Regular Monoidal Languages

Matthew Earnshaw
Department of Software Science, Tallinn University of Technology, Estonia

Paweł Sobociński
Department of Software Science, Tallinn University of Technology, 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 over string diagrams. We use the algebra of monoidal categories to investigate the properties of regular monoidal languages, and provide sufficient conditions for their recognizability by deterministic monoidal automata.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory; Theory of computation → Categorical semantics

Keywords and phrases monoidal categories, string diagrams, formal language theory, cartesian restriction categories

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.44

Related Version *Full Version*: `https://arxiv.org/abs/2207.00526` [6]

Funding This research was supported by the ESF funded Estonian IT Academy research measure (project 2014-2020.4.05.19- 0001). The second author was additionally supported by the Estonian Research Council grant PRG1210.

Acknowledgements We would like to thank Ed Morehouse for extensive discussions concerning this work, and Tobias Heindel for discussion of his erstwhile project that partially inspired ours.

1 Introduction

Classical formal language theory has been extended to various kinds of algebraic structures, such as infinite words, rational sequences, trees, countable linear orders, graphs of bounded tree width, etc. In recent years, the essential unity of the field has been better understood [1, 16]. Such structures can often be seen as algebras for monads on the category of sets, and sufficient conditions exist [1] for formal language theory to extend to their algebras.

In this paper, we make a first step into a programme of extending language theory to higher-dimensional algebraic structures. Here we make the step from monoids to 2-monoids, better known as monoidal categories.

We introduce a categorial framework for reasoning about languages of morphisms in strict monoidal categories – including their associated grammars and automata. We show how these include classical and tree automata, but also open up a wilder world of string diagram languages. By investigating the 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. Omitted proofs can be found in the full version [6].

2 Related work

Bossut [2] studied rational languages of planar acyclic graphs and proved a Kleene theorem for a class of such languages. Bossut's graph languages feature initial and final states, whereas our languages consist of scalar morphisms, which more neatly generalizes the theory of regular string and tree languages. Bossut introduces a notion of automaton for these languages, but these lack a state machine denotation – being more similar to our grammars.

In [10], Heindel recasts Bossut's approach using monoidal categories. Unfortunately the purported Myhill-Nerode result was incorrect, due to a flawed definition of syntactic congruence. We rectify this in Section 5, but a Myhill-Nerode type theorem remains open.

Zamdzhiev [18] introduced context-free languages of string diagrams using the string graph representation of string diagrams 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. [13] use 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 12.

Walters' note [17] on regular and context-free grammars served as a starting point for our definition of regular monoidal grammar. Rosenthal [12], 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 [5] 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. [7] 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.

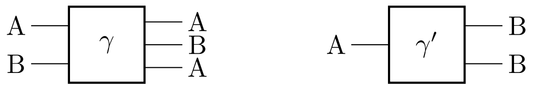
3 Regular monoidal grammars and regular monoidal languages

A *monoidal grammar* is a finite specification for the construction of string diagrams: i.e. morphisms in free monoidal categories (more specifically, free pros). We introduce *regular monoidal grammars*, an analogue of classical (right-) regular grammars, and their equivalent representation as non-deterministic *monoidal automata*. We begin by recalling the notion of monoidal graph and how they present free monoidal categories.

3.1 Monoidal graphs and free pros

▶ **Definition 1.** *A monoidal graph $\mathcal{G}$ consists of sets $E_\mathcal{G}, V_\mathcal{G}$ and functions $dom, cod : E_\mathcal{G} \rightrightarrows V_\mathcal{G}^*$ where $V_\mathcal{G}^*$ is the underlying set of the free monoid. The elements of $E_\mathcal{G}$ are called* generators*, and for a generator $\gamma \in E_\mathcal{G}$, $dom(\gamma), cod(\gamma)$ are the domain, codomain (resp.) types of γ.*

Diagrammatically, a monoidal graph can be pictured as a collection of boxes, labelled by elements of $E_\mathcal{G}$ with wires entering on the left and exiting on the right, labelled by types given by the functions dom, cod. For example, the following depicts the monoidal graph $\mathcal{G}$ with $E_\mathcal{G} = \{\gamma, \gamma'\}, V_\mathcal{G} = \{A, B\}, dom(\gamma) = AB, cod(\gamma) = ABA, dom(\gamma') = A, cod(\gamma') = BB$:



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 $E_{\mathcal{G}}$ and $V_{\mathcal{G}}$ are both finite sets.

