Monoidal Languages

Matt Earnshaw
https://ioc.ee/~matt

Tallinn University of Technology, Estonia

(i)Po(m)set Project Online Seminar
October 6th 2023

What are monoidal languages?

An approach to languages of graphs via the algebra of *monoidal categories*.

A *monoidal language* is a set of morphisms, in a (free, strict) monoidal category.

Morphisms in monoidal categories are *string diagrams*: certain acyclic graphs.

We investigate some well-behaved classes: regular and context-free.

The former includes regular languages of words, bottom-up and top-down trees, and recognizable (Mazurkiewicz) traces.

Regular monoidal languages are recognized by monoidal automata. Not determinizable in general, but interesting partial results.

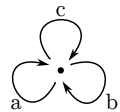
Context-free monoidal languages are described by string diagrams with holes.

Outline

- Introduction to monoidal categories and string diagrams
- Regular monoidal languages: grammars and automata
- Results on determinization
- Mazurkiewicz traces and asynchronous automata as monoidal automata
- String diagrams beyond traces (work in progress)
- Pumping lemma
- Context-free monoidal languages (work in progress)

From monoids to monoidal categories

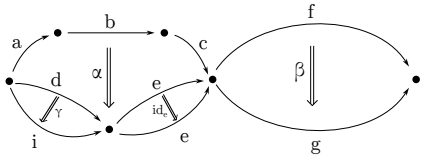
A monoid is equivalent to a category with one object.
Morphisms are the elements, composition is the monoid operation.

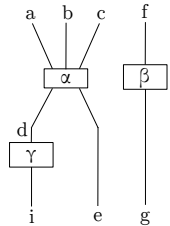


A strict monoidal category is equivalent to a 2-category with one object.

1-cells are the objects, 2-cells are the morphisms. Composition of 1-cells is tensor product of objects, and of 2-cells tensor product of morphisms.

String diagrams are Poincaré dual to 2-dimensional pasting diagrams.

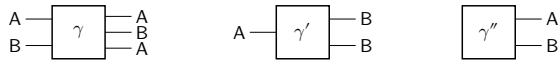




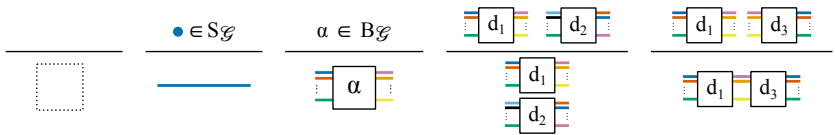
[JS91]: String diagrams are sound for the axioms of strict monoidal categories.

Free monoidal categories and string diagrams

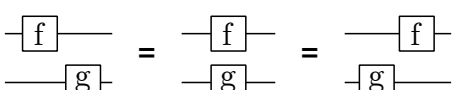
A monoidal graph $\mathcal{G}$ is a kind of multi-input multi-output graph given by: a set S of sorts, a set B of boxes and functions $s, t : B_{\mathcal{G}} \rightrightarrows {S_{\mathcal{G}}}^*$.



A monoidal graph $\mathcal{G}$ generates a free (strict) *monoidal category* $\mathcal{F}\mathcal{G}$ whose morphisms are string diagrams over $\mathcal{G}$. [JS91]: string diagrams are complete.



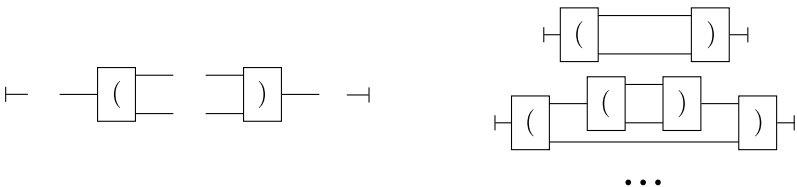
$$(f \otimes \mathrm{id}) \circ (\mathrm{id} \otimes g) = (f \circ \mathrm{id}) \otimes (\mathrm{id} \circ g) = f \otimes g = (\mathrm{id} \circ f) \otimes (g \circ \mathrm{id}) = (\mathrm{id} \otimes f) \circ (g \otimes \mathrm{id})$$



Monoidal languages

Let $\mathcal{G}$ be a finite monoidal graph, our 'monoidal alphabet'.

