

DOCTORAL THESIS

Languages of String Diagrams

Matthew David Earnshaw

TALLINN UNIVERSITY OF TECHNOLOGY
DOCTORAL THESIS
2/2025

Languages of String Diagrams

MATTHEW DAVID EARNSHAW



TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies
Department of Software Science

The dissertation was accepted for the defence of the degree of Doctor of Philosophy on 20 December 2024

Supervisor: Professor Paweł Sobociński,
Department of Software Science,
School of Information Technologies,
Tallinn University of Technology
Tallinn, Estonia

Opponents: Professor Bartek Klin,
University of Oxford
Oxford, United Kingdom

Professor Noam Zeilberger,
Laboratoire d'informatique de l'École Polytechnique
Palaiseau, France

Defence of the thesis: 15 January 2025, Tallinn

Declaration:

Hereby I declare that this doctoral thesis, my original investigation and achievement, submitted for the doctoral degree at Tallinn University of Technology, has not been submitted for any academic degree elsewhere.

Matthew David Earnshaw

signature

Copyright: Matthew David Earnshaw, 2025
ISSN 2585-6898 (publication)
ISBN 978-9916-80-242-7 (publication)
ISSN 2585-6901 (PDF)
ISBN 978-9916-80-243-4 (PDF)
DOI <https://doi.org/10.23658/taltech.2/2025>
Printed by Koopia Niini & Rauam

Earnshaw, M. D. (2025). *Languages of String Diagrams* [TalTech Press].
<https://doi.org/10.23658/taltech.2/2025>

TALLINNA TEHNIKAÜLIKOOL
DOKTORITÖÖ
2/2025

Nöördiagrammide keeled

MATTHEW DAVID EARNSHAW



Contents

Abstract / Kokkuvõte	8
List of Publications	9
Author's Contribution to the Publications	10
1 Introduction	11
1.1 Contributions	12
1.2 Related work	13
1.3 Synopsis	15
2 Background	17
2.1 Monoidal graphs and monoidal categories	17
2.1.1 String diagrams and free monoidal categories	20
2.2 Multigraphs and multicategories	21
2.2.1 Symmetric multicategories	22
2.3 Some monoidal monads	23
2.4 Effectful categories and strong promonads	26
3 Regular Languages of String Diagrams	31
3.1 Grammars as morphisms	32
3.1.1 Regular grammars over arbitrary categories	33
3.2 Languages of string diagrams	34
3.3 Regular monoidal grammars	35
3.3.1 ε -productions	40
3.4 Monoidal pumping lemma	43
3.5 Monoidal automata	45
3.6 Closure properties of regular monoidal languages	49

3.7	Syntactic monoidal category of a language	51
3.8	Deterministically recognizable languages	55
3.8.1	Partial views via cartesian restriction categories	55
3.9	Convex monoidal automata	59
3.10	Mazurkiewicz traces and asynchronous automata	63
3.10.1	Independence and distribution	63
3.10.2	Symmetric monoidal languages over monoidal distributed alphabets	65
3.10.3	Asynchronous automata as monoidal automata	68
3.10.4	Serialization of traces via premonoidal categories	70
4	From Mazurkiewicz Traces to Resourceful Traces	73
4.1	Mazurkiewicz traces are effectful scalars	74
4.2	Presenting effectful categories by devices	76
4.3	Resourceful traces for message passing processes	81
4.4	Independence and distribution for premonoidal categories	83
4.5	Centralizers in effectful categories	84
4.6	Commuting tensor product of effectful categories	86
4.6.1	Presenting the commuting tensor product by devices	87
5	Context-Free Languages of String Diagrams	90
5.1	Context-free grammars as morphisms of multigraphs	91
5.1.1	Context-free languages in monoids and other categories	91
5.2	The splice-contour adjunction: categories and multicategories	92
5.3	Diagram contexts	94
5.4	Context-free monoidal grammars	97
5.5	Optical contour: left adjoint to diagram contexts	100
5.5.1	The multicategory of raw optics	100
5.5.2	Optical contour of a multicategory	102
5.6	Representation theorem for context-free monoidal languages	104
5.6.1	Chomsky-Schützenberger representation theorem	104
5.6.2	Raw representatives of a grammar	105
5.6.3	Regular representation of a grammar	107
5.6.4	Monoidal representation theorem	108
6	Conclusion and Future Work	110
	Acknowledgements	112
	Bibliography	113
	Index	122

A Regular monoidal languages over arbitrary strict monoidal categories	125
B Paper I	129
C Paper II	145
D Paper III	163
E Paper IV	199
Curriculum Vitae / Elulookirjeldus	230

Abstract

In this thesis we introduce and investigate the theory of *regular* and *context-free languages of string diagrams*, considered as morphisms in monoidal categories. The former class includes regular languages of words, trees, and Mazurkiewicz traces. We introduce a pumping lemma and investigate deterministic recognizability. The example of trace languages leads to a refinement of string diagrams for effectful categories, and a construction of their commuting tensor product. Context-free languages of string diagrams include classical context-free languages of words, trees and hypergraphs, when established over appropriate monoidal categories. We prove a representation theorem that links the two classes of languages, inspired by the Chomsky-Schützenberger representation theorem.

Kokkuvõte

Käesolevas doktoritöös toome sisse nõordiagrammide regulaarsete ja kontekstivabade keelte teooria. Taolisi keeli käsitleme morfismidena monoidilistes kategooriates. Regulaarsete nõordiagrammide keelte klassi kuuluvad regulaarsed sõnade, puude ja Mazurkiewiczzi jälgede keeled. Me tõestame pumpamislemma ja uurime deterministlikku keeldekuuluvuse tuvastatavust. Jälgede keelte näide viib peenema variandini nõordiagrammide efektidega kategooriate jaoks. Kontekstivabade nõordiagrammide keelte klassis sisalduvad klassikalised sõnade, puude ja hüpergraafide kontekstivabad keeled konstrueerituna sobivates monoidilistes kategooriates. Me tõestame esitavusteoreemi, mis seostab need kaks keelteklassi, ja on inspireeritud Chomsky ja Schützenberger' esitavusteoreemist.

List of Publications

This thesis includes material from the following articles, which can be found in Appendices B to E.

- I Matthew Earnshaw and Paweł Sobociński. “Regular Monoidal Languages”. In: *47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022)*. Ed. by Stefan Szeider, Robert Ganian and Alexandra Silva. Vol. 241. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022, 44:1–44:14. ISBN: 978-3-95977-256-3. DOI: 10.4230/LIPIcs.MFCS.2022.44
- II Matthew Earnshaw and Paweł Sobociński. “String Diagrammatic Trace Theory”. In: *48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023)*. Ed. by Jérôme Leroux, Sylvain Lombardy and David Peleg. Vol. 272. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023, 43:1–43:15. ISBN: 978-3-95977-292-1. DOI: 10.4230/LIPIcs.MFCS.2023.43
- III Matthew Earnshaw and Paweł Sobociński. “Regular planar monoidal languages”. In: *Journal of Logical and Algebraic Methods in Programming* 139 (2024), p. 100963. ISSN: 2352-2208. DOI: <https://doi.org/10.1016/j.jlamp.2024.100963>
- IV Matthew Earnshaw and Mario Román. “Context-Free Languages of String Diagrams”. In: *28th International Conference on Foundations of Software Science and Computation Structures*. To appear. 2025

Author's Contribution to the Publications

We refer to the list of publications on the previous page.

- I My contribution to this work was to identify the research problem, main definitions, to prove the theorems, and write the manuscript.
- II My contribution to this work was to identify the research problem, main definitions, to prove the main theorem, and write the manuscript.
- III My contribution to this work was to identify the research problem, main definitions, to prove the theorems, and write the manuscript.
- IV My contribution to this work was to identify the research problem and main definitions, to prove the theorems, and write most of the manuscript. The results should be attributed equally to myself and Mario Román, who also contributed to the manuscript.

-

Chapter 1

Introduction

What might a *formal language theory of string diagrams* look like, and what can it do for us? This thesis finds its origins in these simple questions.

Formal language theory is a classical part of computer science that deals with *syntax*; typically the syntax of programming languages, but also of formal languages more generally, such as they appear in mathematics and logic. It provides the tools that enable us to specify formal languages, according to notions of *grammar*, and to mechanically *recognize* given syntactic objects as well-formed or ill-formed, by various notions of *automata*.

Words, i.e. finite sequences of symbols, are the classical unit of syntax in formal language theory. In mathematical terms, words are fruitfully seen as elements of the *free monoid* over an alphabet, and so *monoids* are the central algebraic structure of the classical theory. With this step of algebraic abstraction taken, the methods of language theory have spread over wider notions of syntax: infinite words [73], rational sequences [6], trees [9, 38], graphs of bounded tree width [22], and traces [26], to name a few, each corresponding to different algebraic structures.

Several works have endeavoured to take a further step of abstraction, unifying the language theory of these various structures. Starting in the 1960s, the methods of universal algebra were mobilized towards this end, such as in the work of Eilenberg and Wright [33], and Thatcher and Wright [86]. More recently, a focus on monads has taken this earlier work further [7, 10, 87]. For example, Bojańczyk, Klin and Salamanca [10] have given sufficient conditions on Set-based monads for the correspondence between regularity and definability in monadic second-order logic to extend to languages over their algebras.

In this thesis, we strike out in another direction, continuing the exploration of language theory in the context of more exotic syntactic objects than words, but objects that do not fall under the umbrella of the aforementioned unifying methods.

Instead of monoids, our focus is on languages living in *2-dimensional* 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 seen as 2-categories with one object: “higher” monoids in which there are transformations between the elements.

The natural syntax of monoidal categories is 2-dimensional: these are the *string diagrams* of our title. Accordingly, we call languages in monoidal categories *languages of string diagrams* or *monoidal languages*. A monoidal language in this sense is simply a subset of morphisms in a strict monoidal category, just as a classical formal language is a subset of a monoid.

String diagrams resemble graphical languages commonly found in engineering and science, and indeed, they allow us to reason about Markov kernels [36], linear algebra [12], or quantum processes [1]. In computer science, they provide foundations for visual programming [46, 51]. The use of string diagrams as a syntax in these various domains is one source of inspiration for investigating string diagrams as a formal language.

We start by introducing grammars and automata for languages of string diagrams, defining the class of regular monoidal languages. We show how these include classical and tree automata, but also open up a wilder world of string diagram languages. A key example of such languages is given by languages of Mazurkiewicz traces, which we show to have a natural string diagrammatic presentation. This leads us towards the idea of seeing morphisms in *premonoidal* categories as a generalization of traces. Finally, we introduce context-free grammars of string diagrams, which subsume notions such as context-free hypergraph grammars, and prove a representation theorem linking them to the regular monoidal languages.

1.1 Contributions

We summarize our main contributions.

- Definition 3.3.1 introduces regular monoidal grammars, which define the regular languages of string diagrams.
- Theorem 3.4.1 proves a “pumping lemma” for regular languages of string diagrams, giving a necessary condition for regularity.
- Definition 3.5.10 introduces monoidal automata, which are seen to include classical notions of regular, tree and asynchronous automata as special cases.
- Definition 3.7.11 introduces the syntactic monoidal category of a language, and Theorem 3.7.15 gives a necessary condition for regularity based on it.
- Theorem 3.8.11 provides a necessary condition for regular languages of string diagrams to be deterministically recognizable.

- Theorem 3.9.9 shows that the subclass of *convex monoidal automata* is determinizable by a powerset construction.
- Theorem 3.10.17 characterizes the recognizable trace languages amongst the symmetric regular monoidal languages.
- Definition 4.2.9 defines a notion of presentation for effectful categories. Theorem 4.6.5 uses this notion of presentation to present the commuting tensor product of effectful categories.
- Definition 5.4.1 introduces context-free monoidal grammars, which define the context-free languages of string diagrams.
- Theorem 5.5.6 establishes an adjunction between our notions of *raw optics* and *optical contours*.
- Theorem 5.6.13 uses this adjunction to prove that every context-free language of string diagrams is the image under a monoidal functor of a regular language of string diagrams.

1.2 Related work

Regular languages of string diagrams

Bossut [13] studied rational languages of planar acyclic graphs and proved a Kleene theorem for a class of such languages, which applies to connected languages. We make explicit the fact the languages of graphs investigated by Bossut have an underlying algebra, that of monoidal categories. We are then able to leverage this algebra in our proofs and definitions. This leads us in quite different directions of investigation from Bossut, mostly towards questions of deterministic recognizability. The main difference between our languages of string diagrams and those of Bossut, is that we do not allow the “boundaries” of our graphs to vary, that is, our languages live in a particular hom-set of a (monoidal) category. One reason for this restriction is that we do not have examples that compel us to investigate the broader class of Bossut.

In the preprint [43], 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 definition in Definition 3.7.11, and prove one direction of the result, conjecturing that the converse does not hold.

Winfrey et al. [81] 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 3.3.8.

Walters' note [89] on regular and context-free grammars served as a starting point for our definition of regular monoidal grammar. Rosenthal [80], developing some of the ideas of Walters, defined automata as relational presheaves, which is similar in spirit to our functorial definition of monoidal automata. Functorial treatments of (tree) automata go back at least to the work of Eilenberg and Wright in the 1960s [33], inspired by Lawvere's thesis. More recently, Colcombet and Petrişan [21] have also considered automata as functors, using this perspective to prove correctness of known minimization algorithms. However, all of these works are directed towards questions involving classical word or tree languages, rather than languages of diagrams. Bruggink and König's investigation of recognizable languages of arrows in a category [15] defines a notion of *automaton functor*, which is a functor $\mathbb{C} \rightarrow \text{FinRel}$ equipped with the data of initial and final states. This is similar to our notion of monoidal automaton, which is a *strict monoidal functor* into a certain strict version of FinRel . Our line of development is quite different however, as we are not concerned with the questions of monadic-second order logic and recognizable graph languages investigated by Bruggink and König [15], but more with questions of determinization, which is not always possible in our case.

In the introduction to Joyal & Street's foundational work on string diagrams for monoidal categories [49], it is suggested that string diagrams have a connection to the *heaps* of Viennot [88]. Heaps are known to be equivalent to Mazurkiewicz trace monoids (also known as partially commutative monoids) [50], but a precise formulation of the suggested relation with string diagrams, as formulated in Section 3.10, has not appeared in the literature previously.

The notion of dependence graph [45] has also been used to give a topological presentation of Mazurkiewicz traces. Our use of the algebra of monoidal categories, rather than graphs, has various advantages. For example, we can apply our language theory for monoidal categories to traces, and we see notions such as asynchronous automata arise naturally from this. It also suggests generalizations of trace languages, in particular going beyond the case of atomic actions, an idea which we explore in Chapter 4. Finally, it brings our work into proximity with the semantics of Petri nets and other formalisms for concurrency based on monoidal categories [2, 67].

From atomic traces to resourceful traces

String diagrams for premonoidal categories were introduced by Jeffrey [47, 46], with the theory later refined by Román [77, 78]. The extension of these diagrams to include multiple distinguished strings was suggested by the author, motivated by regular languages of Mazurkiewicz traces. These ideas were first elaborated in the extended abstract joint with Nester and Román [28]. The commuting tensor

product of effectful categories was defined via universal property by Garner and López Franco – here we use our string diagrams to give a simple presentation using string diagrams. Similar string diagrams have recently appeared in the work of Barrett, Heijltjes, and McCusker [4, 3], in the context of the functional machine calculus, an effectful λ -calculus. We hope to elaborate the connection between these ideas in future work.

Context-free languages of string diagrams

The representation of context-free grammars as certain morphisms of multigraphs was introduced by Walters in a short paper [89]. A similar type-theoretical version of this idea has been studied by De Groote [41]. This idea was taken up and substantially refined by Melliès and Zeilberger, in a conference paper [62] and later in an extended version [63].

A different notion of context-free families of string diagrams has been introduced by Zamdzhiev [93]. There, string diagrams are defined combinatorially as *string graphs*, and context-free families are then generated by B-edNCE graph grammars [83]. Though similar, the resulting notion is not directly comparable to ours. Here, we use the native algebra of monoidal categories and their multicategories of contexts to define and investigate languages.

Finally, Heindel’s abstract [44] claims a proof of a Chomsky-Schützenberger theorem for morphisms in symmetric monoidal categories, but the work described in this abstract was never published. Our development is quite different from that outlined in Heindel’s abstract. We prove a stronger representation theorem that does not require an intersection of languages; we work without the assumption of symmetry; and we generalize the categorical machinery of Melliès and Zeilberger.

1.3 Synopsis

Chapter 2 provides a reference on background material, assuming some familiarity with basic notions of category theory. In particular, we define the notions of monoidal category, multicategory and effectful category that shall be our central algebraic ingredients. The main part of the thesis, comprising our original research, is contained in Chapters 3 to 5, and each chapter is prefaced by a short summary of its contents.

Chapter 3 introduces languages of string diagrams and in particular the regular languages of string diagrams which are defined by a notion of regular monoidal grammar and monoidal automaton. The main theoretical focus of this chapter is on deterministic recognizability of languages. We provide a necessary condition for deterministic recognizability, and identify a class of determinizable monoidal automata. We also show in detail how asynchronous automata are an example of

monoidal automata, linking regular languages of string diagrams to languages of Mazurkiewicz traces.

Chapter 4 takes a detour from language theory, elaborating the link between trace languages, seen as languages of string diagrams, and the string diagrams for effectful categories. Our string-diagrammatic view of trace languages suggests a refinement of the latter, in which multiple distinguished strings (termed *devices*) control the interchange of morphisms. As an application, we show how devices facilitate a convenient presentation of the commuting tensor product of effectful categories.

Chapter 5 introduces context-free languages of string diagrams via a notion of grammar, formalizing the intuitive idea of rewrites of “holes” (contexts) into “string diagrams with holes”. Examples include classical context-free languages, and context-free languages of hypergraphs. We exhibit a splice-contour adjunction between monoidal categories and multicategories, which is a variation of a similar adjunction introduced by Melliès and Zeilberger. This allows us to prove a representation theorem for context-free languages of string diagrams, which states that every such language arises as the image under a monoidal functor of a regular language of string diagrams.

Chapter 6 provides some concluding remarks and future directions.

Background

In this chapter, we introduce the basic definitions and propositions concerning the structures used in the following chapters. Specifically,

- Monoidal categories and string diagrams — *Section 2.1*
- Multicategories — *Section 2.2*
- Monoidal monads — *Section 2.3*
- Effectful categories and promonads — *Section 2.4*

2.1 Monoidal graphs and monoidal categories

In this thesis, we will largely be concerned with *strict monoidal categories* presented by generators and equations, in fact in many cases with simply *free* monoidal categories. The appropriate generating data for a monoidal category is a *monoidal graph*: a graph whose edges, which we shall depict as boxes, have multiple input and output strings. Monoidal graphs are also known as *tensor schemes* [49], *monoidal signatures* [24], *polygraphs* [8] and *pre-nets* [2].

Definition 2.1.1. A monoidal graph G is given by a set S_G of *sorts*, a set B_G of *generators* and functions $s, t : B_G \rightrightarrows S_G^*$ assigning source and target (or input and output) words of sorts to each generator. We denote the set of generators with a given source and target by the notation

$$G(X_1, \dots, X_n; Y_1, \dots, Y_m).$$

We depict a monoidal graph as a *string diagram* (Figure 2.1): the generators are drawn as boxes, and their input and output sorts are drawn as labelled strings, with inputs on the left and outputs on the right.

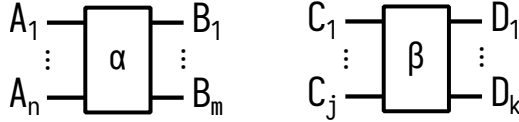


Figure 2.1: A monoidal graph with two generators.

Definition 2.1.2. A morphism of monoidal graphs $\phi : G \rightarrow G'$ is given by a function $\phi_S : S_G \rightarrow S_{G'}$ on sorts and a function $\phi_G : B_G \rightarrow B_{G'}$ on generators, such that $\phi_G \circ s' = s \circ \phi_S^*$ and $\phi_G \circ t' = t \circ \phi_S^*$, or equivalently by a function $\phi : S_G \rightarrow S_{G'}$ on sorts and functions

$$G(A_1, \dots, A_n; B_1, \dots, B_m) \rightarrow G'(\phi(A_1), \dots, \phi(A_n); \phi(B_1), \dots, \phi(B_m))$$

between sets of generators.

The action of a morphism of monoidal graphs is depicted in Figure 2.2. Monoidal graphs and their morphisms form a category MonGraph .

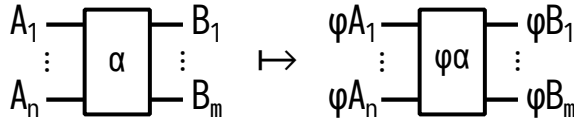


Figure 2.2: The action of a morphism of monoidal graphs.

Monoidal categories provide an algebra in which we can interpret certain diagrams built from monoidal graphs. They are categories equipped with the structure of a (coherently) associative and unital *parallel composition*, which abstracts classical notions such as the cartesian product of sets, or the tensor product of vector spaces. The composition structure of the underlying category gives us a recipe for interpreting diagrams in which all the outputs of a diagram are connected to all the inputs of another. The parallel composition gives us a recipe for interpreting diagrams built from juxtaposition of diagrams. In this thesis, we shall largely be concerned with *strict* monoidal categories: those for which the tensor product is unital and associative on the nose.

Definition 2.1.3. A strict monoidal category \mathbb{C} consists of a monoid of objects, $(\mathbb{C}_{\text{obj}}, \otimes, I)$, and a collection of morphisms $\mathbb{C}(X; Y)$, for every pair of objects $X, Y \in \mathbb{C}_{\text{obj}}$. A strict monoidal category is endowed with (families of) operations for the sequential and parallel composition of morphisms, respectively

$$\begin{aligned} (\circ) : \mathbb{C}(X; Y) \times \mathbb{C}(Y; Z) &\rightarrow \mathbb{C}(X; Z), \text{ and} \\ (\otimes) : \mathbb{C}(X; Y) \times \mathbb{C}(X'; Y') &\rightarrow \mathbb{C}(X \otimes X'; Y \otimes Y'), \end{aligned}$$

and a family of identity morphisms, $\text{id}_X \in C(X; X)$. Strict monoidal categories must satisfy the following axioms;

- sequential composition is unital, $f \circ \text{id}_Y = f$ and $\text{id}_X \circ f = f$,
- sequential composition is associative, $f \circ (g \circ h) = (f \circ g) \circ h$,
- parallel composition is unital, $f \otimes \text{id}_I = f$ and $\text{id}_I \otimes f = f$,
- parallel composition is associative, $f \otimes (g \otimes h) = (f \otimes g) \otimes h$,
- parallel composition and identities interchange, $\text{id}_A \otimes \text{id}_B = \text{id}_{A \otimes B}$, and
- parallel composition and sequential composition interchange, $(f \circ g) \otimes (f' \circ g') = (f \otimes f') \circ (g \otimes g')$.

Parallel composition is also called the tensor product or monoidal product.

In a general (non-strict) monoidal category, objects no longer form a monoid, but only a pseudomonoid; that is, there are natural families of isomorphisms $(A \otimes B) \otimes C \cong A \otimes (B \otimes C)$, $I \cong A \otimes I$ called *associators* and *unitors* respectively, satisfying coherence conditions [59, Chapter VII].

Definition 2.1.4. A symmetric strict monoidal category is a strict monoidal category equipped with a natural family of isomorphisms called *symmetries*

$$\sigma_{X,Y} : X \otimes Y \rightarrow Y \otimes X,$$

such that

- $\sigma_{X,Y} \circ \sigma_{Y,X} = \text{id}_{X \otimes Y}$, and
- $(\sigma_{X,Y} \otimes \text{id}_Z) \circ (\text{id}_Y \otimes \sigma_{X,Z}) = \sigma_{X,Y \otimes Z}$.

These equations have a natural string diagrammatic interpretation: they allow us to cross strings without tangling them. We introduce this in the next section (Figure 2.4).

Definition 2.1.5. A strict monoidal functor $F : (\mathbb{C}, \otimes_{\mathbb{C}}, I_{\mathbb{C}}) \rightarrow (\mathbb{D}, \otimes_{\mathbb{D}}, I_{\mathbb{D}})$ between strict monoidal categories is a functor $F : \mathbb{C} \rightarrow \mathbb{D}$ between the underlying categories, such that $F(X \otimes_{\mathbb{C}} Y) = F(X) \otimes_{\mathbb{D}} F(Y)$ for all pairs of objects $X, Y \in \mathbb{C}_{\text{obj}}$, and $F(I_{\mathbb{C}}) = I_{\mathbb{D}}$. A symmetric strict monoidal functor between symmetric monoidal categories is a strict monoidal functor additionally preserving the symmetry maps, $F(\sigma_{X \otimes Y}) = \sigma_{F(X) \otimes F(Y)}$.

Strict monoidal categories with strict monoidal functors form a category MonCat . In this thesis, we will mostly be concerned with those strict monoidal categories having the property that their monoid of objects is a free monoid: such monoidal categories are known as *pros* and *props*.

Definition 2.1.6. A pro is a strict monoidal category whose monoid of objects is the free monoid on a set (its set of *sorts*). A prop is a symmetric strict monoidal category with this property. Denote by $\mathcal{S}(\mathbb{C})$ the set of sorts of a $\text{pro}(\mathbb{p})$.

Definition 2.1.7. A morphism of $\text{pro}(\mathbb{p})$ s $F : (\mathbb{C}, \otimes_{\mathbb{C}}, I_{\mathbb{C}}) \rightarrow (\mathbb{D}, \otimes_{\mathbb{D}}, I_{\mathbb{D}})$ is a function $f : \mathcal{S}(\mathbb{C}) \rightarrow \mathcal{S}(\mathbb{D})$ between the underlying sorts, and a strict (symmetric) monoidal functor whose action on objects is given by the monoid homomorphism freely induced by the function f .

2.1.1 String diagrams and free monoidal categories

The free strict monoidal category over a monoidal graph can be conveniently presented using *string diagrams*; the morphisms of a free monoidal category are certain string diagrams built from the monoidal graph. String diagrams are often more natural and convenient to work with than the term syntax of the previous section, since axioms such as associativity of composition hold automatically.

Definition 2.1.8. The free strict monoidal category on a monoidal graph G , denoted $\mathcal{F}_{\otimes}G$, has objects given by lists of sorts over S_G and morphisms given by string diagrams built from the generators of G , as in Figure 2.3. A string diagram is an equivalence class of the diagrams generated by Figure 2.3, up to planar isotopy, keeping the left and right boundaries fixed.

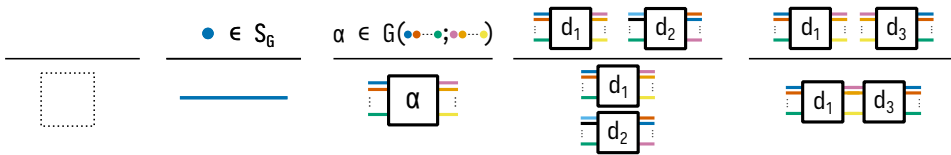


Figure 2.3: The free strict monoidal category over a monoidal graph has morphisms given by equivalence classes (up to planar isotopy) of diagrams inductively generated as above. The leftmost rule denotes the empty diagram, the identity on the monoidal unit I . We use colours here to indicate sorts.

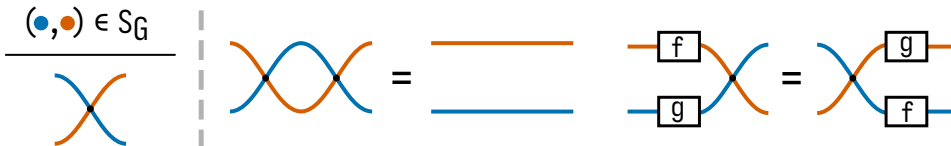


Figure 2.4: The free symmetric strict monoidal category has, as morphisms, string diagrams in which strings may cross without tangling.

Definition 2.1.9. The free symmetric strict monoidal category on a monoidal graph G , denoted $\mathcal{F}_{\chi}G$, has objects given by lists of sorts over S_G and morphisms given by

string diagrams built from the generators of G , as in Figure 2.3 with the addition of the generators and equations of Figure 2.4.

Proposition 2.1.10. *Every (strict) monoidal category has an underlying monoidal graph, and this extends to a functor $\mathcal{U} : \text{MonCat} \rightarrow \text{MonGraph}$.*

Proof. We indicate the action on objects. The underlying monoidal graph $\mathcal{U}\mathbb{C}$ has the same objects as \mathbb{C} and generators

$$\mathcal{U}\mathbb{C}(A_1, \dots, A_n; B_1, \dots, B_m) := \mathbb{C}(A_1 \otimes \dots \otimes A_n; B_1 \otimes \dots \otimes B_m).$$

□

Proposition 2.1.11 (Joyal and Street [49]). $\mathcal{F}_\otimes : \text{MonGraph} \rightarrow \text{MonCat}$ is left adjoint to the forgetful functor $\mathcal{U} : \text{MonCat} \rightarrow \text{MonGraph}$, and similarly for $\mathcal{F}_\chi : \text{SymMonCat} \rightarrow \text{MonGraph}$.

2.2 Multigraphs and multicategories

Multicategories, and in particular multicategories freely generated by *multigraphs*, will provide us with the appropriate algebra to introduce a suitably abstract definition of context-free language of string diagrams. We recall the definitions of multicategory, morphism of multicategories, symmetric multicategory and free multicategory on a multigraph. For a more extensive reference, see for example Leinster [54].

Definition 2.2.1. A multigraph M is given by a set S_M of *objects*, and sets $M(X_1, \dots, X_n; Y)$ of *operations* for every list of objects $X_1, \dots, X_n \in S_M$ and object Y . A multigraph is *finite* just when all of its data are finite sets.

Multicategories formalize the idea of category-like gadgets in which morphisms have *lists* of objects as their domains.

Definition 2.2.2. A morphism of multigraphs $F : M \rightarrow N$ is given by a function $F : S_M \rightarrow S_N$ and functions

$$M(X_1, \dots, X_n; Y) \rightarrow N(FX_1, \dots, FX_n; FY).$$

Multigraphs and their morphisms form a category MultiGraph .

Definition 2.2.3. A multicategory \mathbb{M} consists of

- A collection \mathbb{M}_{obj} of objects,
- collections of multimorphism $\mathbb{M}(X_1, \dots, X_n; Y)$ for every list of objects X_1, \dots, X_n and object Y ,

- for each object X , an identity morphism $\text{id}_X \in \mathbb{M}(X; X)$,
- for each list of objects $\Gamma, \Gamma_1, \Gamma_2$ and objects X, Y , a composition function $\circ_Y: \mathbb{M}(\Gamma; Y) \times \mathbb{M}(\Gamma_1, \Gamma_2; X) \rightarrow \mathbb{M}(\Gamma_1, \Gamma, \Gamma_2; X)$, such that
- composition is associative, $f \circ_X (g \circ_Y h) = (f \circ_X g) \circ_Y h$,
- unital $f \circ_Y \text{id}_Y = f$, $\text{id}_X \circ_X f = f$,
- and satisfies interchange, $(f \circ_X g) \circ_Y h = (f \circ_Y h) \circ_X g$, for every $f: \Gamma_1, X, \Gamma_2, Y, \Gamma_3 \rightarrow Z$ and $g: \Gamma \rightarrow X, h: \Gamma' \rightarrow Y$.

Definition 2.2.4. A morphism of multicategories or *multifunctor* $F: \mathbb{M} \rightarrow \mathbb{N}$ between multicategories \mathbb{M}, \mathbb{N} is an assignment $F: \mathbb{M}_{\text{obj}} \rightarrow \mathbb{N}_{\text{obj}}$ on objects and assignments $\mathbb{M}(X_1, \dots, X_n; Y) \rightarrow \mathbb{N}(FX_1, \dots, FX_n; FY)$ preserving identities and composition.

Definition 2.2.5. The free multicategory on a multigraph G , denoted $\mathcal{F}_\nabla G$ has morphisms inductively generated by

$$\frac{X \in G_{\text{obj}}}{\text{id}_X \in \mathcal{F}_\nabla G(X, X)} \quad \frac{f \in G(X_1, \dots, X_n; Y)}{f \in \mathcal{F}_\nabla G(X_1, \dots, X_n; Y)}$$

$$\frac{g \in \mathcal{F}_\nabla G(Y_1, \dots, Y_m; Z) \quad \{f_i \in \mathcal{F}_\nabla G(X_{i1}, \dots, X_{ik_i}; Y_i)\}_{i=1}^m}{(f_1, \dots, f_m) \circ g \in \mathcal{F}_\nabla G(X_{11}, \dots, X_{1k_1}, \dots, X_{m1}, \dots, X_{mk_m}; Z)},$$

quotiented by the least congruence identifying morphisms up to unitality and associativity.

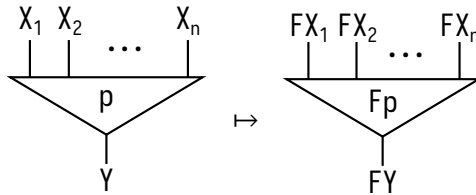


Figure 2.5: Operations of a multigraph/multimorphisms in a multicategory may be depicted as corollas. Here we depict the action of a morphism of multigraphs F on an operation p .

Multicategories and multifunctors form a category MultiCat .

2.2.1 Symmetric multicategories

Many naturally occurring multicategories have the extra structure of *symmetry*, which captures the possibility of coherently permuting the inputs to multimorphisms. For example, in the multicategory \mathbb{M} underlying a symmetric monoidal category, the symmetry map $\sigma_{A,B}: A \otimes B \rightarrow B \otimes A$ induces a bijection $\mathbb{M}(A, B; C) \cong$

$\mathcal{M}(B, A; C)$, and more generally bijections between any two hom-sets whose inputs are permutations of one another.

In Section 5.3, we shall introduce a symmetric multicategory of “string diagrams with holes” or *diagram contexts*. The arities of the multimorphisms will list the types of holes appearing in a string diagram. Just because there is no canonical ordering on the holes in a diagram, this multicategory will be symmetric.

Definition 2.2.6. A symmetric multigraph is a multigraph G equipped with bijections

$$\sigma^* : G(X_1, \dots, X_n; Y) \cong G(X_{\sigma(1)}, \dots, X_{\sigma(n)}; Y)$$

for every list X_1, \dots, X_n of sorts and every permutation σ , satisfying $(\sigma \cdot \tau)^* = \sigma^* \circ \tau^*$ and $\text{id}^* = \text{id}$. A morphism of symmetric multigraphs is a morphism of multigraphs which commutes with the bijections.

Definition 2.2.7 (Leinster [54]). A symmetric multicategory is a multicategory equipped with functions, $- \cdot \sigma : \mathbb{M}(X_1, \dots, X_n; Y) \rightarrow \mathbb{M}(X_{\sigma(1)}, \dots, X_{\sigma(n)}; Y)$, where $\sigma \in S_n$, satisfying $(f \cdot \sigma) \cdot \sigma' = f \cdot (\sigma\sigma')$ and $f = f \cdot \text{id}$, which implies that the functions $- \cdot \sigma$ are bijections. These functions must be compatible with composition in the expected way.

Remark 2.2.8. Every symmetric multicategory has an underlying symmetric multigraph, given by forgetting composition and identities.

Lemma 2.2.9. *The inclusion of symmetric multigraphs into multigraphs has a left adjoint, $\text{clique} : \text{MultiGraph} \rightarrow \text{SymMultiGraph}$. The symmetric multigraph $\text{clique}(M)$ has the same objects as M , and for each $f \in M(X_1, \dots, X_n; Y)$, one freely adds elements*

$$f_\sigma \in \text{clique}(M)(X_{\sigma(1)}, \dots, X_{\sigma(n)}; Y),$$

for every $\sigma \in S_n$, and supplies the bijections $\sigma^ : f_\tau \mapsto f_{\sigma\tau}$, identifying f with f_{id} .*

Proof. A morphism $\phi : \text{clique}(M) \rightarrow N$ of symmetric multigraphs is completely determined by its values on the elements of M , since ϕ must commute with the symmetries, and vice-versa. \square

Proposition 2.2.10. *A symmetric multigraph freely generates a symmetric multicategory, as in Definition 2.2.5.*

2.3 Some monoidal monads

Although we shall not work at the level of generality of arbitrary monads, we shall find it beneficial to structure our definition of monoidal automata in terms of the

powerset, non-empty powerset, and maybe monads, and the expression of the former in terms of a distributive law between the latter two.

Not only will this make some proofs more straightforward, but it makes the relevant conceptual structures explicit, and so the definitions more useful and reusable. For example, while we only investigate deterministic and non-deterministic monoidal automata, use of a probability monad would allow for the definition of stochastic monoidal automata, following Burroni [16].

Definition 2.3.1. The maybe monad $(- + \perp : \mathbf{Set} \rightarrow \mathbf{Set}, \mu : (- + \perp)^2 \rightarrow (- + \perp), \eta : 1 \rightarrow (- + \perp))$ has underlying functor $X \mapsto X + \perp$, where \perp is a singleton set (with element also denoted \perp), multiplication $\mu_X = [\text{id}, \text{inr}] : (X + \perp) + \perp \rightarrow X + \perp$, and unit $\eta_X = \text{inl} : X \rightarrow X + \perp$.

Definition 2.3.2. The powerset monad $(\mathcal{P} : \mathbf{Set} \rightarrow \mathbf{Set}, \mu : \mathcal{P}^2 \rightarrow \mathcal{P}, \eta : 1 \rightarrow \mathcal{P})$ has underlying functor $X \mapsto \mathcal{P}(X)$, multiplication $\mu_X = \bigcup : \mathcal{P}^2(X) \rightarrow \mathcal{P}(X)$, and unit $\eta_X : X \rightarrow \mathcal{P}(X) : x \mapsto \{x\}$.

The non-empty powerset monad \mathcal{P}^+ is defined similarly, but with underlying functor taking only the non-empty subsets.

Recall that for monads T and S on the same category, the composite of the underlying endofunctors $S \circ T$ only has monad structure if we additionally have the structure of a distributive law $\lambda : T \circ S \rightarrow S \circ T$.

Definition 2.3.3. A distributive law from a monad (T, μ^T, η^T) to a monad (S, μ^S, η^S) is a natural transformation $\lambda : T \circ S \rightarrow S \circ T$ satisfying the equations $(T\eta^S) \circ \lambda = \eta^S T$, $(\eta^T S) \circ \lambda = S\eta^T$, $(\lambda S) \circ (S\lambda) \circ (\mu^S T) = (T\mu^S) \circ \lambda$, $(T\lambda) \circ (\lambda T) \circ (S\mu^T) = (\mu^T S) \circ \lambda$. These equations can be rendered as intuitive equations of string diagrams, as in Goy [40].

Definition 2.3.4. A monoidal monad is a monad (T, η, μ) on a monoidal category (\mathbb{C}, \otimes, I) such that $T : \mathbb{C} \rightarrow \mathbb{C}$ has a lax monoidal structure and η, μ are monoidal natural transformations with respect to it. Explicitly, we ask for morphisms

$$\begin{aligned} \epsilon : I &\rightarrow TI \\ \nabla_{C,C'} : TC \otimes TC' &\rightarrow T(C \otimes C') \end{aligned}$$

such that

$$\begin{aligned} \epsilon &= \eta_I \\ (\eta_C \otimes \eta_{C'}) \circ \nabla_{C,C'} &= \eta_{C \otimes C'}, \text{ and} \\ (\mu_C \otimes \mu_{C'}) \circ \nabla_{C,C'} &= \nabla_{TC,TC'} \circ T(\nabla_{C,C'}) \circ \mu_{C \otimes C'}. \end{aligned}$$

Proposition 2.3.5. *The powerset, non-empty powerset and maybe monads are monoidal monads, considering \mathbf{Set} with its cartesian monoidal structure.*

Proof. We describe the structure maps: verification that they satisfy the equations of Definition 2.3.4 is straightforward. For the powerset and non-empty powerset monads, $\epsilon : 1 \rightarrow \mathcal{P}^{(+)}(1)$ picks out the set $\{\bullet\}$, and ∇ is the cartesian product of sets. For the maybe monad, $\epsilon : 1 \rightarrow 1 + \perp$ is the left injection, and ∇ maps the pair x, y to (x, y) only when both $x \neq \perp$ and $y \neq \perp$, and to \perp otherwise. \square

The monoidal structure of a monoidal monad lifts to its Kleisli category, which is where the computations of our automata will run, in Section 3.5.

Proposition 2.3.6 (Day, §4 [23], Power and Robinson, Corollary 4.3 [74]). *Let T be a monoidal monad on a symmetric monoidal category \mathcal{C} . Then the Kleisli category $\text{kl}(T)$ has a symmetric monoidal structure.*

Proof. Define the monoidal product on objects by that in \mathcal{C} , and on morphisms $f : X \rightarrow TA$ and $g : Y \rightarrow TB$ by $X \otimes Y \xrightarrow{f \otimes g} TA \otimes TB \xrightarrow{\nabla} T(A \otimes B)$. Symmetries are given by the morphisms $\sigma_{X,Y} ; \eta_{Y \otimes X}$ in \mathcal{C} . In particular, when \mathcal{C} is symmetric strict monoidal, one easily checks that this data satisfies the axioms of symmetric strict monoidal categories. \square

Definition 2.3.7 (cf. Wolff, Definition 2.1 [92]). A commutative distributive law between monoidal monads is a distributive law $\lambda : T \circ S \rightarrow S \circ T$ such that the following family of diagrams commutes

$$\begin{array}{ccccccc}
 & & TA \otimes TSB & \xrightarrow{\nabla^T} & T(A \otimes SB) & \xrightarrow{T(\eta^T \otimes 1)} & T(SA \otimes SB) & \xrightarrow{T(\nabla^S)} & TS(A \otimes B) \\
 & \nearrow^{1 \otimes \eta^T} & & & & & & & \downarrow \lambda \\
 TA \otimes SB & & & & & & & & \\
 & \searrow^{\eta^S \otimes 1} & STA \otimes SB & \xrightarrow{\nabla^S} & S(TA \otimes B) & \xrightarrow{S(1 \otimes \eta^B)} & S(TA \otimes TB) & \xrightarrow{S(\nabla^T)} & ST(A \otimes B)
 \end{array}$$

Remark 2.3.8. This definition is equivalent to the one given by Wolff [92, Definition 2.1], but expressed in terms of monoidal, rather than the equivalent commutative structure.