▶ **Definition 2.** *A morphism $\Psi : \mathcal{G}' \to \mathcal{G}$ of monoidal graphs is a pair of functions $V_\Psi : V_{\mathcal{G}} \to V_{\mathcal{G}'}, E_\Psi : E_{\mathcal{G}} \to E_{\mathcal{G}'}$ such that $dom \,\mathring{,}\, V_\Psi^* = E_\Psi \,\mathring{,}\, dom$ and $cod \,\mathring{,}\, V_\Psi^* = E_\Psi \,\mathring{,}\, cod$.*

Monoidal graphs and their morphisms form a category MonGraph. Recall that a (coloured) pro is a strict monoidal category whose monoid of objects is free (on the set of "colours"). There is a category Pro with objects pros and morphisms strict monoidal functors whose action on objects is determined by a function between their sets of colours. We call these *pro morphisms*. (Coloured) props are pros that are also *symmetric* (strict) monoidal categories.

Pros (and props) are monadic over monoidal graphs: the forgetful functor $\mathcal{U} : \mathsf{Pro} \to \mathsf{MonGraph}$ has a left adjoint $\mathcal{F} : \mathsf{MonGraph} \to \mathsf{Pro}$, and Pro is equivalent to the category of algebras for the induced monad on MonGraph (see [8, §2.3]). $\mathcal{F}$ sends a monoidal graph $\mathcal{G}$ to a pro $\mathcal{F}\mathcal{G}$ whose set of objects is $V_{\mathcal{G}}^*$ and whose morphisms are *string diagrams* (see [15]).

3.2 Monoidal languages and regular monoidal grammars

Classically, a language over an alphabet Σ is a subset of the free monoid Σ^*. A *monoidal language* is defined similarly, replacing free monoids with free pros over a *monoidal alphabet*:

▶ **Definition 3.** *A monoidal alphabet Γ is a finite monoidal graph where V_Γ is a singleton.*

For a generator γ of a monoidal alphabet, we refer to $dom(\gamma), cod(\gamma)$ as the arity, coarity (resp.) of γ, writing $ar(\gamma)$, $coar(\gamma)$. Such generators are drawn with "untyped" wires.

▶ **Definition 4.** *A monoidal language L over a monoidal alphabet Γ is a subset $L \subseteq \mathcal{F}\Gamma(0,0)$ of morphisms with arity and coarity 0 in the free pro generated by Γ.*

▶ Remark 5. The restriction to arity and coarity zero (i.e. *scalar*) morphisms may appear arbitrary. However, we will see in Section 4 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.

Regular monoidal grammars specify monoidal languages that are an analogue of classical regular languages. They can be obtained by taking Walters' [17] definition of regular language and replacing the adjunction between reflexive graphs and categories with that between monoidal graphs and pros. As shown in Section 4, they include the classical definitions of regular tree and word languages as grammars over monoidal alphabets of a particular shape.

▶ **Definition 6.** *A regular monoidal grammar is a morphism of finite monoidal graphs $\Psi : \mathcal{M} \to \Gamma$ where Γ is a monoidal alphabet.*

Intuitively, a regular monoidal grammar is a labelling of the edges of $\mathcal{M}$ by generators in Γ. Indeed, the vertex function $V_\Psi : V_{\mathcal{M}} \to \{\bullet\}$ is unique, so the grammar is determined by its edge function $E_\Psi : E_{\mathcal{M}} \to E_\Gamma$, sending edges to their labels. In Section 3.4 we show that this data determines a transition system with states words $w \in V_{\mathcal{M}}^*$.

▶ Remark 7. Every regular monoidal grammar determines a pro morphism between free pros, $\mathcal{F}\Psi : \mathcal{F}\mathcal{M} \to \mathcal{F}\Gamma$, which we may also refer to as a regular monoidal grammar.

For any string diagram $s \in \mathcal{F}\Gamma$ over an alphabet Γ, we can think of the set of string diagrams $\mathcal{F}\Psi^{-1}(s)$ as a set of possible "parsings" of that diagram.

▶ Remark 8. We represent regular monoidal grammars diagrammatically by drawing the monoidal graph $\mathcal{M}$ as above, but labelling each box $e \in E_{\mathcal{M}}$ with $E_{\Psi}(e)$. 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.

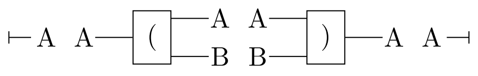
3.3 Regular monoidal languages

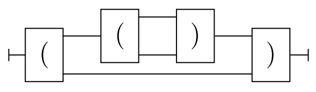
A regular monoidal grammar determines a monoidal language as follows:

▶ **Definition 9.** *Given a regular monoidal grammar $\Psi : \mathcal{M} \to \Gamma$, the image under $\mathcal{F}\Psi$ of the endo-hom-set of the monoidal unit ε in $\mathcal{F}\mathcal{M}$ is a monoidal language $\mathcal{F}\Psi[\mathcal{F}\mathcal{M}(\varepsilon, \varepsilon)] \subseteq \mathcal{F}\Gamma(0, 0)$.*