A (scalar, planar) monoidal language over $\mathcal{G}$ is a set of morphisms in $\mathcal{F}\mathcal{G}$ from the monoidal unit to itself: no dangling wires on the left or right.



Later we will consider more general 'boundary conditions' (initial and final states). Bossut took initial and final *regular languages* over the sorts as boundary languages.

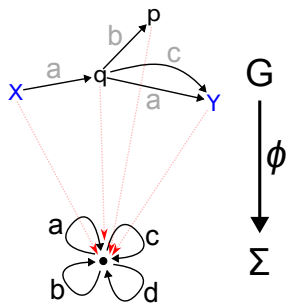
We will also look at non-planar languages. But first, let us define the *regular* monoidal languages.

Grammars according to Walters

Walters (1989)[a] showed how to represent regular grammars as morphisms of graphs:

A *regular grammar* is a morphism of *finite* directed graphs $\phi : G \to \Sigma$, where Σ is a graph with one vertex, along with two distinguished vertices of G.

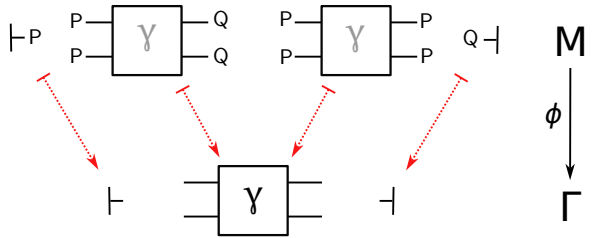
The free category $\mathcal{F}G$ on G has objects the vertices and morphisms the *paths*, with composition given by path concatenation.



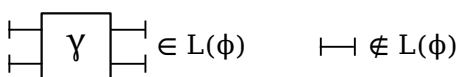
(ϕ, X, Y) determines a subset of Σ^*: take the image under $\mathcal{F}\phi$ of the set of paths $\mathcal{F}G(X, Y)$. The subsets of Σ^* arising in this way are the regular languages over Σ.

[a] A Note on Context-Free Languages, https://doi.org/10.1016/0022-4049(89)90151-5

Regular monoidal grammars and languages



Defines a language: the $0 \to 0$ string diagrams that can be built.

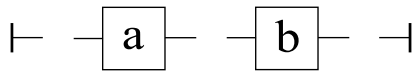


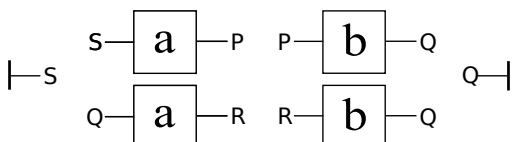
Languages so definable are the (scalar) *regular monoidal languages*.

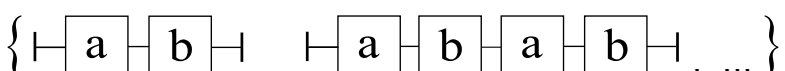
Technically, we pass to the free monoidal functor $\mathcal{F}\phi : \mathcal{F}M \to \mathcal{F}\Gamma$, and take the image of the set of morphisms $\mathcal{F}M(0,0)$.

Regular languages as regular monoidal languages

Take alphabets where each generator has arity and coarity 1.



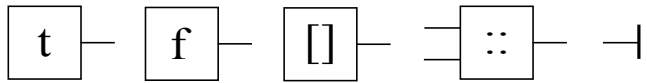


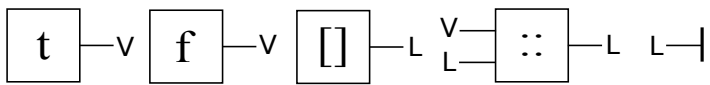


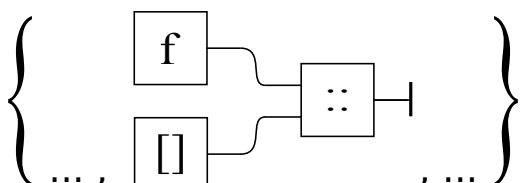
Regular languages are regular monoidal languages over alphabets of this shape.

