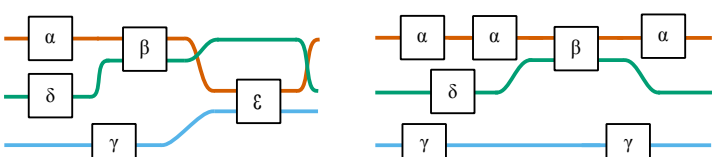
*Regular symmetric monoidal languages* are those recognized by *monoidal automata*.




---

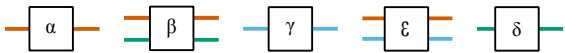



---

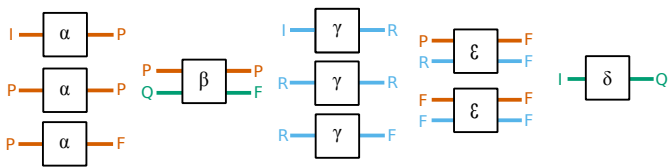


# Regular symmetric monoidal languages

*Regular symmetric monoidal languages* are those recognized by *monoidal automata*.

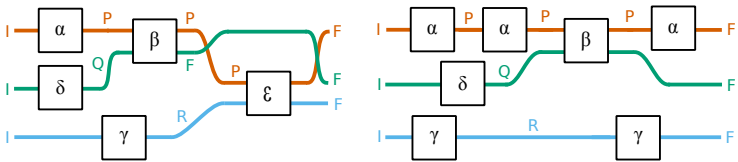



---




$(\mathrm{I},\mathrm{I},\mathrm{I}), (\mathrm{F},\mathrm{F},\mathrm{F}) \in \mathcal{Q}_{\bullet} \times \mathcal{Q}_{\bullet} \times \mathcal{Q}_{\bullet}$

---



# Asynchronous automata are monoidal automata

*Recognizable* trace languages are defined in the literature by an algebraic criterion.

(Zielonka 1987[a]) introduced *asynchronous automata* and proved that these accept exactly the recognizable trace languages.

***Theorem.*** Symmetric monoidal automata over distributed alphabets are precisely Zielonka's asynchronous automata.

Consequently, recognizable trace languages are exactly regular symmetric monoidal languages over distributed alphabets.

Our definition of automaton gives rise to a monoidal functor. By varying the codomain of this functor we recover deterministic and probabilistic asynchronous automata (Jesi, Pighizzini, Sabadini 1996[b]).

---

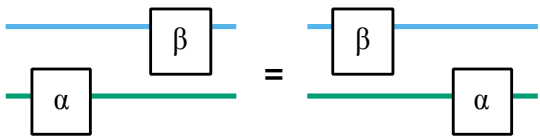
[a] *Notes on Finite Asynchronous Automata*, Informatique théorique et applications

[b] *Probabilistic asynchronous automata*, Mathematical Systems Theory

# Serialization via string diagrams for premonoidal categories

Often useful to consider the possible serializations of a trace.

We can do this using string diagrams for premonoidal categories, which equip boxes with a new distinguished wire, preventing interchange.



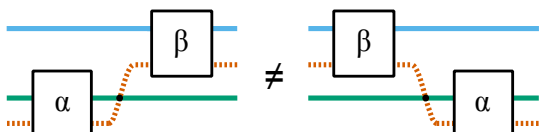

We can define a map from the free premonoidal category over a distributed alphabet, to the free prop over the same alphabet, by forgetting the red wire.

***Theorem.*** The preimage of a string-diagrammatic trace language under this morphism is its serialization.

# Serialization via string diagrams for premonoidal categories

Often useful to consider the possible serializations of a trace.

We can do this using string diagrams for premonoidal categories, which equip boxes with a new distinguished wire, preventing interchange.



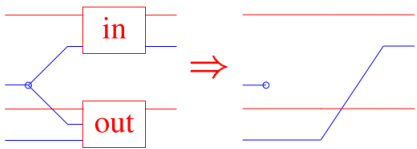

We can define a map from the free premonoidal category over a distributed alphabet, to the free prop over the same alphabet, by forgetting the red wire.

***Theorem.*** The preimage of a string-diagrammatic trace language under this morphism is its serialization.

# Beyond traces: sketch of future work

Many kinds of categorical structure admit a calculus of string diagrams.

Can we find a structure which gives semantics to string diagrams of concurrent systems with actions richer than merely atomic ones? i.e. in which we can distinguish *locations* from *resources*?




String diagrams for *premonoidal categories* (Jeffrey 1998[a], Román 2022[b]) offer a starting point, particularly when seen as embedded in the richer universe of diagrams for (collages of) *bimodular categories* (Braithwaite, Román 2023[c]).

[a] *Premonoidal Categories and a Graphical View of Programs*, preprint

[b] *Promonads and String Diagrams for Effectful Categories*, Proceedings of ACT 2022

[c] *Collages of String Diagrams*, Proceedings of ACT 2023