Theorem 2.3.9 (Wolff, Theorem 2.6 [92]). *If S and T are monoidal monads, and $\lambda : T \circ S \rightarrow S \circ T$ is a distributive law then the composite monad $S \circ T$ is monoidal if and only if λ is a commutative distributive law.*

Proposition 2.3.10. *The powerset monad may be decomposed as a distributive law of the non-empty powerset monad over the maybe monad. The component functions $\lambda_X : (\mathcal{P}^+ \circ \perp)X \rightarrow (\perp \circ \mathcal{P}^+)X$ map all subsets containing only \perp to \perp . This distributive law is moreover a commutative distributive law.*

2.4 Effectful categories and strong promonads

In Chapter 4, we shall introduce a refinement of the known string diagrams for *effectful categories*, introduced by Jeffrey [47, 46] and more recently taken up by Román [77].

Effectful categories, also known as *generalized Freyd categories* [56, 37], refine monoidal categories in two ways. Firstly, the *interchange* axiom is no longer required, and secondly they are equipped with a specified subcategory of central morphisms.

That interchange no longer holds globally in an effectful category means that the string diagrams of Figure 2.6 no longer have a univocal interpretation in a effectful category.



Figure 2.6: These string diagrams are no longer equal in an effectful category. This makes effectful categories suitable for the semantics of programming languages, where the order of *effectful* processes (such as printing) matters.

We recall the definitions of (symmetric) strict premonoidal categories and their functors. For more details, see Power and Robinson, or Román [74, 77].

Definition 2.4.1. A strict premonoidal category is a category \mathcal{C} equipped with:

- for each pair of objects $A, B \in \mathcal{C}$ an object $A \otimes B$,
- for each object $A \in \mathcal{C}$ a functor $A \triangleleft -$ (“left-whiskering with A ”) whose action on objects sends B to $A \otimes B$,
- for each object $A \in \mathcal{C}$ a functor $- \triangleright A$ (“right-whiskering with A ”) whose action on objects sends B to $B \otimes A$, and
- a unit object I ,

such that, for all morphisms f and objects A, B, C the following equations hold:

- $(A \triangleleft f) \triangleright B = A \triangleleft (f \triangleright B)$,
- $(A \otimes B) \triangleleft f = A \triangleleft (B \triangleleft f)$,
- $f \triangleright (A \otimes B) = (f \triangleright A) \triangleright B$,
- $(I \triangleleft f) = f = (f \triangleright I)$,
- $I \otimes A = A = A \otimes I$, and
- $A \otimes (B \otimes C) = (A \otimes B) \otimes C$.

Definition 2.4.2. A morphism $f : A \rightarrow B$ in a premonoidal category is central if and only if for every morphism $g : C \rightarrow D$,

$$(A \triangleleft g) \circledast (f \triangleright D) = (f \triangleright C) \circledast (B \triangleleft g), \text{ and}$$

$$(g \triangleright A) \circledast (D \triangleleft f) = (C \triangleleft f) \circledast (g \triangleright B).$$

Definition 2.4.3. The center of a premonoidal category is the wide subcategory of central morphisms.

Definition 2.4.4. A symmetric strict premonoidal category is a strict premonoidal category additionally equipped with a symmetry in the sense of Definition 2.1.4, such that all the components $\sigma_{A,B}$ are central.

Definition 2.4.5. A *strict premonoidal functor* $F : \mathbb{X} \rightarrow \mathbb{Y}$ between (strict) premonoidal categories is a functor that is a monoid homomorphism on objects, i.e. $F(A \otimes B) = F(A) \otimes F(B)$ and $F(I) = I$, and preserves whiskerings, i.e. $F(A \triangleleft f) = F(A) \triangleleft F(f)$ and $F(f \triangleright A) = F(f) \triangleright F(A)$.

Definition 2.4.6. A strict effectful category (over a strict monoidal category \mathbb{V}) is given by a strict premonoidal category \mathbb{C} and an identity on objects, strict premonoidal functor $\eta : \mathbb{V} \rightarrow \mathbb{C}$.

Note that the image of such a functor necessarily lies in the center of \mathbb{C} . We think of the category \mathbb{V} as a subcategory of the center of \mathbb{C} , and indeed every premonoidal category is always an effectful category over its center.

The free (symmetric) premonoidal category on a monoidal graph was described using string diagrams by Román [77]. The idea is simple: the string diagrams are the same as for props, but an extra string (a “runtime” or “global effect”) threads through each morphism of \mathbb{C} , preventing interchange, as in Figure 2.7.



Figure 2.7: In an effectful category, morphisms of \mathbb{C} are augmented by a string on a new object (dashed red). This prevents interchange (cf. Figure 2.6).

In particular, the runtime string appears only once in each vertical slice through the string diagram, reflecting the fact that premonoidal categories do not have a tensor product on morphisms.

Definition 2.4.7. A strict effectful functor $F : E \rightarrow E'$ between effectful categories $E : \mathbb{V} \rightarrow \mathbb{C}$ and $E' : \mathbb{V} \rightarrow \mathbb{C}'$ over \mathbb{V} is a functor $F : \mathbb{C} \rightarrow \mathbb{C}'$ strictly preserving premonoidal structure and satisfying $E \circledast F = E'$.

Effectful categories are equivalent to *strong promonads*. We shall find it helpful to be able to switch between these two points of view in Chapter 4.

Definition 2.4.8. A profunctor $P : \mathbb{C} \nrightarrow \mathbb{D}$ is a functor $P : \mathbb{D}^{\text{op}} \times \mathbb{C} \rightarrow \text{Set}$.

Definition 2.4.9. A promonad (P, η, μ) on a category \mathbb{C} is a profunctor $P : \mathbb{C} \nrightarrow \mathbb{C}$, equipped with natural transformations $\eta : \text{Hom}_{\mathbb{C}} \rightarrow P$ and $\mu : P \circ P \rightarrow P$ satisfying associativity and unit laws analogous to those in a monad.

Definition 2.4.10. A promonad morphism $\alpha : (P : \mathbb{C} \nrightarrow \mathbb{C}, \eta^P, \mu^P) \rightarrow (Q : \mathbb{C} \nrightarrow \mathbb{C}, \eta^Q, \mu^Q)$ between promonads over \mathbb{C} is given by a natural transformation $\alpha : P \rightarrow Q$ such that the following diagrams, making α a monoid homomorphism, commute

$$\begin{array}{ccc}
 \text{Hom}_{\mathbb{C}} & \xrightarrow{\eta^P} & P \\
 & \searrow \eta^Q & \downarrow \alpha \\
 & & Q
 \end{array}
 \qquad
 \begin{array}{ccc}
 P \circ P & \xrightarrow{\alpha \star \alpha} & Q \circ Q \\
 \mu^P \downarrow & & \downarrow \mu^Q \\
 P & \xrightarrow{\alpha} & Q
 \end{array}
 \tag{2.1}$$

Definition 2.4.11. The Kleisli category $\text{kl}(P)$ of a promonad $(P : \mathbb{C} \nrightarrow \mathbb{C}, \eta, \mu)$ has

- objects, those of \mathbb{C} ,
- morphisms, $\text{kl}(P)(A; B) := P(A; B)$,
- identities given by $\eta_{A,A}(\text{id}_A)$,
- composition given by $\mu_{A,C} : \int^{B \in \mathbb{C}} P(A; B) \times P(B; C) \rightarrow P(A; C)$.

Definition 2.4.12. An identity-on-objects functor is a family of functions $F_{x,y}$ between categories \mathbb{C} and \mathbb{D} having the same family of objects,

$$F_{x,y} : \mathbb{C}(x, y) \rightarrow \mathbb{D}(x, y),$$

preserving identities and composition.

Remark 2.4.13. Note that an identity-on-objects functor is not a functor with a special property since in general we do not have the requisite notion of object equality. Rather, it is a collection of data that induces a functor.

It is folklore that promonads on a category correspond to the same data as identity-on-objects functors out of that category (*cf.* Román [77, Theorem 3.9]). More than this, we have an isomorphism of categories, as follows.

Proposition 2.4.14. *The category of promonads on a category \mathbb{V} and promonad morphisms is isomorphic to the category of identity-on-objects functors out of \mathbb{V} and commuting triangles.*

Proof. We first establish a bijection on objects. Let (P, η, μ) be a promonad on a category \mathbb{V} , then there is an identity-on-objects functor $\mathbb{V} \rightarrow \mathbf{kl}(T)$ whose action on hom-sets is given by the components $\eta_{A,B} : \mathbb{V}(A; B) \rightarrow \mathbf{kl}(T)(A; B)$. This is functorial by the definition of $\mathbf{kl}(T)$ (Definition 2.4.11).

Conversely, let $F : \mathbb{V} \rightarrow \mathbb{C}$ be an identity-on-objects functor. Then $F^* \circledast F_* : \mathbb{V} \rightarrow \mathbb{V}$ is a profunctor, where $F_*(c; v) := \mathbb{C}(v; Fc)$ and $F^*(v; c) := \mathbb{C}(Fc; v)$ are the (co)representable profunctors determined by F , giving $F^* \circledast F_*(v; v') = \mathbb{C}(Fv; Fv')$. The components of the unit are given by the action of F on hom-sets, and multiplication is given by composition in \mathbb{C} .

Let $\alpha : (P, \eta^P, \mu^P) \rightarrow (Q, \eta^Q, \mu^Q)$ be a promonad morphism. Define the assignment $a : \mathbf{kl}(P) \rightarrow \mathbf{kl}(Q)$ to be identity on objects and act as the components of α on hom-sets – functoriality of this assignment follows from the equations making α a promonad morphism.

Conversely, let

$$\begin{array}{ccc} \mathbb{V} & \xrightarrow{F} & \mathbb{D} \\ & \searrow G & \downarrow A \\ & & \mathbb{E} \end{array}$$

be a commutative triangle of identity-on-objects functors. F and G define two promonads on \mathbb{V} as above. Then the action of A on hom-sets defines the components of a natural transformation $\alpha : F^* \circledast F_* \rightarrow G^* \circledast G_*$, which is moreover a morphism of promonads. It is clear that these processes define bijections on objects and morphisms and hence an isomorphism of categories. \square

Definition 2.4.15. A promonad $(P : \mathbb{C} \rightarrow \mathbb{C}, \eta, \mu)$ on a monoidal category \mathbb{C} is left-strong if the profunctor P is equipped with a family of functions

$$\ell_{u,c,d} : P(c; d) \rightarrow P(u \otimes c; u \otimes d)$$

natural in c and d and dinatural in u , and such that the strengths for $u = I$ and $u = a \otimes b$ are coherent with the unitors and associators of \mathbb{C} .

Similarly, a right-strong promonad is equipped with a family of functions

$$r_{u,c,d} : P(c; d) \rightarrow P(c \otimes u; d \otimes u)$$

satisfying the analogous conditions.

Definition 2.4.16. A strong promonad (also known as *bistrong* promonad) is a compatibly left- and right-strong promonad in the sense that the following family of

diagrams commutes, where a is the associator of the underlying monoidal category.

$$\begin{array}{ccc}
 & P(u \otimes c; u \otimes d) & \xrightarrow{r_v} P((u \otimes c) \otimes v; (u \otimes d) \otimes v) \\
 \ell_u \nearrow & & \downarrow P(a^{-1}, a) \\
 P(c; d) & & \\
 r_v \searrow & & \\
 & P(c \otimes v; d \otimes v) & \xrightarrow{\ell_u} P(u \otimes (c \otimes v); u \otimes (d \otimes v))
 \end{array}$$

Definition 2.4.17. A morphism of strong promonads is a morphism of promonads $\alpha : (P : \mathbb{C} \rightarrow \mathbb{C}, \eta^P, \mu^P) \rightarrow (Q : \mathbb{C} \rightarrow \mathbb{C}, \eta^Q, \mu^Q)$ which is moreover compatible with the strengths in the sense that the following family of diagrams commutes

$$\begin{array}{ccc}
 P(c; d) & \xrightarrow{\ell_u^P} P(u \otimes c; u \otimes d) & P(c; d) & \xrightarrow{r_u^P} P(c \otimes u; d \otimes u) \\
 \alpha_{c,d} \downarrow & & \downarrow \alpha_{c,d} & \\
 Q(c; d) & \xrightarrow{\ell_u^Q} Q(u \otimes c; u \otimes d) & Q(c; d) & \xrightarrow{r_u^Q} Q(c \otimes u; d \otimes u)
 \end{array}$$

Compatible left- and right-strengths provide exactly the data needed to define premonoidal whiskerings on the Kleisli category of a promonad. Conversely, premonoidal structure on the codomain of an identity-on-objects functor provides exactly the data required to equip the induced promonad with compatible left- and right-strengths.

Proposition 2.4.18. *The category of strong promonads on a strict monoidal category \mathbb{V} and morphisms of strong promonads is isomorphic to the category of strict effectful categories over \mathbb{V} and strict effectful functors.*

Proof. Following Proposition 2.4.14, it suffices to show that strength for a promonad exactly determines a premonoidal structure on the Kleisli category, that compatibility of morphisms of promonads with strengths corresponds to preservation of premonoidal structure, and vice-versa. Let $\text{kl}(T)$ be the Kleisli category of a strong promonad $T : \mathbb{V} \rightarrow \mathbb{V}$. We have a strictly associative and unital tensor product of objects given by that in \mathbb{V} . Define the left whiskering by the left strength $u \triangleleft (f : c \rightarrow d) := \ell_{u,c,d}(f)$, and similarly by the right strength for the right whiskering. Functoriality is guaranteed by the naturality of strengths, and similarly for the converse. Let $\alpha : T \rightarrow T'$ be a morphism of strong promonads over \mathbb{V} . Then the naturality squares of Definition 2.4.17 correspond exactly to the condition that the induced functor between Kleisli categories strictly preserves premonoidal structure, and similarly for the converse. \square

Chapter 3

Regular Languages of String Diagrams

In this chapter, we introduce languages of string diagrams, and in particular the regular class of these languages. In more detail,

- We recall the categorical formulation of regular languages. — Section 3.1
- We introduce *languages of string diagrams*. — Section 3.2
- We define *regular languages of string diagrams* via a notion of grammar. — Section 3.3
- We prove a pumping lemma for regular languages of string diagrams. — Section 3.4
- We introduce automata for regular languages of string diagrams. — Section 3.5
- We prove some closure properties of regular languages of string diagrams. — Section 3.6
- We introduce the syntactic monoidal category of a language. — Section 3.7
- We investigate when languages are deterministically recognizable. — Section 3.8
- We define a determinizable class of monoidal automata. — Section 3.9
- We show how Mazurkiewicz traces and asynchronous automata can be treated as monoidal languages and monoidal automata. — Section 3.10

The results of this chapter have been published in the papers [30, 31, 32].

3.1 Grammars as morphisms

Finite-state automata are often represented as finite *directed graphs*, whose vertices are *states* and edges *transitions*, with the edges being labelled by an alphabet.

Definition 3.1.1. A directed graph is given by a set E of edges, V of vertices, and two functions $s, t : E \rightrightarrows V$ giving the source and target of each edge. A graph is finite just when E and V are finite sets.

It is convenient to view a labelled graph not as a graph with extra structure, but rather as a particular *morphism* of graphs.

Definition 3.1.2. A morphism of (directed) graphs $\phi : (E, V, s, t) \rightarrow (E', V', s', t')$ is given by functions $\phi_e : E \rightarrow E', \phi_v : V \rightarrow V'$ such that $\phi_e \circ s' = s \circ \phi_v$ and $\phi_e \circ t' = t \circ \phi_v$.

Definition 3.1.3. A Σ -labelled graph is a morphism of graphs $\phi : G \rightarrow \Sigma$ where Σ is a graph with one vertex.

The idea is that such a morphism is trivial on vertices, and on edges assigns labels, namely, the edges of Σ . Walters [89, 90], following ideas of Lawvere [52], demonstrated that this idea elegantly captures classical notions of regular and context-free grammar – we return to the latter in Chapter 5.

Definition 3.1.4 (Walters [89, 90]). A regular grammar (Σ, ϕ, i, f) over an alphabet Σ is a Σ -labelled graph $\phi : G \rightarrow \Sigma$, where G and Σ are finite, along with distinguished vertices i, f of G .

To allow the possibility of labellings by the empty word ε , one can instead consider morphisms of *reflexive graphs* – we return to this in Section 3.3.1. Following Walters, we call such a morphism 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 will appear in the *domain* of a morphism. In other words, a grammar is *fibered* over the alphabet, whereas in an automaton the alphabet *indexes* transitions.

There are various advantages to this reframing of grammars as morphisms of graphs when it comes to language theory. Firstly, we can define the language of a grammar in terms of the associated functor between free categories. The idea is that a derivation in a regular grammar $G \rightarrow \Sigma$ corresponds to a path in G , and the accepted word is given by the labelling of the path. Paths in a directed graph are precisely the morphisms of the *free category* over that graph. 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 $\phi : G \rightarrow \Sigma$ gives rise to a functor between the associated free categories. Given two chosen vertices i, f in G , 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 $\mathcal{F}\phi$.

We can also use this representation of grammars to neatly describe operations on them, and so easily prove closure properties. For example, closure under morphisms of alphabets $\Sigma \rightarrow \Sigma'$ is witnessed immediately by post-composition. Certain “dual” closure properties are more easily addressed from the dual point of view of automata, which we introduce in Section 3.5.

It also suggests various generalizations of regular languages, 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, to which we return in Chapter 5. In this chapter, starting in Section 3.3, we shall consider the case of multi-input, multi-output graphs, which shall lead us to *regular languages of string diagrams*.

We might also consider replacing Σ by a more general graph, such as the graph underlying an arbitrary category. This has been explored in recent work by Melliès and Zeilberger [62, 63], giving rise to regular languages over arbitrary categories. We start by recalling some notions from this work, as they will be useful in our later discussions.

3.1.1 Regular grammars over arbitrary categories

Melliès and Zeilberger [63] introduce various refinements of Walters’ ideas, including extending Definition 3.1.4 to a notion of regular grammar of morphisms in an arbitrary category. Naively, we might expect this to take the form of a morphism of graphs $Q \rightarrow |\mathbb{C}|$, giving rise to a functor $\mathcal{F}Q \rightarrow \mathbb{C}$ into the underlying graph of a category \mathbb{C} . However, this is too general: we should also require that every production sequence (or run of an automaton), i.e. arrow in $\mathcal{F}Q$, whose labelling factors as sequence of morphisms in \mathbb{C} , itself factors uniquely as a sequence of productions labelled by the factors. This property is axiomatized by functors having the *unique lifting of factorizations* (ULF) property:

Definition 3.1.5 (Lawvere [53]). A functor $p : \mathbb{D} \rightarrow \mathbb{C}$ has the unique lifting of factorizations (ULF) property if for every morphism f in \mathbb{D} and factorization $p(f) = u \circ v$, there exists a unique pair of morphisms g, h in \mathbb{D} such that $f = g \circ h$, $p(g) = u$ and $p(h) = v$.

Moreover, we should ask for *finitary* functors, in the sense that the fibre over every object and every morphism is finite, generalizing the condition that Q is finite in the case of the one-object categories of the previous section.

Definition 3.1.6 (Melliès and Zeilberger [63]). A non-deterministic finite state automaton $(\mathbb{C}, \mathbb{Q}, p, i, f)$ over a category \mathbb{C} is a finitary, ULF functor $p : \mathbb{Q} \rightarrow \mathbb{C}$ and objects i, f of \mathbb{C} . The regular language of arrows of a grammar is the image of the hom-set of runs $\mathbb{Q}(i, f)$ under p .

Note that we would rather call this a regular grammar over \mathbb{C} , in line with the terminology introduced in the previous section. Finitary ULF functors over free categories are precisely those arising from morphisms of finite graphs.

Proposition 3.1.7 (Melliès and Zeilberger [63]). *A functor $\phi : Q \rightarrow \mathcal{F}\Sigma$ is a ULF functor if and only if $Q = \mathcal{F}G$ and $\phi = \mathcal{F}h$ for some morphism of graphs $h : G \rightarrow \Sigma$.*

Proof. The *if* direction follows by structural induction on the morphisms of $Q = \mathcal{F}G$. For the *only if* direction, we construct the graph G by taking the morphisms q of Q such that $\phi(q) = \sigma$ for some $\sigma \in \Sigma$ and easily verify the ULF property. \square

3.2 Languages of string diagrams

Just as a classical word language is a subset of a (finitely generated, free) monoid, a *language of string diagrams* or *monoidal language* is a subset of morphisms in a strict monoidal category, which will be a finitely generated free monoidal category in most of our examples. We shall investigate both *planar* and *symmetric* languages of string diagrams, corresponding to whether or not we consider our monoidal category to have symmetries: that is, whether or not we permit strings to cross. In Chapter 5, we consider languages in further “doctrines” of monoidal categories, such as cartesian and hypergraph monoidal categories.

Definition 3.2.1. A monoidal language or language of string diagrams is a subset L of some hom-set $\mathbb{C}(X; Y)$ in a strict monoidal category (\mathbb{C}, \otimes, I) .

In this chapter, we will exclusively be concerned with the case that \mathbb{C} is a finitely generated, free monoidal category – we have not yet found it necessary to investigate the more general notions in detail. However, we indicate in Appendix A how the definitions of Section 3.1.1 generalize to the setting of arbitrary (strict) monoidal categories. Let us introduce some terminology for the case of free monoidal categories.

Definition 3.2.2. A monoidal alphabet is a finite monoidal graph, considered as the alphabet for a monoidal language, and usually denoted by Γ .

In most of our examples, our monoidal alphabets will be single-sorted monoidal graphs, with the important exception of Mazurkiewicz traces as treated in Section 3.10.

Definition 3.2.3. A planar monoidal language L over a monoidal alphabet Γ is a subset L of some hom-set $\mathcal{F}_\otimes \Gamma(X; Y)$ in the free strict monoidal category generated by Γ .

Definition 3.2.4. A symmetric monoidal language L over a monoidal alphabet Γ is a subset L of some hom-set $\mathcal{F}_\chi\Gamma(X;Y)$ in the free symmetric strict monoidal category generated by Γ .

We shall speak simply of monoidal languages over Γ to refer to both the planar and symmetric cases, writing $\mathcal{F}\Gamma$ to indicate either the planar or symmetric case. In the following sections, we will see many examples, in particular examples of monoidal languages which we call *regular* (this chapter) and *context-free* (Chapter 5). Let us introduce some terminology to refer to languages that live in particular hom-sets.

Definition 3.2.5. A monoidal language is rooted if it is a subset of a hom-set from the monoidal unit, $L \subseteq \mathbb{C}(I; X)$, co-rooted if it is a subset of a hom-set into the monoidal unit, $L \subseteq \mathbb{C}(X; I)$, and scalar if it is a subset of the hom-set from the monoidal unit to itself, $L \subseteq \mathbb{C}(I; I)$.

3.3 Regular monoidal grammars

Regular monoidal grammars specify a class of monoidal languages analogous to regular languages. The idea is simple: a regular monoidal grammar is just like a regular grammar in the sense of Definition 3.1.4, except we replace graphs with monoidal graphs.

Definition 3.3.1. A regular monoidal grammar (M, Γ, Ψ, i, f) over a monoidal alphabet Γ is a morphism of monoidal graphs $\Psi : M \rightarrow \Gamma$ where M and Γ are finite monoidal graphs, and i (“start symbol”) and f (“end symbol”) are objects of M . We write $i \sqsubset a$ to mean $\Psi(i) = a$.

A regular monoidal grammar is a labelling of the generators of a finite monoidal graph M by the generators of Γ . We can represent regular monoidal grammars diagrammatically by drawing the monoidal graph M as above, but labelling each box b with $\Psi(b)$. The resulting diagram is not in general a monoidal graph, since it may contain boxes with the same label but different domain or codomain sorts. When Γ has many sorts, we will use colours to indicate them, and colour the states in M by their image under Ψ , as in Example 3.3.10. Consider the following monoidal alphabet (Figure 3.1) comprising four generators: white and gray boxes, along with start and end markers.

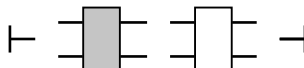


Figure 3.1: A monoidal alphabet Γ containing four generators.

A possible regular monoidal grammar over this alphabet is given in Figure 3.2, with initial and final objects given by ε . The monoidal graph M for this grammar contains eight generators, of types $\varepsilon \rightarrow H_1$; $\varepsilon \rightarrow V_1$; $H_1, V_1 \rightarrow V_0, H_0$; $H_0, V_0 \rightarrow V_0, H_0$; $H_0, V_1 \rightarrow V_1, H_1$; $H_1, V_0 \rightarrow V_1, H_1$; $H_1 \rightarrow \varepsilon$; $V_1 \rightarrow \varepsilon$. Their images under the morphism Ψ are as suggested in Figure 3.2.

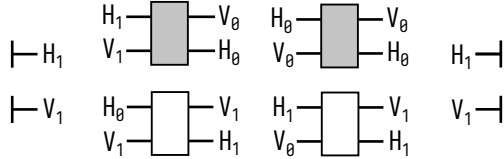


Figure 3.2: A regular monoidal grammar over the alphabet Γ of Figure 3.1, with initial and final objects ε .

Informally, the language defined by a regular monoidal grammar is the set of all string diagrams from the initial to the final object that can be built from the typed generators of the grammar. Every grammar determines both a symmetric and a planar monoidal language, depending on whether we allow our the strings in our diagrams to cross or not. For example, Figure 3.3 shows an element of the language determined by the grammar of Figure 3.2: this language contains the Sierpiński triangles of arbitrary iteration depth.

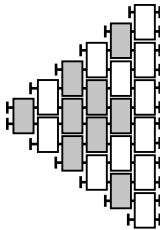


Figure 3.3: An element of the planar regular monoidal language determined by the grammar of Figure 3.2.

Formally, a regular monoidal grammar determines planar and symmetric monoidal languages as follows:

Definition 3.3.2. The planar monoidal language L of a regular monoidal grammar $(\Psi : M \rightarrow \Gamma, \Gamma, i, f)$ is the image under $\mathcal{F}_\otimes \Psi$ of the hom-set $\mathcal{F}_\otimes M(i, f)$, that is, $L := \mathcal{F}_\otimes \Psi[\mathcal{F}_\otimes M(i, f)] \subseteq \mathcal{F}_\otimes \Gamma(\Psi(i), \Psi(f))$. Similarly, by taking the free symmetric monoidal functor $\mathcal{F}_\chi \Psi$, we obtain the symmetric monoidal language of a grammar. We call the class of languages determined by regular monoidal grammars the (planar or symmetric) *regular monoidal languages*.

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. From

another perspective, we can think of a string diagram $s \in \mathcal{FM}$ as representing a specification for the construction of the string diagram $\mathcal{F}\Psi(s) \in \mathcal{F}\Gamma$ 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. Let us see some more examples.

Example 3.3.3 (Regular languages of words). Let Γ be a monoidal alphabet containing only generators of type $1 \rightarrow 1$. Then regular monoidal languages $L \subseteq \mathcal{F}_\otimes \Gamma(1, 1)$ over such an alphabet are exactly regular languages of words over the generators $1 \rightarrow 1$, and every regular language arises from a regular monoidal grammar of this form.

Example 3.3.4 (Regular languages of trees). Let Γ be a monoidal alphabet containing only generators of type $1 \rightarrow n$, for $n \geq 0$. Then (co-rooted) regular monoidal languages $L \subseteq \mathcal{F}_\otimes \Gamma(1, 0)$ over such an alphabet are exactly regular languages of trees. We investigate this and the previous example in more detail from the perspective of automata in Examples 3.5.4 and 3.5.5.

Example 3.3.5 (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 $1 \rightarrow 2$ and $2 \rightarrow 1$:

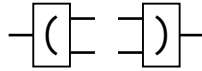


Figure 3.4: A monoidal alphabet of parentheses.

The regular monoidal grammar for balanced parentheses over this alphabet is given in Figure 3.5, where $i = f = A$. An example of a morphism in the planar language defined by this grammar is shown on the right in Figure 3.5.

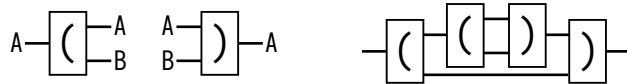


Figure 3.5: A regular monoidal grammar for balanced parentheses (left), and a morphism in the language (right).

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

The following two examples (Examples 3.3.6 and 3.3.7) are introduced in order to set up Example 3.3.8.

Example 3.3.6 (Brick walls). We can define a two-colour variant of the “brick wall” language introduced by Bossut [13] over the “brick” alphabet introduced in Figure 3.1, as in Figure 3.6. An example of a morphism in the regular planar monoidal language defined by this grammar is shown in Figure 3.7.

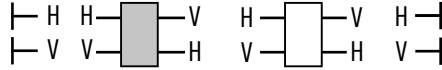


Figure 3.6: The grammar for two-coloured brick walls with $i = f = \varepsilon$.

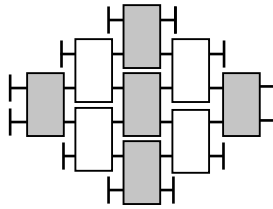


Figure 3.7: An element of the language determined by Figure 3.6.

Example 3.3.7 (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 Rothmund, Papadakis, and Winfree [81] in the context of self-assembly of DNA tiles. The output of each tile is the duplication of the XOR of the inputs.

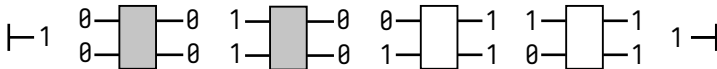
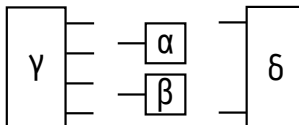


Figure 3.8: The XOR language grammar.

Example 3.3.8 (Sierpiński triangles). We already saw the grammar for Sierpiński triangles above. This example is inspired by the cellular automaton introduced by Rothmund, Papadakis, and Winfree [81], and arises as the intersection of the languages generated by the grammars of Examples 3.3.6 and 3.3.7. The corresponding monoidal grammar is given in Figure 3.2.

Example 3.3.9 (An inherently non-deterministic language). Consider the following monoidal alphabet:



The grammar over this alphabet in Figure 3.9 has a language whose elements with one connected component are exactly two (Figure 3.10). This grammar will serve as a running counterexample in Sections 3.8 and 3.9, as it defines a language that cannot be deterministically recognized (Figure 3.9). In particular it does not satisfy the property of *partial view closure*, a necessary condition for deterministic recognizability which we introduce in Section 3.8.

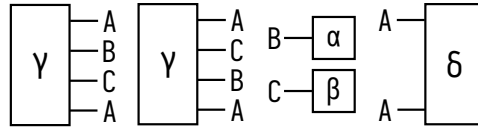


Figure 3.9: This grammar (with $i = f = \varepsilon$) is “non-deterministic”: there are two possible transitions from the empty word when encountering γ .

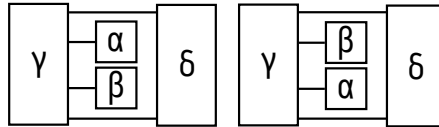


Figure 3.10: The connected string diagrams in the language of Figure 3.9.

Example 3.3.10 (Mazurkiewicz traces). In the above examples, we are interested in the planar languages of the grammars, and moreover all of the monoidal alphabets are single-sorted. An important example of grammars over multi-sorted alphabets and their *symmetric* languages is given by Mazurkiewicz traces [61]. We treat this example in detail in Section 3.10, but introduce it briefly now. The sorts are *locations*, and generators of the alphabet are atomic actions whose arities and coarities are equal to the set of locations at which the action acts. Regular monoidal grammars over alphabets of this form are regular languages of Mazurkiewicz traces. Figures 3.11 and 3.12 show a small example.

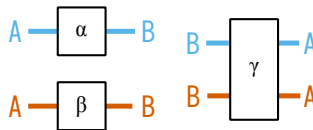


Figure 3.11: Grammar for a language of Mazurkiewicz traces.

Remark 3.3.11. The proper two dimensional generalization of regular grammars considered as *graphs over single-vertex graphs* (Definition 3.1.4) would appear at first to be *2-graphs over single-region 2-graphs*, where a *2-graph* comprises a set of regions, a set of arrows between any two regions, and a set of 2-arrows between any

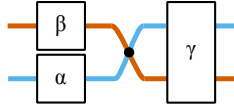


Figure 3.12: Elements of the symmetric monoidal language defined by Figure 3.11 are Mazurkiewicz traces. Here, α and β are independent actions, and so can commute, but both depend on γ .

two paths of arrows. Diagrammatically, a 2-graph is a monoidal graph in which the blank regions may be coloured, and monoidal graphs coincide with 2-graphs with a single colour of region, hence a 2-graph over a “one-colour” 2-graph defines a monoidal language. However, this does not provide any extra generality over monoidal graphs, since the types of the strings determine the types of the regions.

3.3.1 ε -productions

A regular grammar, in the sense of Definition 3.1.4, gives us the same class of languages whether we allow productions labelled by words over the alphabet, $w \in \Sigma^*$, or only labelled by the alphabet, $a \in \Sigma$. A key difference of the former case however is that such grammars allow the possibility of labellings by the empty word, or ε -productions. The proof idea is straightforward: any word-labelling may be factored into a sequence of letter-labellings by the introduction of new, intermediate states. Similarly, ε -labellings may be eliminated by the addition of appropriate letter-labelled transitions.

Allowing ε -productions is particularly useful in language theory – for example, for proving closure properties of languages. As such, we would like to consider regular monoidal grammars with ε -productions, but here we must be more careful. In order to add ε -productions, it might be tempting to consider morphisms of monoidal graphs of the form $M \rightarrow |\mathcal{F}\Gamma|$, and thus the possibility of labelling transitions by string diagrams over Γ . However, the languages obtained in this way, corresponding to arbitrary monoidal functors $\mathcal{F}M \rightarrow \mathcal{F}\Gamma$, are strictly more expressive. Let us call a grammar of the form $(\phi : M \rightarrow |\mathcal{F}\Gamma|, i, f)$ quasiregular.

Proposition 3.3.12. *Quasiregular monoidal grammars are strictly more expressive than regular monoidal grammars.*

Proof. Any regular monoidal grammar induces a quasiregular monoidal grammar via the inclusion $\Gamma \hookrightarrow |\mathcal{F}\Gamma|$. For the converse, we give an explicit counterexample. Consider the alphabet $\Gamma = \{a : \varepsilon \rightarrow \varepsilon, b : \varepsilon \rightarrow \varepsilon\}$, monoidal graph $M = \{t : \varepsilon \rightarrow \varepsilon\}$ and quasiregular grammar $(\phi : M \rightarrow |\mathcal{F}\Gamma|, \varepsilon, \varepsilon)$ with $\phi : t \mapsto a \otimes b$. It is clear that the language obtained is $\{(a \otimes b)^n \mid n \geq 0\}$. However, any regular monoidal grammar over Γ whose language contains $a \otimes b$ must contain a and b , since by

monoidal functoriality, the production of $a \otimes b$ has the form $p \otimes p'$ where p witnesses the production of a and p' the production of b . \square

There are good reasons not to take quasiregular grammars as defining our class of regular monoidal languages. We shall see later (Proposition 5.6.10) that quasiregular monoidal grammars are at least as expressive as context-free monoidal grammars. More fundamentally, just as regular grammars over free categories in the sense of Melliès and Zeilberger are exactly the grammars coming from morphisms of graphs (Proposition 3.1.7), and not arbitrary functors $\mathcal{F}G \rightarrow \mathcal{F}\Sigma$, arbitrary strict monoidal functors $\mathcal{F}M \rightarrow \mathcal{F}\Gamma$ need not satisfy the generalized unique lifting of factorizations property introduced in Appendix A. As witnessed by Lemma A.0.4, it is precisely those that come from morphisms of monoidal graphs that do.

Instead then, we shall incorporate ε -productions by upgrading monoidal graphs to certain *reflexive* monoidal graphs.

Definition 3.3.13. A reflexive monoidal graph is a monoidal graph $s, t : E \rightrightarrows V^*$ equipped with a function $\varepsilon : V^* \rightarrow E$, satisfying $\varepsilon \circ s = \varepsilon \circ t = 1_{V^*}$.

Definition 3.3.14. A morphism of reflexive monoidal graphs $q : M \rightarrow M'$ is a morphism of the underlying monoidal graphs additionally satisfying $\varepsilon_M \circ q_E = q_V^* \circ \varepsilon_{M'}$. Reflexive monoidal graphs and their morphisms form a category RMonGraph .

The function $\varepsilon : V^* \rightarrow E$ specifies an edge at each word which will act as an ε -production. A reflexive monoidal graph cannot be a finite monoidal graph, since it follows from the definition that E must be infinite. However, we can freely complete any monoidal graph to a reflexive one, and such completions of finite monoidal graphs will provide a suitable notion.

Proposition 3.3.15. *The forgetful functor $U : \text{RMonGraph} \rightarrow \text{MonGraph}$ has a left adjoint $F : \text{MonGraph} \rightarrow \text{RMonGraph}$.*

Proof. For a monoidal graph $G = (E, V, s, t)$, define

$$F(G) = (E + V^*, V, [s, \text{id}], [t, \text{id}], \varepsilon = \text{inr}).$$

For a morphism of monoidal graphs, define $F(\phi)$ to have component functions $F(\phi)_E := \phi_E + \phi_V^*$ and $F(\phi)_V := \phi_V$, which is easily verified to be a morphism of reflexive monoidal graphs. We need to show that morphisms of reflexive monoidal graphs $\phi : F(G) \rightarrow G'$ are in natural bijection with morphisms of monoidal graphs $\hat{\phi} : G \rightarrow U(G')$. The former comprises functions $\phi_E : E + V^* \rightarrow E'$ and $\phi_V : V \rightarrow V'$ satisfying the equations $(\varepsilon = \text{inr}) \circ \phi_E = \phi_V^* \circ \varepsilon'$, $\phi_E \circ s' = [s, \text{id}] \circ \phi_V^*$, and $\phi_E \circ t' = [t, \text{id}] \circ \phi_V^*$ while the latter comprises functions $\hat{\phi}_E : E \rightarrow E'$ and $\hat{\phi}_V : V \rightarrow V'$ satisfying equations $s \circ \hat{\phi}_V^* = \hat{\phi}_E \circ s'$ and $t \circ \hat{\phi}_V^* = \hat{\phi}_E \circ t'$. By the universal property of coproducts, ϕ_E is equivalently two functions $p : E \rightarrow E'$ and

$q : V^* \rightarrow E'$ satisfying $\text{inr} \circ \phi_\varepsilon = q$ and $\text{inl} \circ \phi_\varepsilon = p$. We have that p satisfies the equations for $\hat{\phi}_\varepsilon$, since $p \circ s' = \text{inl} \circ \phi_\varepsilon \circ s' = \text{inl} \circ [s, \text{id}] \circ \phi_V^* = s \circ \phi_V^*$ and similarly for the other equation. The converse reasoning is analogous. \square

Now instead of morphisms of finite monoidal graphs, we can ask for those morphisms of reflexive monoidal graphs which arise from this process of free reflexive completion.

Definition 3.3.16. A regular monoidal grammar with ε -productions is a morphism of reflexive monoidal graphs $\phi : M \rightarrow \Gamma$, in the image of the functor $\text{FinMonGraph} \xrightarrow{i} \text{MonGraph} \xrightarrow{F} \text{RMonGraph}$, along with two objects i, f of M .

Note that when $i = f$ in a regular monoidal grammar ϕ (without ε -productions), then the language of the grammar contains the identity morphism $\text{id}_{\phi(i)}$, and when $i \neq f$ but $\phi(i) = \phi(f)$, then the language cannot contain the identity. The only expressive difference of regular monoidal grammars with ε -productions is that in the latter case, their languages may indeed contain the identity.

Lemma 3.3.17. *Let $(\phi : M \rightarrow \Gamma, a, a)$ be a regular monoidal grammar with ε -productions and the same start and end symbol $i = f = a$. Then there exists a regular monoidal grammar without ε -productions having the same language.*

Proof. This is essentially the usual ε -elimination procedure. For an ε -production $p \xrightarrow{\varepsilon} q$, and rules $a \xrightarrow{t} p$, we take a rule $a \xrightarrow{t} q$, etc., see for example Sipser, [85, p. 109]. Note also that, by functoriality, the language of any regular monoidal grammar with $i = f = a$ contains $\text{id}_{\phi(a)}$. \square

Lemma 3.3.18. *Let $(\phi : M \rightarrow \Gamma, i, f)$ be a regular monoidal grammar with ε -productions, where $i \neq f$. Then there exists a regular monoidal grammar (without ε -productions) either with the same language, or differing only in that it does not contain $\text{id}_{\phi(i)}$.*

Proof. In case the grammar ϕ does not actually make use of ε -productions, it is clear that we can simply forget reflexiveness. Otherwise, we can apply the usual strategy for ε -elimination, except for the case that we have $t : i \rightarrow f$ in M such that $\phi(t)$ is some $b \rightarrow b$ in the image of ε_Γ , i.e. a ε -production from i to f . \square

In the following, when speaking of regular monoidal grammars, we shall mean regular monoidal grammars with ε -productions, unless indicated otherwise.

