Regular Planar Monoidal Languages[⋆]

Matthew Earnshaw[1], Paweł Sobociński[1]

[a] *Tallinn University of Technology, Department of Software Science, Akadeemia tee 21/1, 12618, Tallinn, Estonia*

Abstract

We introduce regular languages of morphisms in free monoidal categories, with their associated grammars and automata. These subsume the classical theory of regular languages of words and trees, but also open up a much wider class of languages of planar string diagrams. We give a pumping lemma for monoidal languages, generalizing the one for words and trees. We use the algebra of monoidal and cartesian restriction categories to investigate the properties of regular monoidal languages, and provide sufficient conditions for their recognizability by deterministic monoidal automata.

Keywords: monoidal categories, string diagrams, formal language theory, automata, cartesian restriction categories

1. Introduction

Free monoids play a central role in classical formal language theory, but language theory has been extended to many algebraic structures, such as infinite words [29], rational sequences [1], trees [21, 2], countable linear orders [7], graphs of bounded tree width [11], etc. Recently, several works have appeared that aim at unifying the language theory of these diverse structures [3, 38].

One approach to unification is to view these structures as algebras for monads on the category of sets, and then to develop language theory at the level of monads. In this vein, Bojańczyk, Klin and Salamanca [3] have given sufficient conditions on a monad for the correspondence between regularity and definability in monadic second-order logic to extend to languages over its algebras. Previously, universal algebra has also been fruitfully applied to the problem of giving a unified presentation of automata over diverse algebras, for example in the classical work of Eilenberg and Wright [18], and Thatcher and Wright [37].

However, there are many algebraic structures arising neither as algebras for monads on the category of sets, nor as structures in classical universal algebra. In particular, this is true of many structures in category theory, such as monoidal categories. At the same time, these structures can be considered as natural

⋆This research was supported by the ESF funded Estonian IT Academy research measure (project 2014-2020.4.05.19-0001) and Estonian Research Council grant PRG1210.

generalizations of monoids to higher dimensions, and so offer promise for an algebraic approach to higher-dimensional formal languages.

A natural first step, which we take in this paper, is to replace monoids with 2-monoids, better known as strict monoidal categories. Monoids can be seen as *categories* with one object, in which morphisms are the elements of the monoid. Strict monoidal categories can be defined as 2-categories with one object: "higher" monoids in which there are now additionally transformations between the elements. We call languages in these categories *monoidal languages*.

We introduce grammars and automata for monoidal languages, defining the class of regular planar monoidal languages. We show how these include classical and tree automata, but also open up a wilder world of string diagram languages. In fact, our framework is flexible enough to treat any structures arising as algebras for monads over multi-input, multi-output graphs known as *monoidal graphs*. We indicate some future directions along these lines in our conclusion.

By investigating morphisms in monoidal categories from the perspective of language theory, this work contributes to research into the computational manipulation of string diagrams, and so their usage in industrial strength applications.

Outline. In Section 3, we cover some preliminaries, recalling the basic algebraic ingredients needed for the rest of the paper: *monoidal graphs* and *monoidal categories*, along with their *string diagrams*, a graphical formalism for representing their morphisms. In Section 4, we introduce *monoidal languages* and *regular monoidal grammars*, a finitary specification of monoidal languages defining the class of *regular monoidal languages*. In Section 5, we introduce the pumping lemma for regular monoidal languages, and use it to analyze a non-regular monoidal language. In Section 6, we introduce *non-deterministic monoidal automata* and their associated monoidal languages, which give an operational characterization of regular monoidal languages. In Section 7, we show how regular word and tree automata are special cases of monoidal automata. In Section 8, we give some closure properties of regular monoidal languages. These are the usual closure properties of regular languages, with the exception of complements. In Section 9, we introduce the *syntactic pro* of a monoidal language by analogy with the syntactic monoid of a regular language. In Section 10, we introduce deterministic monoidal automata and the concept of causal history, which is used to investigate deterministic recognizability and provide a necessary condition on a language for it to be deterministically recognizable. We also give an algebraic condition on a language sufficient for deterministic recognizability, using the syntactic pro. In Section 11, we introduce *convex relations* and give a powerset construction for a class of monoidal automata.

Comparison with the conference paper. This work is an extended version of the conference paper [16]. Besides reorganization and additional examples, there are several new sections, extensions of concepts, and corrections. In particular, we introduce a pumping lemma for regular monoidal languages (Lemma 5.1), and apply this to a new example of a monoidal language (Definition 5.4), showing that it is not regular (Proposition 5.6). We correct the details around convex relations: the lift Δ^* need not be unique, as claimed in [16]. Prerequisites are covered in more detail, making the paper more self-contained. We also include all proofs omitted in the conference version.

2. Related work

Bossut [4] studied rational languages of planar acyclic graphs and proved a Kleene theorem for a class of such languages. Bossut introduced a notion of automaton for these languages, but these lack a state machine denotation – being more similar to our grammars. Bossut's graph languages feature initial and final states, whereas in this paper we do away with these by considering *scalar* morphisms, which more neatly generalizes the theory of regular string and tree languages. We make explicit the fact the languages of graphs investigated by Bossut have an underlying algebra, that of monoidal categories, and hence fully leverage this algebra in our proofs and definitions. This leads us in quite different directions of investigation from Bossut.

In the preprint [22], Heindel recasts Bossut's approach using monoidal categories, and this serves as a starting inspiration for ours, although our definitions and direction of development differ. Unfortunately the purported Myhill-Nerode result was incorrect, due to a flawed definition of syntactic congruence. We rectify this in Section 9, but a Myhill-Nerode type theorem remains open.

Zamdzhiev [40] introduced context-free languages of string diagrams using a combinatorial representation of string diagrams called *string graphs*, and the machinery of context-free graph grammars. In contrast, our approach does not require an intermediate representation of string diagrams as graphs: we work directly with morphisms in monoidal categories. This allows us to use the algebra of monoidal categories to reason about properties of monoidal languages.

Winfree et al. [33] used DNA self-assembly to simulate cellular automata and Wang tile models of computation. The kinds of two-dimensional languages obtained in this way can be seen quite naturally as regular monoidal languages, as illustrated in Example 4.10.

Walters' note [39] on regular and context-free grammars served as a starting point for our definition of regular monoidal grammar. Rosenthal [32], developing some of the ideas of Walters, defined automata as relational presheaves, which is similar in spirit to our functorial definition of monoidal automata. The framework of Colcombet and Petrişan [10] considering automata as functors is also close in spirit to our definition of monoidal automata. However, all of these papers are directed towards questions involving classical one-dimensional languages, rather than languages of diagrams as in the present paper.

Fahrenberg et al. [19] investigated languages of higher-dimensonal automata, a well-established model of concurrency. We might expect that the investigations of the present paper correspond to a detailed study of a particular low-dimensional case of such languages, but the precise correspondence between these notions is unclear.

3. Monoidal Graphs, Pros, and their String Diagrams

In this section we introduce the main algebraic structures including the kinds of monoidal categories known as *pros*. Morphisms in monoidal categories can be represented using *string diagrams*, a kind of "graph with interfaces". Monoidal graphs are intuitive to work with and often lead to shorter proofs [23, 35]. String diagrams can be used to present monoidal categories. The basic building blocks for string diagrams are given by *monoidal graphs*, a kind of multi-input, multi-output graph:

Definition 3.1. *A monoidal graph $\mathcal{G}$ is a set $B_\mathcal{G}$ of boxes, a set $S_\mathcal{G}$ of sorts and functions $s, t : B_\mathcal{G} \rightrightarrows {S_\mathcal{G}}^*$ to the free monoid over $S_\mathcal{G}$, giving source and target boundaries of each box.*

Diagrammatically, a monoidal graph can be pictured as a collection of boxes, labelled by elements of $B_\mathcal{G}$ with strings entering on the left and exiting on the right, labelled by types given by the source and target functions s, t. For example, Figure 1 depicts the monoidal graph $\mathcal{G}$ with $B_\mathcal{G} = \{\gamma, \gamma'\}, S_\mathcal{G} = \{A, B\}, s(\gamma) = AB, t(\gamma) = ABA, s(\gamma') = A, t(\gamma') = BB$:

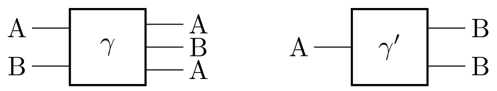


Figure 1: Example of a monoidal graph. Monoidal graphs form the building blocks of string diagrams: a diagrammatic notation for morphisms in monoidal categories.

Given that we are interested in finite state machines over finite alphabets, we shall work exclusively with finite monoidal graphs, i.e. those in which $B_\mathcal{G}$ and $S_\mathcal{G}$ are both finite sets. We call $s(\gamma)$ and $t(\gamma)$ the arity and coarity of γ, respectively, of a box γ. We will write $\gamma : s(\gamma) \to t(\gamma)$ when considering a box along with its source and target types. These are the generators of (free) monoidal categories, and so we also call them *generators*.

Definition 3.2. *A* strict monoidal category *is a category $\mathcal{C}$, equipped with a functor $\otimes : \mathcal{C} \times \mathcal{C} \to \mathcal{C}$ (the* monoidal product*) and a unit object $I \in \mathcal{C}$, such that $\otimes$ is associative and unital: $A \otimes (B \otimes C) = (A \otimes B) \otimes C$ and $A \otimes I = A = I \otimes A$ for all objects A, B, C.*

The monoidal product turns the sets of objects and morphisms in $\mathcal{C}$ into monoids.

Definition 3.3. *A* pro *is a strict monoidal category whose monoid of objects is a free monoid (whose generators are* sorts*).*

Although the data of a strict monoidal category can seem intimidating to the non-expert, they admit an intuitive graphical calculus of *string diagrams*. Given a monoidal graph, we can construct the *free pro* on it using string diagrams, generated by the diagrammatic presentation of the monoidal graph:

Definition 3.4. *The free pro $\mathscr{F}\mathcal{G}$ on a monoidal graph $\mathcal{G}$ has monoid of objects $S^*_{\mathcal{G}}$ and morphisms* string diagrams *inductively defined as in Figure 2. The monoidal product ($\otimes$) is given on objects by concatenation, on diagrams by juxtaposition, and the unit is the empty word. Composition ($\fatsemi$) is given by joining wires. Note that, since we give a presentation via string diagrams rather than terms, the reader may verify that the required equations hold automatically. Specifically, these are: the associativity and unitality of the monoidal product, the associativity and unitality of composition, and the interchange law: $(d_1 \otimes d_2) \fatsemi (d_3 \otimes d_4) = (d_1 \fatsemi d_3) \otimes (d_2 \fatsemi d_4)$ whenever these composites exist.*

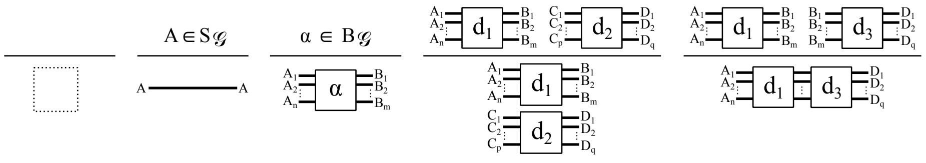


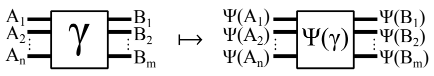
Figure 2: Inductive definition of morphisms in the free pro on a monoidal graph. Left to right: the empty string diagram is a string diagram (emphasized here with a dotted box); for each sort, the identity string is a string diagram; the string diagram for every generator α is a string diagram; for any two string diagrams their vertical juxtaposition is a string diagram; and for any two string diagrams with matching right and left boundaries, the string diagram obtained by joining the matching wires is a string diagram (their composition).

The basic idea is straightforward: we treat generators like circuit components that we can plug together in series, or place in parallel. String diagrams are topological objects: they are invariant up to planar isotopy. Furthermore, they are sound and complete: an equation between morphisms of strict monoidal categories follows from their axioms if and only if it holds between string diagrams up to planar isotopy [23, 35]. The structural equations of strict monoidal categories such as associativity and unitality of tensor hold automatically in string diagrams, and this often leads to shorter, less bureaucratic proofs.