Bottom-up regular tree languages as regular monoidal languages

Take alphabets where each generator has coarity 1 (plus an 'end' symbol).



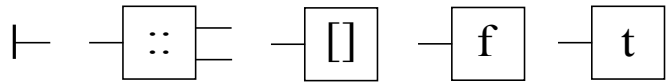


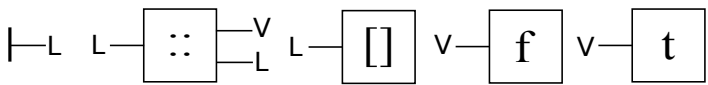


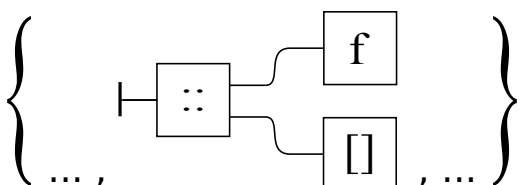
Bottom-up regular tree languages are regular monoidal languages over alphabets of this shape.

Top-down regular tree languages as regular monoidal languages

Take alphabets where each generator has arity 1 (plus a 'start' symbol):

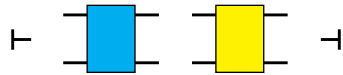


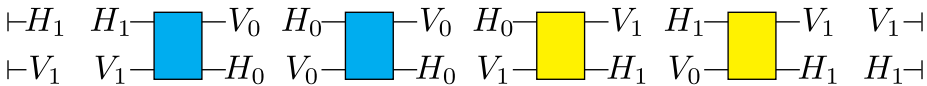


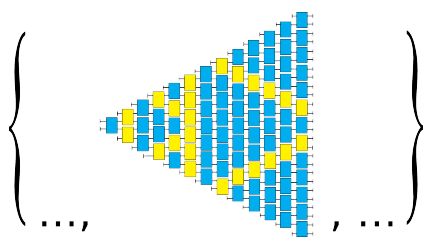


Top-down regular tree languages are regular monoidal languages over alphabets of this shape.

Regular monoidal language of Sierpinski triangles







Non-deterministic monoidal automata

- V, finite set
- Γ, monoidal alphabet
- $\Delta_\Gamma = \{V^{\mathsf{ar}(\gamma)} \xrightarrow{\Delta_\gamma} \mathcal{P}(V^{\mathsf{coar}(\gamma)})\}_{\gamma \in E_\Gamma}$, set of transition relations

Inductively extends to a strict monoidal functor $\mathcal{F}\Gamma \to \mathsf{Rel}_V$

String diagrams $0 \to 0$ map to a function $V^0 \to \mathcal{P}(V^0)$ (accept/reject).

By restricting Γ we recover:

- Ordinary non-deterministic automata
- Top-down tree automata
- Bottom-up tree automata

The problem of determinization

Top-down tree automata cannot be determinized in general, so we cannot expect to determinize non-deterministic monoidal automata.

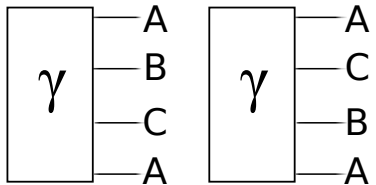
Challenge

Characterize the deterministically recognizable RMLs.