Note that for grammars over arbitrary categories as considered by Melliès and Zeilberger (Section 3.1.1), accounting for ε -productions remains an open question, since the ULF property implies that the only morphisms over identities are identities.

3.4 Monoidal pumping lemma

The classical pumping lemmas for regular languages of words and trees are useful for proving certain languages to be non-regular. For regular monoidal languages, we can prove a pumping lemma which relies on an extra parameter related to the *width* of factorizations of elements of the language. For this section, we consider single-sorted monoidal alphabets, but the ideas could be adapted to arbitrary monoidal alphabets.

Theorem 3.4.1 (Monoidal pumping lemma). *Let L be a regular monoidal language. Then $\forall k \in \mathbb{N}^+, \exists n > 0$ such that for any $\boxed{\gamma} \in L$ which may be factorized (as follows) into $m \geq n$ morphisms with boundaries (k_{i-1}, k_i) of width $1 \leq k_i \leq k$ and such that no $\boxed{\gamma_i}$ is an identity morphism:*

$$k_0 \left\{ \boxed{\gamma_0} \right\} k_1 \dots k_{i-1} \left\{ \boxed{\gamma_i} \right\} k_i \dots k_{m-1} \left\{ \boxed{\gamma_{m-1}} \right\} k_m$$

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

$$\boxed{\alpha} \boxed{\ell} \left(\boxed{\beta} \right)^p \boxed{\ell} \boxed{\delta} \in L, \forall p \geq 0$$

where $(\beta)^p$ in the diagram indicates sequential repetition of β , p times, and $\alpha = \gamma_0; \dots; \gamma_i$, $\beta = \gamma_{i+1}; \dots; \gamma_j$, and $\delta = \gamma_{j+1}; \dots; \gamma_m$.

Proof. Let L be the language of the grammar $\varphi : M \rightarrow \Gamma$. 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 $w \in L$, such that it has a factorization of the form above. The chosen n guarantees that some vector of sorts (S_1, \dots, S_{k_i}) where $S_i \in S_M$ must be repeated when generating w according to the grammar. That is, by the pigeonhole principle, we will have i, j, ℓ as required in the lemma. \square

Corollary 3.4.2 (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 $w'w''^pw''' \notin L$, then L is not regular monoidal.*

Observation 3.4.3. *This reduces to the usual pumping lemmas for words and trees [39, Proposition 5.2], when φ is a regular word or regular tree grammar (Examples 3.3.3 and 3.3.4), taking $k = 1$.*

Let us use the monoidal pumping lemma to prove that languages of *unbraids* on n strings, considered as planar 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 3.4.4 (Unbraid languages). The language of unbraids on $n \geq 2$ strings is a planar monoidal language $\text{Unbraid}_n \subseteq \mathcal{F}_\otimes \Gamma(n; n)$ over the monoidal alphabet Γ containing an under-braid and an over-braid generator (Figure 3.13). The elements



Figure 3.13: Monoidal alphabet for Unbraid_n .

of Unbraid_n are defined to be string diagrams $n \rightarrow n$ which could be unbraided to give two parallel strings, by planar isotopy, while keeping the ends fixed. For example, Figure 3.14 (left) is an element of Unbraid_2 but Figure 3.14 (right) is not.



Figure 3.14: (Left) Example of an element in Unbraid_2 : it could be untangled by planar deformations. (Right) A string diagram not in Unbraid_2 : we cannot uncross the strings using only planar moves while keeping the ends fixed.

To prove that the languages Unbraid_n are not regular monoidal, we will make use of the following regular planar monoidal language:

Definition 3.4.5 (Over-under language). $(O^+U^*)_n$ is the family of regular planar monoidal languages $(O^+U^*)_n \subseteq \mathcal{F}_\otimes \Gamma(n; n)$ over the braid alphabet (Figure 3.13) whose elements are one or more over-braidings of two strings followed by an arbitrary number of under-braidings of two strings, in parallel with $n-2$ identities. A grammar for this language is the following, with $i = ABB^{n-2}$, $f = CDB^{n-2}$,



Figure 3.15: A regular monoidal grammar for the monoidal languages $(O^+U^*)_n$.

Proposition 3.4.6. *The languages Unbraid_m ($m \geq 2$) are not regular monoidal.*

Proof. Consider the intersection $\text{Unbraid}_m \cap (O^+U^*)_m$. An element of $(O^+U^*)_m$ is an unbraid just when it comprises p under-braidings followed by p over-braidings

(in parallel with $n - 2$ identities), and thus these are the elements of the intersection, which we denote by $(O^p U^p)_m$. Figure 3.15 witnesses the regularity of $(O^+ U^*)_m$, and the intersection of regular monoidal languages is regular (Lemma 3.6.2), thus it will suffice to prove that $(O^p U^p)_m$ is not regular monoidal.

We use Corollary 3.4.2. Let $k = m$ and $n > 0$ be given. Take $p \geq n$ and let $\gamma \in (O^p U^p)_m$ be the element having p over-braidings (O) followed by p under-braidings (U), in parallel with $m - 2$ identities. Consider the decomposition $\gamma = (O \otimes \text{id})^p \circ (U \otimes \text{id})^p$, which is of width m and sequential size $\binom{2p+1}{2} \geq n$.

We have the following cases for the sub-decompositions $d : m \rightarrow m$: either it consists of $j - i$ O s, $j - i$ U s, or some number of O s followed by some number of U s. In the first two cases, pumping the section leads to a term with more O s than U s or vice-versa, and in the last case it will no longer be that all O s come before all U s. \square

In Chapter 5, we shall define the class of context-free monoidal languages, and we shall see that unbraids fall into this class.

3.5 Monoidal automata

Monoidal automata give an alternative specification of the class of regular monoidal languages: they are analogues of finite-state automata in which transitions now have multiple inputs and multiple outputs. Instead of transition functions $Q \xrightarrow{\Delta_a} \mathcal{P}(Q)$, we shall have transition functions $Q^n \xrightarrow{\Delta_\gamma} \mathcal{P}(Q^m)$, taking vectors of states to (sets of) vectors of states.

In this section, we introduce monoidal automata over monoidal graphs and show how these recognize monoidal languages. By specializing the shape of the monoidal graph, we recover classical word and tree automata, as well as the asynchronous automata of Zielonka [94] (Section 3.10).

Definition 3.5.1. A non-deterministic monoidal automaton comprises

- an input alphabet, given by a finite monoidal graph Γ ,
- a family of finite sets $\mathbf{Q} := \{Q_c\}_{c \in S_\Gamma}$ of states indexed by the sorts of Γ ,
- for each generator $\gamma : c_1 \dots c_n \rightarrow c'_1 \dots c'_m$ in Γ , a transition function

$$\Delta_\gamma : \prod_{i=0}^n Q_{c_i} \rightarrow \mathcal{P} \left(\prod_{j=0}^m Q_{c'_j} \right)$$

- initial and final words of states i, f in $(\bigcup_{c \in S_\Gamma} Q_c)^*$

Let us unpack this definition for the case in which Γ has a single sort, as it does in many of our examples (with the notable exception of asynchronous automata). In this case, in addition to the monoidal graph Γ , we have a finite set of states Q , and for every generator $\gamma : n \rightarrow m$ we have a transition function $\Delta_\gamma : Q^n \rightarrow \mathcal{P}(Q^m)$, and initial and final words over Q^* . Note that in particular, the initial and final words might be the empty word.

For classical NFAs, the assignment $a \in \Sigma \mapsto \Delta_a$ of letters to transition relations extends uniquely to a functor $\Sigma^* \rightarrow \text{Rel}$, by the free-forgetful adjunction between graphs and categories, giving the inductive extension of the transition structure from letters to runs over words. Monoidal automata extend to string diagrams in much the same way. Figure 3.16 provides some intuition for this, illustrating an accepting run over a string diagram in the monoidal automaton corresponding to the grammar of Example 3.3.5. The run starts with a *word* of states, whose subwords are modified by transitions corresponding to generators. Identity strings do not modify the states (in the absence of ε -transitions), and symmetries, if present, permute adjacent states.

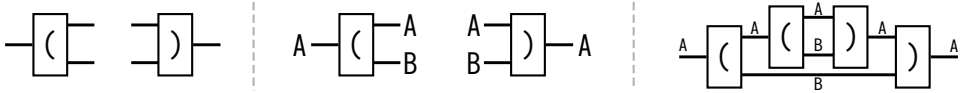


Figure 3.16: A run in the monoidal automaton for balanced parentheses.

A label next to a string indicates the state carried by that string. The accepted term is what is left if we erase these labels. In this example, the state vector undergoes the following transformations $A \rightarrow (A, B) \rightarrow (A, B, B) \rightarrow (A, B) \rightarrow A$. First, we define the strict monoidal category $\text{Rel}_{\mathbf{Q}}$ that will serve as our codomain.

Definition 3.5.2. For a family of finite sets $\mathbf{Q} := \{Q_c\}_{c \in S_\Gamma}$ indexed by the sorts of Γ then $\text{Rel}_{\mathbf{Q}}$ is the pro with:

- set of objects S_Γ^* ,
- morphisms $c_1 \dots c_n \rightarrow c'_1 \dots c'_m$ are functions $\prod_{i=1}^n Q_{c_i} \rightarrow \mathcal{P}(\prod_{j=1}^m Q_{c'_j})$,
- composition is the Kleisli composition of relations, i.e. $f \circ g := \mu \circ \mathcal{P}(g) \circ f$,
- \otimes is given on objects by concatenation, and
- on morphisms $f : \otimes_i c_i \rightarrow \otimes_j c'_j$ and $g : \otimes_k d_k \rightarrow \otimes_l d'_l$ by $f \otimes g := \nabla \circ (f \times g)$, where ∇ is the cartesian product of sets.

Moreover, $\text{Rel}_{\mathbf{Q}}$ can be equipped with symmetries, making it a prop:

- symmetries $\sigma : c_1 c_2 \rightarrow c_2 c_1$ are functions $Q_{c_1} \times Q_{c_2} \rightarrow \mathcal{P}(Q_{c_2} \times Q_{c_1}) : (q, q') \mapsto \{(q', q)\}$.

Note that a non-deterministic monoidal automaton amounts to a morphism of monoidal graphs $\Gamma \rightarrow \mathcal{U}(\text{Rel}_{\mathbf{Q}})$. In order to allow ε -transitions, we can just as well consider a morphism of reflexive monoidal graphs. The adjunctions $\mathcal{F} \dashv \mathcal{U}$ of Proposition 2.1.11 imply that this data extends uniquely to a strict monoidal functor $\mathcal{F}_{\otimes} \Gamma \rightarrow \text{Rel}_{\mathbf{Q}}$ and also to a symmetric strict monoidal functor $\mathcal{F}_{\chi} \Gamma \rightarrow \text{Rel}_{\mathbf{Q}}$, and indeed that such strict monoidal functors are in natural bijection with non-deterministic monoidal automata. These functors map a string diagram (with or without symmetries, respectively) to a relation. When this relation relates the initial word to the final word, the string diagram is *accepted*:

Definition 3.5.3. Let $\Delta : \mathcal{F}\Gamma \rightarrow \text{Rel}_{\mathbf{Q}}$ be a non-deterministic monoidal automaton with initial and final states $i, f \in (\cup_c Q_c)^*$. Then the symmetric monoidal language accepted by Δ is the set of morphisms $\mathcal{L}(\Delta) := \{\alpha \in \mathcal{F}\Gamma \mid f \in \Delta(\alpha)(i)\}$.

Classical non-deterministic *finite-state automata* and *tree automata* can be seen as non-deterministic monoidal automata over alphabets of particular shapes (Examples 3.5.4 and 3.5.5). This is also true of *asynchronous automata*, which we treat in detail in Section 3.10.

Example 3.5.4. Non-deterministic monoidal automata over monoidal alphabets having only generators of arity and coarity 1 (word monoidal alphabets) correspond to classical non-deterministic finite state automata.

Let an NFA $A = (Q, \Sigma, \Delta, i, F)$ be given. We build a monoidal automaton as follows. Form the monoidal alphabet Σ' by taking generators $\boxed{\sigma}$ for each $\sigma \in \Sigma$. For each $\boxed{\sigma}$, take the transition function $\Delta_{\sigma} := \Delta(\sigma, -) : Q \rightarrow \mathcal{P}(Q)$. Finally, we take ε -transitions from every $f \in F$ to a fresh final state f' – the converse construction is analogous.

There is another way to encode NFAs as monoidal automata that avoids the use of ε -transitions. If, in addition to the generators $\boxed{\sigma}$, we take generators \vdash and \dashv that mark the start and end of a word, then the corresponding relations are exactly functions $1 \rightarrow \mathcal{P}(Q)$ picking a set of initial states, and $Q \rightarrow \mathcal{P}(1)$ defining a set of final states.

Example 3.5.5. Non-deterministic finite tree automata can be given either as *bottom-up* or *top-down* recognizers, 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 $(\Sigma, r : \Sigma \rightarrow \mathbb{N})$, a set of final states $F \subseteq Q$, and for each $\sigma \in \Sigma$ a transition function $\Delta_{\sigma} : Q^{r(\sigma)} \rightarrow \mathcal{P}(Q)$. A non-deterministic top-down tree automaton, instead, has a set of initial states $I \subseteq Q$ and transition functions $\Delta_{\sigma} : Q \rightarrow \mathcal{P}(Q^{r(\sigma)})$. We can recover these as non-deterministic monoidal automata over *tree monoidal alphabets*:

Definition 3.5.6. A top-down tree monoidal alphabet is a monoidal alphabet having only generators of arity 1 (and arbitrary coarities ≥ 0), $\boxed{\sigma} \prec$. Analogously, a

bottom-up tree monoidal alphabet is a monoidal alphabet having only generators of coarity 1 (and arbitrary arities ≥ 0).

As with the previous example, we have:

Observation 3.5.7. *Bottom-up tree automata are exactly non-deterministic monoidal automata over bottom-up tree monoidal alphabets, and likewise for top-down tree automata.*

There is an evident correspondence between non-deterministic monoidal automata and regular monoidal grammars. The graphical representation of a grammar (such as Figure 3.16) 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 3.5.8. *Given a regular monoidal grammar $(\Psi : M \rightarrow \Gamma, i, f)$, define a monoidal automaton with $Q = S_M$, $w(\Delta_\gamma)w' \Leftrightarrow \exists \sigma \in \Psi^{-1}(\gamma)$ such that $s(\sigma) = w, t(\sigma) = w'$ and initial and final states i, f . Conversely given a monoidal automaton (Q, Δ_Γ) , define a regular monoidal grammar with $S_M = Q$ and take an edge $w \rightarrow w'$ over $\gamma \Leftrightarrow w(\Delta_\gamma)w'$. This correspondence of grammars and automata preserves the recognized language.*

We can further abstract our definition of monoidal automaton by noting that Rel_Q is exactly the *endomorphism pro*(p) for the family $\{Q_c\}$ in the Kleisli category of the powerset monad.

Definition 3.5.9. Given a family $\{Q_c\}_{c \in X}$ of objects in a monoidal category \mathbb{C} indexed by a set X , the endomorphism pro $\text{End}_X(\mathbb{C})$ for this family has, as objects, sequences of elements of X , and morphisms

$$\text{End}_X(\mathbb{C})(c_1, \dots, c_n; d_1, \dots, d_m) := \mathbb{C} \left(\bigotimes c_i, \bigotimes d_j \right).$$

If moreover \mathbb{C} is a symmetric monoidal category, $\text{End}_X(\mathbb{C})$ is a prop, with symmetries corresponding to those of \mathbb{C} .

Taking the endomorphism pro corresponding to the Kleisli category of any monoidal monad and a family of objects in it, gives a suitable semantic universe for interpreting computations of automata over string diagrams. For example, taking the maybe monad instead of the powerset monad, we obtain deterministic monoidal automata. Let us spell out the data of deterministic monoidal automata, and the definition of the endomorphism pro of partial functions to which they extend.

Definition 3.5.10. A deterministic monoidal automaton comprises

- an input alphabet, given by a finite monoidal graph Γ ,
- a family of finite sets $\mathbf{Q} := \{Q_c\}_{c \in S_\Gamma}$ of states indexed by the sorts of Γ ,

- for each generator $\gamma : c_1 \dots c_n \rightarrow c'_1 \dots c'_m$ in Γ , a transition function

$$\Delta_\gamma : \prod_{i=0}^n Q_{c_i} \rightarrow \left(\prod_{j=0}^m Q_{c'_j} \right) + \perp$$

- initial and final words of states i, f in $(\bigcup_{c \in S_\Gamma} Q_c)^*$

Deterministic monoidal automata amount to morphisms of monoidal graphs $\Gamma \rightarrow \mathcal{U}(\text{Par}_\mathbf{Q})$ where $\text{Par}_\mathbf{Q}$ is defined as follows.

Definition 3.5.11. For a family of finite sets $\mathbf{Q} := \{Q_c\}_{c \in S_\Gamma}$ indexed by the sorts of Γ then $\text{Par}_\mathbf{Q}$ is the pro with:

- set of objects S_Γ^* ,
- morphisms $c_1 \dots c_n \rightarrow c'_1 \dots c'_m$ functions $\prod_{i=1}^n Q_{c_i} \rightarrow (\prod_{j=1}^m Q_{c'_j}) + \perp$,
- composition is the Kleisli composition of partial functions, i.e. $f \circ g := \mu \circ (g + \perp) \circ f$,
- \otimes is given on objects by concatenation,
- and on morphisms $f : \otimes_i c_i \rightarrow \otimes_j c'_j$ and $g : \otimes_k d_k \rightarrow \otimes_l d'_l$ by $f \otimes g := \nabla \circ (f \times g)$, where ∇ is the monoidal multiplication of the maybe monad.

Moreover, $\text{Par}_\mathbf{Q}$ can be equipped with symmetries, making it a prop:

- symmetries $\sigma : c_1 c_2 \rightarrow c_2 c_1$ are functions $Q_{c_1} \times Q_{c_2} \rightarrow (Q_{c_2} \times Q_{c_1}) + \perp : (q, q') \mapsto (q', q)$.

Once again, there is a natural bijection between deterministic monoidal automata and strict monoidal functors $\mathcal{F}_\otimes \Gamma \rightarrow \text{Par}_\mathbf{Q}$, and also symmetric strict monoidal functors $\mathcal{F}_\chi \Gamma \rightarrow \text{Par}_\mathbf{Q}$, which determine the planar and symmetric monoidal languages of an automaton:

Definition 3.5.12. Let $\Delta : \mathcal{F}\Gamma \rightarrow \text{Par}_\mathbf{Q}$ be a deterministic monoidal automaton with initial and final states $i, f \in (\sqcup_c Q_c)^*$. Then the (symmetric) monoidal language accepted by Δ is the set of morphisms $\mathcal{L}(\Delta) := \{\alpha \in \mathcal{F}\Gamma \mid f \in \Delta(\alpha)(i)\}$.

3.6 Closure properties of regular monoidal languages

Regular languages are closed under operations such as union, concatenation and Kleene star. Indeed, these closure properties can be used to characterize the class of regular languages as such.

In this section, we record some closure properties of regular monoidal languages. In some proofs the use of grammars is expedient, whereas in others the use of automata is more natural. In this section, we work with single-sorted monoidal alphabets, but the proof ideas hold more generally.

Lemma 3.6.1 (Closure under union). *Let $L, L' \subseteq \mathcal{FT}(a, b)$ 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 $(\Psi : M \rightarrow \Gamma, i, f), (\Psi' : M' \rightarrow \Gamma, i', f')$ respectively, where $i, i' \sqsubset a$, and $f, f' \sqsubset b$. Let E be the grammar over Γ with fresh sorts $i'' \sqsubset a$ and $f'' \sqsubset b$ and four ε -labelled transitions $i'' \rightarrow i, i'' \rightarrow i'$ and $f \rightarrow f'', f' \rightarrow f''$.

Define the grammar $([\Psi, \Psi', E] : M + M' + E \rightarrow \Gamma, i'', f'')$, where the morphism is the cotupling of Ψ and Ψ' and E . Graphically, this is suspending the disjoint union of two grammars between appropriate ε -productions unifying their initial and final states, and it is clear that the language defined in this way is the union of the languages defined by the two grammars. \square

Lemma 3.6.2 (Closure under intersection). *Let $L, L' \subseteq \mathcal{FT}(a, b)$ be regular monoidal languages over Γ . Then $L \cap L'$ is a regular monoidal language over Γ .*

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

Remark 3.6.3. The Sierpiński triangle language (Example 3.3.8) is the intersection of the brick wall language (Example 3.3.6) and the XOR language (Example 3.3.7): this explains the origin of the states in the grammar shown in Example 3.3.8.

Lemma 3.6.4 (Closure under composition). *Let $L \subseteq \mathcal{FT}(a, b)$ and $L' \subseteq \mathcal{FT}(b, c)$ be regular monoidal languages over Γ . Then the language $L \cdot L' = \{f \circledast g \mid f \in L, g \in L'\}$ is a regular monoidal language.*

Proof. Let L and L' be given by the regular monoidal grammars $(\Psi : M \rightarrow \Gamma, i, f), (\Psi' : M' \rightarrow \Gamma, i', f')$ respectively. Similarly to Lemma 3.6.1, we can use an ε -production $f \xrightarrow{\varepsilon} i'$ to compose the grammars. \square

Lemma 3.6.5 (Closure under sequential Kleene star). *Let $L \subseteq \mathcal{FT}(a, a)$ be a regular monoidal language on an object a . Then the language $L^* = \{f \circledast \dots \circledast f \mid f \in L, n \geq 0\}$ is a regular monoidal language.*

Proof. Similarly to Lemma 3.6.4, if the boundaries of the grammar are i, f , we add an ε -production $f \xrightarrow{\varepsilon} i$ to the grammar. \square

Lemma 3.6.6 (Closure under images of alphabets). *Let $L \subseteq \mathcal{FT}(a, b)$ be a regular monoidal language over Γ , and $\Gamma \xrightarrow{h} \Gamma'$ be a morphism of monoidal alphabets. Then $(\mathcal{F}h)L \subseteq \mathcal{FT}(ha, hb)$ is a regular monoidal language over Γ' .*

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

Lemma 3.6.7 (Closure under preimages of functors). *Let $L \subseteq \mathcal{F}\Gamma(a, b)$ be a regular monoidal language over Γ , and $F : \mathcal{F}\Gamma' \rightarrow \mathcal{F}\Gamma$ be a monoidal functor. Then the inverse image of L , $F^{-1}L$, is a regular monoidal language over Γ' .*

Proof. Let $(Q, \{\Delta_\gamma\}_{\gamma \in \Gamma}, i, f)$ be a non-deterministic monoidal automaton recognizing L with inductive extension $\Delta : \mathcal{F}\Gamma \rightarrow \text{Rel}_Q$. Consider the monoidal automaton given by the composite $(\Delta \circ F : \mathcal{F}\Gamma' \rightarrow \text{Rel}_Q, i, f)$. We have $\mathcal{L}(\Delta \circ F) = F^{-1}\mathcal{L}(\Delta) = F^{-1}L$, 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 by Bojańczyk, Klin, and Salamanca [10], it is a *theorem* that the class of recognizable languages relative to a monad on Set is closed under complement. However, given that every scalar regular monoidal language $L \subseteq \mathbb{C}(I; I)$ contains the identity on the monoidal unit, we have that:

Observation 3.6.8. *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 by Bojańczyk, Klin, and Salamanca [10] to extend to monoidal categories, since these are not algebras for a monad on Set .

3.7 Syntactic monoidal category of a language

Every formal language of words $L \subseteq \Sigma^*$ gives rise to an invariant called its *syntactic monoid*. The study of the connection between syntactic properties of a language and algebraic properties of its syntactic monoid is the content of *algebraic language theory*. In this section, we lift this idea to the setting of monoidal languages. Let us first recall the basics of this theory – for more details, see for example Pin [72].

Definition 3.7.1 (Syntactic congruence). Given a language $L \subseteq \Sigma^*$, define the relation \sim_L on Σ^* by

$$w \sim_L w' := \forall x, y \in \Sigma^*, xwy \in L \Leftrightarrow xw'y \in L$$

In other words, two words are equated if there is no context that distinguishes them, in terms of membership of the language. It is easily verified that \sim_L is an

equivalence relation, and hence gives rise to the quotient Σ^*/\sim_L . Moreover, it is a congruence for the monoid operation of Σ^* .

Proposition 3.7.2 (Syntactic monoid). *Σ^*/\sim_L is a monoid with $[w] \cdot [v] := [w \cdot v]$ and unit $[\varepsilon]$.*

Proof. We show that the operation is well-defined. Let $w \sim_L w'$ and $v \sim_L v'$, and let $xwvy \in L$ for all $x, y \in \Sigma^*$. Then $xw'vy \in L$ by the first equivalence and $xw'v'y \in L$ by the second equivalence. The converse follows the bi-implications in reverse. Therefore $[w \cdot v] = [w' \cdot v']$. \square

Definition 3.7.3 (Recognition by a monoid). A monoid M recognizes a language $L \subseteq \Sigma^*$ just when there is a monoid homomorphism $\phi : \Sigma^* \rightarrow M$ and a subset $N \subseteq M$ such that $L = \phi^{-1}(N)$.

Proposition 3.7.4. *The quotient map $q : \Sigma^* \rightarrow \Sigma^*/\sim_L$ recognizes L , and every other surjective recognizer factors through it.*

Proof. We claim that $L = q^{-1}(q(L))$. One inclusion is trivial, we show the other, namely $q^{-1}(q(L)) \subseteq L$. Let $w \in q^{-1}(q(L))$, then $qw \in q(L)$ and there exists $w' \in L$ such that $qw = qw'$, i.e. $w \sim_L w'$. But then $w \in L$ by taking $x = y = \varepsilon$. Now let $p : \Sigma^* \rightarrow M$ be another surjective monoid homomorphism recognizing L via some subset $N \subseteq M$. Then $pw = pw' \Rightarrow w \sim_L w'$, and so (using the surjectivity of p) we have a function $r : M \rightarrow \Sigma^*/\sim_L$ such that $q = r \circ p$, which is easily checked to be a monoid homomorphism. \square

Theorem 3.7.5 (Myhill, Nerode [65, 66]). *A language $L \subseteq \Sigma^*$ is regular if and only if its syntactic monoid is finite.*

Proof. Let L be regular and hence recognized by a DFA D . Let $w, v \in \Sigma^*$ induce the same transition in D , that is, $\delta(i, w) = \delta(i, v)$. Then $w \sim_L v$ follows immediately, since D is deterministic. Therefore the number of equivalence classes of \sim_L must be at most the number of states of D and hence finite. Let L be a language with a finite syntactic monoid. Then the equivalence classes of its syntactic monoid give the states of a DFA with transitions $[q] \xrightarrow{w} [qw]$, initial state $[\varepsilon]$ and final states $[w]$ for each $w \in L$. It is clear this automaton accepts exactly L . \square

We can define a version of the syntactic congruence for monoidal languages, using the following notion of string diagram context, generalizing the contexts “ x_y ” for words.

Definition 3.7.6. A context $p \rightarrow q$ of type $(a_1 \otimes \dots \otimes a_n, b_1 \otimes \dots \otimes b_m)$ in a strict monoidal category \mathbb{C} where $a_i, b_j \in \mathbb{C}$, is a scalar string diagram $p \rightarrow q$ with a hole of type $(a_1 \otimes \dots \otimes a_n, b_1 \otimes \dots \otimes b_m)$, as in Figure 3.17.

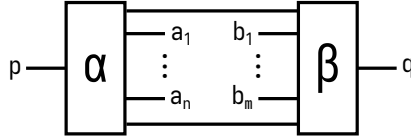


Figure 3.17: A context $p \rightarrow q$ of type $(a_1 \otimes \dots \otimes a_n, b_1 \otimes \dots \otimes b_m)$. α and β stand for arbitrary string diagrams arity and coarity p and q , respectively.

Given a context of type (a, b) in \mathbb{C} , we can fill the hole with a string diagram $\gamma : a \rightarrow b$ in \mathbb{C} . Write $C[\gamma]$ for the resulting morphism of \mathbb{C} . Note that the empty diagram is a context $0 \rightarrow 0$, the empty context of type $(0, 0)$, and more generally for every pair of types (p, q) there is an identity context $p \rightarrow q$ of type (p, q) in which $\alpha = \beta = \text{id}$. We shall treat contexts in more formal detail in Chapter 5, Section 5.3. Contexts allow us to define contextual equivalence of string diagrams. Let us first recall the definition of congruence on a monoidal category.

Definition 3.7.7. A congruence on a monoidal category \mathbb{C} is an equivalence relation $f \sim g$ on every hom-set $\mathbb{C}(x, x')$ compatible with composition and monoidal product, that is,

- $f \sim g \Rightarrow k \circ f \circ h \sim k \circ g \circ h$, whenever these composites are defined, and,
- $f \sim g \Rightarrow p \otimes f \otimes q \sim p \otimes g \otimes q$.

Proposition 3.7.8 (Syntactic congruence). *Given a monoidal language $L \subseteq \mathcal{FT}(p, q)$ there is a congruence \equiv_L defined as follows. Let γ, δ be morphisms in $\mathcal{FT}(r, s)$. Then $\gamma \equiv_L \delta$ whenever $C[\gamma] \in L \Leftrightarrow C[\delta] \in L$, for all contexts $C : p \rightarrow q$ in \mathcal{FT} of type (r, s) .*

Proof. Since \equiv_L is clearly an equivalence relation, we check its compatibility with composition and monoidal product. Let $\gamma \equiv_L \delta$ and $\epsilon : r' \rightarrow r$, $\zeta : s \rightarrow s'$ be morphisms, such that $C[\zeta \circ \gamma \circ \epsilon] \in L$ for an arbitrary context $C : p \rightarrow q$ of type (r', s') . By composing ζ and ϵ with the context, we obtain a new context $C' : p \rightarrow q$ of type (r, s) with $C'[\gamma] \in L$, and so $C'[\delta] \in L$ by assumption. But then $C[\zeta \circ \delta \circ \epsilon] \in L$, by moving ζ and ϵ out of the context, and a similar argument applies for the tensor product. \square

Remark 3.7.9. Compared with the definition of congruence given by Heindel [43], we only ask for equivalence relations on hom-sets, rather than the global set of morphisms. This is necessary, since otherwise syntactic congruences need not exist in general.

Proposition 3.7.10. *Given a congruence on a strict monoidal category \mathbb{C} , there is a quotient monoidal category \mathbb{C}/\sim with*

- objects those of \mathbb{C} ,
- hom-sets $(\mathbb{C}/\sim)(x, x') := \mathbb{C}(x, x')/\sim$, and
- composition, monoidal unit and monoidal product lifted from \mathbb{C} .

The quotient map $\mathbb{C} \rightarrow \mathbb{C}/\sim$ is a full, strict monoidal functor.

Definition 3.7.11. The syntactic monoidal category of a monoidal language $L \subseteq \mathcal{FT}(n, m)$ is the quotient monoidal category \mathcal{FT}/\equiv_L under the syntactic congruence. The quotient functor $S_L : \mathcal{FT} \rightarrow \mathcal{FT}/\equiv_L$ is the syntactic morphism of L .

Definition 3.7.12. A strict monoidal functor $R : \mathcal{FT} \rightarrow M$ recognizes a monoidal language $L \subseteq \mathcal{FT}(i, f)$ if there exists a subset $N \subseteq M(Ri, Rf)$ such that $L = R^{-1}(N)$.

Definition 3.7.13. A monoidal language $L \subseteq \mathcal{FT}(i, f)$ is finitely recognizable (or simply recognizable) if there exists a *locally finite* strict monoidal category M and a strict monoidal functor $R : \mathcal{FT} \rightarrow M$ recognizing L .

Proposition 3.7.14. The syntactic morphism $S : \mathcal{FT} \rightarrow \mathcal{FT}/\equiv_L$ of a monoidal language $L \subseteq \mathcal{FT}(n, m)$ recognizes that language.

Proof. We claim that $L = S^{-1}(S(L))$. The left-to-right inclusion is trivial, we show the converse inclusion. Let $u \in S^{-1}(S(L))$, so $Su \in S(L)$ and there exists $u' : n \rightarrow m \in L$ such that $Su = Su'$, that is, $u \equiv_L u'$. Taking the identity context $n \rightarrow m$ of type (n, m) , it follows that $u \in L$. \square

Theorem 3.7.15. If a monoidal language L is regular, then its syntactic monoidal category \mathcal{FT}/\equiv_L is locally finite (i.e. has finite hom-sets).

Proof. It suffices to exhibit a full strict monoidal functor into \mathcal{FT}/\equiv_L from a locally finite strict monoidal category. Let L be a regular monoidal language recognized by $(\Delta : \mathcal{FT} \rightarrow \text{Rel}_{\mathbf{Q}}, i, f)$. Δ induces a congruence \sim on \mathcal{FT} defined by $\alpha \sim \beta \Leftrightarrow \Delta(\alpha) = \Delta(\beta)$, which implies that \mathcal{FT}/\sim is locally finite, since $\text{Rel}_{\mathbf{Q}}$ is locally finite. Define the strict monoidal functor $\mathcal{FT}/\sim \rightarrow \mathcal{FT}/\equiv_L$ 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 the (monoidal) functoriality of Δ , $C[\beta] \in L$. Clearly it is full, so \mathcal{FT}/\equiv_L is locally finite. \square

Recall that in the case of languages of words, the converse holds true (Theorem 3.7.5), and the proof proceeds by taking the elements of the syntactic monoid as states for a DFA. Here however, it is not clear how to do something similar.

In the case of series-parallel pomset languages accepted by branching automata, the converse holds only when one restricts to languages whose terms have bounded width [57]. We conjecture that the same is true for monoidal languages.

Conjecture 3.7.16. Let $L \subseteq \mathcal{FT}(n, m)$ be a monoidal language whose elements have bounded width. Then L is regular monoidal if and only if its syntactic monoidal category is locally finite.

3.8 Deterministically recognizable languages

In this section, we provide a necessary condition for co-rooted regular monoidal languages to be deterministically recognizable. 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 [38, §2.11]. For string diagrams, we call the generalization of paths through a tree the *partial views* of a diagram. Our result then says that if a co-rooted regular monoidal language is deterministically recognizable then it is closed under grafting partial views (Theorem 3.8.11).

First, we briefly recall the machinery of (cartesian) *restriction categories* [19] that we shall use to define partial views. Restriction categories axiomatize the category of sets and partial functions, and provide us with a diagrammatic calculus for reasoning about deterministic recognition of regular monoidal languages.

3.8.1 Partial views via cartesian restriction categories

Definition 3.8.1 (Cockett and Lack, Theorem 5.2 [20]). A cartesian restriction category is a symmetric monoidal category in which every object is equipped with a commutative comonoid structure that is coherent, and for which the comultiplication is natural. We write $\dashv\bullet$ for the counit of the comonoid on an arbitrary object, $\dashv\lrcorner$ for the comultiplication of the comonoid on an arbitrary object, and \bowtie 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 hold (respectively: coassociativity, commutativity, and left unitality):

$$\begin{array}{c} \text{---} \end{array} \dashv\bullet = \begin{array}{c} \text{---} \end{array} \dashv\lrcorner \quad \begin{array}{c} \text{---} \end{array} \dashv\lrcorner = \begin{array}{c} \text{---} \end{array} \dashv\lrcorner \quad \begin{array}{c} \text{---} \end{array} \dashv\lrcorner = \begin{array}{c} \text{---} \end{array} \dashv\lrcorner \quad \text{---} = \begin{array}{c} \text{---} \end{array} \dashv\bullet \quad (3.1)$$

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

Then δ^* factors uniquely as:

$$\begin{array}{ccc} \mathcal{F}\Gamma & \xrightarrow{\delta^*} & \text{Par}_Q \\ & \searrow [-] & \nearrow \delta_{\downarrow}^* \\ & \mathcal{F}_{\downarrow}\Gamma & \end{array}$$

where $[-] : \mathcal{F}\Gamma \rightarrow \mathcal{F}_{\downarrow}\Gamma$ is the identity-on-objects quotient functor, sending string diagrams to their equivalence class in $\mathcal{F}_{\downarrow}\Gamma$.

We now introduce the notion of *partial view* of a string diagram. Recall that any restriction category is poset-enriched: $s \leq p$ if s is “less defined” than p , i.e. if s coincides with p on s ’s domain of definition [19, §2.1.4]. For string diagrams $n \rightarrow 0$, this amounts to the following equation, with special case $s = p \otimes s$ in case $n = 0$.

$$n \text{---} \boxed{s} = n \text{---} \begin{array}{c} \boxed{p} \\ \boxed{s} \end{array}$$

Definition 3.8.5. Let γ be a string diagram in $\mathcal{F}\Gamma(n, 0)$. We call a string diagram p in $\mathcal{F}_{\downarrow}\Gamma(n, 0)$ a partial view of γ if $[\gamma] \leq p$ in $\mathcal{F}_{\downarrow}\Gamma(n, 0)$.

A partial view represents the possible causal influence of parts of a diagram on generators appearing “later” in the diagram. For example, the five string diagrams of Figure 3.18 are partial views of the rightmost string diagram below, taken from the language introduced in Example 3.3.9. In particular, every diagram is a partial view of itself.

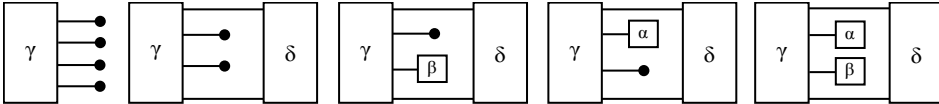


Figure 3.18: Partial views of the rightmost diagram.

In the case of tree languages, the partial views of a tree include in particular all of its *branches*, obtained by discarding (via \rightarrow) all but one child at each node.

Definition 3.8.6. The grafting $p_1 \times p_2$ of two string diagrams $p_1, p_2 \in \mathcal{F}_{\downarrow}\Gamma(n, 0)$ is the following string diagram in $\mathcal{F}_{\downarrow}\Gamma(n, 0)$, with special case $p_1 \times p_2 := p_1 \otimes p_2$ in case $n = 0$.

$$n \text{---} \boxed{p_1 \times p_2} := n \text{---} \begin{array}{c} \boxed{p_1} \\ \boxed{p_2} \end{array}$$

For example, the leftmost diagram in Figure 3.19 depicts the grafting of two partial views determined by the counterexample language of Example 3.3.9. By the equational theory of cartesian restriction categories (Definition 3.8.1), this is equal to the string diagrams in the center and on the right, where we first apply the naturality of $\dashv\vdash$ (for γ), then unitality (twice), then naturality of $\dashv\vdash$ (for δ).

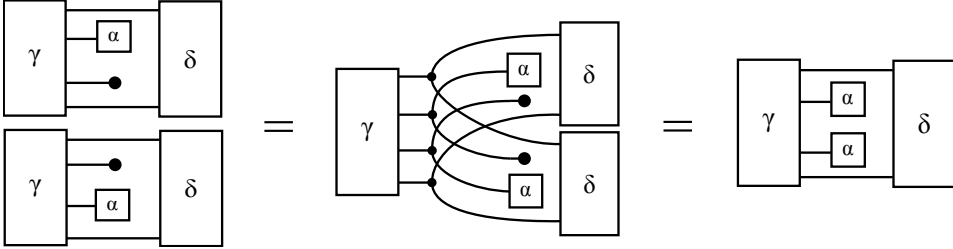


Figure 3.19: The grafting of two partial views of elements of the language from Example 3.3.9 (left). This is equal to a string diagram (right) in the image of the quotient functor $[-] : \mathcal{F}\Gamma \rightarrow \mathcal{F}_{\downarrow}\Gamma$ (i.e. expressible without using the cartesian restriction structure). This is an element of the partial view closure of the language.

Definition 3.8.7. The set $\mathcal{G}(L)$ of graftings of a monoidal language $L \subseteq \mathcal{F}\Gamma(n, 0)$ is the set of all graftings of all partial views of the elements of L ,

$$\mathcal{G}(L) := \{p_1 \times p_2 \mid \gamma, \gamma' \in L, [\gamma] \leq p_1, [\gamma'] \leq p_2\} \subseteq \mathcal{F}_{\downarrow}\Gamma(n, 0).$$

We are interested in those graftings of partial views that are equal to $[\gamma]$ for some γ in $\mathcal{F}\Gamma$. For example, consider the grafting of partial views arising from the language of Example 3.3.9 shown in Figure 3.19. The rightmost form of the diagram exhibits this morphism as being in the image of the quotient functor; its preimage under this functor is the same diagram in $\mathcal{F}\Gamma$. Crucially, this diagram is not in the original language: we say that the language is not closed under grafting partial views.

Definition 3.8.8. The partial view closure of a co-rooted monoidal language L is the preimage under $[-] : \mathcal{F}\Gamma \rightarrow \mathcal{F}_{\downarrow}\Gamma$ of the set $\mathcal{G}(L)$ of graftings of partial views of L (Definition 3.8.7). A monoidal language L is closed under partial views if and only if it equals its partial view closure, i.e. $[\mathcal{G}(L)]^{-1} = L$.