Since the monoidal unit is the empty word (denoted 0 when $\mathcal{G}$ is single sorted, and by ε in general), morphisms from or to the monoidal unit are string diagrams with no "dangling wires" on their left or right, respectively. Note that when $\mathcal{G}$ is single-sorted, we do not need labels on the strings. Alphabets for monoidal languages will be single-sorted finite monoidal graphs: we call such monoidal graphs *monoidal alphabets.*

We will make extensive use of morphisms of monoidal graphs:

Definition 3.5. *A morphism $\Psi : \mathcal{G}' \to \mathcal{G}$ of monoidal graphs is a pair of functions $S_\Psi : S_{\mathcal{G}} \to S_{\mathcal{G}'}, B_\Psi : B_{\mathcal{G}} \to B_{\mathcal{G}'}$ such that $S^*_\Psi \circ s = s \circ B_\Psi$ and $S^*_\Psi \circ t = t \circ B_\Psi$, where S^*_Ψ is the unique monoid homorphism determined by S_Ψ. Graphically, a morphism of monoidal graphs acts as follows:*



Monoidal graphs and their morphisms form a category MonGraph. Moreover, just as a monoidal graph freely generates a pro, a morphism of monoidal graphs freely generates a *morphism of pros.* Recall that a *strict monoidal functor* is a functor $F : \mathcal{C} \to \mathcal{D}$, where $\mathcal{C}$ and $\mathcal{D}$ are monoidal categories, and $F(X \otimes Y) = F(X) \otimes F(Y)$, $F(I_C) = I_{\mathcal{D}}$. A morphism of pros is simply a strict monoidal functor whose action on objects is determined by a function between their sets of generating sorts.

A pro has an underlying monoidal graph, given by forgetting composition and monoidal product:

Definition 3.6. *The underlying monoidal graph $\mathscr{U}M$ of a pro M is defined to have $S_{\mathscr{U}M}$ the set of generators of the monoid of objects of M, and $B_{\mathscr{U}M} = \bigcup \mathsf{Mor}(M)$, the morphisms of M with $s, t : B_{\mathscr{U}M} \rightrightarrows S^*_{\mathscr{U}M}$ assign each morphism its source and target sorts.*

There is a "free-forgetful" adjunction $\mathscr{F} \dashv \mathscr{U}$: Pro $\to$ MonGraph (see for example, [25, Proposition 6.1]). We will use this later in Section 6 to define inductive extensions of monoidal automata.

Remark 3.7. *In the introduction we remarked that strict monoidal categories might also be seen as* 2-dimensional monoids*, and hence a natural candidate for the algebra of higher-dimensional formal languages. In general we can define n-monoid to mean strict n-category with one object, as in Burroni [5] who introduced the* word problem for n-monoids*. Monoids are equivalent to one-object categories, and moving up a dimension corresponds to introducing morphisms* between *the elements of a monoid.*

4. Planar Monoidal Languages and Regular Monoidal Grammars

Just as a classical word language is a subset of a finitely generated free monoid, a monoidal language is a set of morphisms in a finitely generated free pro. In particular, we are interested in the *scalar* morphisms: those from the monoidal unit to itself, in other words, string diagrams with no dangling wires. Recall that a *monoidal alphabet* is defined to be single-sorted finite monoidal graph.

Definition 4.1. *A planar* monoidal language *L over a monoidal alphabet Γ is a subset $L \subseteq \mathscr{F}\Gamma(0, 0)$ of morphisms with arity and coarity 0 in the free pro generated by Γ.*

The restriction to arity and coarity zero (i.e. *scalar*) morphisms may appear arbitrary. However, we will see in Section 7 that this captures and explains the classical definitions of finite-state automata over words and trees. It also leads to more concise definitions in our theory. We will usually drop the qualifier planar, since they are the main object of investigation. In Section 12 we highlight some work in progress on *symmetric monoidal languages*, which permit non-planar string diagrams.

Regular monoidal grammars specify a class of monoidal languages analogous to regular languages. The starting point of our definition of regular monoidal grammar and their associated monoidal languages is inspired by Walters [39], so we briefly explain his elegant idea.

Walters [39] introduced an algebraic definition of regular grammars as morphisms of finite graphs. It is commonplace to represent finite-state automata as labelled directed graphs and this is the starting intuition. A labelled directed graph can be seen as a morphism of graphs $\phi : G \to \Sigma$, from a graph G whose vertices are *states* and edges *transitions* to a graph Σ with one vertex whose edges are labels. A *morphism* of graphs is a function on edges and a function on vertices, commuting with source and target assigments. On vertices a morphism $G \to \Sigma$ is trivial: it must send every state to the single vertex of Σ. On edges of G, it assigns labels, which are elements of the alphabet.

Following Walters, such a morphism is called a grammar rather than an automaton, because Σ appears in the codomain: when we consider Σ as a set of *inputs* for an automaton, it should (and will, in Definition 6.1) appear in the *domain* of a morphism. Roughly speaking, a grammar is *fibered* over the alphabet, whereas in an automaton the alphabet *indexes* transitions.

There are many advantages to this innocent reframing of grammars as morphisms of graphs when it comes to language theory. It allows us to neatly describe operations on grammars, such as in proving closure operations in Section 8. It suggests various generalizations, by replacing graphs with other kinds of structure. Walters does this with multi-input, single-output graphs, obtaining context-free languages using a similar construction. Using monoidal graphs instead leads to definition of regular monoidal grammars:

Definition 4.2. *A* regular monoidal grammar *is a morphism of monoidal graphs $\Psi : \mathcal{M} \to \Gamma$ where $\mathcal{M}$ is finite, and Γ is a monoidal alphabet.*

Intuitively, a regular monoidal grammar is a labelling of the edges of $\mathcal{M}$ by generators in Γ. Indeed, since Γ is single-sorted, the sort function $_{\Psi} : S_{\mathcal{M}} \to \{\bullet\}$ is unique, and the grammar is determined by its box function $B_{\Psi} : B_{\mathcal{M}} \to B_{\Gamma}$, sending boxes to their labels. In Section 6 we show that this data determines a transition system with states words $w \in S^*_{\mathcal{M}}$.

Walters' definition also allows us to concisely describe the language associated to a grammar. A derivation in a regular grammar $G \to \Sigma$ corresponds to a path in G, and the accepted word is given by the labelling of the path. Formally, we can use the fact that any directed graph G generates a *free category* $\mathcal{F}G$, having objects the vertices and morphisms the *paths.* In particular, the free category on a single vertex graph Σ is the free monoid over the set of edges. Furthermore, any morphism of graphs generates a functor. If i, f are chosen vertices of G, then the language of the grammar is simply the image of the set of morphisms from i to f in $\mathcal{F}G$ under the associated functor $\mathcal{F}\phi$, giving a subset of Σ^*.

Remark 4.3. *Regular monoidal grammars determine morphisms between free pro(p)s, $\mathscr{F}\Psi : \mathscr{F}\mathcal{M} \to \mathscr{F}\Gamma$. We may also refer to these morphisms as grammars.*

Pros are monadic over monoidal graphs: the forgetful functor $\mathscr{U} : \mathsf{Pro} \to \mathsf{MonGraph}$ has a left adjoint $\mathscr{F} : \mathsf{MonGraph} \to \mathsf{Pro}$, and Pro is equivalent to the category of algebras for the induced monad on $\mathsf{MonGraph}$ (see [20, §2.3]). $\mathscr{F}$ sends a monoidal graph $\mathcal{G}$ to a pro $\mathscr{F}\mathcal{G}$ whose set of objects is $V_{\mathcal{G}}^*$ and whose morphisms are *string diagrams* (see [35]).

For any string diagram $s \in \mathscr{F}\Gamma$ over an alphabet Γ, we can think of the set of string diagrams $\mathscr{F}\Psi^{-1}(s)$ as a set of possible "parsings" of that diagram. From another perspective, we can think of a string diagram $s \in \mathscr{F}\mathcal{M}$ as representing a specification for the construction of the string diagram $\mathscr{F}\Psi(s) \in \mathscr{F}\Gamma$ to which the grammar maps it: the specification is a decomposition of the desired diagram into generators with typed boundaries that specify how they should be composed.

Remark 4.4. *We represent regular monoidal grammars diagrammatically by drawing the monoidal graph $\mathcal{M}$ as above, but labelling each box $b \in B_{\mathcal{M}}$ with $B_{\Psi}(e)$. The resulting diagram is not in general a diagram of a monoidal graph, since it may contain boxes with the same label but different domain or codomain types. Examples are given below.*

4.1. Regular monoidal languages

Regular monoidal grammars specify monoidal languages that are an analogue of classical regular languages. They can be obtained by taking Walters' [39] definition of regular language and replacing the adjunction between reflexive graphs and categories with that between monoidal graphs and pros. As shown in Section 7, they include the classical definitions of regular tree and word languages as grammars over monoidal alphabets of a particular shape.

Regular monoidal grammars specify a subset of monoidal languages, which specialize to the usual regular languages when $\mathcal{M}$ is free over a signature containing $1 \to 1$ generators (Section 7).

A regular monoidal grammar determines a monoidal language as follows:

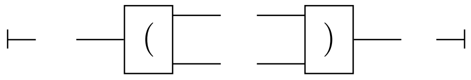
Definition 4.5. *Given a regular monoidal grammar $\Psi : \mathcal{M} \to \Gamma$, the image under $\mathscr{F}\Psi$ of the endo-hom-set of the monoidal unit ε in $\mathscr{F}\mathcal{M}$ is a planar monoidal language $\mathscr{F}\Psi[\mathscr{F}\mathcal{M}(\varepsilon, \varepsilon)] \subseteq \mathscr{F}\Gamma(0,0)$. We call the class of languages determined by regular monoidal grammars the* regular planar monoidal languages.

The basic idea is that a "word" is a scalar string diagram, i.e. one with no "dangling strings". The language of a monoidal grammar then consists of those scalar string diagrams that can be given a parsing. Parsings can be visually explained using the graphical notation for grammars (Remark 4.4). A morphism in the language defined by a grammar is any string diagram that can be built using the "typed" building blocks, such that there are no dangling strings, and then erasing the types on the strings. The following examples of regular monoidal grammars illustrate this idea:

Example 4.6 (Regular languages of words)**.** *Let Γ be a monoidal alphabet containing only generators of type $1 \to 1$, along with "start" and "end" generators*

$0 \to 1$ and $1 \to 0$ respectively. Then regular monoidal languages over such an alphabet are exactly regular languages of words over the generators $1 \to 1$, and every regular language arises from a regular monoidal grammar of this form. We investigate this in more detail from the perspective of automata in Section 7.

Example 4.7 (Balanced parentheses)**.** *Recall that the Dyck language, the language of balanced parentheses, is a paradigmatic example of a non-regular word language. We can simulate recognition of the Dyck language using a regular monoidal grammar over the following monoidal alphabet. Note that we do not obtain exactly the classical Dyck language of words, since our "brackets" have arities $1 \to 2$ and $2 \to 1$:*



The regular monoidal grammar for balanced parentheses over this alphabet is given in Figure 3. An example of a morphism in the language defined by this grammar is shown on the right in Figure 3.

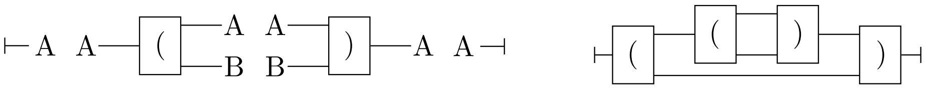


Figure 3: A regular monoidal grammar for balanced parentheses (left), and a morphism in the language (right).

This illustrates how regular monoidal grammars permit unbounded concurrency. Here, as one scans from left to right, the (unbounded) size of the internal boundary of a string diagram keeps track of the number of open left parentheses.

The following two examples (Examples 4.8 and 4.9) are introduced in order to set up Example 4.10.

Example 4.8 (Brick walls)**.** *The following two colour brick alphabet allows us to define a two-colour variant of the "brick wall" language introduced by Bossut [4].*

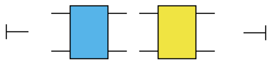


Figure 4 gives the grammar for two-coloured brick walls. An example of a morphism in the language defined by this grammar is shown on the right in Figure 4.

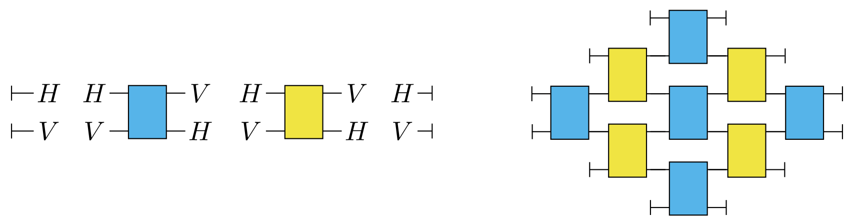


Figure 4: The grammar for two-coloured brick walls (left), and a morphism in the language (right).

Example 4.9 (XOR)**.** *The following grammar over the two colour brick alphabet is determined by computation of the XOR gate, based on the cellular automaton appearing in the work of Rothemund, Papadakis, and Winfree [33] in the context of self-assembly of DNA tiles. The output of each tile is the duplication of the XOR of the inputs.*

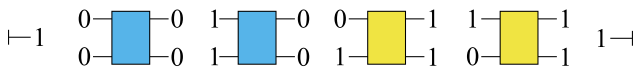


Figure 5: The XOR language grammar.