Partial answers:

- convex automata
- necessary property of deterministic language
- algebraic invariant

Partial answer I: Convex automata

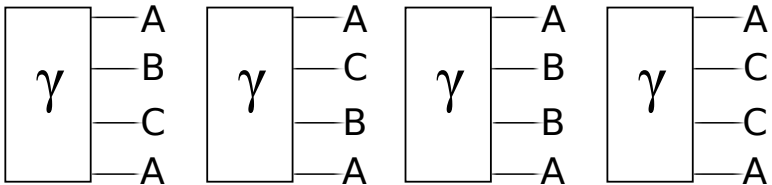


$\gamma : V^0 \to \mathcal{P}(V^4)$ is not convex

A monoidal automaton is convex if its transition relations are convex.

Theorem. Convex automata can be determinized, by an analogue of the powerset construction. E.g. word automata and bottom-up tree automata.

Partial answer I: Convex automata

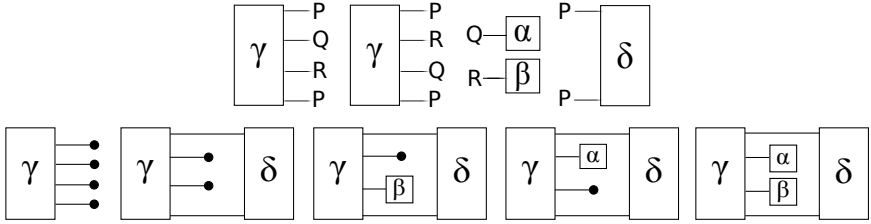


$\gamma : V^0 \to \mathcal{P}(V^4)$ is convex

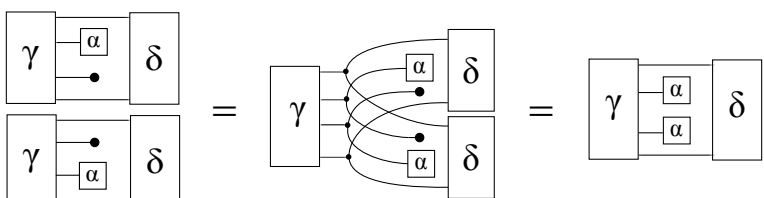
A monoidal automaton is convex if its transition relations are convex.

Theorem. Convex automata can be determinized, by an analogue of the powerset construction. E.g. word automata and bottom-up tree automata.

Partial answer II: Causal closure



Causal histories recombinable via equations in cartesian restriction categories

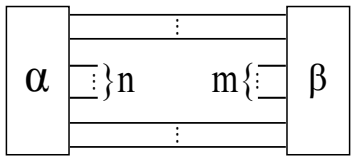


Theorem. Deterministically recognizable languages are causally closed.

Syntactic monoidal category

Recall that for a word language, its syntactic monoid is finite iff the language is regular.

We can define the syntactic monoidal category via the following equivalence relation:



$$\gamma \equiv_L \delta \text{ if } C[\gamma] \in L \iff C[\delta] \in L, \text{ for all contexts } C.$$

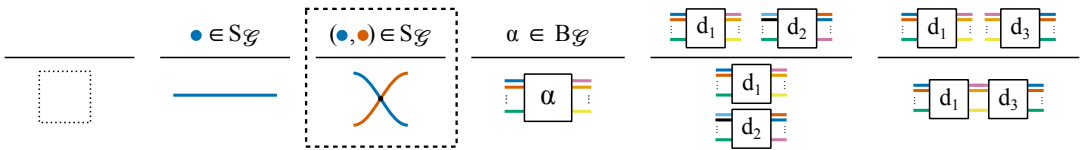
Two string diagrams are equivalent if they are not distingushed by any context.

Theorem. If L is regular monoidal then its syntactic monoidal category has finite homsets.

Theorem. If the syntactic monoidal category is a cartesian restriction category, then the language is deterministically recognizable.

Symmetric monoidal languages

Our story works also for languages of non-planar diagrams. We allow our wires to cross without tangling:



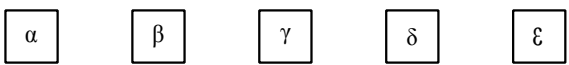
This gives string diagrams for symmetric monoidal categories.