We call the class of languages determined by regular monoidal grammars the *regular monoidal languages.* We shall see that they are precisely the languages accepted by *non-deterministic monoidal automata* (Section 3.4). The basic idea is that a "word" is a scalar string diagram, i.e. one with no "dangling wires". 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 8). 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 wires, and then erasing the types on the wires. The following examples of regular monoidal grammars illustrate this idea:

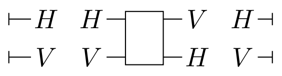
▶ **Example 10** (Balanced parentheses)**.** Recall that the Dyck language, the language of balanced parentheses, is a paradigmatic example of a non-regular word language. However, we can recognize balanced parentheses using the regular monoidal grammar shown below left. An example of a morphism in the language defined by this grammar is shown on the right.

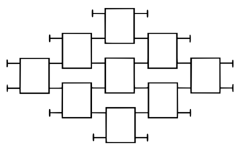




This illustrates how regular monoidal grammars permit unbounded concurrency. Here, as one scans from left to right, the (unbounded) size of the internal boundary of a string diagram keeps track of the number of open left parentheses.

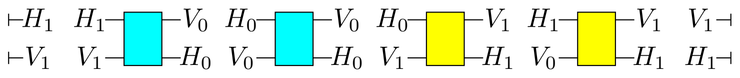
▶ **Example 11** (Brick walls)**.** A variant on the "brick wall" language introduced by [2] is given by the following grammar (left below). An example of a morphism in the language defined by this grammar is shown on the right.

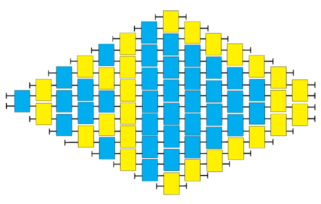




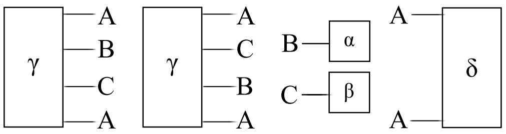
In Section 3.5 we will see how this language of "brick walls" allows us to construct the following example as an intersection of two languages:

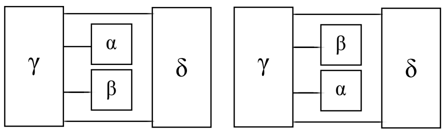
▶ **Example 12** (Sierpiński gasket)**.** In [13], self-assembly of DNA tiles was used to realize the behaviour of a cellular automaton that computes the Sierpiński gasket fractal, based on the computation of the XOR gate. [13] implicitly depicts a monoidal grammar, and so Sierpiński gaskets of arbitrary iteration depth (e.g. right below) are in fact the monoidal language over this grammar (left below, where we use colours for the alphabet):





▶ **Example 13.** We define a grammar (left below) that will serve as a running counterexample in Section 7, as it defines a language that cannot be deterministically recognized. The connected string diagrams in this language are exactly two (right below).





▶ Remark 14. If the monoidal graph $\mathcal{M}$ has no edges whose domain is ε and no edges whose codomain is ε, a regular monoidal grammar $\Psi : \mathcal{M} \to \Gamma$ 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.

3.4 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 $Q \xrightarrow{\Delta_a} \mathcal{P}(Q)$. Non-deterministic *monoidal automata* do not have initial and final states; string diagrams are simply accepted or rejected depending on their shape. In Section 4, we will see that initial and final states derive from this definition, when the alphabet is of a particular form.

▶ **Definition 15.** *A non-deterministic monoidal automaton $\Delta = (Q, \Delta_\Gamma)$ over a monoidal alphabet Γ is given by a finite set Q, together with a set of transition functions indexed by generators $\Delta_\Gamma = \{Q^{ar(\gamma)} \xrightarrow{\Delta_\gamma} \mathcal{P}(Q^{coar(\gamma)})\}_{\gamma \in E_\Gamma}$.*

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

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

▶ **Lemma 17** ([11], Corollary 4.3). *Let T be a commutative monad on a symmetric monoidal category $\mathcal{C}$. Then the Kleisli category $\mathsf{Kl}(T)$ has a canonical monoidal structure, which is given on objects by the monoidal product in $\mathcal{C}$, 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 given by the commutativity of T.*

▶ Remark 18. The maybe monad $(-)_\perp$ is also commutative, so its Kleisli category, equivalent to the category Par of sets and partial functions, also has a canonical monoidal structure, and for each set Q there is an endomorphism pro Par_Q. We will come back to Par_Q in Section 6.