Example 4.10 (Sierpiński triangles)**.** *Rothemund, Papadakis, and Winfree [33] introduce a cellular automaton that computes the Sierpiński triangle fractal. This arises as the intersection of the languages generated by the grammars of Examples 4.8 and 4.9. The corresponding monoidal grammar is given in Figure 6.*

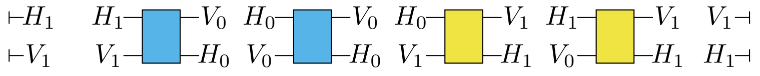


Figure 6: The regular monoidal grammar of Sierpiński triangles.

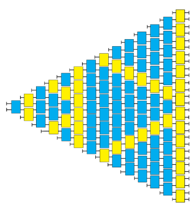
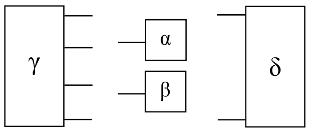


Figure 7: An element of the regular monoidal language of Sierpiński triangles.

Example 4.11. *Consider the following monoidal alphabet:*



The grammar over this alphabet in Figure 8 has a language whose elements with one connected component are exactly two (Figure 9). This grammar will serve as a running counterexample in Sections 10 and 11, as it defines a language that cannot be deterministically recognized (Figure 8). In particular it does not satisfy the property of causal closure*, a necessary condition for deterministic recognizability which we introduce in Section 10.*

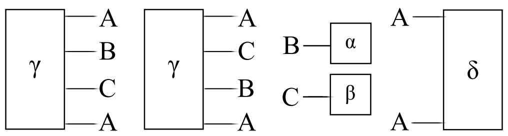


Figure 8: This grammar is "non-deterministic": there are two possible transitions from the empty word when encountering γ.

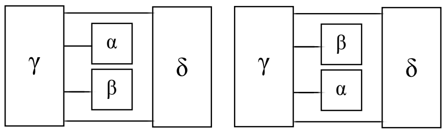


Figure 9: The connected string diagrams in the language of Figure 8.

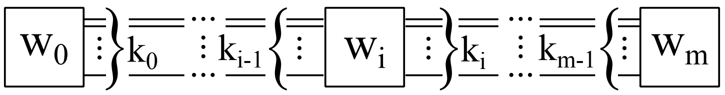
Again, in this paper we will often drop "planar" and just speak of *regular monoidal languages*, since all of the languages treated here are planar. We shall see in Section 6 that regular monoidal languages are precisely the languages accepted by *non-deterministic monoidal automata.*

Remark 4.12. *If the monoidal graph $\mathcal{M}$ has no edges whose domain is ε or no edges whose codomain is ε, a regular monoidal grammar will define a language containing only the identity on the monoidal unit, i.e. the empty string diagram (denoted $\boxed{}$). In fact,* every *monoidal language contains the empty string diagram.*

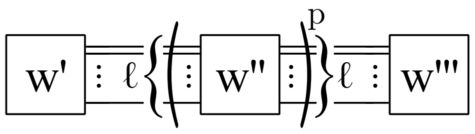
5. Pumping lemma for regular monoidal languages

In this section we prove a pumping lemma for regular monoidal languages (Lemma 5.1). We use this lemma to show that the language of *unbraids* is a monoidal language which is not regular monoidal.

Lemma 5.1 (Monoidal pumping lemma)**.** *Let L be a regular monoidal language. Then $\forall k \in \mathbb{N}^+, \exists n > 0$ such that for any $\boxed{\mathrm{w}} \in L$ where $\boxed{\mathrm{w}}$ may be factorized (as* *follows) into $m \geqslant n$ morphisms with boundaries (k_{i-1}, k_i) of width $1 \leqslant k_i \leqslant k$ and such that no $\boxed{\mathrm{w_i}}$ is an identity morphism:*



there exists $i < j$ such that $k_i = k_j = \ell$ and



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

where $(w'')^p$ in the diagram indicates sequential repetition of w'', p times, and $w' = w_0; ...; w_i$, $w'' = w_{i+1}; ...; w_j$, and $w''' = w_{j+1}; ...; w_m$.

Proof. Let L be the language of the grammar $\varphi : M \to \Gamma$. If L has a finite number of connected elements, then for any k take n to be longer than the longest factorization over all diagrams in L, then the lemma holds vacuously. Otherwise let k be given, then take $n = \sum_{i=0}^{k} |S_M|^i$, where $|S_M|$ is the number of sorts in M. Let $\boxed{\mathrm{w}} \in L$, such that it has a factorization of the form above. The chosen n guarantees that some vector of sorts $(S_1, ..., S_{k_i})$ where $S_l \in S_M$ must be repeated when generating $\boxed{\mathrm{w}}$ according to the grammar. That is, by the pigeonhole principle, we will have i, j, ℓ as required in the lemma. $\square$

Corollary 5.2 (Contrapositive form)**.** *Let L be a monoidal language and suppose that $\exists k \in \mathbb{N}^+$ such that $\forall n > 0$ there exists a morphism $w \in L$ that factorizes as above and for all $i < j$ such that $k_i = k_j = \ell$, there exists a p such that the pumped morphism $w'w''^p w''' \notin L$, then L is not regular monoidal.*

Observation 5.3. *This reduces to the usual pumping lemmas for words and trees, when φ is a regular word or regular tree grammar (Example 7), taking $k = 1$.*

We can use the monoidal pumping lemma to prove that languages of "unbraids" on n-strings, considered as monoidal languages, are not regular monoidal. The "crossing" generators in the following are syntax for *braidings*: under- and over- crossings of strings, allowing them to tangle.

Definition 5.4 (Unbraid languages)**.** *The language of unbraids on $m \geqslant 2$ strings, $\mathsf{Unbraid}_m$, is a monoidal language defined over a monoidal alphabet with an under-braid, over-braid, and start, end generators with k prongs for $0 \leqslant k \leqslant m$. For example, taking $m = 4$ we have the alphabet in Figure 10.*

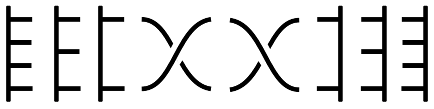


Figure 10: Monoidal alphabet for $\mathsf{Unbraid}_4$.

Connected elements of $\mathsf{Unbraid}_m$ are defined to be string diagrams with one start and one end generator, between which there is a configuration of at most m strings which could be "unbraided" to give two parallel strings, by planar isotopy. For example, Figure 11 (left) is an element of $\mathsf{Unbraid}_m$ for all $m \geqslant 2$, but Figure 11 (right) is not. The non-connected elements of the language are tensor products (juxtapositions of string diagrams) of these connected elements.

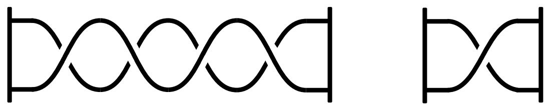


Figure 11: (Left) Example of an element in $\mathsf{Unbraid}_m$: by it could be untangled by planar deformations. (Right) An string diagram not in $\mathsf{Unbraid}_m$: we cannot uncross the strings using only planar moves.

To prove that $\mathsf{Unbraid}_m$ is not regular monoidal, we will make use of the following regular monoidal language:

Definition 5.5 (Over-under language on two strings). $(O^*U^* + U^*O^*)_2$ *is the regular monoidal language whose connected elements are an arbitrary number of over-braidings of two strings followed by an arbitrary number of under-braidings of two strings, or vice-versa. A grammar for this language is the following:*

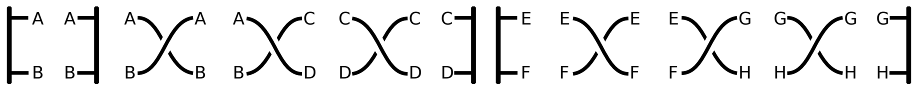


Figure 12: A regular monoidal grammar for the monoidal language $(O^*U^* + U^*O^*)_2$.

Proposition 5.6. *The languages $\mathsf{Unbraid}_m$ ($m \geqslant 2$) are not regular monoidal.*

Proof. Consider the intersection $\mathsf{Unbraid}_m \cap (O^*U^* + U^*O^*)_2$. An element of $(O^*U^* + U^*O^*)_2$ is an unbraid just when it comprises p under-braidings followed by p over-braidings, or vice-versa and thus these are the elements of the intersection, which we denote by $(O^pU^p + U^pO^p)_2$. Figure 12 witnesses the regularity of $(O^*U^* + U^*O^*)_2$, and the intersection of regular monoidal languages is regular (Lemma 8.2), thus it will suffice to prove that $L = (O^pU^p + U^pO^p)_2$ is not regular monoidal.

We use Corollary 5.2. Let $k = 2$ and $n > 0$ be given, and let $w \in L$ be the connected element having n over-braidings (O) followed by n under-braidings (U), with factorization $w = O^nU^n$. Finally let $i < j$ be given (all i, j satisfy $k_i = k_j = 2$). Then we have three cases for the pumping section w'': either it consists of $j - i$ Os, $j - i$ Us, or some number of Os followed by some number of Us. In the first two cases, pumping the section leads to a term with more Os than Us or vice-versa, and in the last case it will no longer be that all Os come before all Us. □

Remark 5.7. *Proposition 5.6 raises the question of whether context-free monoidal languages could be defined. Alongside his algebraic definition of regular grammar, Walters [39] introduced a similar definition of context-free grammar using the notion of multicategory. This has been revisited in the recent work of Melliès and Zeilberger [26], who gave a conceptual proof of the Chomsky-Schützenberger representation theorem using the algebra of multicategories. Earnshaw, Hefford, and Román [15] recently extended this algebra to the setting of monoidal categories, providing a foundation for a notion of context-free monoidal grammar to be pursued in future work.*

6. Non-deterministic monoidal automata

Recall that a non-deterministic finite automaton (NFA) is given by a finite set Q of states, an initial state $i \in Q$, a set of final states $F \subseteq Q$, and for each $a \in \Sigma$, a function $Q \xrightarrow{\Delta_a} \mathscr{P}(Q)$. Automata accepting monoidal languages, or *monoidal automata* are defined similarly, but the type of the transition function will be different for each generator $\gamma \in \Gamma$: for $\gamma : n \to m$, we have a transition function $Q^n \xrightarrow{\Delta_\gamma} \mathscr{P}(Q^m)$, taking vectors of states to (sets of) vectors of states.

Our *monoidal automata* do not have initial and final states; these are effectively taken over by transitions corresponding to generators of arity 0 and coarity 0, respectively. This corresponds to our languages consisting of *scalar* string diagrams; those with no dangling wires. For example, Figure 13 illustrates an accepting run over the automaton corresponding to the grammar of Example 4.7.

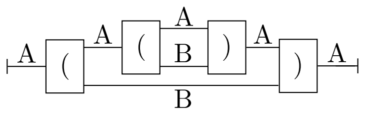


Figure 13: A run in the monoidal automaton for balanced parentheses.

A label next to a wire indicates the state "carried" by that wire. The accepted term is what is left if we erase these labels. In this example, the state vector undergoes the following transformations $\varepsilon \to A \to (A, B) \to (A, B, B) \to (A, B) \to A \to \varepsilon$. We now proceed to the formal definitions.

Definition 6.1. *A non-deterministic monoidal automaton over a monoidal alphabet* Γ *comprises:*

- *a finite set of states* Q*,*
- *for each generator* $\gamma : n \to m$ *in* Γ*, a transition function* $\Delta_\gamma : Q^n \to \mathscr{P}(Q^m)$*.*

In Section 7, we will see that initial and final states for regular and tree automata derive from this definition, when the alphabet is of a particular shape.

For classical NFAs, the assignment $a \mapsto \Delta_a$ extends uniquely to a functor $\Sigma^* \to \mathsf{Rel}$, the inductive extension of the transition structure from letters to words. Similarly, every non-deterministic monoidal automaton determines a monoidal language.

We define the inductive extension of monoidal automata from generators to string diagrams. First recall the definition of the endomorphism pro of an object in a monoidal category:

Definition 6.2. *Let $\mathscr{C}$ be a monoidal category, and Q an object of $\mathscr{C}$. The endomorphism pro of Q, $\mathscr{C}_Q$, has natural numbers as objects, hom-sets $\mathscr{C}_Q(n, m) := \mathscr{C}(Q^n, Q^m)$, composition and identities as in $\mathscr{C}$. The monoidal product is addition on objects, and as in $\mathscr{C}$ on morphisms.*

The codomains of our inductive extension will be endomorphism pros of finite sets Q in Rel, the Kleisli category of the powerset monad $\mathscr{P}$. The morphisms $X \to Y$ in this Kleisli category are functions $X \to \mathscr{P}(Y)$, which we think of as relations between sets X and Y. Since $\mathscr{P}$ is a commutative monad (with respect to the cartesian product of sets, with $\mathscr{P}X \times \mathscr{P}Y \to \mathscr{P}(X \times Y)$ given by the product of subsets), the following lemma gives us the monoidal structure on Rel:

Lemma 6.3 ([30], Corollary 4.3)**.** *Let T be a commutative monad on a symmetric monoidal category $\mathscr{C}$. Then the Kleisli category $\mathsf{Kl}(T)$ has a canonical monoidal structure, which is given on objects by the monoidal product in $\mathscr{C}$, and on morphisms $f : X \to TA, g : Y \to TB$ by $X \otimes Y \xrightarrow{f \otimes g} TA \otimes TB \xrightarrow{\nabla} T(A \otimes B)$, where ∇ is the monoidal multiplication given by the commutativity of T.*

For the powerset monad, the monoidal product $\otimes$ is the cartesian product of sets, and monoidal multiplication ∇ is given by the cartesian product of sets.