We will now see how Mazurkiewicz traces are symmetric monoidal languages.

Mazurkiewicz traces with string diagrams

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

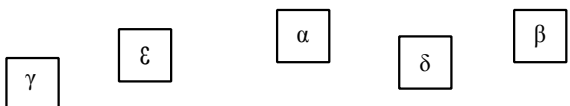
Words are the behaviour of sequential machines with atomic actions. Trace generalize words by allowing specified pairs of actions to commute.



Mazurkiewicz traces with string diagrams

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

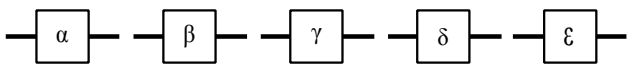
Words are the behaviour of sequential machines with atomic actions. Trace generalize words by allowing specified pairs of actions to commute.



Mazurkiewicz traces with string diagrams

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

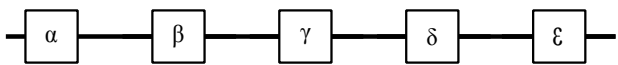
Words are the behaviour of sequential machines with atomic actions. Trace generalize words by allowing specified pairs of actions to commute.



Mazurkiewicz traces with string diagrams

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

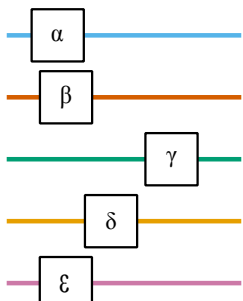
Words are the behaviour of sequential machines with atomic actions. Trace generalize words by allowing specified pairs of actions to commute.



Mazurkiewicz traces with string diagrams

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

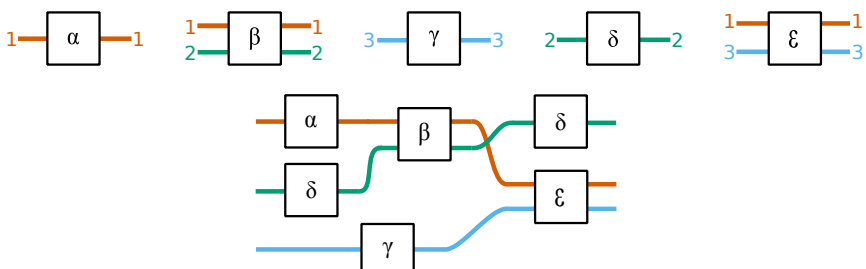
Words are the behaviour of sequential machines with atomic actions. Trace generalize words by allowing specified pairs of actions to commute.



Mazurkiewicz traces with string diagrams

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.

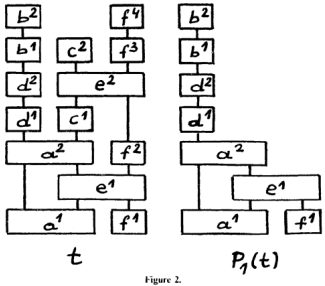
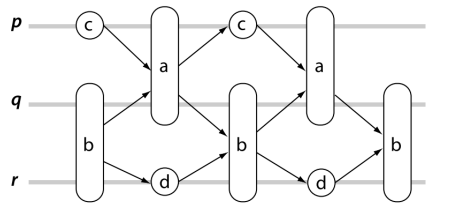


Figure 2.

Zielonka 1987



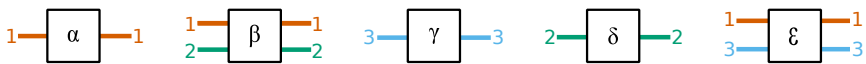
Krishna, Muscholl 2013

Idea: these are elements of symmetric monoidal languages over certain alphabets.

Symmetric monoidal automata over such alphabets are Zielonka's asynchronous automata.

Distributed alphabets

To obtain trace languages, we take multi-sorted alphabets of a special shape.