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

▶ **Observation 19.** *The assignment of generators to transition functions $\gamma \mapsto \Delta_\gamma$ in Definition 15 determines a morphism of monoidal graphs $\Gamma \to |\mathsf{Rel}_Q|$. Such morphisms are in bijection with pro morphisms $\Delta : \mathcal{F}\Gamma \to \mathsf{Rel}_Q$. 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 $\{\bullet\} \to \mathcal{P}(\{\bullet\})$, which represent accepting or rejecting computations, and thus can be used to define the language of the automaton:

▶ **Definition 20.** *Let $\Delta : \mathcal{F}\Gamma \to \mathsf{Rel}_Q$ be a non-deterministic monoidal automaton. Then the monoidal language accepted by Δ is $\mathcal{L}(\Delta) := \{\alpha \in \mathcal{F}\Gamma(0,0) \mid \Delta_\alpha(\bullet) = \{\bullet\}\}$.*

There is an evident correspondence between regular monoidal grammars and non-deterministic monoidal automata. 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 21.** *Given a regular monoidal grammar $\Psi : \mathcal{M} \to \Gamma$, define a monoidal automaton with $Q = V_{\mathcal{M}}$, $w(\Delta_\gamma)w' \iff \exists\sigma \in E_\Psi^{-1}(\gamma)$ such that $dom(\sigma) = w, cod(\sigma) = w'$. Conversely given a monoidal automaton (Q, Δ_Γ), define a regular monoidal grammar with $V_{\mathcal{M}} = Q$ and take an edge $w \to w'$ over $\gamma \iff w(\Delta_\gamma)w'$. This correspondence of grammars and automata preserves the recognized language.*

▶ Remark 22. In automata theory it is often convenient to consider automata with ε-transitions, or word-labelled transitions more generally. As monoidal grammars, these correspond to arbitrary functors $\mathcal{F}\mathcal{M} \to \mathcal{F}\Gamma$, that is (by the adjunction $\mathcal{U} \dashv \mathcal{F}$), to morphisms of finite monoidal graphs $\mathcal{M} \to \mathcal{U}\mathcal{F}\Gamma$. The corresponding generalization of monoidal automata requires considering Rel_Q as a monoidal 2-category with 2-cells the inclusions. Identity on objects, strict monoidal lax 2-functors $\mathcal{F}\Gamma \to \mathsf{Rel}_Q$ (where $\mathcal{F}\Gamma$ is considered as equipped with identity 2-cells), then give the refined notion of monoidal automaton. Such a lax 2-functor need no longer send the identity on n wires to the identity relation on Q^n, but merely to a relation that includes the identity; this corresponds to allowing silent transitions. Similarly, *lax* preservation of composition corresponds to allowing "term-labelled" transitions.

3.5 Closure properties of regular monoidal languages

We record some closure properties of regular monoidal languages.

▶ **Lemma 23** (Closure under union). *Let L and L' be regular monoidal languages over Γ. Then $L \cup L'$ is a regular monoidal language over Γ.*

▶ **Lemma 24** (Closure under intersection). *Let L and L' be regular monoidal languages over Γ. Then $L \cap L'$ is a regular monoidal language over Γ.*

▶ Remark 25. The Sierpiński gasket language (Example 12) is the intersection of the brick wall language (Example 11) and an "XOR gate" language: this explains the origin of the states in the grammar shown in Example 12.

▶ **Lemma 26** (Closure under monoidal product and factors). *Let L be a regular monoidal language. Then $\alpha, \beta \in L \iff \alpha \otimes \beta \in L$.*

▶ **Lemma 27** (Closure under images of alphabets). *Let L a be regular monoidal language over Γ, and $\Gamma \xrightarrow{h} \Gamma'$ be a morphism of monoidal alphabets. Then $(\mathcal{F}h)L$ is a regular monoidal language over Γ'.*

▶ **Lemma 28** (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, $(\mathcal{F}h)^{-1}(L)$ is a regular monoidal language over Γ'.*

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 [1], 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 29.** *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 [1] to extend to monoidal categories, since these are not algebras for a monad on Set. Moreover, taking inspiration from classical examples in Section 4, one could also refine what is meant by complement, for instance focussing on the set of non-empty connected scalar diagrams – see below for more details.

4 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.

4.1 Finite-state automata

▶ **Definition 30.** *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 31.** *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 $\Delta_\sigma := \Delta(\sigma, -) : Q \to \mathcal{P}(Q)$. For $-\!\!\dashv$ take the transition function $Q \to \mathcal{P}(Q^0)$ 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 $Q^0 \to \mathcal{P}(Q)$ to pick out the singleton $\{i\}$. This defines a monoidal automaton $A' := (Q, \Delta'_{\Sigma'})$, and a simple induction shows that $\mathcal{L}(A) = \mathcal{L}(A')$, 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 $\{\bullet\} \to \mathcal{P}(Q)$. We can always "normalize" such an automaton into an equivalent NFA with one initial state (see [14, §2.3.1]). This shows how NFA initial and final states are captured by this particular shape of monoidal alphabet.