Lemma 3.8.9. Let $(Q, \Gamma, \delta, i, f)$ be a deterministic monoidal automaton. Then its cartesian restriction extension, $\delta_{\downarrow}^* : \mathcal{F}_{\downarrow}\Gamma \rightarrow \text{Par}_Q$ accepts the grafting $p \times p'$, if and only if it accepts p and p' .

Proof. Let δ_{\downarrow}^* accept p and p' . Using the monoidal functoriality of δ_{\downarrow}^* , a run over $p \times p'$ amounts to a run over p and p' in parallel, so $p \times p'$ is accepted. Conversely,

let $p \times p'$ be accepted. Then it follows from the monoidal functoriality of δ_{\downarrow}^* that p and p' must be accepted. \square

Lemma 3.8.10. *Let $(Q, \Gamma, \delta, i, f)$ be a deterministic monoidal automaton. Then its cartesian restriction extension, $\delta_{\downarrow}^* : \mathcal{F}_{\downarrow}\Gamma \rightarrow \text{Par}_Q$ accepts every partial view of elements of $\mathcal{L}(\delta)$.*

Proof. Let $\gamma \in \mathcal{L}(\delta)$, and let $p_{\gamma} \in \mathcal{F}_{\downarrow}\Gamma$ be a partial view of γ , that is, $p_{\gamma} \times [\gamma] = [\gamma]$. By Lemma 3.8.4, δ_{\downarrow}^* accepts $[\gamma]$, therefore δ_{\downarrow}^* also accepts p_{γ} by Lemma 3.8.9. \square

Theorem 3.8.11. *Let L be a regular monoidal language. If L is deterministically recognizable then it is closed under partial views.*

Proof. Let L be recognized by a deterministic monoidal automaton $\delta^* : \mathcal{F}\Gamma \rightarrow \text{Par}_Q$. This factorizes as $\delta^* = \delta_{\downarrow}^* \circ [-]$ as in Lemma 3.8.4. We wish to show that $L = [\mathcal{G}(L)]^{-1}$. We have that $L \subseteq [\mathcal{G}(L)]^{-1}$, by taking $p_1 = p_2 = [x]$ for any given $x \in L$. For the converse, it suffices to show that δ_{\downarrow}^* accepts every $[x] \in \mathcal{G}(L)$, where $[x] = p_1 \times p_2$ and p_1, p_2 are partial views of elements of L . By Lemma 3.8.10, δ_{\downarrow}^* accepts p_1 and p_2 , and so by Lemma 3.8.9 it accepts $[x]$. \square

3.9 Convex monoidal automata

In this section, we study a class of determinizable monoidal automata which we term *convex*, and introduce a powerset construction for them. The classical powerset construction is given conceptually by composition with the functor $\text{Rel} \rightarrow \text{Set}$, sending sets to their powersets and relations to their corresponding functions, right adjoint to the inclusion $\text{Set} \hookrightarrow \text{Rel}$. This right adjoint is only *lax monoidal* (with respect to the cartesian product monoidal structures on Set and FinRel), so this strategy does not extend to for monoidal automata, as we shall see below. For this section, we work with single-sorted monoidal automata for clarity, but this is not an essential restriction.

Both non-deterministic finite state automata for words and bottom-up trees can be determinized via a form of powerset construction. However, top-down tree automata cannot be determinized in general,

Proposition 3.9.1 (Gécseg and Steinby, §2.11 [38]). *If a deterministic top-down tree (monoidal) automaton accepts the trees $\sigma(x, y)$ and $\sigma(y, x)$ then it must accept $\sigma(x, x)$.*

Proof. This is clear from inspecting all the ways to annotate the morphisms $\sigma(x, y)$ and $\sigma(y, x)$ with states (i.e. checking possible runs in an automaton). \square

The problem is that a top-down tree automaton cannot synchronize its computations on different branches of an input tree. An immediate consequence is that

monoidal automata also cannot be determinized in general (Example 3.5.5), and so we cannot hope to obtain an analogue of the above functor: the obvious candidate is only lax monoidal, and so does not preserve the language. Nevertheless, there are interesting examples of deterministically recognizable monoidal languages that are not tree languages, such as the monoidal Dyck language (Example 3.3.5) and Sierpiński fractals (Example 3.3.8), and it is an intriguing theoretical challenge to characterize such languages. Towards this end, we describe a suitable subcategory of $\text{Rel}_{\mathcal{Q}}$ for which determinization is functorial, that of *convex relations*. Again, for simplicity we work in this section with automata over single-sorted signatures, but there is no essential obstacle to extending the ideas to the multi-sorted setting.

Definition 3.9.2. A relation $\Delta : Q^n \rightarrow \mathcal{P}(Q^m)$ is convex if there is a morphism Δ^* such that the following square commutes:

$$\begin{array}{ccc} (\mathcal{P}Q)^n & \xrightarrow{\Delta^*} & (\mathcal{P}Q)^m \\ \nabla_{\mathcal{P}} \downarrow & & \downarrow \nabla_{\mathcal{P}} \\ \mathcal{P}(Q^n) & \xrightarrow{\Delta^\#} & \mathcal{P}(Q^m) \end{array}$$

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

Remark 3.9.3. Such a lift Δ^* is not necessarily unique, since $\nabla_{\mathcal{P}}$ is not injective: any tuple containing the empty set is mapped to the empty set. Nevertheless, we shall be able to use these lifts to implement a well-defined determinization procedure in Lemma 3.9.8.

Example 3.9.4. The relation $\Delta_\gamma : Q^0 \rightarrow \mathcal{P}(Q^4)$ induced by the grammar in Example 3.3.9 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.

Definition 3.9.5 (Convex automaton). A non-deterministic monoidal automaton is convex just when its transition relations are convex.

Lemma 3.9.6. *Convex relations determine a monoidal sub-category $\text{CRel}_{\mathcal{Q}} \hookrightarrow \text{Rel}_{\mathcal{Q}}$.*

Proof. 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 $\Delta_\alpha : Q^a \rightarrow \mathcal{P}(Q^b)$, $\Delta_\beta : Q^b \rightarrow \mathcal{P}(Q^c)$, and take $(\Delta_\beta \diamond \Delta_\alpha)^* = \Delta_\beta^* \circ \Delta_\alpha^*$, where \diamond is composition in $\text{Kl}(\mathcal{P})$.

Consider the following diagram:

$$\begin{array}{ccccc}
 (\mathcal{P}Q)^a & \xrightarrow{\Delta_\alpha^*} & (\mathcal{P}Q)^b & \xrightarrow{\Delta_\beta^*} & (\mathcal{P}Q)^c \\
 \downarrow \nabla & & \downarrow \nabla & & \downarrow \nabla \\
 \mathcal{P}(Q^a) & \xrightarrow{\Delta_\alpha^\#} & \mathcal{P}(Q^b) & \xrightarrow{\Delta_\beta^\#} & \mathcal{P}(Q^c) \\
 \searrow \mathcal{P}(\Delta_\alpha) & & \nearrow \mu & & \searrow \mathcal{P}(\Delta_\beta) \\
 & & \mathcal{P}^2(Q^b) & & \mathcal{P}^2(Q^c) \\
 & & \searrow \mathcal{P}^2(\Delta_\beta) & & \nearrow \mathcal{P}(\mu) \\
 & & & & \mathcal{P}^3(Q^c)
 \end{array}$$

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:

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

by naturality of μ ,

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

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

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

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

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

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

Lemma 3.9.7. *The equation $\nabla_{\mathcal{P}} = \perp \nabla_{\mathcal{P}^+} \circ \nabla_{\perp}$ holds between the monoidal multiplications of the monads \mathcal{P} , \mathcal{P}^+ , and \perp .*

Proof. Since the distributive law of \mathcal{P}^+ over \perp is commutative (Proposition 2.3.5), this follows immediately from Wolff [92, Proposition 2.3]. \square

We can now give the powerset construction on convex automata. We use the non-empty powerset monad \mathcal{P}^+ to avoid duplication of the failure state, which is \emptyset in Rel_Q , but \perp in $\text{Par}_{\mathcal{P}^+(Q)}$:

Lemma 3.9.8. *For each set Q there is a strict monoidal functor $\mathcal{D}_Q : \text{CRel}_Q \rightarrow \text{Par}_{\mathcal{P}^+(Q)}$ which is identity on objects and acts as follows on morphisms:*

$$\begin{array}{c} \Delta_\alpha : Q^n \rightarrow \mathcal{P}(Q^m) \\ \Downarrow \\ \mathcal{P}^+(Q)^n \xrightarrow{\eta^n} (\perp \mathcal{P}^+(Q))^n \xrightarrow{\Delta_\alpha^*} (\perp \mathcal{P}^+(Q))^m \xrightarrow{\nabla_{\perp}} \perp(\mathcal{P}^+(Q))^m \end{array}$$

where we elide the isomorphisms $\mathcal{P}(Q)^n \cong (\perp \mathcal{P}^+(Q))^n$. \perp is the maybe monad, η is the unit of this monad, and ∇_{\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. 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 $\nabla_{\perp} \circ \Delta_\alpha^* = \nabla_{\perp} \circ \hat{\Delta}_\alpha^*$. From Lemma 3.9.7, $\nabla_{\mathcal{P}} = \perp \nabla_{\mathcal{P}^+} \circ \nabla_{\perp}$. Furthermore, $\perp \nabla_{\mathcal{P}^+}$ is injective, so our conclusion follows, using the definition of convexity: $\nabla_{\mathcal{P}} \circ \Delta_\alpha^* = \Delta_\alpha^\# \circ \nabla_{\mathcal{P}} = \nabla_{\mathcal{P}} \circ \hat{\Delta}_\alpha^*$.

We need to show that this mapping is a strict monoidal functor. It is clear that identities are preserved. It remains to show that composition and monoidal product are preserved. Let $\Delta_\alpha : Q^a \rightarrow \mathcal{P}(Q^b)$, $\Delta_\beta : Q^b \rightarrow \mathcal{P}(Q^c)$. We require $\mathcal{D}_Q(\Delta_\beta \diamond \Delta_\alpha) = \mathcal{D}_Q(\Delta_\beta) \diamond \mathcal{D}_Q(\Delta_\alpha)$. 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_Q .

$$\begin{array}{ccc} (\perp \mathcal{P}^+(Q))^b & \xrightarrow{\eta^b} & (\perp \perp \mathcal{P}^+(Q))^b \\ \nabla_{\perp} \downarrow & \searrow \eta & \downarrow \nabla_{\perp} \\ \perp \mathcal{P}^+(Q)^b & \xrightarrow{\perp \eta^b} & \perp(\perp \mathcal{P}^+(Q))^b \end{array}$$

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

Determinization of a convex automaton $(\Delta : \mathcal{FT} \rightarrow \mathbf{CRel}_Q, \vec{i}, \vec{f})$ is now given by post-composition with the functor $\mathcal{D}_Q : \mathbf{CRel}_Q \rightarrow \mathbf{Par}_{\mathcal{P}^+(Q)}$. The new initial state becomes $\{i_1\} \dots \{i_n\}$, and we add ε -transitions from every (F_1, \dots, F_m) where $f_i \in F_i$ to a fresh final state \vec{f}' .

Theorem 3.9.9. *Determinization of convex automata preserves the accepted language: let $(\Delta : \mathcal{FT} \rightarrow \mathbf{CRel}_Q, \vec{i}, \vec{f})$ be a convex automaton, then $\mathcal{L}(\Delta) = \mathcal{L}(\mathcal{D}_Q \circ \Delta)$, with the initial and final states described above.*

Proof. Let $\alpha \in \mathcal{L}(\Delta)$, i.e. $\vec{f} \in \Delta_\alpha(\vec{i})$. Then we must have $\Delta_\alpha^*(\{i_1\} \dots \{i_n\}) = F_1 \dots F_m$, with $f_i \in F_i$ and so $(\mathcal{D}_Q \circ \Delta)_\alpha(\vec{i}) = \vec{f}'$. Conversely let $\alpha \in \mathcal{L}(\mathcal{D}_Q \circ \Delta)$, i.e. $(\mathcal{D}_Q \circ \Delta)_\alpha(\{i_1\} \dots \{i_n\}) = f'$. Then we must have that $\Delta_\alpha^*(\{i_1\} \dots \{i_n\}) = \{f_1\} \dots \{f_m\}$, and so $\alpha \in \mathcal{L}(\Delta)$. \square

Example 3.9.10. Non-deterministic monoidal automata over word monoidal alphabets (Example 3.5.4) are convex: for a relation $\Delta : Q \rightarrow \mathcal{P}(Q)$, $\Delta^* = \Delta^\#$ is the Kleisli extension of Δ . Similarly, non-deterministic monoidal automata over bottom-up tree monoidal alphabets (Definition 3.5.6) are convex, with $\Delta^* := \Delta^\# \circ \nabla_{\mathcal{P}}$. This reflects the well known determinizability of word and bottom-up tree automata. For top-down tree monoidal alphabets, the general obstruction to convexity (and thus determinizability) is seen as the inexistence of a left inverse of $\nabla_{\mathcal{P}}$, viz. the cartesian product.

3.10 Mazurkiewicz traces and asynchronous automata

The theory of Mazurkiewicz traces [26, 60, 64] provides a simple but powerful model of concurrent systems. Traces are a generalization of *words* in which specified pairs of letters can commute. If we think of letters as corresponding to atomic *actions*, then commuting letters reflect the *independence* of those particular actions and so their possible concurrent execution: ab is observationally indistinguishable from ba if a and b are independent.

In this section, we show that trace languages are symmetric monoidal languages over monoidal graphs of a particular form that we call *monoidal distributed alphabets*. In Section 3.10.3 we turn to *asynchronous automata* [94], a well-known model accepting exactly the *recognizable* trace languages, and show that these automata are precisely symmetric monoidal automata over monoidal distributed alphabets.

3.10.1 Independence and distribution

We recall some definitions from Mazurkiewicz trace theory, before recasting them in terms of monoidal languages. Fix a finite set Σ , an alphabet thought of as a set of atomic actions.

Definition 3.10.1. An independence relation on Σ is a symmetric, irreflexive relation I . The induced dependence relation, D_I , is the complement of I .

Definition 3.10.2. For I an independence relation, let \equiv_I be the least congruence on Σ^* such that $\forall a, b: (a, b) \in I \Rightarrow ab \equiv_I ba$. The quotient monoid $\mathcal{T}(\Sigma, I) := \Sigma^* / \equiv_I$ is the trace monoid.

Definition 3.10.3. A (Mazurkiewicz) trace language over (Σ, I) is a subset of the trace monoid $\mathcal{T}(\Sigma, I)$.

An element of $\mathcal{T}(\Sigma, I)$ or *trace over* (Σ, I) is thus an equivalence class of words up to commutation of independent letters. A trace language may be thought of as the set of possible observations of a concurrent system's behaviour, in which independent letters stand for actions which may occur concurrently. Independence relations correspond to *distributions*:

Definition 3.10.4 (Mukund [64]). A distribution of an alphabet Σ is a finite tuple of non-empty alphabets $(\Sigma_1, \dots, \Sigma_k)$ such that $\bigcup_{i=1}^k \Sigma_i = \Sigma$.

Proposition 3.10.5 (Mukund [64]). A distribution of Σ corresponds to a function $\text{loc} : \Sigma \rightarrow \mathcal{P}^+(\{1, \dots, k\}) : \sigma \mapsto \{i \mid \sigma \in \Sigma_i\}$.

Such a function gives the set of “locations” of each action $\sigma \in \Sigma$. In terms of concurrency, we can consider this to be a set of memory locations, threads of execution, or runtimes in which σ participates. In particular, every action has a non-empty set of locations.

A well-known construction (see, e.g. Mukund [64, §9.6]) allows us to move between independence relations and distributions: locations correspond to *maximal cliques* in the graph of the dependence relation. We recall this construction in the proof of Proposition 3.10.8, which refines this correspondence.

Definition 3.10.6. Ind_Σ is the poset of independence relations on Σ , with order the inclusion of relations.

Definition 3.10.7. Dist_Σ is the preorder on distributions of Σ up to permutation, with $[(\Sigma_1, \dots, \Sigma_p)] \leq [(\Sigma'_1, \dots, \Sigma'_q)]$ if and only if for each pair of distinct elements $a, b \in \Sigma$, if there exists $1 \leq j \leq q$ such that Σ'_j contains both a and b , then there exists an Σ_i containing both a and b .

Proposition 3.10.8. *There is a Galois insertion $\text{Ind}_\Sigma \hookrightarrow \text{Dist}_\Sigma$.*

Proof. We construct an injective monotone function $i : \text{Ind}_\Sigma \rightarrow \text{Dist}_\Sigma$. Let an independence relation I over Σ be given, with induced dependence relation D_I . Construct the undirected *dependence graph*: vertices are elements of Σ and there is an edge (a, b) for every $(a, b) \in D_I$. Choose an ordering of maximal cliques of

D_I , and define a distributed alphabet by taking Σ_i to be the elements of Σ in the maximal clique i . Different orderings give the same distribution up to permutation, and so equivalent elements of Dist_Σ . This is injective since distinct independence relations induce distinct dependence graphs. It is monotone since if $I \subseteq I'$ then the dependence graph D_I is at least as connected as $D_{I'}$, so if a, b both belong to a maximal clique of $D_{I'}$ then they will both belong to a maximal clique of D_I .

We construct a monotone function $r : \text{Dist}_\Sigma \rightarrow \text{Ind}_\Sigma$. Let $(\Sigma_1, \dots, \Sigma_k)$ be a distribution. Define a relation I by $(a, b) \in I \Leftrightarrow \text{loc}(a) \cap \text{loc}(b) = \emptyset$. This is irreflexive and symmetric, and so an independence relation. r is also clearly well-defined and monotone. Finally, it is easy to check that $r \circ i : \text{Ind}_\Sigma \rightarrow \text{Ind}_\Sigma$ is the identity.

□

In other words, though the same independence relation may be induced by many different distributions, independence relations correspond bijectively with the distributions in the image of $i \circ r$, that is, the distributions obtained via the maximal clique construction.

3.10.2 Symmetric monoidal languages over monoidal distributed alphabets

We now turn to the interpretation of these notions in terms of symmetric monoidal languages. A distribution can be seen as a monoidal graph in which sorts are the locations (runtimes).

Definition 3.10.9. A monoidal distributed alphabet is a finite monoidal graph Γ with the following properties:

- Γ has set of sorts a finite ordinal $S_\Gamma = \{1 < 2 < \dots < k\}$ for $k \geq 1$,
- sorts $i \in S_\Gamma$ appear in order in the sources and targets of each generator $\gamma \in B_\Gamma$,
- each sort $i \in S_\Gamma$ appears at most once in each source and target,
- for each generator $\gamma \in B_\Gamma$, the sources and targets are non-empty and equal: $s(\gamma) = t(\gamma)$.

In brief, every generator in the alphabet is equipped with some set of runtimes, which serve as its source and target, and the runtimes are conserved. Figure 3.20 gives an example.

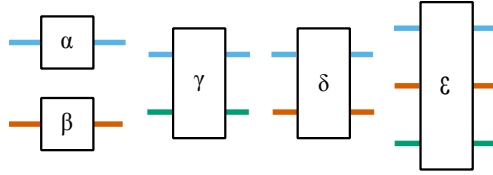


Figure 3.20: An example of a monoidal distributed alphabet. For example, γ and β are independent but γ and α are not. We use colours to stand for sorts, here blue = 1 < red = 2 < green = 3.

This gives us a way of representing distributions as monoidal graphs and vice-versa, if the graph is a monoidal distributed alphabet. Following Proposition 3.10.5, we will use $\text{loc}(\boxed{\gamma})$ to mean the arity (= coarity) of a generator $\boxed{\gamma}$. Since we choose a finite ordinal for the sorts, we have that:

Proposition 3.10.10. *Distributions of alphabets are in bijection with monoidal distributed alphabets.*

Since the *ordering* of the runtimes is ultimately not relevant to the structure of a trace, we should allow them to freely cross each other in our string diagrams: this is precisely what is enabled by taking the symmetric monoidal languages over these alphabets. We also need each runtime to appear once in each element of these languages, so we take the boundaries to be $1 \otimes \dots \otimes n$, which we will write as $\frac{1}{n}$.

Definition 3.10.11. A monoidal trace language is a symmetric monoidal language of the form $L \subseteq \mathcal{F}_\chi \Gamma \left(\frac{1}{n}, \frac{1}{n} \right)$ where Γ is a monoidal distributed alphabet.

Figure 3.21 gives an example of an element in a monoidal trace language over the monoidal distributed alphabet in Figure 3.20. We call such morphisms *monoidal traces*, and indeed we shall see below that they are exactly Mazurkiewicz traces. The corresponding string diagram gives an intuitive representation of traces as topological objects.

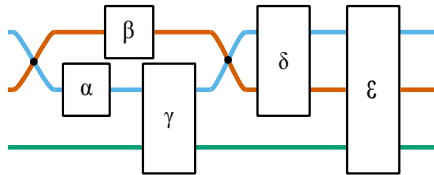


Figure 3.21: An example of a monoidal trace. β is independent of α and γ , but not δ or ϵ . Thus $\alpha\gamma\beta\delta\epsilon$ and $\beta\alpha\gamma\delta\epsilon$ are two possible serializations of this trace, corresponding to sliding β past α and γ in the string diagram. We use colours for sorts, blue = 1 < red = 2 < green = 3.

We now show that monoidal trace languages correspond precisely to Mazurkiewicz trace languages (Theorem 3.10.14), by establishing an isomorphism of monoids

between trace monoids and monoids of string diagrams generated by monoidal distributed alphabets. Fix a monoidal distributed alphabet Γ . Recall that endomorphism hom-sets in a category are monoids under composition, and that the hom-set $\mathcal{F}_\chi\Gamma\left(\begin{smallmatrix} 1 \\ \vdots \\ n \end{smallmatrix}, \begin{smallmatrix} 1 \\ \vdots \\ n \end{smallmatrix}\right)$ has elements string diagrams $\begin{smallmatrix} 1 \\ \vdots \\ n \end{smallmatrix} \rightarrow \begin{smallmatrix} 1 \\ \vdots \\ n \end{smallmatrix}$ over Γ .

Lemma 3.10.12. *The hom-set $\mathcal{F}_\chi\Gamma\left(\begin{smallmatrix} 1 \\ \vdots \\ n \end{smallmatrix}, \begin{smallmatrix} 1 \\ \vdots \\ n \end{smallmatrix}\right)$ admits the following presentation as a monoid:*

- *Generators: For each $\boxed{\gamma} \in \Gamma$, the string diagram $N(\gamma) : 1 \otimes \dots \otimes n \rightarrow 1 \otimes \dots \otimes n$ built from symmetries, followed by $\boxed{\gamma}$ tensored with identities, followed by the inverse symmetry. See Figure 3.22 for an example.*
- *Equations: $N(\alpha) \circledast N(\beta) = N(\beta) \circledast N(\alpha) \Leftrightarrow \text{loc}(\boxed{\alpha}) \cap \text{loc}(\boxed{\beta}) = \emptyset$, where \circledast denotes composition of string diagrams in diagrammatic (left-to-right) order.*

Proof. We construct an isomorphism between the monoids. Let $s \in \mathcal{F}_\chi\Gamma\left(\begin{smallmatrix} 1 \\ \vdots \\ n \end{smallmatrix}, \begin{smallmatrix} 1 \\ \vdots \\ n \end{smallmatrix}\right)$ be a string diagram. We can use interchange (Figure 2.6) to impose a linear order of generators from left to right in the diagram, e.g. $\boxed{\gamma_1}, \dots, \boxed{\gamma_n}$. This is called putting s in *general position*, by perturbing generators at the same horizontal position [49]. We then split the string diagram into a sequence of slices, each containing one generator. For a slice with right (or left) boundary $\begin{smallmatrix} k_1 \\ \vdots \\ k_n \end{smallmatrix}$, we can use the permutation $\begin{smallmatrix} k_1 \\ \vdots \\ k_n \end{smallmatrix} \rightarrow \begin{smallmatrix} 1 \\ \vdots \\ n \end{smallmatrix}$ followed by its inverse (or vice-versa) to finally obtain s as a sequence $N(\gamma_1) \circledast \dots \circledast N(\gamma_n)$. Any other possible sequence of generators is obtainable by repeatedly interchanging generators: this is possible if and only if their locations are disjoint. Consequently, this defines a function from $\mathcal{F}_\chi\Gamma\left(\begin{smallmatrix} 1 \\ \vdots \\ n \end{smallmatrix}, \begin{smallmatrix} 1 \\ \vdots \\ n \end{smallmatrix}\right)$ to the monoid presented above. Given that, as argued above, the slicing construction is unique up to interchanging independent generators, this defines a homomorphism. Conversely, given a generator $N(\gamma)$ in the presentation, we map this to the same string diagram in $\mathcal{F}_\chi\Gamma\left(\begin{smallmatrix} 1 \\ \vdots \\ n \end{smallmatrix}, \begin{smallmatrix} 1 \\ \vdots \\ n \end{smallmatrix}\right)$. Again, it follows from interchange that this extends to a homomorphism, inverse to that above. \square

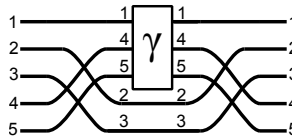


Figure 3.22: An example of a generator $N(\gamma)$ as in Lemma 3.10.12.

We now show that trace monoids are isomorphic to the endomorphism monoids $\mathcal{F}_\chi\Gamma\left(\begin{smallmatrix} 1 \\ \vdots \\ n \end{smallmatrix}, \begin{smallmatrix} 1 \\ \vdots \\ n \end{smallmatrix}\right)$.

Lemma 3.10.13. *Let I be an independence relation on an alphabet Σ , and Γ the monoidal distributed alphabet induced by the corresponding distribution (Proposition 3.10.10). Then there is an isomorphism of monoids,*

$$\mathcal{T}(\Sigma, I) \cong \mathcal{F}_\chi \Gamma \left(\begin{array}{c} 1 \\ \vdots \\ n \end{array}, \begin{array}{c} 1 \\ \vdots \\ n \end{array} \right).$$

Proof. We use the presentation of the endomorphism monoid given in Lemma 3.10.12. Define a homomorphism $\alpha : \mathcal{F}_\chi \Gamma \left(\begin{array}{c} 1 \\ \vdots \\ n \end{array}, \begin{array}{c} 1 \\ \vdots \\ n \end{array} \right) \rightarrow \mathcal{T}(\Sigma, I)$ by mapping generators $N(\gamma) \mapsto [\gamma]$. Let $N(\gamma) \circ N(\gamma') = N(\gamma') \circ N(\gamma)$, then it follows $[\gamma\gamma'] = [\gamma'\gamma]$ in $\mathcal{T}(\Sigma, I)$, since the former holds if and only if $\text{loc}(\gamma) \cap \text{loc}(\gamma') = \emptyset$, and so this extends to a homomorphism. Define a homomorphism $\beta : \mathcal{T}(\Sigma, I) \rightarrow \mathcal{F}_\chi \Gamma \left(\begin{array}{c} 1 \\ \vdots \\ n \end{array}, \begin{array}{c} 1 \\ \vdots \\ n \end{array} \right)$ by mapping generators $[\gamma] \mapsto N(\gamma)$. $[\gamma\gamma'] = [\gamma'\gamma]$ holds iff $\text{loc}(\gamma) \cap \text{loc}(\gamma') = \emptyset$, iff $\text{loc}(\boxed{\gamma}) \cap \text{loc}(\boxed{\gamma'}) = \emptyset$, iff $N(\gamma) \circ N(\gamma') = N(\gamma') \circ N(\gamma)$. Finally it is clear that α and β are inverses, and so witness an isomorphism of monoids. □

Theorem 3.10.14. *Monoidal trace languages are exactly Mazurkiewicz trace languages.*

Proof. This is immediate from Lemma 3.10.13: given a monoidal trace language $L \subseteq \mathcal{F}_\chi \Gamma \left(\begin{array}{c} 1 \\ \vdots \\ n \end{array}, \begin{array}{c} 1 \\ \vdots \\ n \end{array} \right)$ we obtain a trace language $L' \subseteq \mathcal{T}(\Sigma, I)$ using one direction of the isomorphism, and vice-versa. □

Lemma 3.10.13 also shows that composition of traces corresponds simply to concatenation of the corresponding monoidal traces. Diagrams like Figure 3.21 are commonplace in the trace literature [25, 94]. Theorem 3.10.14 gives a formal basis for these diagrams as elements of symmetric monoidal languages.

Remark 3.10.15. The monoidal category $\mathcal{F}_\chi \Gamma$ for a monoidal distributed alphabet Γ contains a number of other trace languages of interest, namely those obtained by restricting the set of locations. In this way, one obtains the sub-languages of traces in which only certain locations are involved.

Remark 3.10.16. Free props over monoidal distributed alphabets considered as monoidal categories with *multiple runtimes* suggest a further generalization of string diagrams for effectful categories. A similar proposal is sketched as a setting for concurrency by Jeffrey [46, Section 9.4]. We return to this in Chapter 4.

3.10.3 Asynchronous automata as monoidal automata

Asynchronous automata were introduced by Zielonka [94] as a true-concurrent operational model of recognizable trace languages, a well-behaved subclass of trace languages analogous to regular languages. In this section, we show they are precisely

monoidal automata over monoidal distributed alphabets, which leads to the following theorem:

Theorem 3.10.17. *Recognizable trace languages are exactly symmetric regular monoidal languages over monoidal distributed alphabets.*

We recall the definition of asynchronous automata, before turning to monoidal automata.

Definition 3.10.18 (Asynchronous automaton [94]). Let $(\Sigma_1, \dots, \Sigma_k)$ be a distribution of an alphabet Σ . For each $1 \leq i \leq k$, let Q_i be a non-empty finite set of states, and for each $\sigma \in \Sigma$ take a transition relation

$$\Delta_\sigma : \prod_{i \in \text{loc}(\sigma)} Q_i \rightarrow \mathcal{P} \left(\prod_{i \in \text{loc}(\sigma)} Q_i \right).$$

This defines a global transition relation on the set $Q := \prod_{i=1}^k Q_i$ as follows:

$(q_1, \dots, q_k) \xrightarrow{\sigma} (q'_1, \dots, q'_k) \Leftrightarrow q_i = q'_i$ for $i \notin \text{loc}(\sigma)$ and $(q'_{i_1}, \dots, q'_{i_j}) \in \Delta_\sigma(q_{i_1}, \dots, q_{i_j})$ where $\{i_1, \dots, i_j\} \in \text{loc}(\sigma)$. Finally let $\vec{i} \in Q, F \subseteq Q$ be initial and final words of states.

The global transition relation for σ leaves unchanged those states at locations in the complement of $\text{loc}(\sigma)$, and otherwise acts according to the local transition Δ_σ . An asynchronous automaton has a language over Σ given by the extension of the transition relation to words. Moreover, asynchronous automata have a language of Mazurkiewicz traces over the distribution of Σ : a trace in $\mathcal{T}(\Sigma, I)$ is accepted when all of its serializations are accepted, which happens when one of its serializations is accepted [94, p. 109]. *Recognizable trace languages* are defined algebraically as those whose syntactic congruence is of finite index [94]. Zielonka's theorem says that they also have an operational characterization:

Theorem 3.10.19 (Zielonka [94]). *Asynchronous automata accept precisely the recognizable trace languages.*

Definition 3.10.18 closely resembles that of monoidal automata. Indeed, asynchronous automata are precisely monoidal automata over monoidal distributed alphabets:

Proposition 3.10.20. *For an asynchronous automaton \mathcal{A} , there is a monoidal automaton over a monoidal distributed alphabet with the same trace language, and vice-versa.*

Proof. An asynchronous automaton with multiple final state words can be normalized to a single final state word in the usual way by introducing a new final

state word and modifying transitions appropriately. Then a monoidal automaton can be constructed by taking the monoidal distributed alphabet associated to the distribution of Σ (Proposition 3.10.10), the same transition relations, initial and final state words. We show that the languages coincide. Let $w \in \mathcal{L}(\mathcal{A})$, and consider the corresponding trace $[w]$. Using Lemma 3.10.13, we can produce the corresponding monoidal trace. By construction, this is accepted by the monoidal automaton defined above. The converse is analogous. \square

As a corollary, we can invoke Theorem 3.10.19 to obtain Theorem 3.10.17. In contrast to asynchronous automata, the constructed monoidal automaton directly accepts traces qua string diagrams, rather than a language of words corresponding to a trace language.

Remark 3.10.21. Jesi, Pighizzini, and Sabadini [48] introduced probabilistic asynchronous automata. Initial and final states, and transition relations are replaced by initial and final distributions, and stochastic transitions. These are precisely what are obtained if the powerset monad in our definition of non-deterministic monoidal automaton is replaced with the *distribution monad* [71], whose Kleisli category has morphisms stochastic matrices.

Remark 3.10.22. Melliès and Zeilberger have shown how to capture a related notion of asynchronous automata, namely Bednarczyk’s *asynchronous systems* [5], in the framework of regular grammars over arbitrary categories (Section 3.1.1) [63, Example 2.14]. The idea is to take grammars in the sense of Definition 3.1.6 over the trace monoid considered as a one-object category. Asynchronous systems are simply finite state automata in which every path $q_1 \xrightarrow{a} q_2 \xrightarrow{b} q_3$ completes to a diamond via $q_1 \xrightarrow{b} q_4 \xrightarrow{a} q_3$ whenever a and b are independent. Like asynchronous automata, *asynchronous systems* also accept precisely the regular trace languages.

3.10.4 Serialization of traces via premonoidal categories

Trace theorists often consider trace languages to be word languages with the property of *trace-closure* with respect to an independence relation [58]: if $u \in L$ and $u \equiv_I v$ then $v \in L$. These languages arise as preimages of trace languages along the quotient map $q_{\Sigma, I} : \Sigma^* \rightarrow \mathcal{T}(\Sigma, I)$. For $L \subseteq \mathcal{T}(\Sigma, I)$ a trace language, $q_{\Sigma, I}^{-1}(L) \subseteq \Sigma^*$ is its *flattening* or *serialization*.

In this section, we show that the serialization of monoidal trace languages can be carried out using the algebra and string diagrams of (symmetric) premonoidal categories. Recall from Section 2.4 that premonoidal categories are like monoidal categories, except interchange (Figure 2.6) does not hold in general.

We first define the *runtime monoidal graph* over a monoidal graph, which augments the generators with a new wire:

Definition 3.10.23. Let \mathcal{G} be a monoidal graph. Let R be a sort disjoint from $S_{\mathcal{G}}$. The runtime monoidal graph \mathcal{G}_R has sorts $S_{\mathcal{G}} + \{R\}$ and for each generator $\gamma : S_1 \dots S_n \rightarrow S'_1 \dots S'_m$ in \mathcal{G} a generator $\gamma : RS_1 \dots S_n \rightarrow RS'_1 \dots S'_m$.

Graphically we can depict \mathcal{G}_R as in Figure 3.23 (right):



Figure 3.23: Left: A monoidal graph \mathcal{G} . Right: the associated runtime monoidal graph \mathcal{G}_R , where the new sort R is drawn as a dashed string.

Definition 3.10.24. The symmetric *runtime monoidal category* is the free prop $\mathcal{F}_{\chi}\mathcal{G}_R$ on \mathcal{G}_R .

Theorem 3.10.25. *The free symmetric strict premonoidal category $\mathcal{F}_p\mathcal{G}$ on a monoidal graph \mathcal{G} has set of objects $S_{\mathcal{G}}$ and a morphism $S_1 \otimes \dots \otimes S_n \rightarrow S'_1 \otimes \dots \otimes S'_m$ is a morphism $R \otimes S_1 \otimes \dots \otimes S_n \rightarrow R \otimes S'_1 \otimes \dots \otimes S'_m$ in the symmetric runtime monoidal category.*

Proof. The proof follows Román [77, Theorem 2.14], in the case where \mathcal{V} is empty, and taking instead the free symmetric strict monoidal category. \square

Given a monoidal distributed alphabet Γ , the endomorphism monoid $\mathcal{F}_p\Gamma\left(\begin{smallmatrix} 1 & 1 \\ \vdots & \vdots \\ n & n \end{smallmatrix}\right)$ in the free symmetric premonoidal category is now the free monoid over the boxes of Γ , since the runtime prevents interchange:

Proposition 3.10.26. *Let Γ be a monoidal distributed alphabet. Then $\mathcal{F}_p\Gamma\left(\begin{smallmatrix} 1 & 1 \\ \vdots & \vdots \\ n & n \end{smallmatrix}\right) \cong B_{\Gamma}^*$, where B_{Γ} is the set of boxes of Γ .*

Proof. By augmenting the generators of Γ with a new runtime, we create a distributed monoidal alphabet in which every generator depends on every other, that is, the independence relation is empty. Thus the corresponding trace monoid is simply B_{Γ}^* . From here, we can follow the idea of Lemma 3.10.13. \square

We can define a morphism of monoids $\eta_{\Gamma} : \mathcal{F}_p\Gamma\left(\begin{smallmatrix} 1 & 1 \\ \vdots & \vdots \\ n & n \end{smallmatrix}\right) \rightarrow \mathcal{F}_{\chi}\Gamma\left(\begin{smallmatrix} 1 & 1 \\ \vdots & \vdots \\ n & n \end{smallmatrix}\right)$ by presenting $\mathcal{F}_p\Gamma\left(\begin{smallmatrix} 1 & 1 \\ \vdots & \vdots \\ n & n \end{smallmatrix}\right)$ as in Lemma 3.10.12, and defining η_{Γ} on generators by erasing the runtime string. Theorem 3.10.27 then follows immediately from the definitions along with Lemma 3.10.13 and Proposition 3.10.26:

Theorem 3.10.27. *For every alphabet B_Γ , the following square of monoid homomorphisms commutes, where q is the quotient monoid homomorphism.*

$$\begin{array}{ccc}
 B_\Gamma^* & \xrightarrow{q} & \mathcal{T}(B_\Gamma, I) \\
 \cong \downarrow & & \downarrow \cong \\
 \mathcal{F}_p \Gamma \left(\begin{array}{c} 1 \\ \vdots \\ n \end{array}, \begin{array}{c} 1 \\ \vdots \\ n \end{array} \right) & \xrightarrow{q_\Gamma} & \mathcal{F}_\chi \Gamma \left(\begin{array}{c} 1 \\ \vdots \\ n \end{array}, \begin{array}{c} 1 \\ \vdots \\ n \end{array} \right)
 \end{array}$$

As a result, the preimage of a monoidal trace language under the morphism q_Γ corresponds to the serialization of that language.

Chapter 4

From Mazurkiewicz Traces to Resourceful Traces

In this chapter, we elaborate the link between the string-diagrammatic Mazurkiewicz traces introduced in the previous chapter, and string diagrams for effectful categories. This leads us to the idea of devices, which facilitate a convenient notion of presentation for effectful categories, suited to their use in the semantics of programming languages. In more detail,

- We revisit the account of trace languages from Section 3.10, showing how it fits more naturally into the setting of effectful categories. — *Section 4.1*
- We introduce *device presentations* of effectful categories, which lets us see their morphisms as *resourceful traces*. — *Section 4.2*
- We give an example of resourceful traces arising from the theory of message passing concurrency. — *Section 4.3*
- We introduce the notion of *centralizer* of a set of morphisms in an effectful category, relating it to device presentations. — *Section 4.5*
- We show how the *commuting tensor product* of effectful categories admits a convenient presentation in terms of devices. — *Section 4.6*

4.1 Mazurkiewicz traces are effectful scalars

In Section 3.10, we showed how Mazurkiewicz traces arise as string diagrams in certain free symmetric monoidal categories. In general, we only cared about a small part of these categories, namely a single endo-hom-set on the set of locations with an arbitrary fixed ordering (although as mentioned in Remark 3.10.15, there are languages that are natural to consider in other hom-sets of the category). The crucial invariant of our string-diagrammatic Mazurkiewicz traces is the appearance of each location string only once in each vertical slice through the diagram: this was ensured by construction. In particular, monoidal product is not an operation on traces, in the sense that what we obtain is no longer a trace.

This situation strongly resembles the properties of string diagrams for premonoidal categories, or more generally, effectful categories, as recalled in Sections 2.4 and 3.10.4. There, the distinguished string, drawn as a dashed string, acts as a “global location”. By construction, it can only appear once in every vertical slice through a string diagram. There is no longer a monoidal product of morphisms, but only a premonoidal product. In this chapter, we take this analogy seriously.

We start, in this section, by showing how effectful categories provide a setting for Mazurkiewicz traces: given any alphabet with an independence relation, there is a one-object effectful category whose hom-set is the set of Mazurkiewicz traces. This is somewhat trivial: we know that traces form a monoid, and we can give this monoid a trivial premonoidal structure. However, this is merely a warm-up for the rest of the chapter; our intention is only to show the basic construction working in a simple case, not to record a deep result. In the following sections, we shall provide a more general construction of which this trivial one is a special case; the construction of effectful categories by *device presentations*.

The idea is simple: we consider a variation of string diagrams for effectful categories in which there are *multiple* distinguished strings: in this section these will correspond to *locations*, but we later give them the more generic term of *devices*. When there are no resource strings, but only devices, we have precisely Mazurkiewicz traces.

For the rest of this section, let us fix an alphabet (Σ, I) equipped with an independence relation I , whose induced distribution $\text{loc} : \Sigma \rightarrow \mathcal{P}^+(\{1, \dots, k\})$ has locations $\mathcal{L} = \{1, \dots, k\}$.