Definition 6.1 amounts to a morphism of monoidal graphs from Γ, to the underlying monoidal graph of the endomorphism pro Rel_Q, defined as follows:

Definition 6.4. *Let Q be a set of states, then Rel_Q is the pro with:*

- *set of objects $\mathbb{N}$,*
- *morphisms $n \to m$ are functions $Q^n \to \mathscr{P}(Q^m)$,*
- *composition is Kleisli composition, i.e. the usual composition of relations $f \circ g := \mu \circ \mathscr{P}(g) \circ f$, where μ takes the union of subsets,*
- *monoidal product of objects given by addition, and*
- *monoidal product of morphisms $f : n \to m$ and $g : p \to q$ by $\nabla \circ (f \times g) : n + p \to m + q$, where ∇ is the monoidal multiplication of $\mathcal{P}$, i.e. the cartesian product of sets.*

Remark 6.5. *The maybe monad $(-)_\perp$ is also commutative, so its Kleisli category also has a canonical monoidal structure. The objects of this category are* *sets, and the morphisms $X \to Y$ are functions $X \to Y + \{\bot\}$, which we think of as partial functions $X \to Y$. The canonical monoidal structure is given on objects by the cartesian product of sets and on morphisms by first taking their cartesian product, then collapsing $(\bot, \bot)$ to $\bot$. We return to this in Section 10.* Stochastic *monoidal automata could be obtained by use of the* distribution monad *[27], whose Kleisli category has* stochastic matrices *as morphisms.*

Now we can define the inductive extension of a non-deterministic monoidal automaton:

Observation 6.6. *The assignment of generators to transition functions $\gamma \mapsto \Delta_\gamma$ in Definition 6.1 determines a morphism of monoidal graphs $\Gamma \to \mathscr{U}(\mathsf{Rel}_Q)$. By the adjunction $\mathscr{F} \dashv \mathscr{U}$ (Section 4), such morphisms are in bijection with pro morphisms $\Delta : \mathscr{F}\Gamma \to \mathsf{Rel}_Q$. We will also refer to the inductive extension Δ as a non-deterministic monoidal automaton, and sometimes write Δ_α for the relation $\Delta(\alpha : n \to m)$.*

A scalar string diagram is mapped to one of the two possible nullary relations $\{\bullet\} \to \mathscr{P}(\{\bullet\})$, which represent accepting or rejecting computations, and thus can be used to define the language of the automaton:

Definition 6.7. *Let $\Delta : \mathscr{F}\Gamma \to \mathsf{Rel}_Q$ be a non-deterministic monoidal automaton. Then the monoidal language accepted by Δ is $\mathscr{L}(\Delta) := \{\alpha \in \mathscr{F}\Gamma(0,0) \mid \Delta_\alpha(\bullet) = \{\bullet\}\}$.*

There is an evident correspondence between non-deterministic monoidal automata and regular monoidal grammars. The graphical representation of a grammar makes this most clear: it can also be thought of as the "transition graph" of a non-deterministic monoidal automaton. More explicitly we have:

Proposition 6.8. *Given a regular monoidal grammar $\Psi : \mathcal{M} \to \Gamma$, define a monoidal automaton with $Q = S_{\mathcal{M}}$, $w(\Delta_\gamma)w' \iff \exists \sigma \in B_\Psi^{-1}(\gamma)$ such that $s(\sigma) = w, t(\sigma) = w'$. Conversely given a monoidal automaton (Q, Δ_Γ), define a regular monoidal grammar with $S_{\mathcal{M}} = Q$ and take an edge $w \to w'$ over $\gamma \iff w(\Delta_\gamma)w'$. This correspondence of grammars and automata preserves the recognized language.*

7. Regular word and tree languages as regular monoidal languages

Classical non-deterministic finite-state automata and tree automata can be seen as non-deterministic monoidal automata over alphabets of a particular shape.

To make the correspondence precise, in the following we restrict monoidal languages to their *connected* string diagrams. Strictly speaking, the language of a monoidal automaton always contains only the empty diagram or is countably infinite, because if α is accepted by the automaton, so are arbitrary finite monoidal products $\alpha \otimes \cdots \otimes \alpha$. However, it is of course possible for a monoidal language to consist of a finite number of connected string diagrams.

From another perspective, without restricting to connected components, we can say that the monoidal automata corresponding to finite-state and tree automata have the power of an unbounded number of such classical automata running in parallel.

7.1. Finite-state automata

Definition 7.1. *A word monoidal alphabet is a monoidal alphabet having only generators of arity and coarity 1,* $-\boxed{\sigma}-$ *, along with a single "start" generator* $\vdash$ *of arity 0 and coarity 1, and "end" generator* $\dashv$ *of arity 1 and coarity 0.*

Observation 7.2. *Non-deterministic monoidal automata over word monoidal alphabets correspond to classical NFAs.*

Let an NFA $A = (Q, \Sigma, \Delta, i, F)$ be given. We build a monoidal automaton as follows. Form the monoidal alphabet Σ' by starting with generators $\vdash$, $\dashv$ and adding generators $-\boxed{\sigma}-$ for each $\sigma \in \Sigma$. For each $-\boxed{\sigma}-$, take the transition function $\Delta_\sigma := \Delta(\sigma,-) : Q \to \mathscr{P}(Q)$. For $\dashv$ take the transition function $Q \to \mathscr{P}(Q^0)$ to be the characteristic function of $F \subseteq Q$, sending elements of F to $\{\bullet\}$ and to $\varnothing$ otherwise, and for $\vdash$ take the function $Q^0 \to \mathscr{P}(Q)$ to pick out the singleton $\{i\}$. This defines a monoidal automaton $A' := (Q, {\Delta'}_{\Sigma'})$, and a simple induction shows that $\mathscr{L}(A) = \mathscr{L}(A')$, if one restricts to connected string diagrams.

Conversely, the data of a monoidal automaton over a word monoidal alphabet corresponds to the data of an NFA, the only difference being that the transition function associated to $\vdash$ picks out a *set* of initial states $\{\bullet\} \to \mathscr{P}(Q)$. We can always "normalize" such an automaton into an equivalent NFA with one initial state (see [34, §2.3.1]). This shows how NFA initial and final states are captured by this particular shape of monoidal alphabet.

7.2. Tree automata as monoidal automata

Recall that non-deterministic finite tree automata come in two flavours, bottom-up and top-down, depending on whether they process a tree starting at the leaves or at the root, respectively. A non-deterministic bottom-up finite tree automaton is given by a finite set of states Q, a "ranked" alphabet $(\Sigma, r : \Sigma \to \mathbb{N})$, a set of final states $F \subseteq Q$, and for each $\sigma \in \Sigma$ a transition function $\Delta_\sigma : Q^{r(\sigma)} \to \mathscr{P}(Q)$. A non-deterministic top-down tree automaton, instead, has a set of initial states $I \subseteq Q$ and transition functions $\Delta_\sigma : Q \to \mathscr{P}(Q^{r(\sigma)})$. We can recover these as non-deterministic monoidal automata over *tree monoidal alphabets*:

Definition 7.3. *A top-down tree monoidal alphabet is a monoidal alphabet having only generators of arity 1 (and arbitrary coarities $\geqslant 0$),* $-\boxed{\sigma}\!<\!\vdots$ *, along with a single "root" generator* $\vdash$ *. Analogously, a bottom-up tree monoidal alphabet is a monoidal alphabet having only generators of coarity 1 (and arbitrary arities $\geqslant 0$),* $\vdots\!>\!\boxed{\sigma}-$ *, along with a single "root" generator* $\dashv$ *.*

Observation 7.4. *Bottom-up tree automata are exactly non-deterministic monoidal automata over bottom-up tree monoidal alphabets, and likewise for top-down tree automata.*

Of course, when the coarities of the generators in a top-down tree alphabet are all 1 (or likewise for the arities of a bottom-up tree alphabet), trees are just *words*, and we obtain finite-state automata over words.

For example, consider the following graph of a monoidal automaton over a bottom-up tree monoidal alphabet, recognizing trees corresponding to terms of the inductive type of lists of boolean values (a list may be empty, [], or be a boolean value "consed" onto a list via ::).

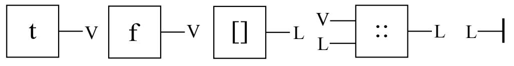


Figure 14: Transition graph of a monoidal automata for a tree language, accepting trees corresponding to lists of boolean values.

Intuitively, the connected scalar string diagrams determined by this language are trees, with leaves on the left, and the root on the right. Monoidal automata over top-down tree monoidal alphabets have a similar form, but are mirrored horizontally. Thus morphisms in the language have the root on the left, leaves on the right, and monoidal automata start at the root.

8. Closure properties of regular monoidal languages

We record some closure properties of regular monoidal languages, making use of their representation as grammars and as automata, where appropriate.

Lemma 8.1 (Closure under union)**.** *Let L and L' be regular monoidal languages over Γ. Then $L \cup L'$ is a regular monoidal language over Γ.*

Proof. Let L and L' be given by the regular monoidal grammars $\Psi : \mathcal{M} \to \Gamma, \Psi' : \mathcal{M}' \to \Gamma$ respectively. Define the grammar $\Psi + \Psi' : \mathcal{M} + \mathcal{M}' \to \Gamma$, where $B_{\mathcal{M}+\mathcal{M}'} := B_{\mathcal{M}} + B_{\mathcal{M}'}, S_{\mathcal{M}+\mathcal{M}'} := S_{\mathcal{M}} + S_{\mathcal{M}'}$, and $B_{\Psi+\Psi'} := [\Psi, \Psi']$ (the copairing of Ψ and Ψ'). Graphically, this is just taking the disjoint union of two grammars, and it is clear that the language defined in this way is the union of the languages defined by the two grammars. $\Box$

Lemma 8.2 (Closure under intersection)**.** *Let L and L' be regular monoidal languages over Γ. Then $L \cap L'$ is a regular monoidal language over Γ.*

Proof. Let L and L' be recognized by non-deterministic automata $(Q, \{\Delta_\gamma\}_{\gamma\in\Gamma})$, $(Q', \{\Delta'_\gamma\}_{\gamma\in\Gamma})$ respectively. Consider the product automaton $(Q \times Q', \{(\Delta \times \Delta')_\gamma\}_{\gamma\in\Gamma}$, with $(\Delta \times \Delta')_\gamma := \nabla \circ (\Delta_\gamma \times \Delta'_\gamma)$, where ∇ maps pairs of subsets to their cartesian product. Then α is accepted by the product automaton just when it is accepted by both, so $\mathscr{L}(\Delta \times \Delta') = L \cap L'$. $\Box$

Remark 8.3. *The Sierpiński triangle language (Example 4.10) is the intersection of the brick wall language (Example 4.8) the XOR language (Example 4.9): this explains the origin of the states in the grammar shown in Example 4.10.*

Lemma 8.4 (Closure under monoidal product and factorizations)**.** *Let L be a regular monoidal language. Then $\alpha, \beta \in L \iff \alpha \otimes \beta \in L$.*

Proof. Let $(Q, \{\Delta_\gamma\}_{\gamma \in \Gamma})$ be an automaton accepting both α and β. Since the inductive extension Δ is a strict monoidal functor, $\Delta(\alpha \otimes \beta) = \Delta(\alpha) \otimes \Delta(\beta)$, so we must have $\Delta(\alpha \otimes \beta)(\bullet) = \{\bullet\}$, and conversely. $\square$

Lemma 8.5 (Closure under images of alphabets)**.** *Let L be a regular monoidal language over Γ, and $\Gamma \xrightarrow{h} \Gamma'$ be a morphism of monoidal alphabets. Then $(\mathscr{F}h)L$ is a regular monoidal language over Γ'.*

Proof. Let L be given by the regular monoidal grammar $\Psi : \mathcal{M} \to \Gamma$, that is $L = \mathscr{F}\Psi[\mathscr{F}\mathcal{M}(\varepsilon, \varepsilon)]$. Consider the grammar given by the composite $h \circ \Psi : \mathcal{M} \to \Gamma'$. Since $\mathscr{F}$ is a functor we have: $\mathscr{F}(h \circ \Psi)[\mathscr{F}\mathcal{M}(\varepsilon, \varepsilon)] = (\mathscr{F}h \circ \mathscr{F}\Psi)[\mathscr{F}\mathcal{M}(\varepsilon, \varepsilon)] = (\mathscr{F}h)L$, thus $h \circ \Psi$ is a grammar for $(\mathscr{F}h)L$. $\square$