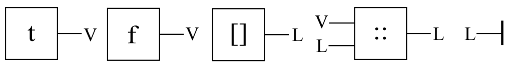
4.2 Tree automata

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 $\Delta_\sigma : Q^{r(\sigma)} \to \mathcal{P}(Q)$. A non-deterministic top-down tree automaton, instead, has a set of initial states $I \subseteq Q$ and transition functions $\Delta_\sigma : Q \to \mathcal{P}(Q^{r(\sigma)})$. We can recover these as non-deterministic monoidal automata over *tree monoidal alphabets*:

▶ **Definition 32.** *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 33.** *Bottom-up tree automata are exactly non-deterministic monoidal automata over bottom-up tree monoidal alphabets, and likewise for top-down tree automata.*

The idea is similar to that sketched above for NFAs. 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 ::).

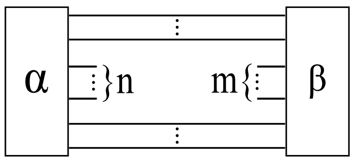


Intuitively, the connected scalar string diagrams determined by this language are trees, with leaves on the left, and the root on the right. Monoidal automata over top-down tree monoidal alphabets have a similar form, but are mirrored horizontally, and thus morphisms in the language have the root on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

5 The syntactic pro of a monoidal language

In this section we introduce the *syntactic congruence* on monoidal languages and the corresponding *syntactic pro*, by analogy with the syntactic congruence on classical regular languages and their associated syntactic monoid. In Section 7.2 we will give an algebraic property of the syntactic pro sufficient for the language to be deterministically recognizable.

▶ **Definition 34.** *A context of capacity (n, m), where $n, m \geqslant 0$, is a scalar string diagram with a hole – as illustrated below – with zero or more additional wires exiting the first box and entering the second (indicated by ellipses).*



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

▶ **Definition 35** (Syntactic congruence). *Given a monoidal language $L \subseteq \mathcal{F}\Gamma(0,0)$ we define its syntactic congruence $\equiv_L$ as follows. Let α, β be morphisms in $\mathcal{F}\Gamma(n,m)$. Then $\alpha \equiv_L \beta$ whenever $C[\alpha] \in L \iff C[\beta] \in L$, for all contexts C of capacity (n,m).*

▶ **Definition 36.** *The syntactic pro of a monoidal language L is the quotient pro $\mathcal{F}\Gamma/{\equiv_L}$. The quotient functor $S_L : \mathcal{F}\Gamma \to \mathcal{F}\Gamma/{\equiv_L}$ is the syntactic morphism of L. For more details, see the full version [6].*

▶ Remark 37. 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 38.** *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 26. 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 59:

▶ **Lemma 39.** *If a monoidal language L is regular, then its syntactic pro $\mathcal{F}\Gamma/{\equiv_L}$ is locally finite (i.e. has finite hom-sets).*

Proof. It suffices to exhibit a full pro morphism into $\mathcal{F}\Gamma/{\equiv_L}$ from a locally finite pro. Let L be a regular monoidal language recognized by $\Delta : \mathcal{F}\Gamma \to \mathsf{Rel}_Q$. Δ induces a congruence $\sim$ on $\mathcal{F}\Gamma$ defined by $\alpha \sim \beta \iff \Delta(\alpha) = \Delta(\beta)$, which implies that $\mathcal{F}\Gamma/{\sim}$ is locally finite, since Rel_Q is locally finite. Define the pro morphism $\mathcal{F}\Gamma/{\sim} \to \mathcal{F}\Gamma/{\equiv_L}$ 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 $\mathcal{F}\Gamma/{\equiv_L}$ is locally finite. ◀

6 Deterministic monoidal automata

Classically, the expressive equivalence of deterministic and non-deterministic finite-state automata for string languages is well known, but already for trees, top-down deterministic tree automata are less expressive than bottom-up deterministic tree automata. Therefore we cannot expect to determinize non-deterministic monoidal automata. However, we have already seen monoidal languages that are deterministically recognizable (Examples 10, 11, 12, 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 7 we consider the question of determinizability.