In Section 3.10, in order to obtain a monoid, we made a *choice* of ordering on the set of locations. Thus for example, although the three traces in Figure 4.1 are morally the same trace, by requiring a fixed ordering of locations, we are choosing a particular representative.

To avoid this, we will take the object of our one-object effectful category to be the *clique* of locations, which contains every permutation of the locations. This idea follows the work of Román in the case of a single device (or *runtime*) [77, 78].

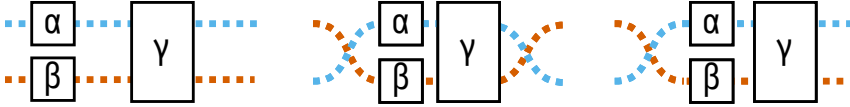


Figure 4.1: These traces are morally the same: we should not be concerned about the order of the locations on the boundaries.

Indeed, Definitions 4.2.3 and 4.2.4 are straightforward generalizations of similar constructions given by Román, from a single device to multiple devices.

Definition 4.1.1. Given a finite set \mathcal{L} , the clique $\text{Cl}_{\mathcal{L}}$ on \mathcal{L} is the groupoid whose objects are all total orders of \mathcal{L} and whose isomorphisms are the permutations.

Instead of asking for endomorphisms of the monoidal product of locations in a chosen order (or indeed between two arbitrary orders), we take morphisms from the clique of locations to itself to be *families* of traces differing only by permutations of their boundaries, as in Figure 4.1: they take into account all of the permutations at once. Readers familiar with the strictification of monoidal categories will note that the idea here is very much in the same spirit. Note that a single element of the family determines the rest, by pre-/post-composition with a permutation.

Proposition 4.1.2. *There is a single-object strict premonoidal category $\text{Dev}(\Sigma, I)$ with object the clique of locations $\text{Cl}_{\mathcal{L}}$ and morphisms given by families of string-diagrammatic traces over (Σ, I) , commuting with the isomorphisms of the clique of locations.*

Proof. Identities are given by families of identity string diagrams. Composition is induced by composition of string diagrams, and these are automatically associative and unital. $\text{Cl}_{\mathcal{L}}$ is the unit object, and the left- and right- whiskering functors are the identity functors. \square

Proposition 4.1.3. *The monoid of morphisms of $\text{Dev}(\Sigma, I)$ is isomorphic to the trace monoid $\mathcal{T}(\Sigma, I)$.*

Proof. This is essentially Lemma 3.10.13, along with the fact that one component of each family comprising the morphisms of $\text{Dev}(\Sigma, I)$ uniquely determines the rest of the family. \square

In the next section, we place this construction in a more general setting, where $\text{Dev}(\Sigma, I)$ is seen to be the free effectful category on a *scalar* signature, in the sense that the signature only has generators from the premonoidal unit to itself.

4.2 Presenting effectful categories by devices

Monoidal categories can be presented by monoidal graphs of generators and equations between string diagrams over the monoidal graph, as in Section 2.1. In this section, we consider an augmentation of monoidal graphs by distinguished strings which control the commutation of generators, and show how they present effectful categories. In the context of traces we called these strings locations, but in the following more general context we shall term these special strings devices. Devices may be thought of as effects, and in particular (memory) locations can be thought of as particular devices.

A device is nothing more than a string in a string diagram with the special property that it may only appear once in every vertical slice through the diagram, and it may appear at any position, that is, it may freely braid with other devices and with other strings: we shall ensure these properties by construction.

Following the work of Jeffrey [46], Román [77, 78] has shown that free effectful categories may be presented by string diagrams in which the (effectful) morphisms are augmented by a single global device. However, as the case of traces makes clear, the introduction of *multiple devices* allows us to capture, via the geometry of string diagrams, certain families of equations that arise when particular families of generators commute, such as independent actions in a trace, or more generally commuting effects.

For example, consider a programming language with mutable memory cells. We can picture the `get` and `set` actions associated to a cell as string diagrams augmented with devices, again depicted as dashed strings. When we only have a single global device for our effectful processes, certain equations that we would like to hold no longer do so, as in Figure 4.2.



Figure 4.2: `set` actions for two distinct memory cells, using the string diagrams for effectful categories introduced by Jeffrey [46] and Román [77, 78]. We would like these morphisms to be equal.

Instead, we can introduce a distinct device for each memory cell, giving the following *device graph*,



which should be furthermore subject to equations expressing, for example, that two sequential `get` operations on the same cell amount to copying the result of one `get` operation.

Treating these diagrams as generators, we can build programs as string diagrams over them. If we enforce the condition that every device appear at most once in each vertical section through the diagram, then the *independence* of operations on distinct cells is reflected topologically. For example, we can freely reorder `set` operations on distinct cells, without changing the semantics of the program:



Note that these actions are no longer *scalars*: they have resource strings that carry a value of the type of the memory cell. It is in this sense that we consider these morphisms as *resourceful traces*, that is, traces in which actions are not merely atomic names, as in Mazurkiewicz traces, but also have input and output types, e.g. $\text{set} : X \rightarrow I$, for a memory cell storing data of type X .

We begin to formalize these pictures by introducing the following notion of signature, *device graphs*. These are monoidal graphs equipped with a set of devices and an assignment of a subset of devices to each process.

Definition 4.2.1. A device graph is given by

- a monoidal graph, $s, t : P \rightrightarrows R^*$,
- a finite set of *devices* D , and
- a function $d : P \rightarrow \mathcal{P}(D)$, specifying a set of devices used by each generator.

We shall speak of the sorts R of the monoidal graph as *resources*. Intuitively, the effectful category generated by a device graph will have morphisms the string diagrams built from the device graph, with the restriction that each device appears exactly once in every slice of the diagram. Note also that, unlike the resources in the source and target of a generator, devices, which occur both in the source and target of the corresponding string diagrams, do not occur in any particular order. In Definition 4.2.4, we will fix an order, in order to treat a device graph as an ordinary monoidal graph. However, the use of cliques, as in the previous section, will allow us to use diagrams in which devices occur in any order to reason about the morphisms of effectful categories.

The graphical representation of a generator α in a device graph is as for monoidal graphs, but for every device used by the generator we add a dashed string entering

and exiting the box, corresponding to the device. We shall generally use colours to represent distinct devices.

Definition 4.2.2. A morphism of device graphs $f : (P, R, \mathcal{D}, d) \rightarrow (P', R', \mathcal{D}', d')$ comprises

- a morphism of the underlying monoidal graphs $(f_S : R \rightarrow R', f_G : P \rightarrow P')$,
- such that for all $g, h \in P$, if $d(g) \cap d(h) = \emptyset$, then $d'(f_G(g)) \cap d'(f_G(h)) = \emptyset$.

Device graphs and their morphisms form a category, DevGraph .

This definition of device graph morphism reflects an important property of strict premonoidal functors, namely that they preserve interchanging morphisms.

Our construction of the effectful category freely generated by a device graph will be very much along the lines of Section 4.1. We take as objects the following cliques.

Definition 4.2.3. The device clique, $\text{Cl}_{\mathcal{D}}[X_1, \dots, X_n]$, on a list of resources $\vec{X} = X_1, \dots, X_n \in R$ in a device graph with devices \mathcal{D} is a full subcategory of a certain freely generated monoidal category, which we describe first. This monoidal category has the following generators, for every device D_i, D_k and distinct resource $X_j \in \vec{X}$,

- $\sigma_{i,j} : D_i, X_j \rightarrow X_j, D_i$
- $\sigma_{i,j}^{-1} : X_j, D_i \rightarrow D_i, X_j$
- $\sigma'_{i,k} : D_i, D_k \rightarrow D_k, D_i$
- $\sigma'^{-1}_{i,k} : D_k, D_i \rightarrow D_i, D_k$

quotiented by the equations that witness σ^{-1} and σ'^{-1} as left- and right-inverses of σ and σ' . In other words, we may swap any two resources and devices, and any two devices, but we do not allow resources to swap. The set of objects on which we take the full subcategory is the set of all the shufflings of the ordered list of resources into permutations of the list of all devices,

$$\coprod_{\sigma \in \text{Perm}(\mathcal{D})} \text{Shuf}([X_1, \dots, X_n], \sigma\mathcal{D}).$$

An element of $\text{Shuf}([X_1, \dots, X_n], \sigma\mathcal{D})$ is a list of length $n + |\mathcal{D}|$, which contains all the elements of each list exactly once, preserving their relative order.

For example, let $\mathcal{D} = \{D_1, D_2\}$, then the clique $\text{Cl}_{\mathcal{D}}[X_1, X_2]$ has twelve objects: $D_1D_2X_1X_2$; $D_1X_1D_2X_2$; $D_1X_1X_2D_2$; $X_1X_2D_1D_2$; $X_1X_2D_2D_1$; $X_1D_2X_2D_1$; $X_1D_1X_2D_2$; $X_1D_1D_2X_2$; $X_1D_2D_1X_2$; $D_2X_1D_1X_2$; $D_2X_1X_2D_1$; $D_2D_1X_1X_2$.

For morphisms, we proceed in two stages. Firstly, we treat the device graph as a monoidal graph, freely generating a monoidal category subject to braiding of device strings over resource strings. Then, the morphisms of the effectful category freely generated by a device graph will be families of certain morphisms of this category, namely those in which each device appears exactly once.

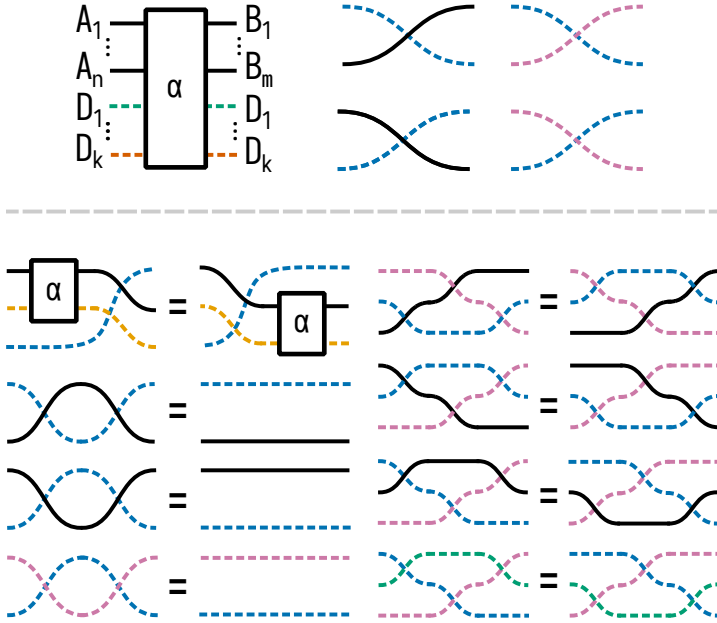


Figure 4.3: Generators and equations for the device monoidal category on a device graph (Figure 4.3). Generators and equations with pairwise distinct devices are implied to be taken over every pair of devices.

Definition 4.2.4. The device monoidal category on a device graph \mathcal{D} is the free monoidal category presented by the generators and equations of Figure 4.3. To define the monoidal graph in Figure 4.3, we fix an order on the set of devices. For each α with $s(\alpha) = A_1 \dots A_n$, $t(\alpha) = B_1 \dots B_m$ and $\mathcal{D} = \{D_1, \dots, D_k\}$, we take a generator α as above, where $D_1 < \dots < D_k$ in the chosen order on devices. We also take braidings of every device over every object, and braidings of every device over every other device. We ask that whenever α does not use a certain device that we can slide it past this device, that the braidings are symmetries, and that Yang-Baxter equations hold (Figure 4.3, right).

Note that the extra complication of explicitly allowing device strings to braid could be obviated by simply asking for the free symmetric monoidal category on the generators α , as in the definition of the runtime monoidal graph in Section 3.10.4.

However, this would also allow resources to braid. While this may be acceptable in some cases, we choose to present the general case.

Definition 4.2.5. A device clique morphism $\text{Cl}_{\mathcal{D}}[X_1, \dots, X_n] \rightarrow \text{Cl}_{\mathcal{D}}[Y_1, \dots, Y_m]$ over a device graph \mathcal{D} is a family of morphisms in the device monoidal category on \mathcal{D} from each of the objects of the first clique, to each of the objects of the second, which moreover commute with the isomorphisms of the braid cliques.

The requirement that each morphism of the family commutes with the isomorphisms of the device cliques, means that one component of the family determines the others.

Proposition 4.2.6. *There is a strict premonoidal category $\mathcal{F}_p\mathcal{D}$ with objects the device cliques over \mathcal{D} and morphisms the device clique morphisms.*

Proof. Identities are determined by identity string diagrams, and composition is induced by composition of string diagrams. The unit object is the device clique on the empty list. For device cliques $\text{Cl}_{\mathcal{D}}[X_1, \dots, X_n]$ and $\text{Cl}_{\mathcal{D}}[Y_1, \dots, Y_m]$, define $\text{Cl}_{\mathcal{D}}[X_1, \dots, X_n] \otimes \text{Cl}_{\mathcal{D}}[Y_1, \dots, Y_m] := \text{Cl}_{\mathcal{D}}[X_1, \dots, X_n, Y_1, \dots, Y_m]$. Left- and right-whiskerings are induced by the string diagrams of Figure 4.4. \square

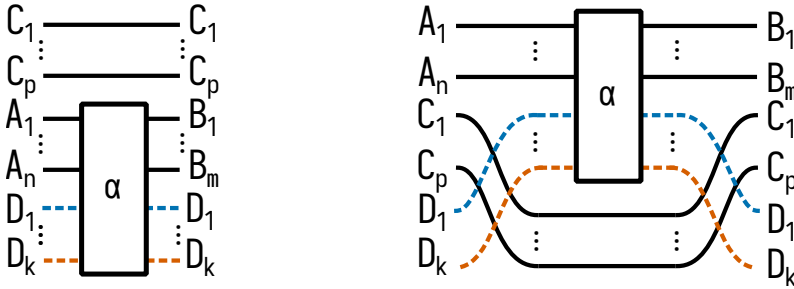


Figure 4.4: Left- and right- whiskerings in $\mathcal{F}_p\mathcal{D}$, denoted $\text{Cl}_{\mathcal{D}}[C_1, \dots, C_p] \triangleleft \alpha, \alpha \triangleright \text{Cl}_{\mathcal{D}}[C_1, \dots, C_p]$ respectively.

Proposition 4.2.7. *A device graph \mathcal{D} freely generates an effectful category*

$$\mathcal{F}_e\mathcal{D} : \mathcal{F}_{\otimes}\mathcal{D}_{\emptyset} \rightarrow \mathcal{F}_p\mathcal{D}$$

where $\mathcal{F}_{\otimes}\mathcal{D}_{\emptyset}$ is free monoidal category on the generators of the device graph having no devices, and $\mathcal{F}_p\mathcal{D}$ is the premonoidal category with objects the device cliques and morphisms the device clique morphisms constructed in Proposition 4.2.6.

Proof. We define a functor $\mathcal{F}_e\mathcal{D} : \mathcal{F}_{\otimes}\mathcal{D}_{\emptyset} \rightarrow \mathcal{F}_p\mathcal{D}$ strictly preserving the premonoidal structure, and with central image. On objects, we define $X_1 \otimes \dots \otimes X_n \mapsto$

$\text{Cl}_{\mathcal{D}}[X_1, \dots, X_n]$, and the action on morphisms is induced by the mapping in Figure 4.5, where D_1, \dots, D_k are all of the devices of \mathcal{D} . By drawing the necessary string diagrams, one easily sees that the image is central, using the equations of Figure 4.3. Preservation of left whiskering is immediate, and preservation of right whiskering follows from the symmetry equations of Figure 4.3. \square

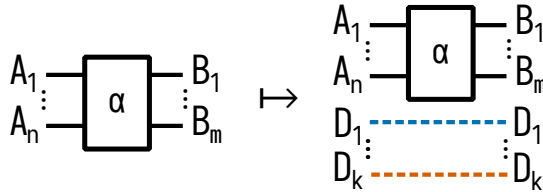


Figure 4.5: Inclusion of central morphisms.

Remark 4.2.8. Although this construction is morally the free effectful category over a device graph, more work is required to show that this construction extends to a left adjoint functor. Since, as we shall see in Section 4.4, not every effectful category has an underlying device graph (since the putative number of devices may be infinite), this poses a challenge to the construction of the right adjoint, if we are to use string diagrammatic syntax. We leave the extension of these constructions to the case of an infinite number of devices to future work.

Definition 4.2.9. A device presentation of an effectful category is given by a device graph, along with a set of equations between string diagrams over the device graph.

Since Mazurkiewicz traces – the actions of which are atomic, i.e. merely names – are the morphisms of the free effectful category presented by device graphs having no resources (Section 4.1), we can think of the morphisms of effectful categories presented by devices as *resourceful traces*, that is traces in which actions are no longer merely names, but also may transform resources. In the next section, we look at an example arising from the theory of message passing concurrency.

4.3 Resourceful traces for message passing processes

In this section, we give an example of resourceful traces (i.e. an effectful category presented by devices) arising from a model of communicating concurrent processes based on the *free cornering* of a monoidal category. We only give a high-level overview of the concepts involved – for more details on the free cornering, see Nester [67, 68] (though note that our diagrams below swap the horizontal and vertical dimensions).

The *free cornering* of a monoidal category is formally defined as a particular (single object) *double category*, which can be given a convenient presentation using

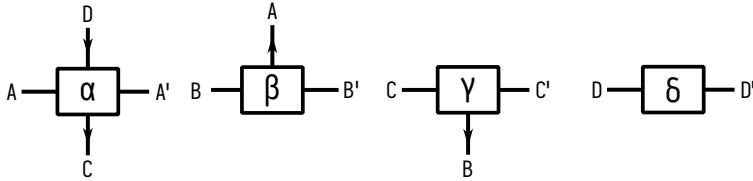


Figure 4.6: Shapes of cells in the free cornering of a monoidal category.

string diagrams in which morphisms (*cells*) additionally have strings entering from the top and exiting from the bottom. Figure 4.6 shows the relevant four types of cells present in a double category such as the free cornering of a monoidal category.

In the free cornering of a monoidal category, the objects labelling the strings entering and exiting the top and bottom of a cell are *polarizations* of the objects in the monoidal category: we indicate the polarization by annotating the strings with arrow heads. This notation suggests the intended semantics of the cells: they are processes which, in addition to transforming resources from left to right, may also receive a resource from the top or bottom, or pass a resource to the top or bottom. In a double category, we can compose cells horizontally, as in a monoidal category, but also vertically. In the free cornering, vertical composition is precisely synchronization of resource passing between two processes.

Importantly, cells of the free cornering do not interchange in general, as in Figure 4.7.

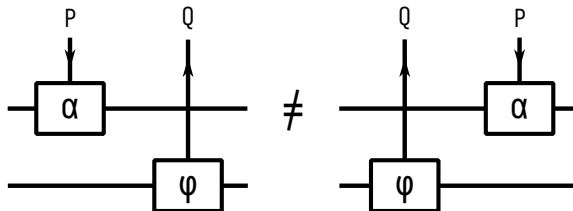


Figure 4.7: Cells in the free cornering do not interchange in general.

The left-hand cell of Figure 4.7 denotes a process that first receives a P and later outputs a Q , whereas this is reversed in the other cell: the resource passing protocols are different from the perspective of the actor on the top. On the other hand, certain pairs of cells do commute, as in Figure 4.8.

The situation is summarized by the fact that the free cornering has an underlying effectful category presented by two devices, one corresponding to the top, and one to the bottom, which we might conceive of as the ends of a bidirectional channel. This gives rise to a device graph as shown in Figure 4.9.

The effectful category presented by this device graph gives a simplified representation of the processes in the free cornering: it abstracts away from the message

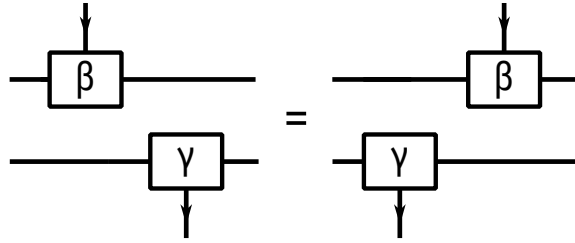


Figure 4.8: We can interchange cells when it does not change the top and bottom boundary protocols.

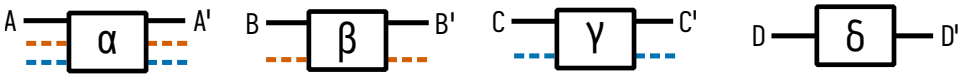


Figure 4.9: Device graph corresponding to Figure 4.6.

passing protocols, recording only the transformations of resources from left to right, but in a way that records topologically the independence between the actions in the process.

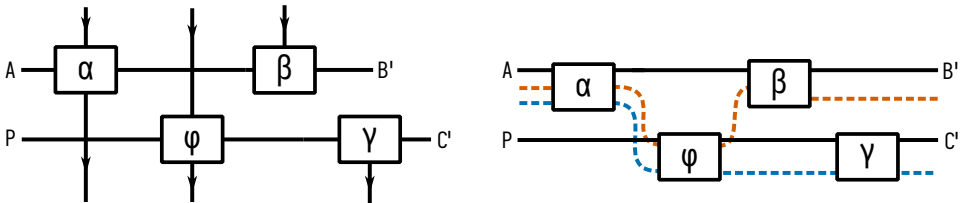


Figure 4.10: A cell in the free cornering (left) and the corresponding resourceful trace (right).

4.4 Independence and distribution for premonoidal categories

In this section, we show how a device presentation can be extracted from a premonoidal category. We use a similar construction to that of Section 3.10.1, where we obtained a distribution of an alphabet from an independence relation on it by taking maximal cliques in the graph of the associated dependence relation. The graph we consider here is the following.

Definition 4.4.1. The interference graph of a premonoidal category \mathbb{C} has vertices the morphisms of \mathbb{C} , and an edge $f \leftrightarrow g$ just when f and g do not interchange.

The interference graph of a premonoidal category sometimes determines a device presentation which we call its intrinsic device presentation.

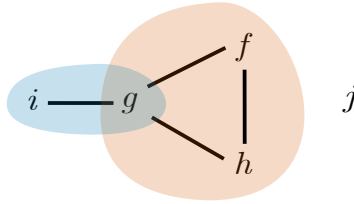


Figure 4.11: Illustrative interference graph of a premonoidal category.

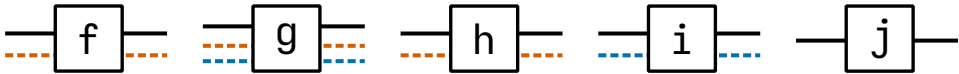


Figure 4.12: The device graph corresponding to the interference graph of Figure 4.11. Types of the resource strings are omitted.

Definition 4.4.2. When the interference graph of a premonoidal category \mathbb{C} has a finite number of non-trivial maximal cliques, it determines a device presentation called the intrinsic device presentation, with set of devices equal to the set of non-trivial maximal cliques, underlying monoidal graph given by the morphisms of \mathbb{C} , and assignment of devices to a morphism by the set of maximal cliques to which it belongs. Finally, the equations are given by all those holding in \mathbb{C} .

Example 4.4.3. Monoidal categories have intrinsic device presentations with no devices: since every pair of morphisms interchanges, every maximal clique is trivial.

Proposition 4.4.4. *Not every premonoidal category admits an intrinsic device presentation.*

Proof. By presenting a premonoidal category with a countably infinite number of morphisms \mathbb{N} , in which 0 interferes with every $i \in \mathbb{N} \setminus \{0\}$ but all other pairs of morphisms are independent, then 0 lies in an infinite number of maximal cliques and so the putative set of devices is not finite. \square

4.5 Centralizers in effectful categories

Given an effectful morphism equipped with some set of devices, we can ask for all the processes that interchange with it: intuitively, these are precisely the morphisms not using any device from the set. More generally, for any set of morphisms in an effectful category, we obtain a subcategory of processes interchanging with every morphism in the set.

When an effectful category admits an intrinsic device presentation (Section 4.4), it stratifies into a partial order of centralizers, each of which models a subset of its devices.

Definition 4.5.1. A morphism $f : a \rightarrow b$ in a (strict) effectful category $E : \mathbb{V} \rightarrow \mathbb{C}$ *interchanges* or *commutes* with a morphism $g : c \rightarrow d$ just when the following equations (Figure 4.13) hold in \mathbb{C} .

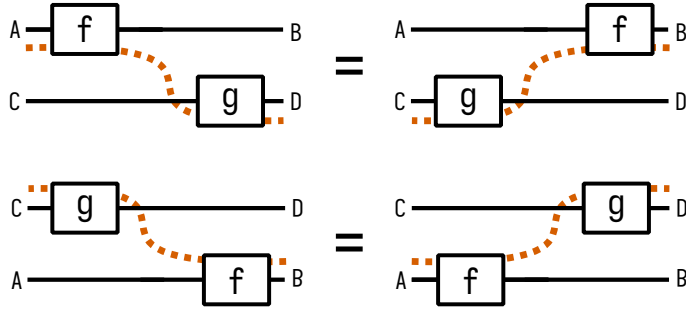
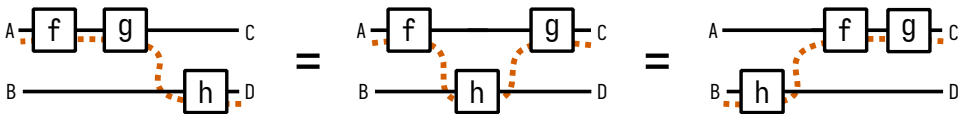


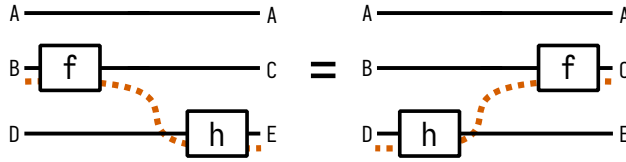
Figure 4.13: f and g interchange.

Proposition 4.5.2. Let $E : \mathbb{V} \rightarrow \mathbb{C}$ be an effectful category, and D a set of morphisms in \mathbb{C} . Define the centralizer \mathbb{C}_D of \mathbb{C} with respect to D to have objects those of \mathbb{C} and morphisms all those of \mathbb{C} interchanging with every morphism of D . The centralizer \mathbb{C}_D is an effectful subcategory over \mathbb{V} .

Proof. Let $f : a \rightarrow b, g : b \rightarrow c$ be morphisms in \mathbb{C}_D , then their composite in \mathbb{C} must also be in \mathbb{C}_D , as witnessed by the following string diagrams, using first the interchanging property of g then f (with the same argument holding for morphisms h tensored on the top),



Furthermore, identities and structural isomorphisms of \mathbb{C} are *central* and hence give the identities and structural isomorphisms of \mathbb{C}_D . We can now define the premonoidal structure on this category. The tensor product of objects is given by that in \mathbb{C} . Likewise we define left and right-whiskerings to be those of \mathbb{C} . This is well-defined, since as witnessed by the following string diagrams, if f is in \mathbb{C}_D then so is $a \triangleleft f$, with a similar argument holding for right-whiskering.



□

Definition 4.5.3. Let \mathbb{C} be an effectful category presented by devices, and let D be a subset of the devices. The device centralizer \mathbb{C}_D with respect to D is the centralizer of \mathbb{C} with respect to the set of morphisms whose devices are contained in D .

Intuitively, the device centralizer \mathbb{C}_D of a set of devices D contains all those processes that do not use any of the devices in D , which leads to the following proposition.

Proposition 4.5.4. *Every effectful category \mathbb{C} with an intrinsic device presentation admits a poset of device centralizers, each corresponding to a subset of the devices of \mathbb{C} , with least element its center, greatest element \mathbb{C} , and $\mathbb{C}_D \leq \mathbb{C}_{D'}$ just when $D \supseteq D'$.*

4.6 Commuting tensor product of effectful categories

In their wide-ranging analysis of the notion of *commutativity*, Garner and López-Franco [37] introduced the *commuting tensor product* of categories enriched in a duoidal category. In the case of one-object categories enriched in the duoidal category of strong profunctors (with respect to composition and convolution), this implies the existence of a notion of commuting tensor product of strong promonads, i.e. of effectful categories (termed *generalized Freyd categories* by Garner and López-Franco). We agree that this notion “may be of interest to computer scientists”; for example, the commuting tensor product of the theory of a memory cell with itself is precisely the theory of two non-interacting memory cells, as sketched in Section 4.2. In this section we show how to construct the commuting tensor product in general, and show that it satisfies the universal property of Garner and López-Franco.

We shall denote the effectful categories corresponding to strong promonads A, B, \dots by calligraphic letters \mathcal{A}, \mathcal{B} . We work with these notions over a fixed monoidal base \mathbb{V} .

Definition 4.6.1 (Garner and López Franco [37], Equation 9.6). A cospan of strong promonads $F : A \rightarrow K \leftarrow B : G$ over \mathbb{V} is a commuting cospan when for each $f \in A(a, a')$ and $g \in B(b, b')$, the following square commutes in the Kleisli category

of K .

$$\begin{array}{ccc}
 a \otimes b & \xrightarrow{Ff \triangleright b} & a' \otimes b \\
 \downarrow a \triangleleft Gg & & \downarrow a' \triangleleft Gg \\
 a \otimes b' & \xrightarrow{Ff \triangleright b'} & a' \otimes b'
 \end{array} \tag{4.1}$$

Example 4.6.2. For disjoint sets of devices D_1, D_2 of an effectful category \mathbb{C} presented by devices, the device centralizers \mathbb{C}_{D_1} and \mathbb{C}_{D_2} , with their inclusions, form a commuting cospan over \mathbb{C} .

Garner and López-Franco characterize the commuting tensor product of duoidally enriched categories via the following universal property:

Definition 4.6.3 (Garner and López Franco [37]). The commuting tensor product $A \odot B$ of \mathcal{V} -categories A and B is the object representing \mathcal{V} -bifunctors from A and B , when it exists. That is, there is a natural isomorphism

$$\mathcal{V}\text{-BiFun}(A, B; -) \cong \mathcal{V}\text{-Cat}(A \odot B; -).$$

In the case of strong promonads considered as one object $\mathcal{V} = \text{StrProf}(\mathbb{V}, \mathbb{V})$ enriched categories, this reduces to the following.

Proposition 4.6.4. *Let A, B be strong promonads over \mathbb{V} . Their commuting tensor product $A \odot B$ is the apex of the initial commuting cospan over A and B in the category of strong promonads over \mathbb{V} .*

$$\begin{array}{ccccc}
 & & A \odot B & & \\
 & \nearrow l & \downarrow q & \nwarrow r & \\
 A & \xrightarrow{f} & K & \xleftarrow{g} & B
 \end{array}$$

We denote the corresponding effectful category by $\mathcal{A} \odot \mathcal{B}$.

Proof. It suffices to recognize that bifunctors from A and B are equivalent to commuting cospans over A and B in the case of one object \mathcal{V} -categories, as remarked by Garner and López Franco [37, §4]. The diagram then follows from the natural isomorphism of Definition 4.6.3. \square

4.6.1 Presenting the commuting tensor product by devices

In this section, we show how to construct the commuting tensor product of effectful categories in a simple manner, using a device presentation, where each factor of the tensor product is assigned a distinct device.

Theorem 4.6.5. *The commuting tensor product $A \odot B$ of two effectful categories $\eta^A : \mathbb{V} \rightarrow \mathcal{A}$ and $\eta^B : \mathbb{V} \rightarrow \mathcal{B}$ over \mathbb{V} admits the device presentation in Figure 4.14.*

$$\begin{array}{ccc}
 \begin{array}{c} A_1 \\ \vdots \\ A_n \end{array} \text{---} \boxed{\alpha} \text{---} \begin{array}{c} A_1' \\ \vdots \\ A_m' \end{array} &
 \begin{array}{c} B_1 \\ \vdots \\ B_n \end{array} \text{---} \boxed{\beta} \text{---} \begin{array}{c} B_1' \\ \vdots \\ B_m' \end{array} &
 \begin{array}{c} V_1 \\ \vdots \\ V_n \end{array} \text{---} \boxed{v} \text{---} \begin{array}{c} V_1' \\ \vdots \\ V_m' \end{array} \\
 \alpha \in \mathbb{A}(A_1 \otimes \dots \otimes A_n; A_1' \otimes \dots \otimes A_m') &
 \beta \in \mathbb{B}(B_1 \otimes \dots \otimes B_n; B_1' \otimes \dots \otimes B_m') &
 v \in \mathbb{V}(V_1 \otimes \dots \otimes V_n; V_1' \otimes \dots \otimes V_m')
 \end{array}$$

$$\begin{array}{c}
 \text{---} \boxed{\gamma_1} \text{---} \boxed{\gamma_2} \text{---} \stackrel{(1)}{=} \text{---} \boxed{\gamma_1; \gamma_2} \text{---} \quad \text{---} \boxed{\text{id}} \text{---} \stackrel{(2)}{=} \text{---} \\
 \text{---} \boxed{\alpha} \text{---} \stackrel{(3)}{=} \text{---} \boxed{\text{id} \otimes \alpha} \text{---} \quad \text{---} \boxed{\alpha} \text{---} \stackrel{(4)}{=} \text{---} \boxed{\alpha \otimes \text{id}} \text{---} \\
 v \text{---} \boxed{v} \text{---} v' \stackrel{(5)}{=} v \text{---} \boxed{\eta(v)} \text{---} v'
 \end{array}$$

Figure 4.14: Presentation of the commuting tensor product of effectful categories by devices: each effectful category becomes a distinct device.

Proof. We show that the strong promonad corresponding to the effectful category presented by Figure 4.14 satisfies the universal property of Proposition 4.6.4. We first describe the cospan $l : A \rightarrow A \odot B \leftarrow B : r$. It will suffice to describe the left leg, as the right is analogous. $l : A \rightarrow A \odot B$ has components $l_{v,v'}$ sending morphisms $v \rightarrow v'$ in \mathcal{A} to the corresponding generator in $A \odot B$. These components are natural as a result of equations (1), (2) and (5), since these allow us to prove the equality in Figure 4.15.

$$\text{---} \boxed{\eta^*(g); \alpha; \eta^*(f)} \text{---} = \text{---} \boxed{\eta^*(g)} \text{---} \boxed{\alpha} \text{---} \boxed{\eta^*(f)} \text{---} = \text{---} \boxed{g} \text{---} \boxed{\alpha} \text{---} \boxed{f} \text{---}$$

Figure 4.15: Naturality of the canonical map $l : A \rightarrow A \odot B$, where α is a morphism of A .

The unit law for promonad morphisms is satisfied as a result of equation (5). The multiplication law for promonad morphisms is satisfied as a result of equations (1) and (2). Equations (3) and (4) ensure that this morphism preserves the strengths. Furthermore, this cospan is commuting (Definition 4.6.1), as witnessed by the equality of string diagrams in Figure 4.16.

We now show that this cospan is the initial commuting cospan. Let $F : A \rightarrow K \leftarrow B : G$ be another commuting cospan. We define the universal morphism of

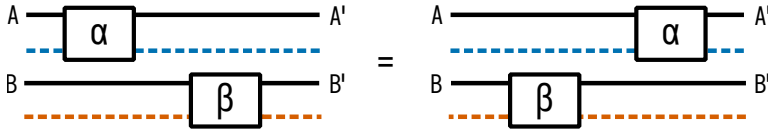


Figure 4.16: Equation (4.1) holds for the universal cospan $l : A \rightarrow A \odot B \leftarrow B : r$.

strong promonads $q : A \odot B \rightarrow K$ on generators as follows

$$\begin{aligned} \alpha : a \rightarrow a' &\mapsto F_{a,a'}(\alpha) \\ \beta : b \rightarrow b' &\mapsto G_{b,b'}(\beta), \\ \nu : v \rightarrow v' &\mapsto F_{v,v'}(\eta_{v,v'}^A(\nu)) = G_{v,v'}(\eta_{v,v'}^B(\nu)) = \eta_{v,v'}^K(\nu) \end{aligned}$$

where the last equalities holds by the unit law of promonad morphisms. We need to show that images of these generators satisfy the corresponding equations of Figure 4.14. Equation (1) follows from the multiplication laws of promonad morphisms, and equation (2) follows from the naturality of the components of F and G . Equations (3) and (4) follow from strength preservation laws of strong promonad morphisms. Equation (5) follows from the unit law of promonad morphisms. Therefore q is a well-defined morphism of strong promonads. The equations $l \circledast q = F$ and $r \circledast q = G$ hold by definition, and q is easily checked to be the unique such morphism making the diagram commute. \square

Remark 4.6.6. We expect that we can give a formal semantics to *string diagrams with devices* via the monoidal category of *pointed bimodular categories*, following the notion of *collages of string diagrams* introduced by Braithwaite and Román [14]. The idea is that effectful categories over \mathbb{V} can be seen as certain categories equipped with compatible left- and right- actions of \mathbb{V} , i.e. as certain *bimodular categories*, as noted by Levy [55].

Chapter 5

Context-Free Languages of String Diagrams

In this chapter, we introduce context-free languages of string diagrams. These include classical context-free languages, but also context-free languages of trees and hypergraphs, when established over appropriate monoidal categories. In more detail,

- We recall the definition of context-free grammars over categories as morphisms of multigraphs. — *Section 5.1*
- We recall the splice-contour adjunction for multicategories. — *Section 5.2*
- We introduce the symmetric multicategory of *diagram contexts* over a monoidal category. — *Section 5.3*
- We introduce *context-free grammars over monoidal categories* along with their key examples. — *Section 5.4*
- We introduce the construction of *raw optics* and its left adjoint, *optical contour*. — *Section 5.5*
- We use this adjunction to prove a representation theorem for context-free languages of string diagrams in terms of regular languages of string diagrams. — *Section 5.6*

The results of this chapter appear in the paper [29].

5.1 Context-free grammars as morphisms of multigraphs

In Chapter 3, we saw how regular grammars are fruitfully thought of as morphisms of directed graphs. In this section, we see that by changing directed graphs to multi-input, single-output multigraphs, we can obtain context-free grammars in a similar fashion. This idea is not original to us; its roots go back to Walters [89], with recent refinement and extension by Melliès and Zeilberger [62, 63]. This point of view, as in the regular case, is simple and powerful. It suggests natural generalizations of context-free grammars, such as we pursue in the central part of this chapter (Section 5.4), and new conceptual tools for reasoning about them, such as the splice-contour adjunction (Sections 5.2 and 5.5).

5.1.1 Context-free languages in monoids and other categories

A rule in a context-free grammar can always be written in the form

$$R \rightarrow w_0 R_1 w_2 \dots R_n w_n,$$

where R, R_i are non-terminals, and w_i are (possibly empty) words over an alphabet Σ . Melliès and Zeilberger [63] noticed that this data may be arranged as an operation $R_1, \dots, R_n \rightarrow R$ in a multigraph over an n -ary operation $w_0 - \dots - w_n$ called a *spliced word*: a word with n gaps, as in Figure 5.1.

More precisely, let us write $|\mathcal{W}\Sigma^*|$ for the multigraph with one object, and whose n -ary operations are spliced words over Σ with n gaps. Context-free grammars over Σ are exactly captured by morphisms of multigraphs $G \rightarrow |\mathcal{W}\Sigma^*|$, where G is a finite multigraph, along with a start symbol given by an object $S \in G$.

The multigraph $|\mathcal{W}\Sigma^*|$ is the underlying multigraph of the following multicategory, which is defined over any category, not necessarily the free monoid Σ^* .

Definition 5.1.1 (Melliès and Zeilberger [63]). The multicategory of spliced arrows, \mathcal{WC} , over a category \mathbb{C} , contains, as objects, pairs of objects of \mathbb{C} , denoted as $\frac{A}{B}$. Its multimorphisms are morphisms of the original category, but with n “gaps” or “holes”, into which other morphisms (with holes) may be *spliced*. More precisely, the multimorphisms of \mathcal{WC} are given by:

$$\mathcal{WC}\left(\frac{A_1}{B_1}, \dots, \frac{A_n}{B_n}; \frac{X}{Y}\right) := \mathbb{C}(X; A_1) \times \prod_{i=1}^{n-1} \mathbb{C}(B_i; A_{i+1}) \times \mathbb{C}(B_n; Y).$$

As a special case, nullary multimorphisms are morphisms of \mathbb{C} , that is $\mathcal{WC}\left(\frac{X}{Y}\right) := \mathbb{C}(X; Y)$. Identities are given by pairs of identities of the original category, multicategorical composition is derived from the composition of the original category.

We can now present a context-free grammar in terms of a morphism of multigraphs from a multigraph of non-terminals to the underlying multigraph of spliced

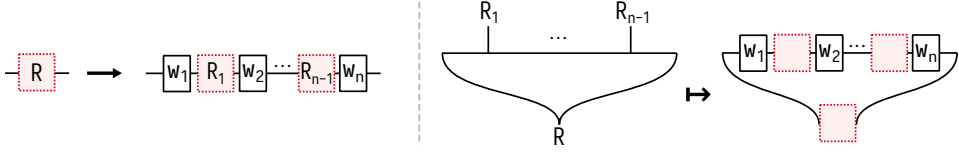


Figure 5.1: (Left) Generic form of a context-free rule. (Right) Context-free rules as a morphism of multigraphs into spliced arrows; here, spliced arrows in a monoid.

arrows, as in Figure 5.1.

Definition 5.1.2 (Melliès and Zeilberger [63]). A context-free grammar of morphisms in a category \mathbb{C} is a morphism of multigraphs $G \rightarrow |\mathcal{WC}|$ and an object S in G (the start symbol).