Lemma 8.6 (Closure under preimages of alphabets)**.** *Let L a regular monoidal language over Γ, and $\Gamma' \xrightarrow{h} \Gamma$ be a morphism of monoidal alphabets. Then the inverse image of L, $(\mathscr{F}h)^{-1}(L)$ is a regular monoidal language over Γ'.*

Proof. Let $(Q, \{\Delta_\gamma\}_{\gamma \in \Gamma})$ be an automaton recognizing L with inductive extension $\Delta : \mathscr{F}\Gamma \to \mathsf{Rel}_Q$. Consider the automaton given by the composite $\Delta \circ \mathscr{F}h : \mathscr{F}\Gamma' \to \mathsf{Rel}_Q$. We have $\mathscr{L}(\Delta \circ \mathscr{F}h) = (\mathscr{F}h)^{-1}(\mathscr{L}(\Delta)) = (\mathscr{F}h)^{-1}(L)$, so the inverse image of L is regular. $\square$

Closure under complement is often held to be an important criterion for what should count as a *recognizable* language. Indeed, for the abstract monadic second order logic introduced in [3], it is a *theorem* that the class of recognizable languages relative to a monad on Set is closed under complement. However, given that every monoidal language contains the empty string diagram, we obviously have that:

Observation 8.7. *Regular monoidal languages are not closed under complement.*

This suggests that there is no obvious account of regular monoidal languages in terms of monadic second order logic. On the other hand, there is no reason we should expect even the general account of monadic second order logic given in [3] to extend to monoidal categories, since these are not algebras for a monad on Set. Moreover, taking inspiration from classical examples in Section 7, one could also refine what is meant by complement, for instance focussing on the set of non-empty connected scalar diagrams.

9. The syntactic pro of a monoidal language

In this section we introduce the *syntactic congruence* on monoidal languages and their corresponding *syntactic pros*, by analogy with the syntactic congruence on classical regular languages and their associated syntactic monoids. In Section 10.1 we will give an algebraic property of the syntactic pro sufficient for a monoidal language to be deterministically recognizable. The syntactic congruence on a classical regular language is defined by examining words in contexts. We start by introducing a notion of context for string diagrams.

Definition 9.1. *A context of capacity (n, m) in a monoidal category $\mathbb{C}$ where $n, m \geqslant 0$, is a scalar string diagram with a hole (Figure 15) with zero or more additional strings exiting the first box and entering the second (indicated by ellipses in Figure 15).*

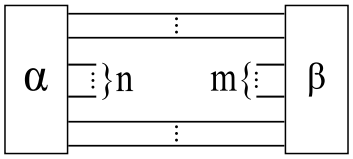


Figure 15: A context of capacity (n, m). α and β stand for arbitrary string diagrams with arity and coarity 0, respectively.

Given a context of capacity (n, m) in $\mathbb{C}$, we can fill the hole with a string diagram $\gamma : n \to m$ in $\mathbb{C}$. Write $C[\gamma]$ for the resulting morphism of $\mathbb{C}$. Note that the empty diagram is a context, the empty context of capacity $(0, 0)$. Contexts allow us to define contextual equivalence of string diagrams:

Definition 9.2 (Syntactic congruence)**.** *Given a monoidal language L we define its syntactic congruence $\equiv_L$ as follows. Let γ, δ be morphisms in $\mathscr{F}\Gamma(n, m)$. Then $\gamma \equiv_L \delta$ whenever $C[\gamma] \in L \iff C[\delta] \in L$, for all contexts C in $\mathscr{F}\Gamma$ of capacity (n, m).*

Definition 9.3. *The syntactic pro of a monoidal language L is the quotient pro $\mathscr{F}\Gamma/{\equiv_L}$. The quotient functor $S_L : \mathscr{F}\Gamma \to \mathscr{F}\Gamma/{\equiv_L}$ is the syntactic morphism of L. See Appendix A for the definition of quotient pro and quotient functor.*

Remark 9.4. *The syntactic congruences for classical regular languages of words and trees are also special cases of this congruence over word and tree monoidal alphabets.*

Lemma 9.5. *L is the inverse image along the syntactic morphism of the equivalence class of the empty diagram.*

Proof. Let $\alpha \in L$. Then $\alpha \equiv_L \boxed{}$, since the empty diagram is in every language and if C is a context of capacity $(0, 0)$ distinguishing α and $\boxed{}$, then we have a contradiction by Lemma 8.4, since $\alpha = \alpha \otimes \boxed{}$. So $\alpha \in S_L^{-1}(\left[\boxed{}\right])$, and conversely. □

In the terminology of algebraic language theory, we say that the syntactic morphism *recognizes* L. A full investigation of algebraic recognizability of monoidal languages is a topic for future work. For now, we record the following lemma which is needed for Theorem 10.11:

Lemma 9.6. *If a monoidal language L is regular, then its syntactic pro $\mathscr{F}\Gamma/{\equiv_L}$ is locally finite (i.e. has finite hom-sets).*

Proof. It suffices to exhibit a full pro morphism into $\mathscr{F}\Gamma/{\equiv_L}$ from a locally finite pro. Let L be a regular monoidal language recognized by $\Delta : \mathscr{F}\Gamma \to \mathsf{Rel}_Q$. Δ induces a congruence $\sim$ on $\mathscr{F}\Gamma$ defined by $\alpha \sim \beta \iff \Delta(\alpha) = \Delta(\beta)$, which implies that $\mathscr{F}\Gamma/{\sim}$ is locally finite, since Rel_Q is locally finite. Define the pro morphism $\mathscr{F}\Gamma/{\sim} \to \mathscr{F}\Gamma/{\equiv_L}$ to be identity on objects and $[\alpha]_\sim \mapsto [\alpha]_{\equiv_L}$ on morphisms. This is well-defined since if $\alpha \sim \beta$ and $C[\alpha] \in L$ for some context C, then by functoriality $C[\beta] \in L$. Clearly it is full, so $\mathscr{F}\Gamma/{\equiv_L}$ is locally finite. $\square$

10. Deterministic monoidal automata

The expressive equivalence of deterministic and non-deterministic finite-state automata for word languages is well-known, but already for trees, top-down deterministic tree automata are less expressive than bottom-up deterministic tree automata [21]. Therefore we cannot expect to determinize non-deterministic monoidal automata. However, we have already seen monoidal languages that are deterministically recognizable: Examples 4.8 and 4.10, interpreted as the transition relations of monoidal automata, are functional relations. Here we introduce deterministic monoidal automata and show that their languages enjoy the property of *causal closure*. In Section 11 we consider the question of determinizability.

Definition 10.1. *A deterministic monoidal automaton $\delta = (Q, \delta_\Gamma)$ over a monoidal graph Γ is given by a finite set Q, together with transition functions $\delta_\Gamma = \{Q^{ar(\gamma)} \xrightarrow{\delta_\gamma} Q_\perp^{coar(\gamma)}\}_{\gamma\in\Gamma}$, where $Q_\perp$ denotes the set $Q + \{\perp\}$.*

We write Par for the Kleisli category of the *maybe monad* $Q \mapsto Q_\perp$, whose morphisms we think of as partial functions. As mentioned in Remark 6.5, this category is equipped with a monoidal structure, given on objects by the cartesian product of sets. Recall from Definition 6.2 the notion of *endomorphism pro* of an object in a monoidal category. In the case of a set Q in Par, we have an endomorphism pro Par_Q whose objects are natural numbers and morphisms $n \to m$ are functions $Q^n \to Q_\perp^m$.

Then as in Observation 6.6, such assignments $\gamma \mapsto \delta_\gamma$ uniquely extend to morphisms of pros $\delta : \mathscr{F}\Gamma \to \mathsf{Par}_Q$, and we will also refer to such functors as deterministic monoidal automata. δ maps scalar string diagrams to one of the two functions $Q^0 \to Q_\perp^0$, and we use this to define the language of the automaton:

Definition 10.2. *Let $\delta : \mathscr{F}\Gamma \to \mathsf{Par}_Q$ be a deterministic monoidal automaton. Then the language accepted by δ is $\mathscr{L}(\delta) := \{\alpha \in \mathscr{F}\Gamma(0,0) \mid \delta_\alpha(\bullet) = \bullet\}$.*

We give a necessary condition for a monoidal language to be recognized by a deterministic monoidal automaton. The idea is to generalize the characterization of top-down deterministically recognizable tree languages as those that are closed under the operation of splitting a tree language into the set of possible paths through the trees, and reconstituting trees by grafting compatible paths [21]. For string diagrams, we call the analogue of paths through a tree the *causal histories* of a diagram (Definition 10.7).

First, we briefly recall the machinery of (cartesian) *restriction categories* [8], that will be necessary in the following. Restriction categories are an abstraction of the category of partial functions, and provide us with a diagrammatic calculus for reasoning about determinization of monoidal languages. Recall that a *prop* is a *pro* which is additionally equipped with a natural family of symmetry morphisms $\sigma_{A,B} : A \otimes B \to B \otimes A$ for every pair of objects [24].

Definition 10.3 (Cockett and Lack, Theorem 5.2 [9])**.** *A cartesian restriction category may be defined as a* counital copy category*: a symmetric monoidal category in which every object is equipped with a commutative comonoid structure (with the counit depicted by* $-\!\bullet$ *, comultiplication by* $-\!\!\bullet\!\!<$ *, and symmetry by* $\times$ *) that is coherent, and for which the comultiplication is natural (see Appendix B for details).*

Proposition 10.4 ([14])**.** *The free cartesian restriction prop on a monoidal graph $\mathcal{M}$, denoted $\mathscr{F}_{\downarrow}\mathcal{M}$ is given by taking the free prop on the monoidal graph $\mathcal{M}$ extended with a comultiplication and counit generator for every object in $V_{\mathcal{M}}$, and quotienting the morphisms by the structural equations of cartesian restriction categories (Appendix B).*

Remark 10.5. *Par is the paradigmatic example of a cartesian restriction category, with $-\!\bullet$ on X given by the function $X \to \{\bullet, \bot\}$ sending every element to $\bullet$, and $-\!\!\bullet\!\!<$ given by the diagonal function $q \mapsto (q, q)$. Par_Q inherits this structure and so is a cartesian restriction prop. Therefore deterministic monoidal automata (Q, δ_Γ) also have inductive extensions to morphisms of cartesian restriction props, $\overline{\delta} : \mathscr{F}_{\downarrow}\Gamma \to \mathsf{Par}_Q$, and these have a obvious notion of associated language, defined similarly to Definition 10.2. These are related by the following lemma, which follows from the universal properties of $\mathscr{F}\Gamma$ and $\mathscr{F}_{\downarrow}\Gamma$:*

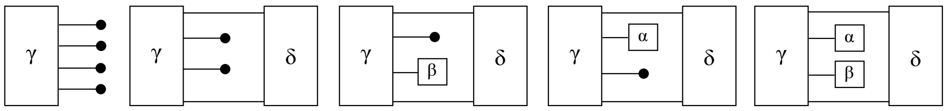
Lemma 10.6. *If (Q, δ_Γ) is a deterministic monoidal automaton, then δ factors through $\overline{\delta}$ as $\delta = \overline{\delta} \circ \mathscr{H}_\Gamma$, where $\mathcal{H}_\Gamma : \mathscr{F}\Gamma \to \mathscr{F}_{\downarrow}\Gamma$ sends morphisms to their equivalence class in $\mathscr{F}_{\downarrow}\Gamma$.*

The idea is that runs in the automaton $\mathscr{F}_{\downarrow}$ can freely duplicate ($-\!\!\bullet\!\!<$) or delete ($-\!\bullet$) an element in the state vector at any point in the run.