▶ **Definition 40.** *A deterministic monoidal automaton $\delta = (Q, \delta_\Gamma)$ over a monoidal alphabet Γ is given by a finite set Q, together with transition functions $\delta_\Gamma = \{Q^{ar(\gamma)} \xrightarrow{\delta_\gamma} Q_\perp^{coar(\gamma)}\}_{\gamma\in\Gamma}$.*

Recall the definition of the pro Par_Q from Remark 18. Then as in Observation 19, such assignments $\gamma \mapsto \delta_\gamma$ uniquely extend to pro morphisms $\delta : \mathcal{F}\Gamma \to \mathsf{Par}_Q$, and we will also refer to such pro morphisms 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 41.** *Let $\delta : \mathcal{F}\Gamma \to \mathsf{Par}_Q$ be a deterministic monoidal automaton. Then the language accepted by δ is $\mathcal{L}(\delta) := \{\alpha \in \mathcal{F}\Gamma(0,0) \mid \delta_\alpha(\bullet) = \bullet\}$.*

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 [9]. For string diagrams, we call the analogue of paths through a tree the *causal histories* of a diagram (Definition 46).

First, we briefly recall the machinery of (cartesian) *restriction categories* [3], 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.

▶ **Definition 42** ([4]). *A cartesian restriction prop is a prop in which every object is equipped with a commutative comonoid structure (with the counit depicted by* —• *, comultiplication by* —•⟨ *, and symmetry by* ⤫ *) that is coherent, and for which the comultiplication is natural (for more details, see the full version [6]).*

▶ **Definition 43.** *The free cartesian restriction prop on a monoidal graph $\mathcal{M}$, denoted $\mathcal{F}_{\downarrow}\mathcal{M}$ 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 (for more details, see the full version [6]).*

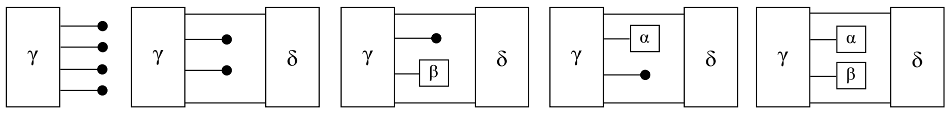
▶ Remark 44. Par is the paradigmatic example of a cartesian restriction category, with —• on X given by the relation $X \to \{\bullet, \bot\}$ sending every element to $\bullet$, and —•⟨ given by the diagonal relation. Par_Q inherits this structure and so is a cartesian restriction prop. Therefore deterministic monoidal automata (Q, δ_Γ) also have inductive extensions to morphisms of cartesian restriction props, $\overline{\delta} : \mathcal{F}_{\downarrow}\Gamma \to \mathsf{Par}_Q$, and these have a obvious notion of associated language, defined similarly to Definition 41. These are related by the following lemma, which follows from the universal properties of $\mathcal{F}\Gamma$ and $\mathcal{F}_{\downarrow}\Gamma$:

▶ **Lemma 45.** *If (Q, δ_Γ) is a deterministic monoidal automaton, then δ factors through $\overline{\delta}$ as $\delta = \mathcal{H}_\Gamma \mathbin{;} \overline{\delta}$, where $\mathcal{H}_\Gamma : \mathcal{F}\Gamma \to \mathcal{F}_{\downarrow}\Gamma$ sends morphisms to their equivalence class in $\mathcal{F}_{\downarrow}\Gamma$.*

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. 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 46.** *Let γ be a string diagram in $\mathcal{F}\Gamma(0,0)$. We call a string diagram h in $\mathcal{F}_{\downarrow}\Gamma(0,0)$ a causal history of γ if $\mathcal{H}_\Gamma(\gamma) \leqslant h$ in $\mathcal{F}_{\downarrow}\Gamma(0,0)$. Let $L \subseteq \mathcal{F}\Gamma(0,0)$ 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 $\mathcal{F}_{\downarrow}\Gamma(0,0)$.*

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 13:

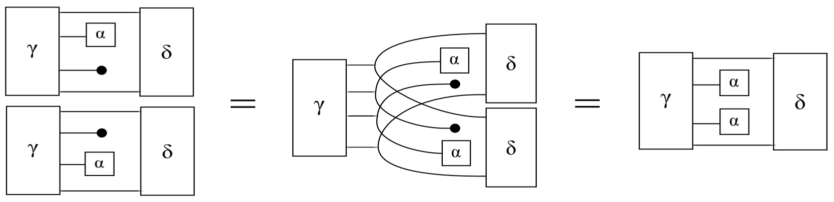


▶ **Lemma 47.** *Let $M = (Q, \delta_\Gamma)$ be a deterministic monoidal automaton, with functors $\delta : \mathcal{F}\Gamma \to \mathsf{Par}_Q$, $\overline{\delta} : \mathcal{F}_{\downarrow}\Gamma \to \mathsf{Par}_Q$. Then if δ accepts γ, $\overline{\delta}$ accepts all causal histories of γ.*

Proof. Since $\delta = \mathcal{H}_\Gamma \mathbin{⨾} \overline{\delta}$, 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 26. ◀

▶ **Definition 48** (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 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 the following figure, which shows part of the derivation of a morphism in the causal closure of the language of Example 13:



The leftmost diagram depicts the monoidal product of two causal histories determined by the counterexample language. By the equational theory of cartesian restriction categories (see the full version [6]), 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 $\mathcal{F}\Gamma$. Since this diagram is not in the original language, the language is not causally closed.