By the free-forgetful adjunction between multicategories and multigraphs, morphisms $\phi : G \rightarrow |\mathcal{WC}|$ and multifunctors $\widehat{\phi} : \mathcal{F}_\nabla G \rightarrow \mathcal{WC}$ are in bijection. This allows for a slick definition of the language of a grammar, akin to Definition 3.3.2.

Definition 5.1.3 (Melliès and Zeilberger [63]). Let $\mathcal{G} = (\phi : G \rightarrow |\mathcal{WC}|, S)$ be a context-free grammar of morphisms in \mathbb{C} . The language of \mathcal{G} is given by the image of the set of derivations $\mathcal{F}_\nabla G(; S)$ under the multifunctor $\widehat{\phi}$.

Remark 5.1.4. When \mathbb{C} is a finitely generated free monoid considered as a one-object category, then context-free grammars over \mathbb{C} correspond precisely to the classical context-free grammars.

5.2 The splice-contour adjunction: categories and multicategories

An important realization of Melliès and Zeilberger is that the operation of forming the multicategory of spliced arrows in \mathbb{C} has a left adjoint. That is, every multicategory gives rise to a category called the contour of \mathbb{M} , and this contouring operation is left adjoint to splicing.

Definition 5.2.1 (Melliès and Zeilberger [63]). The contour category \mathcal{CM} of a multicategory M is the category presented by the following generators and equations.

- Objects are polarized objects of M : X^L and X^R for every $X \in M_{\text{obj}}$,
- morphisms are $(p, i) : X_i^R \rightarrow X_{i+1}^L$ where $0 \leq i \leq n$ and $p : X_1, \dots, X_n \rightarrow X$ is an operation of M , with $X_0^R := X^L$, $X_{n+1}^L := X^R$,

quotiented by the equations

$$(f \circ_i g, j) = \begin{cases} (f, j) & j < i \\ (f, i) \circ (g, 0) & j = i \\ (g, j - i) & i < j < i + m \\ (g, m) \circ (f, i + 1) & j = i + m \\ (f, j - m + 1) & j > i + m \end{cases}$$

$$(f \circ_i c, j) = \begin{cases} (f, j) & j < i \\ (f, i) \circ (c, 0) \circ (f, i + 1) & j = i \\ (f, j + 1) & j > i \end{cases}$$

whenever they are well-typed, where c is an operation of arity 0.

These equations simply say that contours of composites come from composites of contours in the expected way.

Proposition 5.2.2 (Melliès and Zeilberger [63]). *Contour and splice extend to functors $\mathcal{C} : \text{MultiCat} \rightarrow \text{Cat}$ and $\mathcal{W} : \text{Cat} \rightarrow \text{MultiCat}$, and contour is left adjoint to splice, $\mathcal{C} \dashv \mathcal{W}$.*

Contours give a conceptual replacement for Dyck languages in the classical theory of context-free languages: they linearize the shape of derivation trees.

In Section 5.5.2, we define a new contour of multicategories which we call the *optical contour*; we shall use it to prove a representation theorem for languages of string diagrams (Theorem 5.6.13), inspired by the generalized Chomsky-Schützenberger representation theorem proved by Melliès and Zeilberger.

Remark 5.2.3. In work with Román and Hefford [27], we refined the contour-splice adjunction from multicategories to promonoidal categories: the splice of a category has the structure of a *promonoidal category* (which may be viewed as a multicategory with the extra property of *malleability* [75]), and every promonoidal category gives rise to a contour category, whose construction slightly simplifies that of Definition 5.2.1.

Remark 5.2.4. Walters' [89] representation of context-free grammars using multigraphs is slightly different from that of Melliès and Zeilberger, using instead the free category with finite products on a multigraph, following his investigation of its 2-categorical universal property [91]. In particular, it forces grammars to be in a normal form similar to Chomsky normal form.

5.3 Diagram contexts

In order to lift Definition 5.1.2 to categories with monoidal structure, we need an appropriate multicategory of spliced arrows in a monoidal category. Of course, the multicategory of spliced arrows is defined for any category, regardless of whether it may carry any monoidal structures. However, in a monoidal category it is natural to consider more general kinds of holes than provided by the spliced arrows construction.

Figure 5.2 illustrates this more general kind of hole, in which the hole may be surrounded by strings. This kind of monoidal morphism with a hole has appeared in the literature under various names, such as *optics* [18, 76], *contexts*, or *wiring diagrams* [69].

Moreover, it becomes necessary to consider these more general holes in order to naturally capture context-free languages of hypergraphs and trees as examples of context-free monoidal languages.

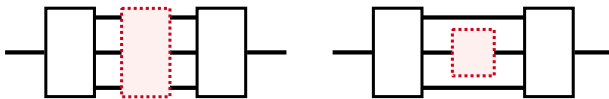


Figure 5.2: (Left) A spliced arrow is a tuple of morphisms. (Right) In a monoidal category, there is the possibility of more general holes, which do not split a morphism into disjoint pieces.

Substituting another diagram context inside a hole induces a symmetric multicategorical structure on the diagrams: symmetry means that we do not distinguish the specific order in which the holes appear. This allows us to avoid declaring a particular ordering of holes when defining a context-free monoidal grammar. The device that allows us to avoid declaring a particular ordering is *shufflings*; their use in categorical logic is inspired by the work of Shulman [84]. As in Section 4.2, we write $\text{Shuf}(\Gamma, \Delta)$ for the set of shufflings of the context Γ into Δ , that is, the set of lists of length $|\Gamma| + |\Delta|$, containing the elements of each list while preserving their relative orders.

For instance, if $\Gamma = [x, y, z]$ and $\Delta = [u, v]$, then $[x, u, y, z, v]$, is an element of $\text{Shuf}(\Gamma, \Delta)$, but not $[y, u, z, x, v]$. The rules for diagram contexts introduce a shuffling every time it mixes two contexts: this way, if a term was derived by combining two contexts, we can always reorder these contexts by a shuffling. For instance, the term $[u, v] \vdash [u] ; [v]$ was derived from composing the axioms $[u] \vdash [u]$ and $[v] \vdash [v]$; by choosing a different shuffling, we can also derive the term $[v, u] \vdash [u] ; [v]$. Let us now formally introduce diagram contexts.

Definition 5.3.1. Diagram contexts $[P]$ over a monoidal graph, \mathcal{P} , are terms-in-context given inductively by the following type theory. In these rules, the variables

$A, B, C, \dots \in \mathcal{P}_{obj}^*$ stand for lists of objects in the monoidal graph.

IDENTITY	GENERATOR	HOLE
$\frac{}{\vdash \text{id} : \frac{A}{A}}$	$\frac{}{\vdash f : \frac{X_1, \dots, X_n}{Y_1, \dots, Y_m}}$	$\frac{}{\boxed{x} : \frac{A}{B} \vdash \boxed{x} : \frac{A}{B}}$
SEQUENTIAL		
$\Gamma \vdash t_1 : \frac{A}{B}$	$\Delta \vdash t_2 : \frac{B}{C}$	$\Psi \in \text{Shuf}(\Gamma; \Delta)$
$\Psi \vdash t_1 \circledast t_2 : \frac{A}{C}$		
PARALLEL		
$\Gamma \vdash t_1 : \frac{A_1}{B_1}$	$\Delta \vdash t_2 : \frac{A_2}{B_2}$	$\Psi \in \text{Shuf}(\Gamma; \Delta)$
$\Psi \vdash t_1 \otimes t_2 : \frac{A_1 \uparrow A_2}{B_1 \uparrow B_2}$		

where \uparrow denotes list concatenation. Every term in a given context has a unique derivation. We consider terms up to α -equivalence and we impose the following equations over the terms whenever they are constructed over the same context: $(t_1 \circledast t_2) \circledast t_3 = t_1 \circledast (t_2 \circledast t_3)$; $t \circledast \text{id} = t$; $t_1 \otimes (t_2 \otimes t_3) = (t_1 \otimes t_2) \otimes t_3$; $(t_1 \circledast t_2) \otimes (t_3 \circledast t_4) = (t_1 \otimes t_3) \circledast (t_2 \otimes t_4)$.

Proposition 5.3.2. *Derivable sequents in \mathcal{P} form a multicategory, which we also denote by \mathcal{P} . The objects of the multicategory are pairs $\frac{A}{B}$ of objects \mathcal{P}_{obj}^* , and multimorphisms $\mathcal{P}(\frac{X_1}{Y_1}, \dots, \frac{X_n}{Y_n}; \frac{S}{T})$ are (equivalence classes) of terms in context,*

$$\boxed{x_1} : \frac{X_1}{Y_1}, \dots, \boxed{x_n} : \frac{X_n}{Y_n} \vdash t : \frac{S}{T}$$

and composition is given by substitution.

Proposition 5.3.3. *The multicategory of derivable sequents in the theory of diagram contexts is symmetric. In logical terms, exchange is admissible in the theory of diagram contexts.*

Proof. Assume we derived a term $\Gamma, \boxed{u}, \boxed{v}, \Delta \vdash t$; let us show we could also derive $\Gamma, \boxed{v}, \boxed{u}, \Delta \vdash t$. We proceed by structural induction, recurring to the first term where the variables \boxed{u} and \boxed{v} appeared at two sides of the rule: this rule must have been of the form $t_1 \circledast t_2$ or $t_1 \otimes t_2$ for $\Gamma_1, \boxed{u}, \Delta_1 \vdash t_1$ and $\Gamma_2, \boxed{v}, \Delta_2 \vdash t_2$, where $\Gamma \in \text{Shuf}(\Gamma_1; \Gamma_2)$ and $\Delta \in \text{Shuf}(\Delta_1; \Delta_2)$. In that case, we can deduce that $\Gamma, \boxed{v}, \boxed{u}, \Delta \in \text{Shuf}(\Gamma_1, \boxed{u}, \Delta_1; \Gamma_2, \boxed{v}, \Delta_2)$; as a consequence, $\Gamma, \boxed{v}, \boxed{u}, \Delta \vdash t$ can be derived. \square

Proposition 5.3.4. *Derivable sequents in the theory of diagram contexts over a monoidal graph \mathcal{P} form the free strict monoidal category over the monoidal graph extended with special “hole” generators, $\mathcal{P} + \{h_{A,B} : A \rightarrow B \mid A, B \in \mathcal{P}_{obj}^*\}$. Derivable sequents over the empty context form the free strict monoidal category*

over the monoidal graph \mathcal{P} . Moreover, there exists a symmetric multifunctor $i : |\mathcal{F}_\otimes \mathcal{P}| \rightarrow \mathcal{P}$ interpreting each monoidal term as its derivable sequent.

Proof. We proceed by structural induction. We first note that the three axioms of the logic correspond to terms of the free strict monoidal category over the monoidal graph $\mathcal{P} + \{h_{A,B} \mid A, B \in \mathcal{P}_{obj}^*\}$. The first corresponds to identities, the second corresponds to generators, and the third, when employed with types A and B , corresponds to the additional generator $h_{A,B}$. We then note that the two binary rules correspond to sequential and parallel composition, thus obtaining the classical algebraic theory of monoidal terms over the monoidal graph $\mathcal{P} + \{h_{A,B} \mid A, B \in \mathcal{P}_{obj}^*\}$.

Quotienting over the equations of monoidal categories, as we do when we impose the equations of the theory of diagram contexts, recovers the free strict monoidal category: in a tautological sense, the free strict monoidal category is precisely the one generated by the operations of a monoidal category quotiented by the axioms of a monoidal category. This contrasts sharply with a much more interesting description of the free strict monoidal category: that using string diagrams. As both are exhibited as satisfying the same universal property, they are necessarily equivalent.

As a particular case, a derivable sequent over the empty context must, by structural induction, avoid any use of the holes. As a consequence of the previous reasoning, it is generated from the monoidal graph \mathcal{P} and it must be a morphism of the free strict monoidal category.

Finally, the symmetric multifunctor can be described by structural induction: it preserves identities, holes, sequential and parallel compositions, and it sends each monoidal term with no holes $h \in |\mathcal{F}_\otimes \mathcal{P}|$ to its derivation under the empty context, $h \in \mathcal{P}$. \square

Definition 5.3.5. The theory of diagram contexts over a finitely presented monoidal category is the theory of diagram contexts over the generators, quotiented by the least congruence on terms generated by the equations of the presentation.

Proposition 5.3.6. Taking diagram contexts in a monoidal category or monoidal graph extends to functors $\square : \text{MonCat} \rightarrow \text{MultiCat}$ and $\square : \text{MonGraph} \rightarrow \text{MultiGraph}$, which moreover commute with the free multicategory \mathcal{F}_∇ and free monoidal category functors \mathcal{F}_\otimes .

Remark 5.3.7. This multicategory is similar to the operad of directed, acyclic wiring diagrams introduced by Patterson, Spivak and Vagner [69], whose operations are generic morphism shapes, rather than holes in a specific monoidal category. Moreover, we expect that this multicategory should be recoverable as a canonical underlying multicategory of the produoidal category of contexts introduced by Hefford, Román and the present author [27].

At this point, the reader may doubt that the formation of diagram contexts has a left adjoint similar to the contour functor for spliced arrows. Indeed, in order to recover a left adjoint, we shall need to introduce another multicategory of diagrams which we call raw optics. This technical device will allow us to prove our main theorem (Theorem 5.6.13). However, let us first see the definition of context-free monoidal grammar, and some examples.

5.4 Context-free monoidal grammars

We now have the ingredients necessary to define context-free monoidal grammars. Intuitively, such a grammar specifies a language of string diagrams via a collection of rewrites between *diagram contexts*, where the non-terminals of a context-free grammar are now (labelled) *holes* in a diagram (e.g. Figure 5.3). Our definition is entirely analogous to Definition 5.1.2, but using our new symmetric multicategory of diagram contexts in a monoidal category, instead of spliced arrows.

Definition 5.4.1. A context-free monoidal grammar over a strict monoidal category (\mathbb{C}, \otimes, I) is a morphism of symmetric multigraphs $\Psi : \mathcal{G} \rightarrow |\mathbb{C}|$, into the underlying multigraph of diagram contexts in \mathbb{C} , where \mathcal{G} is finite, and a start object $S \in \mathcal{G}$.

We shall again use the notation $S \sqsubset \frac{A}{B}$ to indicate that $\Psi(S) = \frac{A}{B}$. A morphism of symmetric multigraphs $\Psi : \mathcal{G} \rightarrow |\mathbb{C}|$ defining a grammar uniquely determines, via the free-forgetful adjunction, a symmetric multifunctor $\hat{\Psi} : \mathcal{F}_{\nabla} \mathcal{G} \rightarrow |\mathbb{C}|$, mapping (closed) derivations to morphisms of \mathbb{C} . The language of a grammar is then defined analogously to Definition 5.1.3:

Definition 5.4.2. Let $(\Psi : \mathcal{G} \rightarrow |\mathbb{C}|, S \sqsubset \frac{A}{B})$ be a context-free monoidal grammar. The language of Ψ is the set of morphisms in $\mathbb{C}(A; B)$ given by the image under $\hat{\Psi}$ of the set of derivations $\mathcal{F}_{\nabla} \mathcal{G}(; S)$. A set of morphisms L in \mathbb{C} is a context-free monoidal language if and only if there exists a context-free monoidal grammar whose language is L .

Let us see some examples.

Example 5.4.3 (Classical context-free languages). Every context-free monoidal grammar of the following form is equivalent to a classical context-free grammar of words. Let Γ be a single-sorted finite monoidal graph whose generators are all of arity and coarity 1. Then context-free monoidal grammars over $\mathcal{F}_{\otimes} \Gamma$ with a start symbol $\phi(S) \sqsubset \frac{1}{1}$ are context-free grammars of words over Γ . Figure 5.3 gives the classical example of balanced parentheses. Similarly, every context-free grammar of words may be encoded as a context-free monoidal grammar in this way.

Example 5.4.4 (Context-free tree grammars). *Context-free tree grammars* [39, 82] are defined over *ranked alphabets* of terminals and non-terminals, which amount

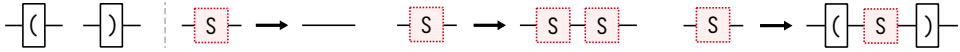


Figure 5.3: Balanced parentheses as a context-free monoidal grammar over the monoidal graph Γ (left).

to monoidal graphs in which the generators have arbitrary arity (the rank) and coarity 1. Productions have the form $A(x_1, \dots, x_m) \rightarrow t$ where the left hand side is a non-terminal of rank m whose frontier is labelled by the variables x_i in order, and whose right hand side is a tree t built from terminals and non-terminals, and whose frontier is labelled by variables from the set $\{x_1, \dots, x_m\}$. Note that t may use the variables non-linearly.

For example, let S be a non-terminal with arity 0, A a non-terminal with arity 2, f a terminal of arity 2, and x a terminal of arity 0 (a leaf). Then a possible rule over these generators is $A(x_1, x_2) \rightarrow f(x_1, A(x_1, x_2))$, where x_1 appears non-linearly. In order to allow such non-linear use of variables in a context-free monoidal grammar, we can consider the free *cartesian* category over Γ . In terms of string diagrams, this amounts to introducing new generators for copying ($\leftarrow \sqsubset$) and deleting variables ($\rightarrow \sqsupset$), which are natural and coherent in the sense of Definition 3.8.1.

Let Γ be a monoidal graph in which generators have arbitrary arity, and coarity 1, as above. Context-free monoidal grammars over the free cartesian category on Γ , with a start symbol $S \sqsubset_1^0$ are context-free tree grammars. In Figure 5.4 we extend the above data to a full example. Note that by allowing start symbols $S \sqsubset_n^0$, we can produce forests of n trees.

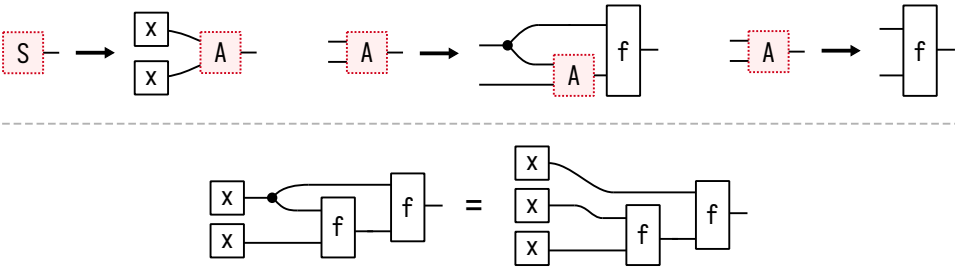


Figure 5.4: Example of a context-free tree grammar as a context-free monoidal grammar. The string diagrams at the bottom are *equal* in the free *cartesian* category over the monoidal graph of terminals.

Example 5.4.5. A language of (directed) series-parallel graphs is given by the context-free monoidal grammar of Figure 5.5.

Example 5.4.6 (Unbraids). We return to the language of unbraids suggested in Definition 3.4.4. Take the grammar over the over- and under-braiding monoidal



Figure 5.5: A context-free monoidal grammar of series-parallel graphs, with start symbol S , over the alphabet containing a “node”, “branch”, and “merge” generators.

graph depicted in Figure 5.6, with start symbol $S \sqsubset \frac{2}{2}$. The language of this grammar consists of unbraids on two strings.

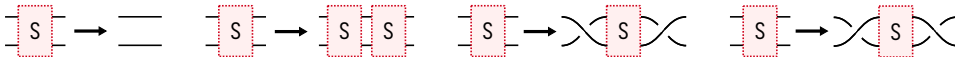


Figure 5.6: A context-free monoidal grammar of unbraids, with start symbol S .

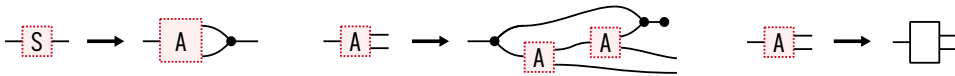


Figure 5.7: A hypergraph grammar for simple *control flow graphs* with branching and looping, as a context-free monoidal grammar. Based on Habel [42, Example 3.3].

Example 5.4.7 (Hyperedge-replacement grammars). Hyperedge-replacement (HR) grammars are a kind of *context-free graph grammar* [34]. We consider HR grammars in *normal form* in the sense of Habel [42, Theorem 4.1]. A production $N \rightarrow R$ of an HR grammar has N a non-terminal with arity and coarity, and R a hypergraph with the same arity and coarity¹, whose hyperedges are labelled by some finite set of terminals and non-terminals.

Just as trees are morphisms in free *cartesian* monoidal categories (Example 5.4.4), hypergraphs are the morphisms of monoidal categories equipped with extra structure, known as *hypergraph categories* [79, 11, 35]. Generators in a monoidal graph are exactly directed hyperedges. The extra structure in a hypergraph category (viz. special commutative Frobenius monoids on every object) amounts to a combinatorial encoding of patterns of wiring between nodes.

Let Γ be a monoidal graph of terminal hyperedges, G a multigraph of non-terminal rules, and $S \in G$ a start symbol. Then context-free monoidal grammars $(G \rightarrow |\mathbf{Hyp}[\Gamma]|, S)$ over the free hypergraph category on Γ are exactly hyperedge replacement grammars over Γ (e.g. Figure 5.7). A hole in a morphism in $\mathbf{Hyp}[\Gamma]$ is a placeholder for an (n, m) hyperedge, the grammar labels these holes by non-terminals, and composition corresponds to hyperedge replacement.

¹A *multi-pointed hypergraph* in Habel’s terminology.

Remark 5.4.8. In order to exhibit regular monoidal grammars as context-free monoidal grammars, we need to equip our symmetric multicategories with the following extra structure, mildly generalizing the class of context-free monoidal languages. First note that symmetric multicategories of diagram contexts $\boxed{\mathbb{C}}$ have a monoid of objects, where $\frac{A}{B} \otimes \frac{C}{D} := \frac{A \otimes C}{B \otimes D}$, and $\frac{I}{I}$ provides the unit. Further, it admits an operation

$$\boxed{\mathbb{C}}(A_1 \cdots A_n, X; Y) \times \boxed{\mathbb{C}}(C_1 \cdots C_m; Z; W) \rightarrow \boxed{\mathbb{C}}(A_1 \cdots A_n, C_1 \cdots C_m; X \otimes Z; Y \otimes W),$$

intuitively corresponding to juxtaposition of diagram contexts. To our knowledge, this structure has not been identified in the literature: in particular, it differs from the concept of *monoidal multicategory* in that it allows combinations of operations of different arities. Instead of taking the free symmetric multicategory on a multigraph G of non-terminals, we must then take the free symmetric multicategory equipped with such extra structure, and morphisms preserving it. General start symbols (and left-hand sides of rules in general) will then be lists $S_1 \otimes \dots \otimes S_n$, and a rule $A_1 \otimes \dots \otimes A_n \xrightarrow{\gamma} B_1 \otimes \dots \otimes B_m$ in a regular monoidal grammar can be represented as a generator $A_1 \otimes \dots \otimes A_n \rightarrow B_1 \otimes \dots \otimes B_m$ over a diagram context $\text{id}_n \ ; \ (\boxed{x_1} \otimes \dots \otimes \boxed{x_n}) \ ; \ \gamma$ (left-linear) or $\gamma \ ; \ (\boxed{x_1} \otimes \dots \otimes \boxed{x_m}) \ ; \ \text{id}_m$ (right-linear).

We leave the rigorous development of this extra structure to future work. Note however, that this mild generalization of context-free monoidal grammars leaves our main result (Theorem 5.6.13) intact: the only difference is that the regular monoidal languages arising in that theorem will generally be of the form

$$S_1^L \otimes \dots \otimes S_n^L \rightarrow S_1^R \otimes \dots \otimes S_n^R.$$

5.5 Optical contour: left adjoint to diagram contexts

Unlike the spliced arrow construction of Melliès and Zeilberger (Definition 5.1.1), we do not expect the formation of diagram contexts to have a left adjoint. The intuitive reason for this is the quotienting built into diagram contexts: they may be seen as *equivalence classes* of tuples of monoidal morphisms under the quotienting of a coend. On the other hand, like the contour operation of Melliès and Zeilberger, we wish to describe our optical contour as being generated by tuples of morphisms. We must therefore first conduct a dissection of diagram contexts into *raw optics*.

5.5.1 The multicategory of raw optics

A raw optic is a tuple of morphisms obtained by cutting a diagram context into a sequence of disjoint pieces. In Section 5.5.2 we shall see that raw optics has a left adjoint, the optical contour, and this will be enough to prove our representation

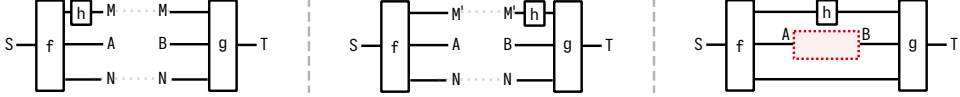


Figure 5.8: Two raw optics (left, centre) in $\text{ROpt}[\mathbb{C}](\frac{A}{B}; \frac{S}{T})$ which quotient to the same diagram context. Note that a raw optic is not the same as a spliced arrow: the types M, N must match.

theorem (Theorem 5.6.13).

Definition 5.5.1. The multicategory of raw optics over a strict monoidal category \mathbb{C} , denoted $\text{ROpt}[\mathbb{C}]$, is defined to have, as objects, pairs $\frac{A}{B}$ of objects of \mathbb{C} , and, as its set of multimorphisms, $\text{ROpt}[\mathbb{C}](\frac{A_1}{B_1}, \dots, \frac{A_n}{B_n}; \frac{S}{T})$, the following set

$$\sum_{M_i, N_i \in \mathbb{C}} \mathbb{C}(S; M_1 A_1 N_1) \times \prod_{i=1}^{n-1} \mathbb{C}(M_i B_i N_i; M_{i+1} A_{i+1} N_{i+1}) \times \mathbb{C}(M_n B_n N_n; T).$$

As a special case, $\text{ROpt}[\mathbb{C}](; \frac{S}{T}) := \mathbb{C}(S; T)$. In other words, a multimorphism, from $\frac{A_1}{B_1}, \dots, \frac{A_n}{B_n}$ to $\frac{S}{T}$, consists of two families of objects, M_1, \dots, M_n and N_1, \dots, N_n , and a family of functions, (f_0, \dots, f_n) , with types $f_0 : S \rightarrow M_1 \otimes A_1 \otimes N_1$; with $f_i : M_i \otimes B_i \otimes N_i \rightarrow M_{i+1} \otimes A_{i+1} \otimes N_{i+1}$; for each $1 \leq i \leq n-1$; and $f_n : M_n \otimes B_n \otimes N_n \rightarrow T$. In the special nullary case, we have a single morphism $f_0 : S \rightarrow T$.

Identities are given by pairs $(\text{id}_A, \text{id}_B)$. Given two raw optics $f = (f_0, \dots, f_n)$ and $g = (g_0, \dots, g_m)$, their composition is defined by

$$f \circledast_i g := (g_0, \dots, g_i \circledast (\text{id} \otimes f_0 \otimes \text{id}), \dots, \text{id} \otimes f_i \otimes \text{id}, \dots, (\text{id} \otimes f_n \otimes \text{id}) \circledast g_{i+1}, \dots, g_m).$$

Every raw optic can be glued into a diagram context, as illustrated in Figure 5.8. More precisely we have,

Proposition 5.5.2. *There is an identity on objects multifunctor $q : \text{ROpt}[\mathbb{C}] \rightarrow \boxed{\mathbb{C}}$ mapping each raw optic to its corresponding diagram context. Equivalently, there is an identity-on-objects symmetric multifunctor $q^* : \text{clique}(\text{ROpt}[\mathbb{C}]) \rightarrow \boxed{\mathbb{C}}$; this symmetric multifunctor is full.*

Proof. The multifunctor q maps a tuple $(f_1 : S \rightarrow M_1 X_1 N_1, f_2 : M_1 Y_1 N_1 \rightarrow M_2 X_2 N_2, \dots, f_n : M_{n-1} Y_{n-1} N_{n-1} \rightarrow T)$ to the derivable sequent

$$\boxed{x_1} : \frac{X_1}{Y_1}, \dots, \boxed{x_{n-1}} : \frac{X_{n-1}}{Y_{n-1}} \vdash f_1 \circledast (M_1 \otimes \boxed{x_1} \otimes N_1) \circledast f_2 \circledast \dots \circledast (M_{n-1} \otimes \boxed{x_{n-1}} \otimes N_{n-1}) \circledast f_n : \frac{S}{T}.$$

Functoriality follows from the unitality, associativity and interchange equations of strict monoidal categories. \square

Proposition 5.5.3. *Raw optics extends to a functor $\text{ROpt} : \text{MonCat} \rightarrow \text{MultiCat}$.*

5.5.2 Optical contour of a multicategory

We now introduce the left adjoint to the formation of raw optics, which we call the *optical contour* of a multicategory. The difference from the contour recalled in Definition 5.2.1 is that additional objects M_i, N_i are introduced which keep track of strings that might surround holes, and this gives rise to a strict monoidal category: the contour category of Definition 5.2.1 does not have a monoidal structure.

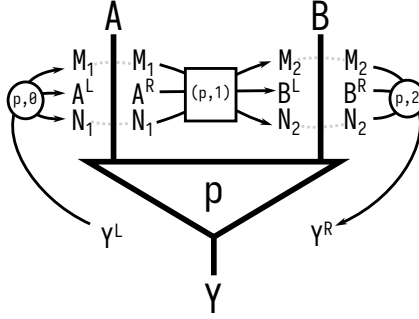


Figure 5.9: A multimorphism $p \in \mathbb{M}(A, B; Y)$ and its three sectors given by optical contour: $(p, 0) : Y^L \rightarrow M_1 \otimes A^L \otimes N_1$, $(p, 1) : M_1 \otimes A^R \otimes N_1 \rightarrow M_2 \otimes B^L \otimes N_2$, $(p, 2) : M_2 \otimes B^R \otimes N_2 \rightarrow Y^R$.

Definition 5.5.4. Let \mathbb{M} be a multicategory. Its optical contour, \mathcal{CM} , is the strict monoidal category presented by a monoidal graph whose generators are given by *contouring* multimorphisms in \mathbb{M} . Each multimorphism gives rise to a set of generators for the monoidal category \mathcal{CM} – its set of sectors, as in Figure 5.9.

Explicitly, for each object $A \in \mathbb{M}$, the optical contour \mathcal{CM} contains a left polarized, A^L , and a right polarized, A^R , version of the object. Additionally, for each multimorphism $f \in \mathbb{M}(X_1, \dots, X_n; Y)$, there exists a family of objects $M_1^f, \dots, M_n^f, N_1^f, \dots, N_n^f$, whose superscripts we omit when they are clear from context. The morphisms are given by the following generators. For each $f \in \mathbb{M}(X_1, \dots, X_n; Y)$, we consider the following $n + 1$ generators:

$$\begin{aligned} (f, 0) &: Y^L \rightarrow M_1^f \otimes X_1^L \otimes N_1^f, \\ (f, i) &: M_i^f \otimes X_i^R \otimes N_i^f \rightarrow M_{i+1}^f \otimes X_{i+1}^L \otimes N_{i+1}^f, \text{ for } 1 \leq i \leq n-1, \text{ and} \\ (f, n) &: M_n^f \otimes X_n^R \otimes N_n^f \rightarrow Y^R. \end{aligned}$$

In particular, for a nullary multimorphism $f \in \mathbb{M}(); Y$, we consider a generator $(f, 0) : Y^L \rightarrow Y^R$. Further, we ask for the following equations which ensure that the optical contour preserves identities and composition: for all $x \in \mathbb{M}$, $(\text{id}_X, 0) = \text{id}_{X^L}$, $(\text{id}_X, 1) = \text{id}_{X^R}$ with $M_1^{\text{id}_X} = N_1^{\text{id}_X} = I$; and given any $f \in \mathbb{M}(X_1, \dots, X_n; Y_i)$

and $g \in \mathbb{M}(Y_1, \dots, Y_m; Z)$,

$$(f \circledast_i g, j) = \begin{cases} (g, j) & j < i, \text{ with } M_j^{f \circledast_i g} = M_j^g, N_j^{f \circledast_i g} = N_j^g \\ (g, i) \circledast (\text{id} \otimes (f, 0) \otimes \text{id}) & j = i, \text{ with } M_i^{f \circledast_i g} = M_i^g \otimes M_0^f \\ \text{id}_{M_i^g} \otimes (f, j - i) \otimes \text{id}_{N_i^g} & i < j < i + n, \text{ with } M_j^{f \circledast_i g} = M_i^g \otimes M_{j-i}^f \\ (\text{id} \otimes (f, n) \otimes \text{id}) \circledast (g, i + 1) & j = i + n + 1, \text{ with } M_j^{f \circledast_i g} = M_i^g \otimes M_n^f \\ (g, j - n) & j > i + n + 1 \text{ with } M_j^{f \circledast_i g} = M_{j-n}^g. \end{cases}$$

In particular, when $f \in \mathbb{M}(\cdot; Y_i)$ is nullary, $(f \circledast_i g, 0) = g_i \circledast f_0 \circledast g_{i+1}$.

Proposition 5.5.5. *Optical contour extends to a functor $\mathcal{C} : \text{MultiCat} \rightarrow \text{MonCat}$.*

Theorem 5.5.6. *Optical contour is left adjoint to raw optics; there exists an adjunction $(\mathcal{C} \dashv \text{ROpt}) : \text{MonCat} \rightarrow \text{MultiCat}$.*

Proof. Let \mathbb{C} be a strict monoidal category and let \mathbb{M} be a multicategory. We will show that there is a bijection between strict monoidal functors $\mathcal{C}\mathbb{M} \rightarrow \mathbb{C}$ and multifunctors $\mathbb{M} \rightarrow \text{ROpt}[\mathbb{C}]$.

- The objects of $\text{ROpt}[\mathbb{C}]$ are pairs of objects. Mapping an object of the multicategory $X \in \mathbb{M}$ to a pair of objects is the same as mapping two objects, X^L and X^R , to the objects of the category \mathbb{C} .
- Mapping a multimorphism $f \in \mathbb{M}(X_1, \dots, X_n; Y)$ to the multicategory of raw optics consists of choosing a family of functions (f_0, \dots, f_n) together with two families of objects M_1, \dots, M_n and N_1, \dots, N_n . This is the same choice we need to map each one of the components of the contour of $f \in \mathbb{M}(X_1, \dots, X_n; Y)$ to that exact family of functions.

That is, we have only checked that, by construction, the maps out of the contour correspond with multifunctors to raw optics. The adjunction remains conceptually interesting because it links two concepts that have different conceptual interpretations, even if, in essence, one has been defined as the adjoint to the other. \square

Lemma 5.5.7. *The following square of adjunctions commutes up to isomorphism.*

$$\begin{array}{ccc} \text{MonCat} & \xleftarrow{\mathcal{C}} & \text{MultiCat} \\ \mathcal{F}_{\otimes} \uparrow \downarrow \mathcal{U}_{\otimes} & \xrightarrow{\text{ROpt}} & \mathcal{F}_{\vee} \uparrow \downarrow \mathcal{U}_{\vee} \\ \text{MonGraph} & \xleftarrow{\mathcal{C}_G} & \text{MultiGraph} \\ & \xrightarrow{\text{ROpt}_G} & \end{array}$$

Proof. It suffices to verify that $\mathcal{U}_{\otimes} \circ \text{ROpt}_G = \text{ROpt} \circ \mathcal{U}_{\vee}$ by definition. The result then follows from composition and uniqueness of adjunctions up to isomorphism. \square

5.6 Representation theorem for context-free monoidal languages

5.6.1 Chomsky-Schützenberger representation theorem

The Chomsky-Schützenberger representation theorem (Theorem 5.6.2) says that every classical context-free language can be obtained as the image under a homomorphism of the intersection of a Dyck language and a regular language [17]. The significance of this theorem is that it tells us how to represent context-free languages in terms of algebraic combinations of simpler parts. It implies, for example, that in order to parse a context-free language (for example, the syntax of a programming language), it suffices to have a machine that can recognize languages of balanced parentheses, and a finite state automaton.

Definition 5.6.1. The *Dyck alphabet* over an alphabet X is the set, $P_X = \{p_x, q_x\}_{x \in X}$, that contains one “opening parenthesis”, p_x , and one “closing parenthesis”, q_x , for each element $x \in X$. The corresponding Dyck language, $D_X \subseteq P_X^*$, is the set of well-parenthesized words over the Dyck alphabet.

Theorem 5.6.2 (Chomsky-Schützenberger [17]). *A language, $L \subseteq \Sigma^*$, is context-free if and only if there exists a Dyck language over some alphabet, D_X ; a regular language over its Dyck alphabet, $R \subseteq P_X^*$; and a monoid homomorphism, $h: P_X^* \rightarrow \Sigma^*$; such that the language L is the image under h of the intersection between the parenthesis language and the regular language, $L = h(R \cap D_X)$.*

Proof sketch. The main idea is to substitute each one of the production rules of the context-free grammar by a family of regular rules: each family of regular rules will open and close parentheses appropriately so as to ensure that it is applied in the correct order, having the same effect that the original primitive rule would have. \square

Melliès and Zeilberger [63] use their splicing-contour adjunction to give a novel proof of this theorem for context-free languages over categories: the classical version is recovered when the category is a free monoid. The role of the Dyck language, providing linearizations of derivation trees, is taken over by *contours* of derivations.

Definition 5.6.3. A context-free grammar over \mathbb{C} is said to be \mathbb{C} -*chromatic* when its non-terminals are given by pairs of objects of \mathbb{C} .

Theorem 5.6.4 (Melliès and Zeilberger [62]). *Every context-free language of morphisms in a category \mathbb{C} is the functorial image of the intersection of a \mathbb{C} -chromatic context-free contour language with a regular language.*

For monoidal languages, we shall see that the Dyck language is not needed because the information that parentheses encode can be carried instead by tensor

products. In this section, we show that a regular monoidal language of *optical contours* is sufficient to reconstruct the original language. Our main theorem states that *every context-free monoidal language is the image under a monoidal functor of a regular monoidal language* (Theorem 5.6.13). In other words, monoidal automata are sufficient to parse canonical encodings of context-free monoidal languages, where the canonical encoding is given by *optical contours*.

This theorem has the flavour of the following two representation results for the case of context-free languages over categories, which we make explicit in order to contrast them with our result.

Proposition 5.6.5. *Every context-free language over a category is the functorial image of a universal context-free language associated to its multigraph G of non-terminals.*

Proof. Given a context-free grammar $(\mathcal{F}_\nabla G \rightarrow \mathcal{WC}, S)$, we can factor it universally through the unit of the splice-contour adjunction, giving $\mathcal{F}_\nabla G \xrightarrow{\eta_{\mathcal{F}_\nabla G}} \mathcal{WC}\mathcal{F}_\nabla G \rightarrow \mathcal{WC}$. Note that the components of the unit are again context-free grammars, now over the category of contours $\mathcal{C}\mathcal{F}_\nabla G$. \square

Proposition 5.6.6. *Every context-free language over a category is the image of a regular tree language.*

Proof. A regular tree language is specified by a morphism of free multicategories, just as in Example 3.5.5. Now given a context-free grammar $(\mathcal{F}_\nabla G \rightarrow \mathcal{WC}, S)$, the trivial factorization $\mathcal{F}_\nabla G \xrightarrow{\text{id}} \mathcal{F}_\nabla G \rightarrow \mathcal{WC}$ witnesses the proposition. \square

In contrast, our representation theorem says that every context-free monoidal language is the image of a regular monoidal language: unlike Proposition 5.6.6 there is not a “level shift” from words (or morphisms) to trees.

To prove this theorem, our strategy will be to first choose a factoring of a grammar through raw optics, corresponding to fixing a particular ordering of the holes in a diagram. Next, we use the optical contour/raw optics adjunction to produce the required monoidal functor. We must first establish that the aforementioned factoring exists. We encourage the reader to refer to Example 5.6.14 in order to gain a better understanding of some of the following constructions.

5.6.2 Raw representatives of a grammar

Given a morphism of symmetric multigraphs underlying a context-free monoidal grammar, $\phi : G \rightarrow |\mathbb{C}|$, our first step will be to seek a factorization $G \rightarrow |\text{ROpt}[\mathbb{C}]| \rightarrow |\mathbb{C}|$. We call the first morphism in such a factorization a *raw representative* of ϕ . It amounts to choosing, for each rule of the grammar, a particular way of representing it in terms of raw optics. The following lemma tells us that raw representatives exist.

Lemma 5.6.7. *Any morphism of symmetric multigraphs underlying a context-free monoidal grammar, $\phi : G \rightarrow \mathbb{C}$, factors (non-uniquely) through the quotienting of raw optics (Proposition 5.5.2); meaning that there exists some multigraph G' satisfying $G = \text{clique}(G')$, and some morphism $\phi_r : G' \rightarrow |\text{ROpt}[\mathbb{C}]|$, such that $\phi = \text{clique}(\phi_r) \circledast q^*$, where $q^* : \text{clique}(\text{ROpt}[\mathbb{C}]) \rightarrow \mathbb{C}$ is the quotient map from Proposition 5.5.2.*

Proof. This is a consequence of the fact that q^* is full. Given any diagram context, we argue that we can obtain a (non-unique) diagram context of the form of a raw optic

$$t_1 \circledast (\text{id}_{M_1} \otimes x_1 \otimes \text{id}_{N_1}) \circledast t_2 \circledast (\text{id}_{M_2} \otimes x_2 \otimes \text{id}_{N_2}) \circledast \dots \circledast (\text{id}_{M_n} \otimes x_n \otimes \text{id}_{N_n}) \circledast t_{n+1}.$$

Indeed, by structural induction, if the diagram is formed by a hole or a generator, it can be put in raw optic form by adding identities; if the diagram is a composition, we can put both factors in raw optic form and check that their composition is again in raw optic form; if the diagram is a tensoring of two diagrams in raw optic form, we can always apply the interchange law and note that whiskering a raw optic by an object returns again a raw optic.