Recall that any restriction category is poset-enriched: $f \leqslant g$ if f is "less defined" than g, i.e. if f coincides with g on f's domain of definition [8, §2.1.4]. For the hom-set from the monoidal unit to itself, we have $f \leqslant g \iff f \otimes g = f$. Now we can define causal histories:

Definition 10.7. *Let γ be a string diagram in $\mathscr{F}\Gamma(0,0)$. We call a string diagram h in $\mathscr{F}_{\downarrow}\Gamma(0,0)$ a causal history of γ if $\mathcal{H}_{\Gamma}(\gamma) \leqslant h$ in $\mathcal{F}_{\downarrow}\Gamma(0,0)$. Let $L \subseteq \mathscr{F}\Gamma(0,0)$ be a regular monoidal language. The set of causal histories of L, denoted $ch(L)$, is defined to be $\mathcal{H}_{\Gamma}(L)^{\uparrow}$, the upwards closure of $\mathcal{H}_{\Gamma}(L)$ in the poset $\mathscr{F}_{\downarrow}\Gamma(0,0)$.*

A causal history represents the possible causal influence of parts of a diagram on generators appearing "later" in the diagram. For example, the following five string diagrams are causal histories of the rightmost string diagram below (every diagram is a causal history of itself), taken from the language introduced in Example 4.11:



Lemma 10.8. *Let $M = (Q, \delta_{\Gamma})$ be a deterministic monoidal automaton, with functors $\delta : \mathscr{F}\Gamma \to \mathsf{Par}_Q$, $\overline{\delta} : \mathscr{F}_{\downarrow}\Gamma \to \mathsf{Par}_Q$. Then if δ accepts γ, $\overline{\delta}$ accepts all causal histories of γ.*

Proof. Since $\delta = \overline{\delta} \circ \mathcal{H}_{\Gamma}$, if δ accepts γ, then $\overline{\delta}$ accepts $\mathcal{H}_{\Gamma}(\gamma)$. Let h be a causal history of γ. Then $\overline{\delta}(\mathcal{H}_{\Gamma}(\gamma)) = \overline{\delta}(h \otimes \mathcal{H}_{\Gamma}(\gamma)) = \overline{\delta}(h) \otimes \overline{\delta}(\mathcal{H}_{\Gamma}(\gamma))$. But then $\overline{\delta}$ accepts h by Lemma 8.4. $\square$

Definition 10.9 (Causal closure of a language)**.** *Let L be a monoidal language over a monoidal alphabet Γ. Let $\bigotimes ch(L)$ denote the closure of the set of causal histories of L under monoidal product. Then the causal closure $\mathsf{cl}(L)$ of L is $\mathcal{H}_{\Gamma}^{-1}(\bigotimes ch(L))$. A monoidal language is causally closed if it is equal to its causal closure.*

To illustrate causal closure, consider Figure 16, which shows part of the derivation of a morphism in the causal closure of the language of Example 4.11.

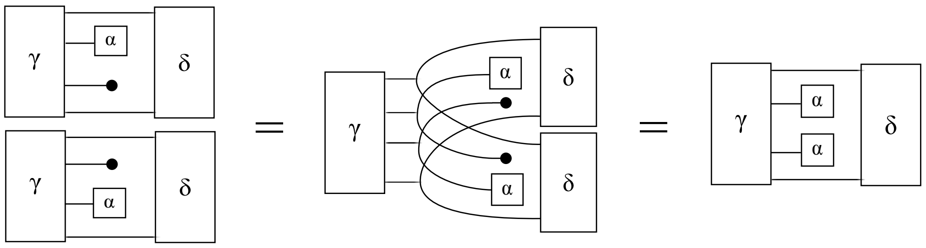
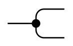
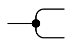


Figure 16: These string diagrams are equal in the equational theory of cartesian restriction categories. On the left, we have the monoidal product of two causal histories of elements of the language from Example 4.11. This determines a string diagram in the image of $\mathcal{H}_{\Gamma}$ (i.e. expressible without using the cartesian restriction structure), which is an element of the causal closure of the language.

The leftmost diagram in Figure 16 depicts the monoidal product of two causal histories determined by the counterexample language. By the equational theory of cartesian restriction categories (Appendix B), this is equal to the string diagrams in the center and on the right, where we first apply the naturality of  (for γ), then unitality (twice), then naturality of  (for δ). The rightmost form of the diagram exhibits this morphism as being in the image of $\mathcal{H}_\Gamma$, and its preimage under $\mathcal{H}_\Gamma$ is the same diagram in $\mathscr{F}\Gamma$. Since this diagram is not in the original language, the language is not causally closed. The following theorem tells us that this is enough to conclude that it is not recognizable by a deterministic monoidal automaton:

Theorem 10.10. *If a monoidal language is recognized by a deterministic monoidal automaton, then it is causally closed.*

Proof. Let L be recognized by a deterministic monoidal automaton $\delta : \mathscr{F}\Gamma \to \mathsf{Par}_Q$. We have $\delta = \overline{\delta} \circ \mathcal{H}_\Gamma$ and from Lemma 10.8 that $\overline{\delta}$ accepts causal histories of morphisms in L. Since languages are closed under monoidal product (Lemma 8.4), then by definition of the causal closure, δ must accept everything in the causal closure of L. $\square$

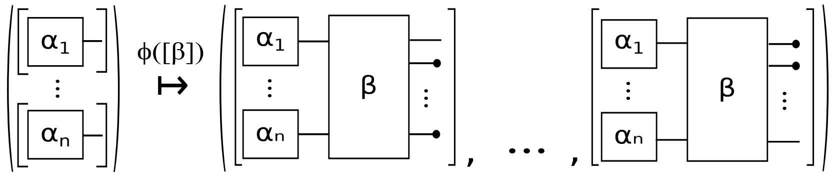
10.1. An algebraic sufficient condition for deterministic recognizability

There are many interesting theorems linking properties of the syntactic monoid of a classical language of words to properties of the language itself. Lemma 9.6 is one result of this type for regular monoidal languages. Here we give another, using Lemma 9.6 to characterize deterministic recognizability in terms of the existence of extra algebraic structure on the syntactic pro:

Theorem 10.11. *If the syntactic pro of a regular monoidal language has the structure of a cartesian restriction prop, then the language is recognizable by a deterministic monoidal automaton.*

Proof. Let L be a monoidal language such that $\mathscr{F}\Gamma/{\equiv_L}$ has a cartesian restriction prop structure. We exhibit a pro morphism $\mathscr{F}\Gamma/{\equiv_L} \xrightarrow{\phi} \mathsf{Par}_Q$ such that $\mathscr{F}\Gamma \xrightarrow{S_L} \mathscr{F}\Gamma/{\equiv_L} \xrightarrow{\phi} \mathsf{Par}_Q$ is a deterministic monoidal automaton accepting exactly L.

Let $Q := \mathscr{F}\Gamma/{\equiv_L}(0,1)$. By Lemma 9.6, this is a finite set. For $m > 0$ and $[\beta] \in \mathscr{F}\Gamma/{\equiv_L}(n,m)$, define $\phi([\beta]) : n \to m$ to be the following map from $Q^n \to Q^m_\perp$:



When $m = 0$, let $\phi([\beta])([\alpha_1], ..., [\alpha_n]) = \bullet$, if $[\beta \circ (\alpha_1 \otimes ... \otimes \alpha_n)] = \left[\,\boxed{}\,\right]$, and $\phi([\beta])([\alpha_1], ..., [\alpha_n]) = \bot$ otherwise. The proof that this defines a morphism of pros is an exercise in diagrammatic reasoning using the equational theory of cartesian restriction categories and can be found in Appendix C. To see that this automaton accepts exactly L, let $\alpha \in \mathscr{L}(\phi \circ S_L)$, then by definition we must have $S_L(\alpha) = \left[\,\boxed{}\,\right]$, and so $\alpha \in L$ (by Lemma 9.5). Conversely let $\alpha \in L$, then $S_L(\alpha) = \left[\,\boxed{}\,\right]$ and by definition $\phi\left(\left[\,\boxed{}\,\right]\right)(\bullet) = \bullet$, so $\alpha \in \mathscr{L}(\phi \circ S_L)$. Therefore $\phi \circ S_L$ is a deterministic monoidal automaton recognizing L. $\square$

Example 10.12. *A simple example is given by the language L of "bones" over the monoidal alphabet $\Gamma = \{ \bullet\!\!-, \; -\!\!\bullet \}$, having one connected component: $\bullet\!\!-\!\!\!-\!\!\bullet$. The syntactic pro of this language has a cartesian restriction prop structure, with the counit given by the equivalence class $[-\!\!\bullet]$, comultiplication by $[\,\dashv\!\!\!\bullet\,]$, and symmetry by $[\,\times\,]$. It is clear that $\mathscr{F}\Gamma/{\equiv_L}(0, 1)$ has one equivalence class, $[-\!\!\bullet]$, which becomes the state of the monoidal automaton. The construction above then gives the obvious transition functions required for each generator.*

11. Convex monoidal automata and the powerset construction

Non-deterministic finite state automata for words and bottom-up trees can be determinized via the well known powerset construction. However, top-down tree automata cannot be determinized in general [21, §2.11], so general monoidal automata also cannot be determinized (Observation 7.4). However, there are interesting examples of deterministically recognizable monoidal languages that are not tree languages, such as the monoidal Dyck language (Example 4.7) and Sierpiński triangles (Example 4.10), and it is an intriguing theoretical challenge to characterize such languages.

In this section we study a class of determinizable automata called *convex* automata, and introduce a powerset construction for them. The classical powerset construction is given conceptually by composition with the functor $\mathsf{Rel} \to \mathsf{Set}$, sending sets to their powersets and relations to their corresponding functions, right adjoint to the inclusion $\mathsf{Set} \hookrightarrow \mathsf{Rel}$. As remarked above, we cannot hope to obtain an analogue of this functor for monoidal automata. Thus we describe a suitable subcategory of Rel_Q for which determinization is functorial, that of convex relations.

Definition 11.1. *A relation $\Delta : Q^n \to \mathscr{P}(Q^m)$ is convex if there is a morphism Δ^* such that the following square commutes:*

$$\begin{tikzcd}
(\mathscr{P}Q)^n \arrow[r, "\Delta^*"] \arrow[d, "\nabla_{\mathscr{P}}"'] & (\mathscr{P}Q)^m \arrow[d, "\nabla_{\mathscr{P}}"] \\
\mathscr{P}(Q^n) \arrow[r, "\Delta^{\#}"] & \mathscr{P}(Q^m)
\end{tikzcd}$$

where $\Delta^{\#}$ is the Kleisli lift of Δ, and $\nabla_{\mathscr{P}}$ is the canonical map from tuples of subsets to subsets of tuples given by cartesian product.

Example 11.2. *The relation $\Delta_\gamma : Q^0 \to \mathscr{P}(Q^4)$ induced by the grammar in Example 4.11 is not convex, since (A, B, B, A) and (A, C, C, A), which we can think of as "convex combinations" of the state vectors (A, B, C, A) and (A, C, B, A), are not included in the image of the relation.*

Lemma 11.3. *Convex relations determine a monoidal sub-category $\mathsf{CRel}_Q \hookrightarrow \mathsf{Rel}_Q$.*

Proof. See Appendix C. ☐

Definition 11.4. *An automaton $\Delta : \mathscr{F}\Gamma \to \mathsf{Rel}_Q$ is convex if it factors through CRel_Q.*

The following lemma gives the powerset construction on convex automata. We use the non-empty powerset $\mathscr{P}^+$ to avoid duplication of failure state ($\varnothing$ in Rel_Q, but $\perp$ in $\mathsf{Par}_{\mathscr{P}^+(Q)}$):

Lemma 11.5. *For each set Q there is a strict monoidal functor $\mathscr{D}_Q : \mathsf{CRel}_Q \to \mathsf{Par}_{\mathscr{P}^+(Q)}$ which is identity on objects and acts as follows on morphisms:*

$$\Delta_\alpha : Q^n \to \mathscr{P}(Q^m)$$
$$\downarrow$$
$$\mathscr{P}^+(Q)^n \xrightarrow{\eta^n} (\perp\mathscr{P}^+(Q))^n \xrightarrow{\Delta^*_\alpha} (\perp\mathscr{P}^+(Q))^m \xrightarrow{\nabla_\perp} \perp(\mathscr{P}^+(Q)^m)$$

where we use elide the isomorphisms $\mathscr{P}(Q)^n \cong (\perp\mathscr{P}^+(Q))^n$. $\perp$ is the maybe monad, η is the unit of this monad, and $\nabla_\perp$ is its monoidal multiplication with respect to the cartesian product, sending a tuple to $\perp$ if $\perp$ appears anywhere in the tuple. This action is well-defined, since if there is more than one Δ^*_α witnessing the convexity of Δ_α, the resulting morphisms defined above are equal.