▶ **Theorem 49.** *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 $\delta : \mathcal{F}\Gamma \to \mathsf{Par}_Q$. We have $\delta = \mathcal{H}_\Gamma \mathbin{⨾} \overline{\delta}$ and from Lemma 47 that $\overline{\delta}$ accepts causal histories of morphisms in L. Since languages are closed under monoidal product (Lemma 26), then by definition of the causal closure, δ must accept everything in the causal closure of L. ◀

7 Deterministically recognizable monoidal languages

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 [9, §2.11], so general monoidal automata also cannot be determinized (Observation 33). However, there are interesting examples of deterministically recognizable monoidal languages that are not tree languages, such as the monoidal Dyck language (Example 10) and Sierpiński gaskets (Example 12), and it is an intriguing theoretical challenge to characterize such languages.

In Section 7.1 we study a class of determinizable automata called *convex* automata. In Section 7.2 we give a sufficient condition for a language to be deterministically recognizable.

7.1 Convex automata and the powerset construction

The classical powerset construction is given conceptually by composition with the functor $\mathsf{Rel} \to \mathsf{Set}$, 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 50.** *A relation $\Delta : Q^n \to \mathcal{P}(Q^m)$ is convex if there is a morphism Δ^* such that the following square commutes:*

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

where $\Delta^{\#}$ is the Kleisli lift of Δ, and ∇ is the monoidal multiplication given by the commutativity of the powerset monad.

▶ **Observation 51.** *If Δ is convex, the morphism Δ^* is unique, since ∇ is a monomorphism.*

▶ **Example 52.** The relation $\Delta_\gamma : Q^0 \to \mathcal{P}(Q^4)$ induced by the grammar in Example 13 is not convex, since (A, B, B, A) and (A, C, C, A), which we can think of as "convex combinations" of the other state vectors, are not included in the image of the relation.

▶ **Lemma 53.** *Convex relations determine a sub-pro $\mathsf{CRel}_Q \hookrightarrow \mathsf{Rel}_Q$.*

▶ **Definition 54.** *An automaton $\Delta : \mathcal{F}\Gamma \to \mathsf{Rel}_Q$ is convex if it factors through CRel_Q.*

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

▶ **Lemma 55.** *For each set Q there is a morphism of pros $\mathcal{D}_Q : \mathsf{CRel}_Q \to \mathsf{Par}_{\mathcal{P}^+(Q)}$ which is identity on objects and acts as follows on morphisms:*

$$\Delta_\alpha : Q^n \to \mathcal{P}(Q^m)$$
$$\mapsto$$
$$\mathcal{P}^+(Q)^n \xrightarrow{\eta^n} (\perp\mathcal{P}^+(Q))^n \xrightarrow{\cong} \mathcal{P}(Q)^n \xrightarrow{\Delta^*_\alpha} \mathcal{P}(Q)^m \xrightarrow{\cong} (\perp\mathcal{P}^+(Q))^m \xrightarrow{\nabla} \perp\mathcal{P}^+(Q)^m$$

where $\perp$ is the maybe monad, η is the unit of this monad, and ∇ is its monoidal multiplication with respect to the cartesian product.

Determinization of a convex automaton $\Delta : \mathcal{F}\Gamma \to \mathsf{CRel}_Q$ is now just given by post-composition with the functor $\mathcal{D}_Q$. We show that this preserves the language:

▶ **Theorem 56.** *Determinization of convex automata preserves the accepted language: let $\Delta : \mathcal{F}\Gamma \to \mathsf{CRel}_Q$ be a convex automaton, then $\mathcal{L}(\Delta) = \mathcal{L}(\Delta \,;\, \mathcal{D}_Q)$.*

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

▶ **Example 57.** Non-deterministic monoidal automata over word monoidal alphabets (Definition 30) are convex: for a relation $\Delta : Q \to \mathcal{P}(Q)$, Δ^* is given by the Kleisli extension of Δ. This reflects the well known determinizability of classical finite-state automata.