It is the case that every map $G \rightarrow \text{clique}(H)$ arises as a map $G' \rightarrow H$ for some multigraph G' such that $G = \text{clique}(G')$. Combining both facts, we obtain the desired result. \square

Call the factor $\phi_r : G' \rightarrow |\text{ROpt}[\mathbb{C}]|$ a raw representative of ϕ . It amounts to choosing a fixed ordering of the holes in a diagram context for each rule in the grammar, and a particular splicing into a raw optic. We can also consider the language of ϕ_r , which is exactly that of ϕ .

Lemma 5.6.8. *Let $\mathcal{G} = (\phi, S)$ be a context-free monoidal grammar. Then the language of any raw representative ϕ_r of ϕ (with start symbol S) equals the language of \mathcal{G} . That is, $\phi_r[\mathcal{F}_\nabla G'(; S)] = \phi[\mathcal{F}_\nabla G(; S)]$.*

Proof sketch. A raw representative amounts to choosing a specific ordering of the holes and generators in a diagram context. By definition (Lemma 5.6.7), these quotient to the original diagram contexts. In particular, closed derivations quotient to the same element of \mathbb{C} .

The contour-splice adjunction now gives us that raw representatives are in natural bijection with certain strict monoidal functors.

Lemma 5.6.9. *A raw representative $\phi_r : G' \rightarrow |\text{ROpt}[\mathbb{C}]|$ uniquely determines a strict monoidal functor $I_\phi : \mathcal{F}_\otimes(\mathcal{C}G') \rightarrow \mathbb{C}$.*

Proof. Using the free-forgetful adjunction, the raw representative, ϕ_r , determines a unique multifunctor $\mathcal{F}_\nabla G' \rightarrow \text{ROpt}[\mathbb{C}]$. Using the adjunction of Theorem 5.5.6, this in turn determines a unique monoidal functor $\mathcal{C}(\mathcal{F}_\nabla G') \rightarrow \mathbb{C}$. Finally, using the commutativity of \mathcal{C} with \mathcal{F}_∇ (Lemma 5.5.7), we obtain $I_\phi : \mathcal{F}_\otimes(\mathcal{C}G') \rightarrow \mathbb{C}$. Explicitly, the action of I_ϕ on generators is given by: $A^L \mapsto \pi_1(\phi_r(A))$, $A^R \mapsto \pi_2(\phi_r(A))$, $(f, i) \mapsto \pi_i(\phi_r(f))$, where π are projections. \square

We shall see that this monoidal functor maps a certain regular monoidal language over $\mathcal{C}G$, introduced in the next section, to the language of the original context-free monoidal grammar ϕ . Before doing so, we make good on our promise from Section 3.3.1.

Proposition 5.6.10. *Quasiregular monoidal grammars are at least as expressive as context-free monoidal grammars over free monoidal categories.*

Proof. It suffices to observe that for $\mathbb{C} = \mathcal{F}_\otimes \Gamma$, the monoidal functor of Lemma 5.6.9 corresponds to a quasiregular monoidal grammar $\mathcal{C}G' \rightarrow |\mathcal{F}_\otimes \Gamma|$. \square

5.6.3 Regular representation of a grammar

Given a raw representative of a context-free monoidal grammar, we can consider optical contours of its rules (the domain of the grammar). This gives rise to a regular monoidal grammar which we call the regular representation: this is the grammar whose image will give the language of the original context-free monoidal grammar.

Definition 5.6.11. Let $\mathcal{G} = (\phi : G \rightarrow |\mathbb{C}|, S)$ be a context-free monoidal grammar, and ϕ_r a raw representative with domain G' . Define a regular representation of \mathcal{G} to be the regular monoidal grammar $\mathcal{R} = (\text{id} : \mathcal{C}G' \rightarrow \mathcal{C}G', S^L, S^R)$ over optical contours of G' whose morphism of monoidal graphs is the identity.

Lemma 5.6.12. *Given a multigraph G , there is a bijection between derivations rooted at an object S and optical contours from S^L to S^R , that is $\mathcal{F}_\nabla G(; S) \cong \mathcal{F}_\otimes(\mathcal{C}G)(S^L; S^R)$.*

Proof. Let $d \in \mathcal{F}_\nabla G(; S)$ be a derivation. We define a family of functions $\{C_X : \mathcal{F}_\nabla G(; X) \rightarrow \mathcal{F}_\otimes(\mathcal{C}G)(X^L, X^R)\}_{X \in G}$ by structural recursion. There are two cases: if d is a generating operation $d \in G(; S)$, then $C_S(d) := (d, 0) : S^L \rightarrow S^R$. Otherwise, d is a composite $(p_1, \dots, p_n) \circ g$ where $g \in G(X_1, \dots, X_n; S)$ is a generating operation and $p_i \in \mathcal{F}_\nabla G(; X_i)$, in which case $C_S(d) := (g, 0) \circ C_{X_1}(p_1) \circ (g, 1) \circ \dots \circ C_{X_n}(p_n) \circ (g, n)$.

We define functions C_S^{-1} right-to-left in a similar fashion. Let

$$c \in \mathcal{F}_\otimes(\mathcal{C}G)(S^L; S^R)$$

be an optical contour. If $c = (c', 0)$ is a generating sector then $C_S^{-1}(c) := c'$. Otherwise c is a composite $((g, 0) : S^L \rightarrow M_1 \otimes X_1^L \otimes N_1) \ ; \ c_1 \ ; \ ((g, 1) : M_1 \otimes X_1^R \otimes N_1 \rightarrow M_2 \otimes X_2^L \otimes N_2) \ ; \ \dots \ ; \ c_n \ ; \ ((g, n) : M_1 \otimes X_1^R \otimes N_1 \rightarrow S^R)$ where (g, i) are generating sectors and $c_i \in \mathcal{F}_\otimes(\mathcal{CG})(X_i^L, X_i^R)$, in which case $C_S^{-1}(c) := (C_{X_1}^{-1}(c_1), \dots, C_{X_n}^{-1}(c_n)) \ ; \ g$. It is clear that these functions are mutually inverse and hence form a bijection. \square

5.6.4 Monoidal representation theorem

We are now ready to prove our main theorem.

Theorem 5.6.13. *The language of a context-free monoidal grammar $\mathcal{G} = (\phi : G \rightarrow \boxed{\mathbb{C}}, S)$ equals the image of a regular representation under the monoidal functor I_ϕ of Lemma 5.6.9.*

Proof. By Lemma 5.6.8, the languages $L(\mathcal{G})$ and $L((\phi_r, S))$ are equal for any raw representative ϕ_r of ϕ , where $L((\phi_r, S)) = \phi_r[\mathcal{F}_\nabla G'(; S)]$. It therefore suffices to show that $\phi_r[\mathcal{F}_\nabla G'(; S)] = I_\phi[\mathcal{F}_\otimes(\mathcal{CG}')(S^L; S^R)]$. We show the inclusion left-to-right. Let $d \in \mathcal{F}_\nabla G'(; S)$ be a derivation, and let $C_S(d)$ be the corresponding optical contour given by Lemma 5.6.12. Then by the definition of I_ϕ (Lemma 5.6.9) and C_S , we have $I_\phi(C_S(d)) = \phi_r(d)$. We show the inclusion right-to-left. Let $g \in \mathcal{F}_\otimes(\mathcal{CG}')(S^L; S^R)$ be a contour from S^L to S^R , and let $C_S^{-1}(g)$ be the corresponding derivation given by Lemma 5.6.12. Then just as before we have $\phi_r(C_S^{-1}(g)) = I_\phi(g)$. \square

Theorem 5.6.13 is at first quite surprising, since in comparison with the usual Chomsky-Schützenberger theorem and its generalization to categories [63], one might expect to see an *intersection* of a regular monoidal language and a context-free monoidal language. Instead, this theorem tells us that regular monoidal languages are powerful enough to encode context-free monoidal languages, even while the latter is strictly more expressive than the former. Just as a context-free grammar suffices to specify a programming language which may encode instructions for arbitrary computations, regular monoidal languages can specify arbitrary context-free monoidal languages, with a monoidal functor effecting the “compilation”. The theorem is moreover distinct from the similar sounding Proposition 5.6.6, in which the regular language is a language of *trees*.

Example 5.6.14 (Balanced parentheses). Consider the simplified context-free monoidal grammar for balanced parantheses in Figure 5.10. Such a simple grammar is essentially already given by a raw representative, since we have no choice of ordering of the hole(s) on the right hand sides. The regular representation obtained, i.e. the regular monoidal grammar given by taking optical contours of the rules of the context-free monoidal grammar, is given in Figure 5.11. Finally, Figure 5.12

indicates the action of the strict monoidal functor obtained from the optical contour/raw optics adjunction, whose image is the language of the original grammar. Note the similarity of the regular representation with the grammar introduced in Example 3.3.5. In a sense, this example explains the origin of that grammar.



Figure 5.10: Simplified grammar of balanced parentheses (allowing only one outer pair).

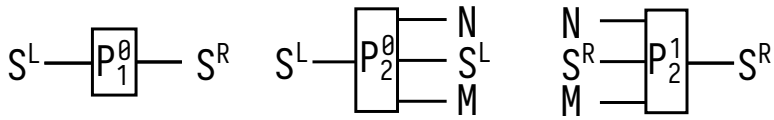


Figure 5.11: Regular representation of the grammar in Figure 5.10.

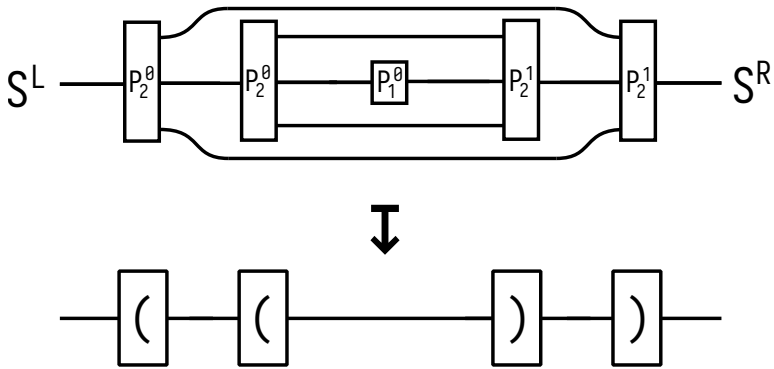


Figure 5.12: Action of the strict monoidal functor obtained from grammar (Lemma 5.6.9).

Chapter 6

Conclusion and Future Work

Classical regular languages are extremely robust, in the sense that they have numerous equivalent definitions. In generalizing regularity to other classes of structure, it is a common theme that analogues of these definitions bifurcate, giving rise to distinct notions of regularity, rationality and (algebraic) recognizability. In this thesis we have investigated both regularity (acceptance by finite-state machine) and algebraic recognizability (acceptance via locally-finite monoidal category), but conjecture that they only coincide for languages of bounded width. Establishing this conjecture is an evident open problem. We have not touched on rationality, that is the definition of monoidal languages by rational (regular) expressions. As mentioned in the introduction, such a line of work exists in the papers of Bossut. However, Bossut’s languages of graphs do not correspond to monoidal languages living in a single hom-set, but rather a (possibly infinite) union of hom-sets, and this is crucial to his account of rational expressions. We have refrained from pushing our definition of monoidal languages to this level of generality for want of examples.

In contrast to regular languages of string diagrams, we have not discussed an operational characterization of context-free languages of string diagrams. In fact, the right notion of “pushdown automaton” accepting languages of morphisms in a category remains an open question for the context-free languages of morphisms defined by Melliès and Zeilberger [63], so this may be a good starting point. Intuitively, one would expect multicategories to be replaced by a similar notion in which composition may only happen at leftmost inputs. The challenge is to identify a good formal account of such structures in relation to the grammars presented

here, just as grammars and automata in the regular case are linked by the passage between fibred and indexed perspectives. It should also be possible to refine the use of multicategories in our definition of context-free monoidal grammars to a notion of multicategory with duoidally structured input. This would allow for the comparison of parsings according to their parallel/sequential cost.

Finally, there is much to be elaborated from the results of Chapter 4, particularly the relation of the commuting tensor product to known tensor products of monads, especially the tensor product of (finitary) monads corresponding to the commuting tensor product of Lawvere theories. We also believe that many robust applications of the notion of *device* can be found, particularly in the domain of computational effects. Tightening the relation with the *functional machine calculus* [3, 4] is a promising direction.

Acknowledgements

Thanks to Pawel for giving me the opportunity to pursue a PhD, for getting me started with many ideas, and encouraging my continued explorations in the topic of this thesis and beyond. It has been a great privilege to be a member of Pawel's group, and to be part of the *Küberneetika Maja* community more broadly. I could not imagine a better group of people to have worked with during these past few years, and I have learned something from all of you. Special thanks to Mario Román, for your generosity and sharing many critical moments at the whiteboard; to Chad Nester for the great dinners, hiking trips, and ongoing collaborations; and to Diana Kessler for sharing in the tribulations of the progress of research and writing.

Thanks also to Tobias Heindel for early encouragement and sharing ideas on the topic of languages in monoidal categories, to Ed Morehouse for many discussions and ideas at an early stage of this work, and to Paolo Brasolin for writing a computer implementation of some ideas of Chapter 3. Thanks to those who reviewed the thesis, whose have comments greatly improved the manuscript, and to the administrative staff who have facilitated the defence, in particular Kristi Ainen. Last but not least, thanks to my parents for supporting me through my change in direction.

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.

Bibliography

- [1] Samson Abramsky and Bob Coecke. “Categorical Quantum Mechanics”. In: *Handbook of Quantum Logic and Quantum Structures*. Ed. by Kurt Engesser, Dov M. Gabbay and Daniel Lehmann. Amsterdam: Elsevier, 2009, pp. 261–323. ISBN: 978-0-444-52869-8. DOI: <https://doi.org/10.1016/B978-0-444-52869-8.50010-4> (cit. on p. 12).
- [2] John C. Baez, Fabrizio Genovese, Jade Master and Michael Shulman. “Categories of nets”. In: *2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. IEEE. 2021, pp. 1–13 (cit. on pp. 14, 17).
- [3] Chris Barrett. “On the Simply-Typed Functional Machine Calculus: Categorical Semantics and Strong Normalisation”. Available at <https://arxiv.org/abs/2305.16073>. PhD thesis. University of Bath, 2023 (cit. on pp. 15, 111).
- [4] Chris Barrett, Willem Heijltjes and Guy McCusker. “The Functional Machine Calculus II: Semantics”. In: *31st EACSL Annual Conference on Computer Science Logic (CSL 2023)*. Ed. by Bartek Klin and Elaine Pimentel. Vol. 252. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023, 10:1–10:18. ISBN: 978-3-95977-264-8. DOI: 10.4230/LIPIcs.CSL.2023.10 (cit. on pp. 15, 111).
- [5] Marek Antoni Bednarczyk. “Categories of asynchronous systems”. PhD thesis. University of Sussex, 1987 (cit. on p. 70).
- [6] Jean Berstel and C Reutenauer. *Rational series and their languages*. en. Eatcs Monographs on Theoretical Computer Science. New York, NY: Springer, Nov. 1988 (cit. on p. 11).
- [7] Achim Blumensath. “Algebraic Language Theory for Eilenberg–Moore Algebras”. In: *Logical Methods in Computer Science* Volume 17, Issue 2 (Apr. 2021). DOI: 10.23638/LMCS-17(2:6)2021 (cit. on p. 11).

- [8] R.F. Blute, J.R.B. Cockett, R.A.G. Seely and T.H. Trimble. “Natural deduction and coherence for weakly distributive categories”. In: *Journal of Pure and Applied Algebra* 113.3 (1996), pp. 229–296. ISSN: 0022-4049. DOI: [https://doi.org/10.1016/0022-4049\(95\)00159-X](https://doi.org/10.1016/0022-4049(95)00159-X) (cit. on p. 17).
- [9] Mikołaj Bojańczyk. “Algebra for Trees”. In: *Handbook of Automata Theory: Volume I*. Ed. by Jean-Éric Pin. Berlin: EMS Press, 2021, pp. 289–355. ISBN: 978-3-98547-002-0 (cit. on p. 11).
- [10] Mikołaj Bojańczyk, Bartek Klin and Julian Salamanca. *Monadic Monadic Second Order Logic*. 2022. DOI: 10.48550/ARXIV.2201.09969 (cit. on pp. 11, 51).
- [11] Filippo Bonchi, Fabio Gadducci, Aleks Kissinger, Pawel Sobocinski and Fabio Zanasi. “String Diagram Rewrite Theory I: Rewriting with Frobenius Structure”. In: *J. ACM* 69.2 (Mar. 2022). ISSN: 0004-5411. DOI: 10.1145/3502719 (cit. on p. 99).
- [12] Filippo Bonchi, Robin Piedeleu, Pawel Sobocinski and Fabio Zanasi. “Graphical Affine Algebra”. In: *34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019, Vancouver, BC, Canada, June 24-27, 2019*. IEEE, 2019, pp. 1–12. DOI: 10.1109/LICS.2019.8785877 (cit. on p. 12).
- [13] Francis Bossut, Max Dauchet and Bruno Warin. “A Kleene Theorem for a Class of Planar Acyclic Graphs”. In: *Inf. Comput.* 117 (Mar. 1995), pp. 251–265. DOI: 10.1006/inco.1995.1043 (cit. on pp. 13, 38).
- [14] Dylan Braithwaite and Mario Román. “Collages of String Diagrams”. In: *Electronic Proceedings in Theoretical Computer Science* 397 (Dec. 2023), pp. 39–53. ISSN: 2075-2180. DOI: 10.4204/eptcs.397.3 (cit. on p. 89).
- [15] H. J. Sander Bruggink and Barbara König. “Recognizable languages of arrows and cospans”. In: *Mathematical Structures in Computer Science* 28.8 (2018), pp. 1290–1332. DOI: 10.1017/S096012951800018X (cit. on p. 14).
- [16] Elizabeth Burroni. “Lois distributives. Applications aux automates stochastiques”. In: *Theory and Applications of Categories* 22.7 (2009), pp. 199–221 (cit. on p. 24).
- [17] Noam Chomsky and Marcel-Paul Schützenberger. “The Algebraic Theory of Context-Free Languages”. In: *Computer Programming and Formal Systems*. Ed. by P. Braffort and D. Hirschberg. Vol. 35. Studies in Logic and the Foundations of Mathematics. Elsevier, 1963, pp. 118–161. DOI: [https://doi.org/10.1016/S0049-237X\(08\)72023-8](https://doi.org/10.1016/S0049-237X(08)72023-8) (cit. on p. 104).
- [18] Bryce Clarke, Derek Elkins, Jeremy Gibbons, Fosco Loregìan, Bartosz Milewski, Emily Pillmore and Mario Román. “Profunctor optics, a categorical update”. In: *CoRR* abs/2001.07488 (2020). arXiv: 2001.07488 (cit. on p. 94).

- [19] J.R.B. Cockett and Stephen Lack. “Restriction categories I: categories of partial maps”. In: *Theoretical Computer Science* 270.1 (2002), pp. 223–259. ISSN: 0304-3975. DOI: [https://doi.org/10.1016/S0304-3975\(00\)00382-0](https://doi.org/10.1016/S0304-3975(00)00382-0) (cit. on pp. 55, 57).
- [20] J.R.B. Cockett and Stephen Lack. “Restriction categories III: colimits, partial limits and extensivity”. In: *Mathematical Structures in Computer Science* 17.4 (2007), pp. 775–817. DOI: [10.1017/S0960129507006056](https://doi.org/10.1017/S0960129507006056) (cit. on p. 55).
- [21] Thomas Colcombet and Daniela Petrişan. “Automata Minimization: a Functorial Approach”. In: *Logical Methods in Computer Science* Volume 16, Issue 1 (Mar. 2020). DOI: [10.23638/LMCS-16\(1:32\)2020](https://doi.org/10.23638/LMCS-16(1:32)2020) (cit. on p. 14).
- [22] 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/CB09780511977619](https://doi.org/10.1017/CB09780511977619) (cit. on p. 11).
- [23] Brian Day. “On closed categories of functors II”. In: *Category Seminar*. Ed. by Gregory M. Kelly. Berlin, Heidelberg: Springer Berlin Heidelberg, 1974, pp. 20–54. ISBN: 978-3-540-37270-7 (cit. on p. 25).
- [24] Ivan Di Liberti, Fosco Loregian, Chad Nester and Paweł Sobociński. “Functorial semantics for partial theories”. In: *Proc. ACM Program. Lang.* 5.POPL (Jan. 2021), pp. 1–28. DOI: [10.1145/3434338](https://doi.org/10.1145/3434338) (cit. on pp. 17, 56).
- [25] Volker Diekert and Anca Muscholl. “On Distributed Monitoring of Asynchronous Systems”. In: *Logic, Language, Information and Computation*. Ed. by Luke Ong and Ruy de Queiroz. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 70–84. ISBN: 978-3-642-32621-9 (cit. on p. 68).
- [26] Volker Diekert and Grzegorz Rozenberg. *The Book of Traces*. World Scientific, 1995. DOI: [10.1142/2563](https://doi.org/10.1142/2563) (cit. on pp. 11, 63).
- [27] Matt Earnshaw, James Hefford and Mario Román. “The Produoidal Algebra of Process Decomposition”. In: *32nd EACSL Annual Conference on Computer Science Logic (CSL 2024)*. Ed. by Aniello Murano and Alexandra Silva. Vol. 288. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024, 10:1–10:18. ISBN: 978-3-95977-264-8. DOI: [10.4230/LIPIcs.CSL.2023.10](https://doi.org/10.4230/LIPIcs.CSL.2023.10) (cit. on pp. 93, 96, 226).
- [28] Matt Earnshaw, Chad Nester and Mario Román. *Presentations of Premonoidal Categories by Devices*. Extended Abstract. Presented at NWPT 2023. 2023 (cit. on p. 14).

- [29] Matthew Earnshaw and Mario Román. “Context-Free Languages of String Diagrams”. In: *28th International Conference on Foundations of Software Science and Computation Structures*. To appear. 2025 (cit. on pp. 9, 90, 195, 226).
- [30] Matthew Earnshaw and Paweł Sobociński. “Regular Monoidal Languages”. In: *47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022)*. Ed. by Stefan Szeider, Robert Ganian and Alexandra Silva. Vol. 241. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022, 44:1–44:14. ISBN: 978-3-95977-256-3. DOI: 10.4230/LIPIcs.MFCS.2022.44 (cit. on pp. 9, 31, 129, 225).
- [31] Matthew Earnshaw and Paweł Sobociński. “Regular planar monoidal languages”. In: *Journal of Logical and Algebraic Methods in Programming* 139 (2024), p. 100963. ISSN: 2352-2208. DOI: <https://doi.org/10.1016/j.jlamp.2024.100963> (cit. on pp. 9, 31, 161, 226).
- [32] Matthew Earnshaw and Paweł Sobociński. “String Diagrammatic Trace Theory”. In: *48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023)*. Ed. by Jérôme Leroux, Sylvain Lombardy and David Peleg. Vol. 272. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023, 43:1–43:15. ISBN: 978-3-95977-292-1. DOI: 10.4230/LIPIcs.MFCS.2023.43 (cit. on pp. 9, 31, 145, 225).
- [33] Samuel Eilenberg and Jesse B. Wright. “Automata in general algebras”. In: *Information and Control* 11.4 (1967), pp. 452–470. ISSN: 0019-9958. DOI: [https://doi.org/10.1016/S0019-9958\(67\)90670-5](https://doi.org/10.1016/S0019-9958(67)90670-5) (cit. on pp. 11, 14).
- [34] Joost Engelfriet. “Context-Free Graph Grammars”. In: *Handbook of Formal Languages: Volume 3 Beyond Words*. Ed. by Grzegorz Rozenberg and Arto Salomaa. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 125–213. ISBN: 978-3-642-59126-6. DOI: 10.1007/978-3-642-59126-6_3 (cit. on p. 99).
- [35] Brendan Fong and David I. Spivak. “Hypergraph categories”. In: *Journal of Pure and Applied Algebra* 223.11 (2019), pp. 4746–4777. ISSN: 0022-4049. DOI: <https://doi.org/10.1016/j.jpaa.2019.02.014> (cit. on p. 99).
- [36] Tobias Fritz. “A synthetic approach to Markov kernels, conditional independence, and theorems on sufficient statistics”. In: *CoRR* abs/1908.07021 (2019). arXiv: 1908.07021 (cit. on p. 12).
- [37] Richard Garner and Ignacio López Franco. “Commutativity”. In: *Journal of Pure and Applied Algebra* 220.5 (2016), pp. 1707–1751 (cit. on pp. 26, 86, 87).

- [38] Ferenc Gécseg and Magnus Steinby. *Tree Automata*. 2015. DOI: 10.48550/arXiv.1509.06233 (cit. on pp. 11, 55, 59).
- [39] Ferenc Gécseg and Magnus Steinby. “Tree Languages”. In: *Handbook of Formal Languages, Vol. 3: Beyond Words*. Berlin, Heidelberg: Springer-Verlag, 1997, pp. 1–68. ISBN: 3540606491 (cit. on pp. 43, 97).
- [40] Alexandre Goy. “Weakening and iterating laws using string diagrams”. In: *Electronic Notes in Theoretical Informatics and Computer Science 1* (2023) (cit. on p. 24).
- [41] Philippe de Groote. “Towards Abstract Categorical Grammars”. In: *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics*. Toulouse, France: Association for Computational Linguistics, July 2001, pp. 252–259. DOI: 10.3115/1073012.1073045 (cit. on p. 15).
- [42] Annegret Habel. *Hyperedge Replacement: Grammars and Languages*. Vol. 643. Lecture Notes in Computer Science. Springer, 1992. ISBN: 3-540-56005-X. DOI: 10.1007/BFB0013875 (cit. on p. 99).
- [43] Tobias Heindel. *A Myhill-Nerode Theorem beyond Trees and Forests via Finite Syntactic Categories Internal to Monoids*. Preprint. 2017 (cit. on pp. 13, 53).
- [44] Tobias Heindel. *The Chomsky-Schützenberger Theorem with Circuit Diagrams in the Role of Words*. Abstract. 2017 (cit. on p. 15).
- [45] H. J. Hoogeboom and G. Rozenberg. “Dependence Graphs”. In: *The Book of Traces*. Ed. by V Diekert and G Rozenberg. World Scientific, 1995 (cit. on p. 14).
- [46] Alan Jeffrey. *Premonoidal categories and a graphical view of programs*. Preprint. 1997 (cit. on pp. 12, 14, 26, 68, 76).
- [47] Alan Jeffrey. “Premonoidal categories and flow graphs”. In: *Electronic Notes in Theoretical Computer Science 10* (1998). HOOTS II, Second Workshop on Higher-Order Operational Techniques in Semantics, p. 51. ISSN: 1571-0661. DOI: [https://doi.org/10.1016/S1571-0661\(05\)80688-7](https://doi.org/10.1016/S1571-0661(05)80688-7) (cit. on pp. 14, 26).
- [48] S. Jesi, G. Pighizzini and N. Sabadini. “Probabilistic asynchronous automata”. In: *Mathematical systems theory 29.1* (Feb. 1996), pp. 5–31. ISSN: 1433-0490. DOI: 10.1007/BF01201811 (cit. on p. 70).
- [49] André Joyal and Ross Street. “The geometry of tensor calculus, I”. In: *Advances in Mathematics 88.1* (1991), pp. 55–112. ISSN: 0001-8708. DOI: [https://doi.org/10.1016/0001-8708\(91\)90003-P](https://doi.org/10.1016/0001-8708(91)90003-P) (cit. on pp. 14, 17, 21, 67).
- [50] C. Krattenthaler. “The theory of heaps and the Cartier-Foata monoid”. In: *Commutation and Rearrangements*. Ed. by P. Cartier and D. Foata. European Mathematical Information Service, 2006 (cit. on p. 14).

- [51] Mohammad Amin Kuhail, Shahbano Farooq, Rawad Hammad and Mohammed Bahja. “Characterizing visual programming approaches for end-user developers: A systematic review”. In: *IEEE Access* 9 (2021), pp. 14181–14202 (cit. on p. 12).
- [52] F. W. Lawvere. “Qualitative Distinctions Between Some Toposes of Generalized Graphs”. In: *Proceedings of AMS Boulder 1987 Symposium on Category Theory and Computer Science*. Contemporary Mathematics. 1989, pp. 261–299 (cit. on p. 32).
- [53] F. W. Lawvere. “State categories and response functors”. Unpublished manuscript. 1986 (cit. on p. 33).
- [54] Tom Leinster. *Higher Operads, Higher Categories*. London Mathematical Society Lecture Note Series. Cambridge University Press, 2004. DOI: 10.1017/CB09780511525896 (cit. on pp. 21, 23).
- [55] Paul Blain Levy. *Call-by-push-value*. en. Semantics Structures in Computation. Dordrecht, Netherlands: Springer, Sept. 2012 (cit. on p. 89).
- [56] Paul Blain Levy, John Power and Hayo Thielecke. “Modelling environments in call-by-value programming languages”. In: *Information and Computation* 185.2 (2003), pp. 182–210. ISSN: 0890-5401. DOI: [https://doi.org/10.1016/S0890-5401\(03\)00088-9](https://doi.org/10.1016/S0890-5401(03)00088-9) (cit. on p. 26).
- [57] K Lodaya and P Weil. “Series-parallel languages and the bounded-width property”. In: *Theoretical Computer Science* 237.1 (2000), pp. 347–380. ISSN: 0304-3975. DOI: [https://doi.org/10.1016/S0304-3975\(00\)00031-1](https://doi.org/10.1016/S0304-3975(00)00031-1) (cit. on p. 54).
- [58] Hendrik Maarand and Tarmo Uustalu. “Reordering derivatives of trace closures of regular languages”. In: *30th International Conference on Concurrency Theory (CONCUR 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2019 (cit. on p. 70).
- [59] Saunders Mac Lane. *Categories for the Working Mathematician*. Second. Vol. 5. Graduate Texts in Mathematics. Springer Verlag, 1998. DOI: 10.1007/978-1-4757-4721-8 (cit. on p. 19).
- [60] Antoni Mazurkiewicz. “Basic notions of trace theory”. In: *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*. Ed. by J. W. de Bakker, W. -P. de Roever and G. Rozenberg. Berlin, Heidelberg: Springer Berlin Heidelberg, 1989, pp. 285–363. ISBN: 978-3-540-46147-0 (cit. on p. 63).

- [61] Antoni Mazurkiewicz. “Trace theory”. In: *Petri Nets: Applications and Relationships to Other Models of Concurrency*. Ed. by W. Brauer, W. Reisig and G. Rozenberg. Berlin, Heidelberg: Springer Berlin Heidelberg, 1987, pp. 278–324. ISBN: 978-3-540-47926-0 (cit. on p. 39).
- [62] 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 (cit. on pp. 15, 33, 91, 104).
- [63] Paul-André Melliès and Noam Zeilberger. “The categorical contours of the Chomsky-Schützenberger representation theorem”. Preprint. Dec. 2023 (cit. on pp. 15, 33, 34, 70, 91–93, 104, 108, 110, 125–127).
- [64] Madhavan Mukund. “Automata on Distributed Alphabets”. In: *Modern Applications of Automata Theory*. World Scientific, 2012, pp. 257–288. DOI: 10.1142/9789814271059_0009 (cit. on pp. 63, 64).
- [65] John Myhill. “Finite automata and the representation of events”. In: *WADD Technical Report 57* (1957), pp. 112–137 (cit. on p. 52).
- [66] Anil Nerode. “Linear automaton transformations”. In: *Proceedings of the American Mathematical Society* 9.4 (1958), pp. 541–544 (cit. on p. 52).
- [67] Chad Nester. “Concurrent Process Histories and Resource Transducers”. In: *Logical Methods in Computer Science* Volume 19, Issue 1 (Jan. 2023). DOI: 10.46298/lmcs-19(1:7)2023 (cit. on pp. 14, 81).
- [68] Chad Nester. “Situated Transition Systems”. In: *Applied Category Theory*. Vol. 372. Electronic Proceedings in Theoretical Computer Science. Open Publishing Association, 2022, pp. 103–115. DOI: 10.4204/EPTCS.372.8 (cit. on p. 81).
- [69] Evan Patterson, David I. Spivak and Dmitry Vagner. “Wiring diagrams as normal forms for computing in symmetric monoidal categories”. In: *Electronic Proceedings in Theoretical Computer Science* 333 (Feb. 2021), pp. 49–64. ISSN: 2075-2180. DOI: 10.4204/eptcs.333.4 (cit. on pp. 94, 96).
- [70] Dusko Pavlović and Samson Abramsky. “Specifying interaction categories”. In: *Category Theory and Computer Science: 7th International Conference, CTCS’97 Santa Margherita Ligure Italy, September 4–6, 1997 Proceedings* 7. Springer. 1997, pp. 147–158 (cit. on p. 125).
- [71] Paolo Perrone. *Distribution monad* (*nLab entry*). <https://ncatlab.org/nlab/show/distribution+monad>, Last accessed 2023-03-13. 2019 (cit. on p. 70).
- [72] Jean-Éric Pin. *Varieties of formal languages*. Plenum Publishing Co., 1986 (cit. on p. 51).

- [73] Jean-Éric Pin and Dominique Perrin. *Infinite Words - Automata, Semigroups, Logic and Games*. Pure and Applied Mathematics (Amsterdam). Elsevier, 2004, pp. 5–73 (cit. on p. 11).
- [74] John Power and Edmund Robinson. “Premonoidal categories and notions of computation”. In: *Mathematical Structures in Computer Science* 7.5 (1997), pp. 453–468. DOI: 10.1017/S0960129597002375 (cit. on pp. 25, 26).
- [75] Mario Román. “Monoidal Context Theory”. PhD thesis. Tallinn University of Technology, 2023 (cit. on p. 93).
- [76] Mario Román. “Open Diagrams via Coend Calculus”. In: *Electronic Proceedings in Theoretical Computer Science* 333 (Feb. 2021), pp. 65–78. ISSN: 2075-2180. DOI: 10.4204/eptcs.333.5 (cit. on p. 94).
- [77] Mario Román. “Promonads and String Diagrams for Effectful Categories”. In: *ACT '22: Applied Category Theory, Glasgow, United Kingdom, 18 - 22 July, 2022*. Vol. abs/2205.07664. 2022. DOI: 10.48550/arXiv.2205.07664. arXiv: 2205.07664 (cit. on pp. 14, 26–28, 71, 74, 76).
- [78] Mario Román and Paweł Sobociński. *String Diagrams for Premonoidal Categories*. 2024. arXiv: 2305.06075 [math.CT] (cit. on pp. 14, 74, 76).
- [79] Robert Rosebrugh, Nicoletta Sabadini and Robert F.C. Walters. “Generic commutative separable algebras and cospans of graphs”. In: *Theory and applications of categories* 15 (2005), pp. 164–177 (cit. on p. 99).
- [80] Kimmo I. Rosenthal. “Quantaloids, enriched categories and automata theory”. In: *Applied Categorical Structures* 3.3 (1995), pp. 279–301. DOI: 10.1007/bf00878445 (cit. on p. 14).
- [81] Paul W. K. Rothmund, Nick Papadakis and Erik Winfree. “Algorithmic Self-Assembly of DNA Sierpinski Triangles”. In: *PLOS Biology* 2.12 (Dec. 2004). DOI: 10.1371/journal.pbio.0020424 (cit. on pp. 13, 38).
- [82] William C. Rounds. “Context-Free Grammars on Trees”. In: *Proceedings of the First Annual ACM Symposium on Theory of Computing*. STOC '69. Marina del Rey, California, USA: Association for Computing Machinery, 1969, pp. 143–148. ISBN: 9781450374781. DOI: 10.1145/800169.805428 (cit. on p. 97).
- [83] Grzegorz Rozenberg. *Handbook Of Graph Grammars And Computing By Graph Transformation, Vol 1: Foundations*. World Scientific Publishing Company, 1997. ISBN: 9789814498104 (cit. on p. 15).
- [84] Michael Shulman. “Categorical logic from a categorical point of view”. In: *Available on the web* (2016) (cit. on p. 94).
- [85] Michael Sipser. *Introduction to the Theory of Computation*. Third. Boston, MA: Course Technology, 2013. ISBN: 113318779X (cit. on p. 42).

- [86] James W. Thatcher and J. B. Wright. “Generalized finite automata theory with an application to a decision problem of second-order logic”. In: *Mathematical systems theory* 2.1 (Mar. 1968), pp. 57–81. ISSN: 1433-0490 (cit. on p. 11).
- [87] Henning Urbat, Jiri Adámek, Liang-Ting Chen and Stefan Milius. “Eilenberg Theorems for Free”. In: *42nd International Symposium on Mathematical Foundations of Computer Science (MFCS 2017)*. Ed. by Kim G. Larsen, Hans L. Bodlaender and Jean-Francois Raskin. Vol. 83. LIPIcs. Dagstuhl, Germany, 2017, 43:1–43:15. ISBN: 978-3-95977-046-0. DOI: 10.4230/LIPIcs.MFCS.2017.43 (cit. on p. 11).
- [88] Gérard Xavier Viennot. “Heaps of pieces, I : Basic definitions and combinatorial lemmas”. In: *Combinatoire énumérative*. Ed. by Gilbert Labelle and Pierre Leroux. Berlin, Heidelberg: Springer Berlin Heidelberg, 1986, pp. 321–350. ISBN: 978-3-540-47402-9 (cit. on p. 14).
- [89] R. F. C. Walters. “A note on context-free languages”. In: *Journal of Pure and Applied Algebra* 62.2 (1989), pp. 199–203. ISSN: 0022-4049. DOI: 10.1016/0022-4049(89)90151-5 (cit. on pp. 14, 15, 32, 91, 93).
- [90] R. F. C. Walters. *Categories and Computer Science*. Cambridge Computer Science Texts. Cambridge University Press, 1992 (cit. on p. 32).
- [91] R. F. C. Walters. “The free category with products on a multigraph”. In: *Journal of Pure and Applied Algebra* 62.2 (1989), pp. 205–210. ISSN: 0022-4049. DOI: [https://doi.org/10.1016/0022-4049\(89\)90152-7](https://doi.org/10.1016/0022-4049(89)90152-7) (cit. on p. 93).
- [92] Harvey Wolff. “Commutative distributive laws”. In: *Journal of the Australian Mathematical Society* 19.2 (1975), pp. 180–195. DOI: 10.1017/S1446788700029487 (cit. on pp. 25, 62).
- [93] Vladimir Zamdzhiev. “Rewriting Context-free Families of String Diagrams”. PhD thesis. University of Oxford, 2016 (cit. on p. 15).
- [94] Wieslaw Zielonka. “Notes on finite asynchronous automata”. en. In: *RAIRO - Theoretical Informatics and Applications - Informatique Théorique et Applications* 21.2 (1987), pp. 99–135 (cit. on pp. 45, 63, 68, 69).