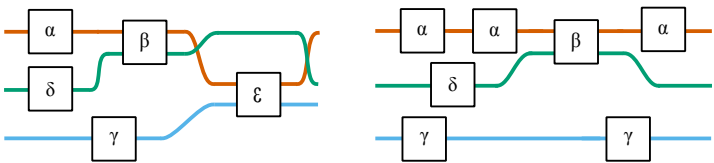
We call a monoidal graph with the following properties a *distributed alphabet*:

- 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: $s(\gamma) = t(\gamma)$.

Trace languages are symmetric monoidal

A *symmetric monoidal language* is a set of morphisms in the free symmetric monoidal $\mathcal{F}G$ 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 isomorphic to the monoid of traces.



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.

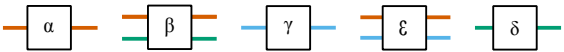
String diagrams can also be understood as certain (open) acyclic graphs.

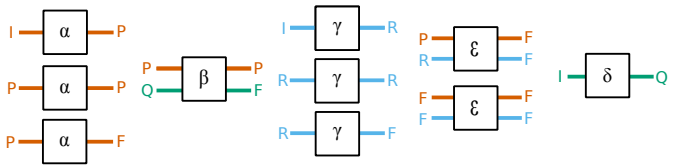
So what are the advantages of string diagrams?

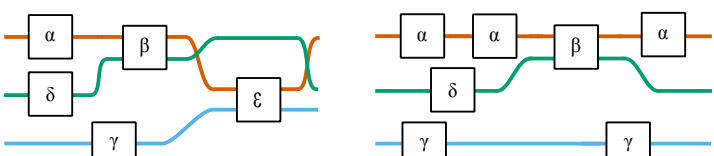
- We can apply our general theory of automata over string diagrams. This recovers *asynchronous automata* and their generalizations.
- Linearization of traces is a diagrammatic operation with algebraic meaning.
- Suggests various generalizations of trace languages, using the powerful algebra of monoidal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

Regular symmetric monoidal languages

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

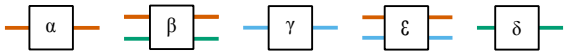


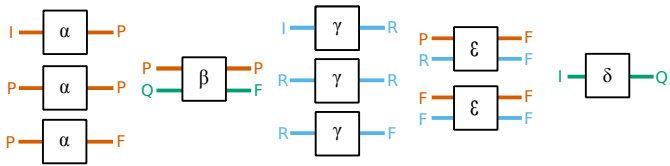


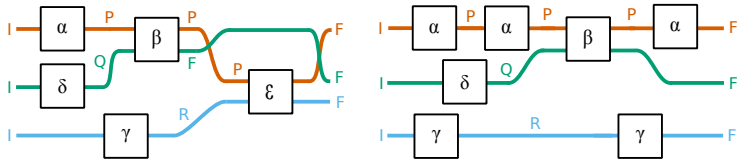


Regular symmetric monoidal languages

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







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]).

---

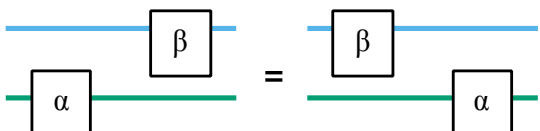
[a] *Notes on Finite Asynchronous Automata*, Informatique théorique et applications

[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.



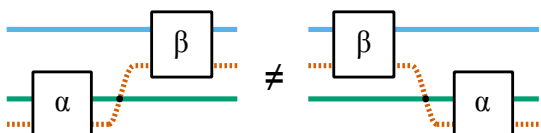
We define a map from the free premonoidal category over a distributed alphabet, to the free symmetric monoidal category over the same alphabet, by forgetting the red wire.

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.



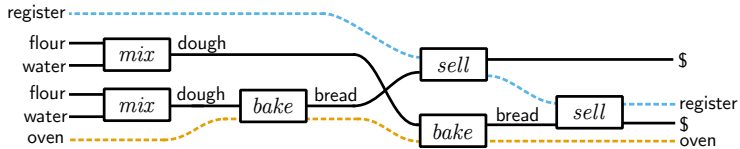
We define a map from the free premonoidal category over a distributed alphabet, to the free symmetric monoidal category over the same alphabet, by forgetting the red wire.