Proof. See Appendix C. ☐

Determinization of a convex automaton $\Delta : \mathscr{F}\Gamma \to \mathsf{CRel}_Q$ is now just given by post-composition with the functor $\mathscr{D}_Q : \mathsf{CRel}_Q \to \mathsf{Par}_{\mathscr{P}^+(Q)}$. We show that this preserves the language:

Theorem 11.6. *Determinization of convex automata preserves the accepted language: let $\Delta : \mathscr{F}\Gamma \to \mathsf{CRel}_Q$ be a convex automaton, then $\mathscr{L}(\Delta) = \mathscr{L}(\mathscr{D}_Q \circ \Delta)$.*

Proof. Let $\alpha \in \mathscr{L}(\Delta)$, i.e. $\Delta_\alpha(\bullet) = \{\bullet\}$. Then we must have $\Delta^*_\alpha(\bullet) = \bullet$, and so $(\mathscr{D}_Q \circ \Delta)_\alpha(\bullet) = \bullet$. Conversely let $\alpha \in \mathscr{L}_{\mathscr{D}}(\mathscr{D}_Q \circ \Delta)$, i.e. $(\mathscr{D}_Q \circ \Delta)_\alpha(\bullet) = \bullet$. Then we must have that $\Delta^*_\alpha(\bullet) = \bullet$, and so $\Delta_\alpha(\bullet) = \{\bullet\}$, that is $\alpha \in \mathscr{L}(\Delta)$. ☐

Example 11.7. *Non-deterministic monoidal automata over bottom-up tree monoidal alphabets (Definition 7.3) are convex, with $\Delta^* := \Delta^\# \circ \nabla_{\mathscr{P}}$. This reflects the well known determinizability of bottom-up tree automata. For top-down tree monoidal alphabets, the general obstruction to convexity (and thus determinizability) is seen as the non-existence of a left inverse of $\nabla_{\mathscr{P}}$, from sets of tuples to tuples of sets.*

12. Conclusion and future work

There are several classical topics in the theory of regular languages, such as regular expressions, the Myhill-Nerode and Kleene theorems, that would be interesting to investigate in the setting of monoidal languages. There is also much room for further investigation of properties of the syntactic pro, paralleling the algebraic theory of automata and languages [28] which links properties of the syntactic *monoid* of word languages to properties of the language itself. Classical language theory also has rich connections with logic [36]; lifting this to the setting of monoidal categories is a intriguing theoretical challenge.

There are also many connections between the word problem for groups and automata theory (see [6] for an overview). Following Burroni's introduction of the word problem for n-monoids [5], many special cases have been studied in detail. In particular, Delpeuch and Vicary give an algorithm for solving the word problem for the string diagrams of planar monoidal categories [13]. Investigation of the links between the theory presented here and the various word problems for string diagrams [12] could enrich the understanding of both.

Monoidal categories can sometimes be equipped with natural *symmetries*, which allows strings to cross: the corresponding string diagrams are non-planar. Our framework can extend to these *symmetric monoidal categories.* In forthcoming work [17], we show how the resulting symmetric monoidal languages are related to Mazurkiewicz traces and asynchronous automata in concurrency theory. *Premonoidal categories* also admit a string diagrammatic presentation over monoidal graphs [31] in which an ordering on generators is defiend. We conjecture that context-free languages of words arise as a particular case of *regular premonoidal languages.* Many results in this paper do not need the assumption of planarity: for example we work at the level of grammars and automata in proving closure properties. Therefore these results should extend to these new flavours of monoidal language.

We have investigated determinization of automata, but have not touched on minimization: it would interesting to see whether the categorical approach to minimization of Colcombet and Petrişan [10] can be lifted to our setting.

We also plan to investigate a notion of context-free monoidal language, using a similar algebraic approach to this paper. The candidate algebra for such languages is the produoidal category of contexts in a monoidal category [15]. Following Melliès and Zeilberger [26], we expect that this should yield a monoidal version of the Chomsky-Schützenberger representation theorem.

13. Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

14. Acknowledgments

We would like to thank Ed Morehouse for extensive discussions concerning this work, Tobias Heindel for discussion of his erstwhile project that partially inspired ours, and the anonymous reviewers for their many detailed comments and suggestions which greatly improved the manuscript.

References

[1] Jean Berstel and C Reutenauer. *Rational series and their languages.* Eatcs Monographs on Theoretical Computer Science. Springer, New York, NY, November 1988.

[2] Mikołaj Bojańczyk. Algebra for trees. In Jean-Éric Pin, editor, *Handbook of Automata Theory: Volume I*, pages 289–355. EMS Press, Berlin, 2021.

[3] Mikołaj Bojańczyk, Bartek Klin, and Julian Salamanca. Monadic monadic second order logic, 2022. URL: `https://arxiv.org/abs/2201.09969`, `doi:10.48550/ARXIV.2201.09969`.

[4] Francis Bossut, Max Dauchet, and Bruno Warin. A Kleene theorem for a class of planar acyclic graphs. *Inf. Comput.*, 117:251–265, 03 1995. `doi:10.1006/inco.1995.1043`.

[5] Albert Burroni. Higher-dimensional word problems with applications to equational logic. *Theoretical Computer Science*, 115(1):43–62, 1993. URL: `https://www.sciencedirect.com/science/article/pii/030439759390054W`, `doi:10.1016/0304-3975(93)90054-W`.

[6] J. W. Cannon, D. F. Holt, S. V. F Levy, Paterson M. S., and W. P. Thurston. *Word Processing in Groups.* Jones and Bartlett, Sudbury, MA, November 1992.

[7] Olivier Carton, Thomas Colcombet, and Gabriele Puppis. Regular languages of words over countable linear orderings. In Luca Aceto, Monika Henzinger, and Jiří Sgall, editors, *Automata, Languages and Programming*, pages 125–136, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[8] J.R.B. Cockett and Stephen Lack. Restriction categories I: categories of partial maps. *Theoretical Computer Science*, 270(1):223–259, 2002. `doi:10.1016/S0304-3975(00)00382-0`.

[9] Robin Cockett and Stephen Lack. Restriction categories III: colimits, partial limits and extensivity. *Mathematical Structures in Computer Science*, 17(4):775–817, 2007. `doi:10.1017/S0960129507006056`.

[10] Thomas Colcombet and Daniela Petrişan. Automata Minimization: a Functorial Approach. *Logical Methods in Computer Science*, Volume 16, Issue 1, March 2020. URL: `https://lmcs.episciences.org/6213`, `doi:10.23638/LMCS-16(1:32)2020`.

[11] Bruno Courcelle and Joost Engelfriet. *Graph Structure and Monadic Second-Order Logic: A Language-Theoretic Approach*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2012. `doi:10.1017/CBO9780511977619`.

[12] Antonin Delpeuch. *Word problems on string diagrams*. PhD thesis, University of Oxford, 2021.

[13] Antonin Delpeuch and Jamie Vicary. Normalization for planar string diagrams and a quadratic equivalence algorithm. *Logical Methods in Computer Science*, Volume 18, Issue 1, January 2022. URL: `https://lmcs.episciences.org/8960`, `doi:10.46298/lmcs-18(1:10)2022`.

[14] Ivan Di Liberti, Fosco Loregian, Chad Nester, and Paweł Sobociński. Functorial semantics for partial theories. *Proc. ACM Program. Lang.*, 5(POPL), jan 2021. `doi:10.1145/3434338`.

[15] M. Earnshaw, M. Román, and J. Hefford. The Produoidal Algebra of Process Decomposition. In *32nd EACSL Annual Conference on Computer Science Logic (CSL 2024)*.

[16] Matthew Earnshaw and Paweł Sobociński. Regular Monoidal Languages. In Stefan Szeider, Robert Ganian, and Alexandra Silva, editors, *47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022)*, volume 241 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 44:1–44:14, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. URL: `https://drops.dagstuhl.de/opus/volltexte/2022/16842`, `doi:10.4230/LIPIcs.MFCS.2022.44`.

[17] Matthew Earnshaw and Paweł Sobociński. String Diagrammatic Trace Theory. In Jérôme Leroux, Sylvain Lombardy, and David Peleg, editors, *48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023)*, volume 272 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 43:1–43:15, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. URL: `https://drops.dagstuhl.de/opus/volltexte/2023/18577`, `doi:10.4230/LIPIcs.MFCS.2023.43`.

[18] Samuel Eilenberg and Jesse B. Wright. Automata in general algebras. *Information and Control*, 11(4):452–470, 1967. `doi:10.1016/S0019-9958(67)90670-5`.

[19] Uli Fahrenberg, Christian Johansen, Georg Struth, and Krzysztof Ziemiański. Languages of higher-dimensional automata. *Mathematical Structures in Computer Science*, 31(5):575–613, 2021. doi:10.1017/S0960129521000293.

[20] Richard Garner and Tom Hirschowitz. Shapely monads and analytic functors. *Journal of Logic and Computation*, 28(1):33–83, 11 2017. doi:10.1093/logcom/exx029.

[21] Ferenc Gécseg and Magnus Steinby. Tree automata, 2015. doi:10.48550/ARXIV.1509.06233.

[22] T. Heindel. A Myhill-Nerode theorem beyond trees and forests via finite syntactic categories internal to monoids. *Preprint*, 2017.

[23] André Joyal and Ross Street. The geometry of tensor calculus, I. *Advances in Mathematics*, 88(1):55–112, 1991. doi:10.1016/0001-8708(91)90003-P.

[24] Saunders MacLane. Categorical algebra. *Bulletin of the American Mathematical Society*, 71(1):40 – 106, 1965.

[25] Jade Master. Petri nets based on lawvere theories. *Mathematical Structures in Computer Science*, 30(7):833–864, 2020. doi:10.1017/S0960129520000262.

[26] Paul-André Melliès and Noam Zeilberger. Parsing as a lifting problem and the Chomsky-Schützenberger representation theorem. In *MFPS 2022 - 38th conference on Mathematical Foundations for Programming Semantics*, Ithaca, NY, United States, July 2022. URL: https://hal.archives-ouvertes.fr/hal-03702762.

[27] Paolo Perrone. Distribution monad (nlab entry), 2019. https://ncatlab.org/nlab/show/distribution+monad, Last accessed 2023-03-13.

[28] Jean-Éric Pin. Syntactic semigroups. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages: Volume 1 Word, Language, Grammar*, pages 679–746. Springer Berlin Heidelberg, Berlin, Heidelberg, 1997. doi:10.1007/978-3-642-59126-6_3.

[29] Jean-Éric Pin and Dominique Perrin. *Infinite Words - Automata, Semigroups, Logic and Games.* Pure and Applied Mathematics (Amsterdam). Elsevier, 2004.

[30] John Power and Edmund Robinson. Premonoidal categories and notions of computation. *Mathematical Structures in Computer Science*, 7(5), 1997. doi:10.1017/S0960129597002375.

[31] Mario Román. Promonads and string diagrams for effectful categories. In *ACT '22: Applied Category Theory, Glasgow, United Kingdom, 18 - 22 July, 2022*, volume abs/2205.07664, 2022. `arXiv:2205.07664`, `doi:10.48550/arXiv.2205.07664`.

[32] Kimmo I. Rosenthal. Quantaloids, enriched categories and automata theory. *Applied Categorical Structures*, 3(3):279–301, 1995. `doi:10.1007/bf00878445`.

[33] Paul W. K Rothemund, Nick Papadakis, and Erik Winfree. Algorithmic self-assembly of DNA Sierpinski triangles. *PLOS Biology*, 2(12), 12 2004. `doi:10.1371/journal.pbio.0020424`.

[34] Jacques Sakarovitch. *Elements of automata theory.* Cambridge University Press, Cambridge New York, 2009.

[35] P. Selinger. A survey of graphical languages for monoidal categories. In B. Coecke, editor, *New Structures for Physics*, pages 289–355. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. `doi:10.1007/978-3-642-12821-9_4`.

[36] Howard Straubing. *Finite automata, formal logic, and circuit complexity.* Progress in Theoretical Computer Science. Birkhauser Verlag AG, Basel, Switzerland, December 1993.

[37] J. W. Thatcher and J. B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical systems theory*, 2(1):57–81, Mar 1968.

[38] Henning Urbat, Jiri Adámek, Liang-Ting Chen, and Stefan Milius. Eilenberg Theorems for Free. In Kim G. Larsen, Hans L. Bodlaender, and Jean-Francois Raskin, editors, *42nd International Symposium on Mathematical Foundations of Computer Science (MFCS 2017)*, volume 83 of *LIPIcs*, pages 43:1–43:15, Dagstuhl, Germany, 2017. `doi:10.4230/LIPIcs.MFCS.2017.43`.

[39] R.F.C. Walters. A note on context-free languages. *Journal of Pure and Applied Algebra*, 62(2):199–203, 1989. `doi:10.1016/0022-4049(89)90151-5`.

[40] Vladimir Zamdzhiev. *Rewriting Context-free Families of String Diagrams.* PhD thesis, University of Oxford, 2016.

Appendix A. Congruence on a monoidal category

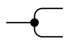
Definition Appendix A.1. *A congruence on a monoidal category $\mathscr{C}$ is an equivalence relation $f \sim g$ on pairs of parallel morphisms $f, g : x \to x'$ compatible with composition and monoidal product:*

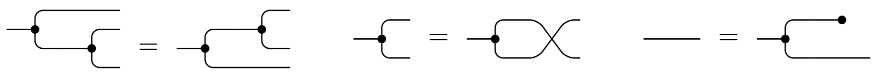
- $f \sim g \implies k \circ f \circ h \sim k \circ g \circ h$ *whenever these composites are defined, and,*

- $f \sim g \implies p \otimes f \otimes q \sim p \otimes g \otimes q$.