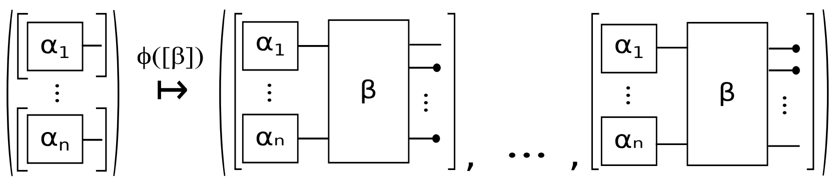
▶ **Example 58.** Similarly, non-deterministic monoidal automata over bottom-up tree monoidal alphabets (Definition 32) are convex, with $\Delta^* := \nabla \,;\, \Delta^{\#}$. For top-down tree monoidal alphabets, the general obstruction to convexity (and thus determinizability) is seen as the non-existence of a left inverse of ∇.

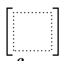
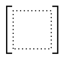
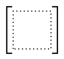
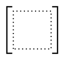
7.2 A sufficient condition for deterministic recognizability

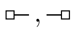
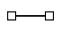
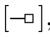
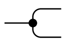
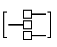
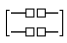
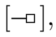
▶ **Theorem 59.** *If the syntactic pro of a regular monoidal language has the structure of a cartesian restriction prop, then the language is recognizable by a deterministic monoidal automaton.*

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

Let $Q := \mathcal{F}\Gamma/{\equiv_L}(0,1)$. By Lemma 39, this is a finite set. For $m > 0$ and $[\beta] \in \mathcal{F}\Gamma/{\equiv_L}(n,m)$, define $\phi([\beta]) : n \to m$ to be the following map from $Q^n \to Q^m_\perp$:



When $m = 0$ (i.e. $[\beta]$ has coarity 0), let $\phi([\beta])([\alpha_1], ..., [\alpha_n]) = \bullet$, if $[(\alpha_1 \otimes ... \otimes \alpha_n) \,⨾\, \beta] =$ , and $\phi([\beta])([\alpha_1], ..., [\alpha_n]) = \perp$ otherwise. The proof that this defines a morphism of pros is an exercise in diagrammatic reasoning using the equational theory of cartesian restriction categories, see the full version [6]. To see that this automaton accepts exactly L, let $\alpha \in \mathcal{L}(S_L \,⨾\, \phi)$, then by definition we must have $S_L(\alpha) =$ , and so $\alpha \in L$ (by Lemma 38). Conversely let $\alpha \in L$, then $S_L(\alpha) =$  and by definition ϕ () $(\bullet) = \bullet$, so $\alpha \in \mathcal{L}(S_L \,⨾\, \phi)$. Therefore $S_L \,⨾\, \phi$ is a deterministic monoidal automaton recognizing L. ◀

▶ **Example 60.** A simple example is the language L of "bones" over the monoidal alphabet $\Gamma = \{$  $\}$, having one connected component: . The syntactic pro $\mathcal{F}\Gamma/{\equiv_L}$ has a cartesian restriction prop structure, with the counit given by the equivalence class , comultiplication  by , and symmetry by . It is clear that $\mathcal{F}\Gamma/{\equiv_L}(0,1)$ has one equivalence class, , which becomes the state of the monoidal automaton. The construction above then gives the obvious transition functions required for each generator.

8 Conclusion and future work

The most immediate open question is to determine necessary and sufficient conditions for determinizability: causal closure is a promising candidate. Furthermore we would like to understand the relation between convexity and Theorem 59. Classical topics in the theory of regular languages such as a Myhill-Nerode theorem are also ripe for future investigation. We also plan to investigate further applications of regular monoidal languages in computer science, for example representing trace languages and look-ahead parsing.

Just as our definition of regular monoidal grammar was obtained from Walters' definition of regular grammar by replacing the adjunction $\mathsf{Cat} \to \mathsf{Graph}$ with the adjunction $\mathsf{Pro} \to \mathsf{MonGraph}$, we might consider other adjunctions and their corresponding notion of grammar. In the first instance, our theory should smoothly generalize to languages in free props, but perhaps also other (higher) categorical structures.

We plan to investigate a notion of context-free monoidal language, using a similar algebraic approach to this paper. One candidate for the algebra of such languages, inspired again by [17], are (monoidal) multicategories of n-hole contexts (in the sense of Definition 34).

References

1 Mikołaj Bojańczyk, Bartek Klin, and Julian Salamanca. Monadic monadic second order logic, 2022. doi:10.48550/ARXIV.2201.09969.

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

3 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.

4 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.

5 Thomas Colcombet and Daniela Petrişan. Automata Minimization: a Functorial Approach. *Logical Methods in Computer Science*, Volume 16, Issue 1, March 2020. doi:10.23638/LMCS-16(1:32)2020.

6 Matthew Earnshaw and Paweł Sobociński. Regular monoidal languages, 2022. doi:10.48550/ARXIV.2207.00526.

7 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.

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

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

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

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

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

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

14 Jacques Sakarovitch. *Elements of automata theory.* Cambridge University Press, Cambridge New York, 2009.

15 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.

16 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.

17 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.

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