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

Beyond traces: work in progress

In our presentation of traces we have used resource wires for locations. By construction, we have prevented their duplication.

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*?



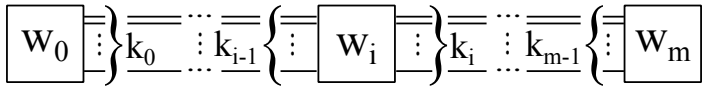
In a recent abstract with Nester and Román, we introduce an extension of string diagrams for premonoidal categories (Jeffrey 1998[a], Román 2022[b]) to include these two kinds of wires.

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

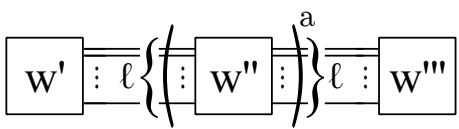
[b] *Promonads and String Diagrams for Effectful Categories*, Proceedings of ACT 2022

Pumping lemma and a non-regular monoidal language

L regular monoidal $\Longrightarrow \forall k \in \mathbb{N}^+, \exists p \geqslant 0$ such that for any $\boxed{\mathrm{w}} \in L$ and factorization

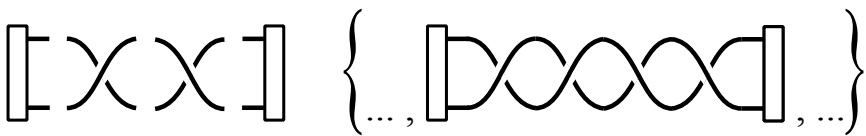


$m \geqslant p, 0 \leqslant k_i \leqslant k$, there exists i, j, ℓ such that $k_i = k_j = \ell$ and



$\in L, \forall a \geqslant 0$

Example: unbraids on n strings. $n = 2$:



Context-free monoidal languages

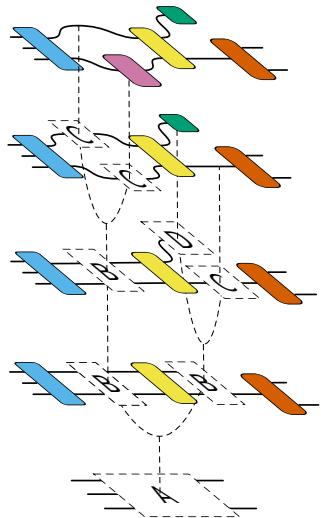
In recent work with Hefford and Román[a], we introduced a category of contexts in monoidal categories.

Contexts provide the notion necessary to define *context-free languages of string diagrams*. These are described by grammars in which contexts can appear both sequentially and in parallel.

We are generalizing the Chomsky-Schützenberger theorem to context-free monoidal languages, following Melliès and Zeilberger[b].

Conjecture: the generic shape of derivation trees is described by a context-free monoidal language that generalizes Dyck languages to two dimensions.

Conjecture: there is a monoidal automaton that checks if a given shape comes from a derivation.



[a] The Produoidal Algebra of Process Decomposition, arxiv.org/abs/2301.11867

[b] Parsing as a Lifting Problem..., arxiv.org/abs/2212.09060

Papers on monoidal languages

[1] Francis Bossut, Max Dauchet, and Bruno Warin. "A Kleene Theorem for a Class of Planar Acyclic Graphs". In: *Inf. Comput.* 117 (Mar. 1995), pp. 251–265. DOI: 10.1006/inco.1995.1043.

[2] Matthew Earnshaw and Paweł Sobociński. "Regular Monoidal Languages". In: *47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022)*. 2022. DOI: 10.4230/LIPIcs.MFCS.2022.44.

[3] Matthew Earnshaw and Paweł Sobociński. "String Diagrammatic Trace Theory". In: *48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023)*. DOI: 10.4230/LIPIcs.MFCS.2023.43.