Given a congruence on a monoidal category $\mathscr{C}$, we can define the quotient monoidal category $\mathscr{C}/{\sim}$ as the category with objects those of $\mathscr{C}$, and homsets $(\mathscr{C}/{\sim})(x, x') := \mathscr{C}(x, x')/{\sim}$, with composition and monoidal product defined in the obvious way. The quotient functor $\mathscr{C} \to \mathscr{C}/{\sim}$ is monoidal, full and bijective on objects (and strict when $\mathscr{C}$ is strict). When $\mathscr{C}$ is a pro, the quotient monoidal category is a pro, and the quotient functor is a pro morphism. One can easily verify with string diagrams that the syntactic congruence (Definition 9.2) is indeed a congruence, and so the syntactic pro is well defined.

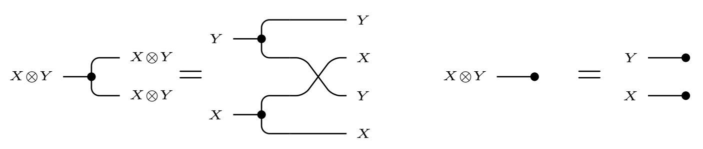
Appendix B. Cartesian restriction categories

A cartesian restriction category [9] can be defined as a symmetric monoidal category in which every object is equipped with a coherent commutative comonoid structure for which comultiplication is natural. The following equations spell out the details of this definition. We write for the counit of the comonoid on an arbitrary object,  for the comultiplication of the comonoid on an arbitrary object, and for the symmetry between two objects. Then to say that there is a commutative comonoid structure on each object is to say that the following equations of string diagrams (CCM) hold (respectively: coassociativity, commutativity, and left unitality):



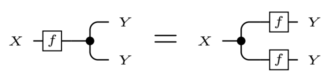
(CCM)

Note that "right unitality" may be derived from these. To say that these comonoid structures are coherent is to say that for all objects X and Y we have the following equations of string diagrams:



(coherent)

Finally to say that comultiplication natural is to say that we can move morphisms through comultiplication as follows:



(natural)

Appendix C. Details for Section 11

Proof of Lemma 11.3. It is clear that identity relations are convex. It remains to show that the composite of convex relations is convex, and that the monoidal product of convex relations is convex. For the former, take convex relations $\Delta_\alpha : Q^a \to \mathscr{P}(Q^b), \Delta_\beta : Q^b \to \mathscr{P}(Q^c)$, and take $(\Delta_\beta \diamond \Delta_\alpha)^* = \Delta_\beta^* \circ \Delta_\alpha^*$, where $\diamond$ is composition in $\mathsf{Kl}(\mathscr{P})$. Consider the following diagram:

$$\begin{tikzcd}
(\mathscr{P}Q)^a \arrow[rr, "\Delta_\alpha^*"] \arrow[d, "\nabla"'] & & (\mathscr{P}Q)^b \arrow[rr, "\Delta_\beta^*"] \arrow[d, "\nabla"] & & (\mathscr{P}Q)^c \arrow[d, "\nabla"] \\
\mathscr{P}(Q^a) \arrow[rr, "\Delta_\alpha^\#"] \arrow[dr, "\mathscr{P}(\Delta_\alpha)"'] & & \mathscr{P}(Q^b) \arrow[rr, "\Delta_\beta^\#"] \arrow[dr, "\mathscr{P}(\Delta_\beta)"'] & & \mathscr{P}(Q^c) \\
& \mathscr{P}^2(Q^b) \arrow[ur, "\mu"'] \arrow[dr, "\mathscr{P}^2(\Delta_\beta)"'] & & \mathscr{P}^2(Q^c) \arrow[ur, "\mu"'] & \\
& & \mathscr{P}^3(Q^c) \arrow[ur, "\mathscr{P}(\mu)"'] & &
\end{tikzcd}$$

We want to show that $\Delta_\beta^\# \circ \Delta_\alpha^\# = (\Delta_\beta \diamond \Delta_\alpha)^\#$, so that the pasting of the two convexity squares at the top witnesses convexity of the composite. By definition of Kleisli extension we have that:

$$\Delta_\beta^\# \circ \Delta_\alpha^\# = \mu \circ \mathscr{P}(\Delta_\beta) \circ \mu \circ \mathscr{P}(\Delta_\alpha)$$

by naturality of μ,

$$\begin{aligned}
&= \mu \circ \mathscr{P}(\mu) \circ \mathscr{P}^2(\Delta_\beta) \circ \mathscr{P}(\Delta_\alpha) \\
&= \mu \circ \mathscr{P}(\mu \circ \mathscr{P}(\Delta_\beta) \circ \Delta_\alpha) \\
&= \mu \circ \mathscr{P}(\Delta_\beta \diamond \Delta_\alpha) \\
&= (\Delta_\beta \diamond \Delta_\alpha)^\#
\end{aligned}$$

Now take convex relations $\Delta_\gamma : Q^{n_1} \to \mathscr{P}(Q^{m_1}), \Delta_\varepsilon : Q^{n_2} \to \mathscr{P}(Q^{m_2})$. Take $(\Delta_\gamma \otimes \Delta_\varepsilon)^* = \Delta_\gamma^* \times \Delta_\varepsilon^*$. We have that:

$$\begin{aligned}
&\mathscr{P}(Q)^{n_1+n_2} \xrightarrow{(\Delta_\gamma \otimes \Delta_\varepsilon)^*} \mathscr{P}(Q)^{m_1+m_2} \xrightarrow{\nabla} \mathscr{P}(Q^{m_1+m_2}) \\
&= \mathscr{P}(Q)^{n_1+n_2} \xrightarrow{\langle \nabla \circ \Delta_\gamma^*, \nabla \circ \Delta_\varepsilon^* \rangle} \mathscr{P}(Q^{m_1}) \times \mathscr{P}(Q^{m_2}) \xrightarrow{\nabla} \mathscr{P}(Q^{m_1+m_2})
\end{aligned}$$

by convexity of $\Delta_\gamma, \Delta_\varepsilon$,

$$\begin{aligned}
&= \mathscr{P}(Q)^{n_1+n_2} \xrightarrow{\nabla \times \nabla} \mathscr{P}(Q^{n_1}) \times \mathscr{P}(Q^{n_2}) \xrightarrow{\mathscr{P}(\Delta_\gamma) \times \mathscr{P}(\Delta_\varepsilon)} \mathscr{P}\mathscr{P}(Q^{m_1}) \times \mathscr{P}\mathscr{P}(Q^{m_2}) \\
&\quad \xrightarrow{\mu \times \mu} \mathscr{P}(Q^{m_1}) \times \mathscr{P}(Q^{m_2}) \xrightarrow{\nabla} \mathscr{P}(Q^{m_1+m_2}) \\
&= \mathscr{P}(Q)^{n_1+n_2} \xrightarrow{\nabla} \mathscr{P}(Q^{n_1+n_2}) \xrightarrow{\mathscr{P}(\Delta_\gamma \times \Delta_\varepsilon)} \mathscr{P}(\mathscr{P}(Q^{m_1}) \times \mathscr{P}(Q^{m_2})) \\
&\quad \xrightarrow{\mathscr{P}(\nabla)} \mathscr{P}\mathscr{P}(Q^{m_1+m_2}) \xrightarrow{\mu} \mathscr{P}(Q^{m_1+m_2}) \\
&= \mathscr{P}(Q)^{n_1+n_2} \xrightarrow{\nabla} \mathscr{P}(Q^{n_1+n_2}) \xrightarrow{\mathscr{P}(\Delta_\gamma \otimes \Delta_\varepsilon)} \mathscr{P}\mathscr{P}(Q^{m_1+m_2}) \xrightarrow{\mu} \mathscr{P}(Q^{m_1+m_2}).
\end{aligned}$$

Hence $\Delta_\gamma \otimes \Delta_\varepsilon$ is convex. □

Lemma Appendix C.1. *The following diagram commutes*

$$\begin{tikzcd}
(\bot\mathscr{P}^+(Q))^n \arrow[d, "\nabla_\bot"'] \arrow[dr, "\nabla_{\mathscr{P}}"] & \\
\bot(\mathscr{P}^+(Q))^n \arrow[r, "\bot\nabla_{\mathscr{P}^+}"'] & (\bot\mathscr{P}^+)(Q^n)
\end{tikzcd}$$

Proof. Reason by cases on elements of $(\bot\mathscr{P}^+(Q))^n$, either no element of the n-tuple is $\bot$, or at least one is. In either case, the functions coincide by definition. □

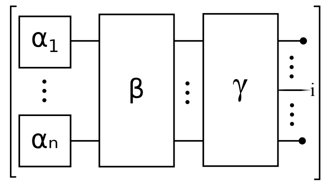
Proof of Lemma 11.5. We first show that the action on morphisms is well-defined. It suffices to show that if Δ^*_α, $\hat{\Delta}^*_\alpha$ are distinct witnesses to the convexity of Δ_α, then $\nabla_\bot \circ \Delta^*_\alpha = \nabla_\bot \circ \hat{\Delta}^*_\alpha$. From Lemma Appendix C.1, $\nabla_{\mathscr{P}} = \bot\nabla_{\mathscr{P}^+} \circ \nabla_\bot$. Furthermore, $\bot\nabla_{\mathscr{P}^+}$ is injective, so our conclusion follows, using the definition of convexity: $\nabla_{\mathscr{P}}\Delta^* = \Delta^\# \circ \nabla_{\mathscr{P}} = \nabla_{\mathscr{P}} \circ \hat{\Delta}^*$.

We need to show that this mapping is a strict monoidal functor. It is clear that identities are preserved. It remains to show that that composition and monoidal product are preserved. Let $\Delta_\alpha : Q^a \to \mathscr{P}(Q^b), \Delta_\beta : Q^b \to \mathscr{P}(Q^c)$. We require $\mathscr{D}_Q(\Delta_\beta \diamond \Delta_\alpha) = \mathscr{D}_Q(\Delta_\beta) \circ \mathscr{D}_Q(\Delta_\alpha)$. This follows from the commutativity of the following diagram (naturality of ∇ and the naturality of η), and the unit law for Kleisli composition in Par.

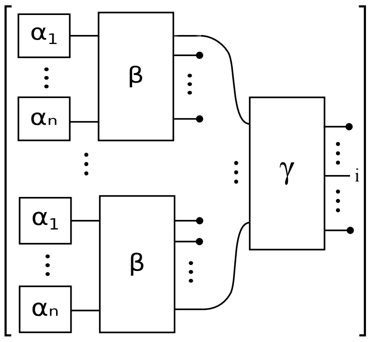
$$\begin{tikzcd}
(\bot\mathscr{P}^+(Q))^b \arrow[r, "\eta^b"] \arrow[d, "\nabla_\bot"'] \arrow[dr, "\eta"] & (\bot\bot\mathscr{P}^+(Q))^b \arrow[d, "\nabla_\bot"] \\
\bot\mathscr{P}^+(Q)^b \arrow[r, "\bot\eta^b"'] & \bot(\bot\mathscr{P}^+(Q))^b
\end{tikzcd}$$

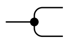
Strict preservation of the monoidal product follows easily from the fact that $(\Delta_\gamma \otimes \Delta_\varepsilon)^* = \Delta^*_\gamma \times \Delta^*_\varepsilon$. □

Proof of Theorem 10.11. We show that the defined mapping is indeed a morphism of pros. For composition, we need to show $\phi([\gamma] \circ [\beta]) = \phi([\gamma]) \circ \phi([\beta])$. The i^{th} component of $\phi([\gamma] \circ [\beta])([\alpha_1], ..., [\alpha_n])$ is the equivalence class:

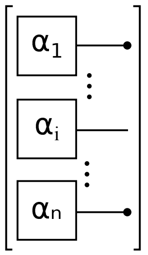


where the i^{th} output of γ is dangling on the right. The i^{th} component of $(\phi([\gamma]) \circ \phi([\beta]))([\alpha_1], ..., [\alpha_n])$ is the equivalence class:

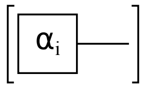


But since $\mathscr{F}\Gamma/{\equiv_L}$ is a cartesian restriction prop, the representatives of these equivalence classes are the same diagram (by repeated applications of the naturality of  and unitality). Hence this is the same equivalence class, and ϕ preserves composition.

For identities, the i^{th} component of $\phi([\text{id}_n])([\alpha_1], ..., [\alpha_n])$ is the equivalence class:



For $\phi([\text{id}_n])$ to be the identity, this needs to be equal to the equivalence class $[\alpha_i]$:



But these must indeed be the same equivalence class, for if there were a context that distinguished these morphisms, we would have a contradiction, since languages are closed under monoidal products (Lemma 8.4). Similar diagrams hold for the preservation of the monoidal product, and thus we have a morphism of pros. $\square$