Index

- automaton, *see also* monoidal automaton
 - asynchronous, **69**
 - convex, 62, 63

- category, *see also* effectful category, *see also* monoidal category, *see also* premonoidal category
 - cartesian restriction, **55**, 56
- central, **27**, 27, 80
- centralizer, 85, 86
 - device, **86**, 86, 87
- commuting cospan, **86**, 87
- commuting tensor product, **87**, 87
- congruence
 - on a monoidal category, **53**
 - syntactic, **53**, 54
- convex
 - relation, 60–63

- dependence relation, **64**
- device
 - clique, 78, 80
 - presentation, **81**, 84, 87
- device clique
 - morphism of, **80**, 80
- device graph, **77**, 77–83
 - morphism of, **78**
- device presentation
 - intrinsic, **84**, 84–86
- diagram contexts, **94**, 95–97, 100, 101, 106

- distributed alphabet, 65
 - monoidal, 63, **65**, 66–71
- distributive law, **24**, 25

- effectful category, 26, **27**, 27, 30, 73–78, 80–82, 85–89
- endomorphism pro, **48**, 48

- functor, *see also* monoidal functor
 - cartesian restriction, **56**, 56
 - effectful, **27**, 30
 - finitary, **127**, 127
 - identity-on-objects, **28**, 28, 29
 - unique lifting of factorizations, **33**, 33

- grafting, **57**, 58
- grammar, *see also* monoidal grammar
 - regular, 32, 35
- graph
 - directed, 32, 35
 - labelled, **32**, 32
 - morphism of, 32, 33

- independence relation, **64**, 64, 65, 68, 70, 83
- interference graph, **83**, 84

- monad
 - maybe, **24**, 24, 25, 48, 49, 62
 - monoidal, **24**, 24, 25, 48
 - non-empty powerset, **24**, 24, 25, 62

powerset, **24**, 24, 25, 48
 monoidal alphabet, **34**, 34, 35, 37–39, 44,
 47, 48, 50, 63
 distributed, 71
 word, **47**, 48, 63
 monoidal automaton
 deterministic, **48**, 49, 56, 58, 59
 non-deterministic, **45**, 47, 48, 50, 51,
 59, 60, 63, 69, 70, 105
 monoidal category
 device, **79**, 79, 80
 free strict, **20**, 34, 79, 80, 95, 96, 125
 free symmetric strict, **20**, 35, 56, 79
 quotient, 53, 54
 strict, **18**, 19–21, 24, 26, 27, 30, 34,
 48, 54, 70, 76, 78, 82, 84, 89, 90,
 94–96, 101, 102, 125, 127
 symmetric strict, **19**, 19, 20, 22, 25,
 48, 55, 74
 syntactic, **54**, 54, 55
 monoidal functor, **19**, 19, 20, 47, 49, 54,
 62, 105–108, 125–127
 symmetric strict, **19**, 47, 49, 56
 unique lifting of factorizations, **126**,
 127
 monoidal grammar
 context-free, 41, 94, **97**, 97–100, 105–
 108
 quasiregular, **40**, 40, 41, 107
 regular, **35**, 35–37, 39, 40, 42, 44, 48,
 51, 100, 107, 108, 125, 127
 regular with ε -productions, 42
 monoidal graph, **17**, 20, 27, 34, 35, 40,
 41, 45, 47, 48, 56, 63, 65, 66, 70,
 71, 76–79, 94, 96–99, 102, 127
 category of, **18**, 96
 morphism of, **18**, 18, 35, 40, 41, 107
 morphism of reflexive, 41
 reflexive, **41**, 41, 42, 47
 runtime, **71**, 79
 monoidal language, **34**, **35**, 35, 40, 45,
 51–55, 58, 63, 104
 co-rooted, **35**, 37, 55, 58
 context-free, 45, **97**, 100, 105, 108
 planar, **34**, **36**, 36, 44, 49
 regular, 37, 43–45, 49–51, 54, 55, 59,
 105, 107, 108, 125
 regular planar, 38, 44
 rooted, **35**
 scalar, **35**, 51
 symmetric, **35**, 36, 40, 49, 63, 65, 66,
 68
 symmetric regular, 69
 trace, **66**, 66, 68, 70, 72
 multicategory, **21**, 21–23, 90, 92, 93, 95
 category of, **22**, 96, 101
 free, 21, **22**
 of spliced arrows, **91**, 91, 92, 94, 97,
 101
 symmetric, 21, **23**, 23
 underlying, 96
 multifunctor, 21, **22**, 103, 107
 symmetric, 96, 101
 multigraph, **21**, 22, 23, 91, 92, 99, 106
 category of, **21**, 96
 morphism of, **21**, 23, 90, 91
 morphism of symmetric, **23**, 97, 105,
 106
 symmetric, **23**, 23, 97
 optical contour, 100, **102**, 102, 103, 105,
 107–109
 partial view, **57**, 57–59
 closure, **58**
 premonoidal category, **26**, 27, 30, 70, 74,
 75, 80, 83–85
 center, **27**, 27
 symmetric, **27**
 pro, **20**, 46, 48, 49
 pro(p)

morphism of, **20**
profunctor, **28**, 28, 29
 strong, **29**, 29, 30, 86
promonad, 28–30
 left-strong, **29**
 morphism of, **28**, 28–30, 88, 89
 morphism of strong, **30**, 88, 89
 strong, **29**, 30, 86–89
prop, **20**, 46, 48, 49

raw optics, 97, 100, **101**, 101–103, 105,
 106, 109
raw representative, **106**, 106–108

string diagram, **20**, 27, 37, 67, 76
syntactic monoid, **52**, 52
syntactic morphism, **54**

trace language, **64**, 66, 68–70, 73
trace monoid, **64**, 64, 67, 71

Regular monoidal languages over arbitrary strict monoidal categories

It is possible to generalize regular monoidal languages over free strict monoidal categories, as in Section 3.3, to regular monoidal languages over arbitrary strict monoidal categories. This follows Melliès and Zeilberger [63], who define an automaton over a category as a finitary unique-lifting-of-factorizations functor, axiomatizing properties of free functors $\mathcal{F}\phi : \mathcal{F}Q \rightarrow \mathcal{F}\Sigma$ on morphisms of finite graphs ϕ , where Q is the transition graph, and Σ is a graph with one object.

Here we do the same for strict monoidal functors. We axiomatize properties of regular monoidal grammars over free monoidal categories $\mathcal{F}_\otimes\phi : \mathcal{F}_\otimes M \rightarrow \mathcal{F}_\otimes\Gamma$, which leads us to a notion of regular grammar of morphisms over an arbitrary monoidal category.

Proposition A.0.1. *Strict monoidal functors $\phi : D \rightarrow T$ are in bijection with lax monoidal lax 2-functors $\hat{\phi} : T \rightarrow \text{Span}(\text{Set})$.*

Proof. We recall the correspondence between functors and lax 2-functors into $\text{Span}(\text{Set})$ [70, Section 2.2]. To construct the lax 2-functor $\hat{\phi}$, one treats the category T as a 2-category whose only 2-cells are identities. We show that monoidal structures on D and T , and a strict monoidal structure on ϕ further induces a lax monoidal structure on $\hat{\phi}$ where T is considered as a monoidal 2-category whose only 2-cells are identities, and $\text{Span}(\text{Set})$ is considered as a cartesian monoidal 2-category.

Consider $(x, y) \in \hat{\phi}(t) \otimes \hat{\phi}(t')$, that is $\phi(x) = t, \phi(y) = t'$, then by strictness, $\phi(x \otimes y) = t \otimes t'$ and so $x \otimes y \in \hat{\phi}(t \otimes t')$ by definition. So we have a function $\hat{\phi}(t) \otimes \hat{\phi}(t') \rightarrow \hat{\phi}(t \otimes t')$ given by \otimes in D . Take this as the right leg of a span $\hat{\phi}(t) \otimes \hat{\phi}(t') \rightarrow \hat{\phi}(t \otimes t')$ with left leg the identity.

For the unit laxator, note that strictness implies $\hat{\phi}(I_T) = \phi^{-1}(I_T)$ is inhabited

by at least I_D . Therefore take the span $1 \rightarrow \hat{\phi}(I_T)$ whose apex is 1, left leg identity and right leg picking out I_D .

Conversely, let $\psi : T \rightarrow \mathbf{Span}(\mathbf{Set})$ be a lax monoidal lax 2-functor, with laxators denoted by spans $\mu_{X,X'} : \psi(X) \times \psi(X') \rightarrow \psi(X \otimes X')$, $\eta : 1 \rightarrow \psi(I_T)$. We show that this structure gives a strict monoidal structure on the corresponding functor $\bar{\psi} : \int \psi \rightarrow T$.

Recall that $\int \psi$ is defined as the category with set of objects the disjoint union $\sum_{X \in T_{\text{obj}}} \psi(X)$ and hom-sets

$$\int \psi(\langle X, x \in \psi(X) \rangle, \langle Y, y \in \psi(Y) \rangle) = \sum_{f \in T(X,Y)} (\psi f)^{-1}(x, y).$$

Note that $\int \psi$ becomes monoidal via the monoidal structure of T and the laxators of ψ with $\langle X, x \rangle \otimes \langle Y, y \rangle := \langle X \otimes Y, \mu_{X,Y}(x, y) \in \psi(X \otimes Y) \rangle$, and unit $\langle I_T, \eta(\bullet) \in \psi(I_T) \rangle$.

The functor $\bar{\psi}$ is the projection $\langle X, x \rangle \mapsto X$, $\langle f, p \rangle \mapsto f$. Then it is immediate that $\bar{\psi}(\langle X, x \rangle) \otimes \bar{\psi}(\langle Y, y \rangle) = \bar{\psi}(\langle X, x \rangle \otimes \langle Y, y \rangle) = X \otimes Y$ and $\bar{\psi}(\langle I_T, \eta(\bullet) \rangle) = I_T$. \square

Proposition A.0.2. *A strict monoidal functor $\phi : D \rightarrow T$ has the monoidal unique lifting of factorizations (monoidal ULF) property if either of the following equivalent conditions hold:*

1. *For any morphism $\alpha \in D$ such that $\phi(\alpha) = \beta \circ \gamma$, there exists a unique pair of morphisms $\delta, \varepsilon \in D$ over β, γ respectively, such that $\alpha = \delta \circ \varepsilon$; and if $\phi(\alpha) = \mu \otimes \nu$, there exists a unique pair of morphisms $\zeta, \eta \in D$ over μ, ν respectively such that $\alpha = \zeta \otimes \eta$.*
2. *The associated lax monoidal lax 2-functor $\hat{\phi} : T \rightarrow \mathbf{Span}(\mathbf{Set})$ is a strong monoidal pseudofunctor.*

Proof. Let $\phi : D \rightarrow T$ satisfy condition (1). Notice in particular, by considering identity morphisms in D , that this implies: for any object $P \in D$ such that $\phi(P) = X \otimes Y$, we have a unique pair $P', P'' \in D$ over X and Y respectively such that $P = P' \otimes P''$.

Now $\hat{\phi}(X) := \phi^{-1}(X)$, so $\hat{\phi}(X) \times \hat{\phi}(Y) = \phi^{-1}(X) \times \phi^{-1}(Y)$ and $\hat{\phi}(X \otimes Y) = \phi^{-1}(X \otimes Y)$. We define a bijection between these sets.

Let $(A, B) \in \hat{\phi}(X) \times \hat{\phi}(Y)$, i.e. $\phi(A) = X$ and $\phi(B) = Y$. Then $A \otimes B \in \hat{\phi}(X \otimes Y)$ by strictness of ϕ . Let $C \in \hat{\phi}(X \otimes Y)$, that is $\phi(C) = X \otimes Y$. Then by condition (1), we have a unique pair $(C', C'') \in \hat{\phi}(X) \times \hat{\phi}(Y)$, and it is clear that these functions form a bijection.

Similarly $1 \cong \hat{\phi}(I_T)$ follows from the fact that identities lift uniquely (take $\gamma = \text{id}$ in a factorization). The rest of this direction of the proof is the claim from Melliès and Zeilberger [63, Proposition 2.2].

Now let condition (2) hold. We first look at the constraints making $\hat{\phi}$ pseudo-functorial, namely the natural families of isomorphisms $\hat{\phi}(f) \circ \hat{\phi}(g) \cong \hat{\phi}(f \circ g)$ and $\text{id}_{\hat{\phi}(X)} \cong \hat{\phi}(\text{id}_X)$. The former says exactly that morphisms over $f \circ g$ are in natural bijection with pairs of morphisms over f and over g , giving the unique lifts of composites. The latter says that elements over X are in bijection with morphisms over id_X . For each object over X , the identity on that object must be over id_X by functoriality. That we have a bijection says that these are the only morphisms over id_X , that is, ϕ has unique lifts of identities.

The constraints making $\hat{\phi}$ strong are the natural families of isomorphisms $\hat{\phi}(X) \times \hat{\phi}(Y) \cong \hat{\phi}(X \otimes Y)$ and $1 \cong \hat{\phi}(I_T)$. The former says that pairs of objects over $X \otimes Y$ are in natural bijection with pairs of objects over X and over Y . Naturality here means that we can uniquely lift morphisms over $f \otimes g$ to pairs of morphisms over f and over g . The latter says that the fibre over I_T contains only I_D , which implies reflection of the identity at I_T . \square

Recall that a functor is finitary if the preimages of every object and every arrow in the codomain are finite [63], or equivalently if the associated lax functor to $\text{Span}(\text{Set})$ factors through $\text{Span}(\text{FinSet})$.

Definition A.0.3. A regular monoidal grammar over a strict monoidal category \mathbb{C} is a strict monoidal functor $\phi : \mathbb{G} \rightarrow \mathbb{C}$ that is finitary and satisfies the monoidal ULF property, along with objects i, f of \mathbb{G} .

Lemma A.0.4. *Let $\phi : \mathcal{M} \rightarrow \mathcal{F}_{\otimes} \Gamma$ be a strict monoidal functor. Then ϕ is monoidal ULF if and only if $\mathcal{M} = \mathcal{F}_{\otimes} M$ for a monoidal graph M and $\phi = \mathcal{F}_{\otimes} h$ for some morphism of monoidal graphs $h : M \rightarrow \Gamma$.*




Proof. The *if* direction follows by structural induction on the morphisms of $\mathcal{F}_{\otimes} M$. For the *only if* direction, we construct the monoidal graph M by taking those morphisms m of \mathcal{M} such that $\phi(m) = \gamma$ for some $\gamma \in \Gamma$, and easily verify the monoidal ULF property. \square

Appendix B

Paper I

Matthew Earnshaw and Paweł Sobociński. “Regular Monoidal Languages”. In: *47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022)*. Ed. by Stefan Szeider, Robert Ganian and Alexandra Silva. Vol. 241. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022, 44:1–44:14. ISBN: 978-3-95977-256-3. DOI: [10.4230/LIPIcs.MFCS.2022.44](https://doi.org/10.4230/LIPIcs.MFCS.2022.44)

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.2022.56

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 categorical 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



© Matthew Earnshaw and Paweł Sobociński;
licensed under Creative Commons License CC-BY 4.0

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).

Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 56; pp. 56:1–56:15



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

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.

Winfrey 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-dimensional 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 $\Psi : \mathcal{G}' \rightarrow \mathcal{G}$ of monoidal graphs is a pair of functions $V_\Psi : V_{\mathcal{G}} \rightarrow V_{\mathcal{G}'}, E_\Psi : E_{\mathcal{G}} \rightarrow E_{\mathcal{G}'}$ such that $\text{dom} \circ V_\Psi^* = E_\Psi \circ \text{dom}$ and $\text{cod} \circ V_\Psi^* = E_\Psi \circ \text{cod}$.

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 $\mathcal{U} : \mathbf{Pro} \rightarrow \mathbf{MonGraph}$ has a left adjoint $\mathcal{F} : \mathbf{MonGraph} \rightarrow \mathbf{Pro}$, 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{FG} 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 $\text{dom}(\gamma), \text{cod}(\gamma)$ as the arity, coarity (resp.) of γ , writing $\text{ar}(\gamma), \text{coar}(\gamma)$. 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 $\Psi : \mathcal{M} \rightarrow \Gamma$ 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 $V_\Psi : V_{\mathcal{M}} \rightarrow \{\bullet\}$ is unique, so the grammar is determined by its edge function $E_\Psi : E_{\mathcal{M}} \rightarrow E_\Gamma$, 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, $\mathcal{F}\Psi : \mathcal{FM} \rightarrow \mathcal{F}\Gamma$, 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.

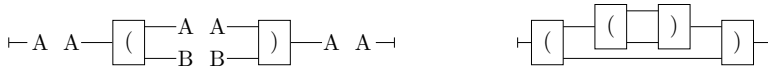
3.3 Regular monoidal languages

A regular monoidal grammar determines a monoidal language as follows:

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

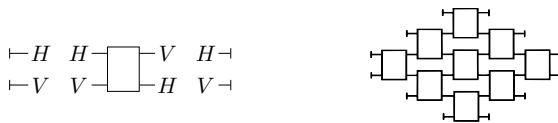
We call the class of languages determined by regular monoidal grammars the *regular monoidal languages*. We shall see that they are precisely the languages accepted by *non-deterministic monoidal automata* (Section 3.4). The basic idea is that a “word” is a scalar string diagram, i.e. one with no “dangling 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.



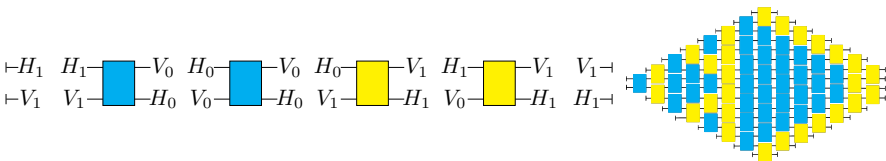
This illustrates how regular monoidal grammars permit unbounded concurrency. Here, as one scans from left to right, the (unbounded) size of the internal boundary of a string diagram keeps track of the number of open left parentheses.

► **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.

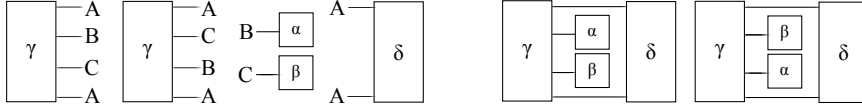


In Section 3.5 we will see how this language of “brick walls” allows us to construct the following example as an intersection of two languages:

► **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 $\Psi : \mathcal{M} \rightarrow \Gamma$ will define a language containing only the identity on the monoidal unit, i.e. the empty string diagram (denoted \square). 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 $Q \xrightarrow{a} \mathcal{P}(Q)$. 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 $\Sigma^* \rightarrow \text{Rel}$, 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 $\mathcal{P}X \times \mathcal{P}Y \rightarrow \mathcal{P}(X \times Y)$ 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 $\text{Kl}(T)$ has a canonical monoidal structure, which is given on objects by the monoidal product in \mathcal{C} , and on morphisms $f : X \rightarrow TA, g : Y \rightarrow TB$ 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 $\Gamma \rightarrow |\text{Rel}_Q|$. Such morphisms are in bijection with pro morphisms $\Delta : \mathbb{F}\Gamma \rightarrow \text{Rel}_Q$. We will also refer to the inductive extension Δ as a non-deterministic monoidal automaton, and sometimes write Δ_α for the relation $\Delta(\alpha : n \rightarrow m)$.

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

► **Definition 20.** *Let $\Delta : \mathcal{F}\Gamma \rightarrow \text{Rel}_Q$ 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 $\Psi : \mathcal{M} \rightarrow \Gamma$, define a monoidal automaton with $Q = V_{\mathcal{M}}$, $w(\Delta_\gamma)w' \iff \exists \sigma \in E_\Psi^{-1}(\gamma)$ such that $\text{dom}(\sigma) = w, \text{cod}(\sigma) = w'$. Conversely given a monoidal automaton (Q, Δ_Γ) , define a regular monoidal grammar with $V_{\mathcal{M}} = Q$ and take an edge $w \rightarrow w'$ 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 $\mathcal{F}\mathcal{M} \rightarrow \mathcal{F}\Gamma$, that is (by the adjunction $\mathcal{U} \dashv \mathcal{F}$), to morphisms of finite monoidal graphs $\mathcal{M} \rightarrow \mathcal{U}\mathcal{F}\Gamma$. 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 $\mathcal{F}\Gamma \rightarrow \text{Rel}_Q$ (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 be a 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 be 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 \mathbf{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 \mathbf{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 $\Delta_\sigma := \Delta(\sigma, -) : Q \rightarrow \mathcal{P}(Q)$. For \dashv take the transition function $Q \rightarrow \mathcal{P}(Q^0)$ to be the characteristic function of $F \subseteq Q$, sending elements of F to $\{\bullet\}$ and to \emptyset otherwise, and for \vdash take the function $Q^0 \rightarrow \mathcal{P}(Q)$ 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 $\{\bullet\} \rightarrow \mathcal{P}(Q)$. 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.

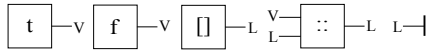
4.2 Tree automata

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 $(\Sigma, r : \Sigma \rightarrow \mathbb{N})$, a set of final states $F \subseteq Q$, and for each $\sigma \in \Sigma$ a transition function $\Delta_\sigma : Q^{r(\sigma)} \rightarrow \mathcal{P}(Q)$. A non-deterministic top-down tree automaton, instead, has a set of initial states $I \subseteq Q$ and transition functions $\Delta_\sigma : Q \rightarrow \mathcal{P}(Q^{r(\sigma)})$. 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 ≥ 0), $\boxed{\sigma} \leftarrow :$, along with a single “root” generator \dashv . Analogously, a bottom-up tree monoidal alphabet is a monoidal alphabet having only generators of coarity 1 (and arbitrary arities ≥ 0), $\vdash \boxed{\sigma}$, 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, $\boxed{\quad}$, or be a boolean value “consed” onto a list via $::$).

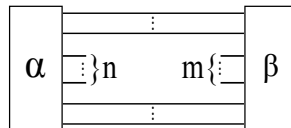


Intuitively, the connected scalar string diagrams determined by this language are trees, with leaves on the left, and the root on the right. Monoidal automata over top-down tree monoidal alphabets have a similar form, but are mirrored horizontally, and thus morphisms in the language have the root on 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 $n, m \geq 0$, 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 $\alpha : n \rightarrow m$. 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 $L \subseteq \mathcal{FT}(0, 0)$ we define its syntactic congruence \equiv_L as follows. Let α, β be morphisms in $\mathcal{FT}(n, m)$. 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{FT}/\equiv_L . The quotient functor $S_L : \mathcal{FT} \rightarrow \mathcal{FT}/\equiv_L$ 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 $\alpha \equiv_L \square$, since the empty diagram is in every language and if C is a context of capacity $(0, 0)$ distinguishing α and \square , then we have a contradiction by Lemma 26. So $\alpha \in S_L^{-1}(\square)$, 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{FT}/\equiv_L is locally finite (i.e. has finite hom-sets).*

Proof. It suffices to exhibit a full pro morphism into \mathcal{FT}/\equiv_L from a locally finite pro. Let L be a regular monoidal language recognized by $\Delta : \mathcal{FT} \rightarrow \mathbf{Rel}_Q$. Δ induces a congruence \sim on \mathcal{FT} defined by $\alpha \sim \beta \iff \Delta(\alpha) = \Delta(\beta)$, which implies that \mathcal{FT}/\sim is locally finite, since \mathbf{Rel}_Q is locally finite. Define the pro morphism $\mathcal{FT}/\sim \rightarrow \mathcal{FT}/\equiv_L$ 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{FT}/\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 \mathbf{Par}_Q from Remark 18. Then as in Observation 19, such assignments $\gamma \mapsto \delta_\gamma$ uniquely extend to pro morphisms $\delta : \mathcal{FT} \rightarrow \mathbf{Par}_Q$, and we will also refer to such pro morphisms as deterministic monoidal automata. δ maps scalar string diagrams to one of the two functions $Q^0 \rightarrow Q_\perp^0$, and we use this to define the language of the automaton:

► **Definition 41.** *Let $\delta : \mathcal{FT} \rightarrow \mathbf{Par}_Q$ be a deterministic monoidal automaton. Then the language accepted by δ is $\mathcal{L}(\delta) := \{\alpha \in \mathcal{FT}(0, 0) \mid \delta_\alpha(\bullet) = \bullet\}$.*

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 \dashv , comultiplication by \dashv , and symmetry by \bowtie) 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]).*

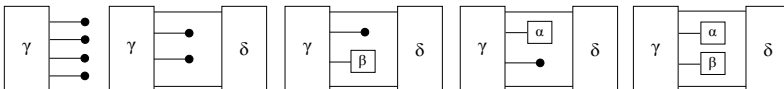
► **Remark 44.** Par is the paradigmatic example of a cartesian restriction category, with \dashv on X given by the relation $X \rightarrow \{\bullet, \perp\}$ sending every element to \bullet , and \dashv given by the diagonal relation. Par_Q inherits this structure and so is a cartesian restriction prop. Therefore deterministic monoidal automata (Q, δ_Γ) also have inductive extensions to morphisms of cartesian restriction props, $\bar{\delta} : \mathcal{F}_\downarrow \Gamma \rightarrow \text{Par}_Q$, 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 $\bar{\delta}$ as $\delta = \mathcal{H}_\Gamma \bar{\delta}$, where $\mathcal{H}_\Gamma : \mathcal{F}\Gamma \rightarrow \mathcal{F}_\downarrow \Gamma$ sends morphisms to their equivalence class in $\mathcal{F}_\downarrow \Gamma$.*

Recall that any restriction category is poset-enriched: $f \leq g$ 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 $f \leq g \iff f \otimes g = f$. 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 $\mathcal{H}_\Gamma(\gamma) \leq h$ 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:

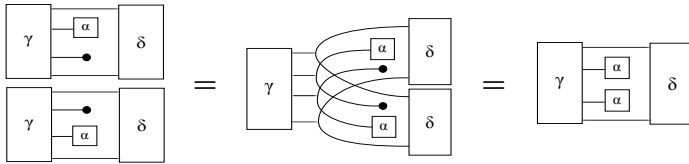


► **Lemma 47.** *Let $M = (Q, \delta_\Gamma)$ be a deterministic monoidal automaton, with functors $\delta : \mathcal{F}\Gamma \rightarrow \text{Par}_Q$, $\bar{\delta} : \mathcal{F}_\downarrow \Gamma \rightarrow \text{Par}_Q$. Then if δ accepts γ , $\bar{\delta}$ accepts all causal histories of γ .*

Proof. Since $\delta = \mathcal{H}_\Gamma \circ \bar{\delta}$, if δ accepts γ , then $\bar{\delta}$ accepts $\mathcal{H}_\Gamma(\gamma)$. Let h be a causal history of γ . Then $\bar{\delta}(\mathcal{H}_\Gamma(\gamma)) = \bar{\delta}(h \otimes \mathcal{H}_\Gamma(\gamma)) = \bar{\delta}(h) \otimes \bar{\delta}(\mathcal{H}_\Gamma(\gamma))$. But then $\bar{\delta}$ accepts h by Lemma 26. ◀

► **Definition 48** (Causal closure of a language). *Let L be a monoidal language over a monoidal alphabet Γ . Let $\otimes ch(L)$ denote the closure of the set of causal histories of L under monoidal product. Then the causal closure of L is $\mathcal{H}_\Gamma^{-1} \otimes ch(L)$. 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 \dashv (for γ), then unitality (twice), then naturality of \dashv (for δ). The rightmost form of the diagram exhibits this morphism as being in the image of \mathcal{H}_Γ , and its preimage under \mathcal{H}_Γ is the same diagram in \mathcal{FT} . 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 $\delta : \mathcal{FT} \rightarrow \text{Par}_Q$. We have $\delta = \mathcal{H}_\Gamma \circ \bar{\delta}$ and from Lemma 47 that $\bar{\delta}$ 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 $\text{Rel} \rightarrow \text{Set}$, right adjoint to the inclusion $\text{Set} \hookrightarrow \text{Rel}$. As remarked above, we cannot hope

56:12 Regular Monoidal Languages

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 $\Delta : Q^n \rightarrow \mathcal{P}(Q^m)$ is convex if there is a morphism Δ^* such that the following square commutes:

$$\begin{array}{ccc} (\mathcal{P}Q)^n & \xrightarrow{\Delta^*} & (\mathcal{P}Q)^m \\ \nabla \downarrow & & \downarrow \nabla \\ \mathcal{P}(Q^n) & \xrightarrow{\Delta^\#} & \mathcal{P}(Q^m) \end{array}$$

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 $\Delta_\gamma : Q^0 \rightarrow \mathcal{P}(Q^4)$ 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 $\text{CRel}_Q \hookrightarrow \text{Rel}_Q$.

► **Definition 54.** An automaton $\Delta : \mathcal{FT} \rightarrow \text{Rel}_Q$ 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 (\emptyset in Rel_Q , but \perp in $\text{Par}_{\mathcal{P}^+(Q)}$):

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

$$\begin{array}{c} \Delta_\alpha : Q^n \rightarrow \mathcal{P}(Q^m) \\ \downarrow \\ \mathcal{P}^+(Q)^n \xrightarrow{\eta^n} (\perp \mathcal{P}^+(Q))^n \xrightarrow{\cong} \mathcal{P}(Q)^n \xrightarrow{\Delta_\alpha^*} \mathcal{P}(Q)^m \xrightarrow{\cong} (\perp \mathcal{P}^+(Q))^m \xrightarrow{\nabla} \perp \mathcal{P}^+(Q)^m \end{array}$$

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 $\Delta : \mathcal{FT} \rightarrow \text{CRel}_Q$ 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 $\Delta : \mathcal{FT} \rightarrow \text{CRel}_Q$ 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 $\Delta : Q \rightarrow \mathcal{P}(Q)$, Δ^* is given by the Kleisli extension of Δ . This reflects the well known determinizability of classical finite-state automata.

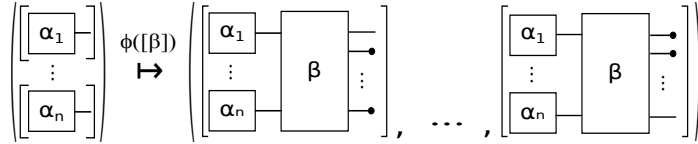
► **Example 58.** Similarly, non-deterministic monoidal automata over bottom-up tree monoidal alphabets (Definition 32) are convex, with $\Delta^* := \nabla \mathcal{D}^\#$. For top-down tree monoidal alphabets, the general obstruction to convexity (and thus determinizability) is seen as the non-existence 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 $\mathcal{F}\Gamma/\equiv_L \xrightarrow{\phi} \text{Par}_Q$ such that $\mathcal{F}\Gamma \xrightarrow{S_L} \mathcal{F}\Gamma/\equiv_L \xrightarrow{\phi} \text{Par}_Q$ 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 $\phi([\beta]) : n \rightarrow m$ to be the following map from $Q^n \rightarrow Q^m$:



When $m = 0$ (i.e. $[\beta]$ has coarity 0), let $\phi([\beta])([\alpha_1], \dots, [\alpha_n]) = \bullet$, if $[(\alpha_1 \otimes \dots \otimes \alpha_n) \circlearrowleft \beta] = \square$, and $\phi([\beta])([\alpha_1], \dots, [\alpha_n]) = \perp$ 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 $\alpha \in \mathcal{L}(S_L \circlearrowleft \phi)$, then by definition we must have $S_L(\alpha) = \square$, and so $\alpha \in L$ (by Lemma 38). Conversely let $\alpha \in L$, then $S_L(\alpha) = \square$ and by definition $\phi(\square)(\bullet) = \bullet$, so $\alpha \in \mathcal{L}(S_L \circlearrowleft \phi)$. Therefore $S_L \circlearrowleft \phi$ is a deterministic monoidal automaton recognizing L . ◀

► **Example 60.** A simple example is the language L of “bones” over the monoidal alphabet $\Gamma = \{ \square, \dashv \}$, having one connected component: $\square \dashv \square$. The syntactic pro $\mathcal{F}\Gamma/\equiv_L$ has a cartesian restriction prop structure, with the counit \dashv given by the equivalence class $[\dashv]$, comultiplication \dashv by $[\dashv]$, and symmetry \times by $[\dashv]$. It is clear that $\mathcal{F}\Gamma/\equiv_L(0, 1)$ has one equivalence class, $[\dashv]$, 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 $\text{Cat} \rightarrow \text{Graph}$ with the adjunction $\text{Pro} \rightarrow \text{MonGraph}$, 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. URL: <https://arxiv.org/abs/2201.09969>, 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, 03 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:[https://doi.org/10.1016/S0304-3975\(00\)00382-0](https://doi.org/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. URL: <https://lmcs.episciences.org/6213>, doi:10.23638/LMCS-16(1:32)2020.
- 6 Matthew Earnshaw and Paweł Sobociński. Regular monoidal languages, 2022. URL: <https://arxiv.org/abs/2207.00526>, 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, 11 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 Rothmund, Nick Papadakis, and Erik Winfree. Algorithmic self-assembly of DNA Sierpinski triangles. *PLOS Biology*, 2(12), 12 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.

Appendix C




Paper II

Matthew Earnshaw and Paweł Sobociński. “String Diagrammatic Trace Theory”.
In: *48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023)*. Ed. by Jérôme Leroux, Sylvain Lombardy and David Peleg. Vol. 272. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023, 43:1–43:15. ISBN: 978-3-95977-292-1. DOI: 10.4230/LIPIcs.MFCS.2023.43

String Diagrammatic Trace Theory

Matthew Earnshaw   

Department of Software Science, Tallinn University of Technology, Estonia

Paweł Sobociński   

Department of Software Science, Tallinn University of Technology, Estonia

Abstract

We extend the theory of formal languages in monoidal categories to the multi-sorted, symmetric case, and show how this theory permits a graphical treatment of topics in concurrency. In particular, we show that Mazurkiewicz trace languages are precisely *symmetric monoidal languages* over *monoidal distributed alphabets*. We introduce *symmetric monoidal automata*, which define the class of regular symmetric monoidal languages. Furthermore, we prove that Zielonka’s asynchronous automata coincide with symmetric monoidal automata over monoidal distributed alphabets. Finally, we apply the string diagrams for symmetric premonoidal categories to derive serializations of traces.

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

Keywords and phrases symmetric monoidal categories, Mazurkiewicz traces, asynchronous automata

Funding Estonian Research Council grant PRG1210 and the European Union under Grant Agreement No. 101087529. Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or European Research Executive Agency. Neither the European Union nor the granting authority can be held responsible for them.

Acknowledgements We thank Chad Nester, Mario Román, and Niels Voorneveld for discussions.

1 Introduction

Monoidal languages [12] generalize formal languages of words to formal languages of *string diagrams*. String diagrams [16, 29] are a graphical representation of morphisms in *monoidal categories*. Monoidal categories can be considered *2-dimensional monoids* [6]: just as monoids are categories with one object, whose morphisms are elements of the monoid, (strict) monoidal categories can be defined as 2-categories with one object. Accordingly, *monoidal languages* are subsets of morphisms in free monoidal categories, just as word languages are subsets of free monoids. *Regular* monoidal languages are those specifiable by finitary grammars or automata. Our paper [12] introduced these devices and examined properties of languages in single-sorted, planar monoidal categories. These include regular languages of words and trees, but also languages of *planar* string diagrams that are neither linear nor tree-like.

In this paper, motivated by concurrency theory, we extend this theory to *coloured props*: multi-sorted monoidal categories with symmetries (Section 2). The resulting theory of *symmetric* monoidal languages (Section 3) captures languages of diagrams having multiple colours of string and in which strings may cross, permitting non-planar diagrams. In terms of concurrency, colours represent different *runtimes*, or *threads of execution*.

Indeed, in Section 4 we show that Mazurkiewicz trace languages [21] are exactly symmetric monoidal languages over alphabets of a particular shape called *monoidal distributed alphabets*. In Section 5 we introduce automata for symmetric monoidal languages, defining the class of *regular* symmetric monoidal languages. Then, in Section 6 we show that these are exactly the asynchronous automata of Zielonka [32] when instantiated over monoidal distributed alphabets. Finally, in Section 7 we use the algebra of symmetric *premonoidal* categories to show how serialization of traces can be treated string-diagrammatically.

Related work

Our previous work [12] introduced monoidal languages in the planar, single-sorted case; that is, languages of morphisms in free *props*. Similar languages of graphs were studied by Bossut [5], but their underlying algebra was not made explicit. Here, we again leverage the algebraic perspective, extending our theory to symmetric multi-sorted monoidal categories (props).

In the introduction to Joyal & Street’s foundational work on string diagrams for monoidal categories [16], it is suggested that string diagrams have a connection to the *heaps* of Viennot [30]. Heaps are known to be equivalent to Mazurkiewicz trace monoids (also known as partially commutative monoids) [17], but a precise formulation of the suggested relation with string diagrams has not appeared in the literature until now.

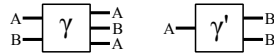
The notion of dependence graph [13] has also been used to give a topological presentation of Mazurkiewicz traces. Our use of the algebra of monoidal categories, rather than graphs, has various advantages. For example, we can apply our language theory for monoidal categories to traces, and we see notions such as asynchronous automata arise naturally from this. It also suggests generalizations of trace languages, in particular going beyond the case of atomic actions (Remark 23). Finally, it brings our work into proximity with the semantics of Petri nets and other formalisms for concurrency based on monoidal categories [2, 24].

2 Monoidal Graphs, Props and their String Diagrams

In this section we recall the basic definitions used in the following, including the specific flavour of monoidal categories known as *props* [20], along with their string diagrams [16, 29]. Just as a category can be presented by a directed graph, (strict) monoidal categories can be presented by *monoidal graphs*, a kind of multi-input, multi-output directed graph.

► **Definition 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.*

The alphabets of monoidal languages will be finite monoidal graphs: those in which $B_{\mathcal{G}}$ and $S_{\mathcal{G}}$ are both finite sets. In fact, since we are interested in finite state machines over finite alphabets, we will work exclusively with finite monoidal graphs. Diagrammatically, a (finite) 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 sorts given by the source and target functions. For example, the following depicts the monoidal graph \mathcal{G} with $B_{\mathcal{G}} = \{\gamma, \gamma'\}$, $S_{\mathcal{G}} = \{A, B\}$, $s(\gamma) = AB, t(\gamma) = ABA, s(\gamma') = A, t(\gamma') = BB$:

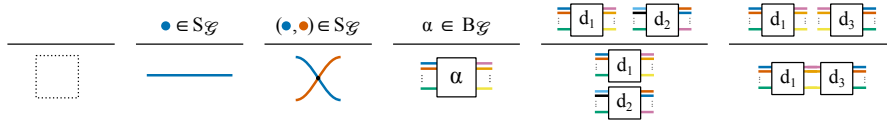


Sorts of a monoidal graph are sometimes called *colours*, since we could equally use different colours of string to represent different sorts, and we shall do so in places below. For a box $\gamma \in B_{\mathcal{G}}$ we call $s(\gamma)$ and $t(\gamma)$ the *arity* and *coarity* of γ , respectively, and write $\gamma : s(\gamma) \rightarrow t(\gamma)$. We will also call γ considered together with its arity and coarity a *generator*.

Monoidal graphs are generating data for monoidal categories. Recall that a *strict monoidal category* is a category \mathcal{C} , equipped with a functor $\otimes : \mathcal{C} \times \mathcal{C} \rightarrow \mathcal{C}$ (the *monoidal product*) and a unit object $I \in \mathcal{C}$, such that \otimes is associative and unital. A strict monoidal category is *symmetric* if there is a natural family of *symmetry morphisms* $\sigma_{A,B} : A \otimes B \rightarrow B \otimes A$, for each pair of sorts, satisfying $\sigma_{B,A} \circ \sigma_{A,B} = 1_{A \otimes B}$. The monoidal product turns the sets of objects and morphisms in \mathcal{C} into monoids. A *prop* is a symmetric strict monoidal category

whose monoid of objects is a free monoid.¹ While the above data can be intimidating to the non-expert, the free prop $\mathcal{F}_\times \mathcal{G}$ on a monoidal graph \mathcal{G} can be described in an intuitive and straightforward way: its arrows are the *string diagrams* generated by \mathcal{G} .

► **Definition 2.** *The free prop $\mathcal{F}_\times \mathcal{G}$ on a monoidal graph \mathcal{G} has monoid of objects $S_\mathcal{G}^*$ and morphisms string diagrams inductively defined as follows:*



Left to right: the empty diagram is a diagram; for every sort, the string on that sort is a diagram; for every pair of sorts, the symmetric braiding is a diagram; the diagram for every generator α is a diagram; for any two diagrams their vertical juxtaposition is a diagram; and for any two diagrams with matching right and left boundaries, the diagram obtained by joining the matching wires is a diagram (their composition). The monoidal product is given on objects by concatenation, on diagrams by juxtaposition, and the unit is the empty word.

The idea is simple: we treat generators like circuit components, and we have a supply of wires (identity morphisms). We also have the ability to cross wires, without tangling them; we do not distinguish over-crossings from under-crossings. A string diagram is then just any (open) circuit that we can build. This notation is 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 [16]. Working with string diagrams rather than the usual term syntax for morphisms is more intuitive, and leads to shorter proofs as the structural equations hold automatically: for example, interchange of morphisms (Figure 1, left), unbraiding of symmetries (centre), and sliding of morphisms past symmetries (right).



■ **Figure 1** These pairs of string diagrams are equal, reflecting the functoriality of \otimes (*interchange*), inverses of symmetries, and naturality of symmetries, respectively.

► **Definition 3.** *A morphism of monoidal graphs $\varphi : \mathcal{H} \rightarrow \mathcal{G}$ is given by functions $B_\varphi : B_\mathcal{H} \rightarrow B_\mathcal{G}$ and $S_\varphi : S_\mathcal{H} \rightarrow S_\mathcal{G}$ compatible with source and target functions: $S_\varphi^* \circ s = s \circ B_\varphi$ and $S_\varphi^* \circ t = t \circ B_\varphi$, where S_φ^* is the unique monoid homomorphism determined by S_φ .*

Morphisms of monoidal graphs freely generate morphisms of props: strict monoidal functors preserving sorts. Every prop has an underlying monoidal graph whose boxes are all the morphisms of the prop. This extends to an adjunction $\mathcal{F}_\times \dashv \mathcal{U}$ between the categories of monoidal graphs and props, where \mathcal{U} takes the underlying monoidal graph of a prop [16].

Monoidal categories have been applied to the study of both computing and physical processes [8, 9, 18, 25]. In these contexts, the monoidal product represents parallel composition of processes, and interchange reflects the *independence* of processes running in parallel. This is the main feature of monoidal categories that we will leverage in our representation of *traces* (Section 4). The use of multi-sorted props will allow fine-grained control of interchange.

¹ Some literature takes prop to mean that the monoid of objects is generated by a single object (and so isomorphic to \mathbb{N}), using the term *coloured prop* for the general case above.

3 Symmetric Monoidal Languages

Our paper [12] treated the case of languages, grammars and automata over single-sorted *pros* (strict monoidal categories *without* symmetries), corresponding to languages of *planar* string diagrams with one string colour. In this section we introduce the *multi-sorted* (or “coloured”) *symmetric monoidal languages*, which will be needed in the following to extend monoidal language theory to trace theory. In Section 5 we introduce the corresponding automata.

Just as a classical formal language is a subset of a free monoid, a symmetric monoidal language is a subset of morphisms in a free prop:

► **Definition 4.** *Let Γ be a finite monoidal graph. A symmetric monoidal language over Γ is a set of morphisms in the free prop $\mathcal{F}_\times\Gamma$ over Γ .*

A morphism of finite directed graphs $G \rightarrow \Sigma$, where Σ is a graph with one vertex, amounts to a labelling of the edges of G by edges of Σ . This is the starting point of Walters’ definition of regular grammar [31], which inspires the following definition:

► **Definition 5.** *A regular monoidal grammar is a morphism of finite monoidal graphs.*

For a regular monoidal grammar $M \xrightarrow{\varphi} \Gamma$, the monoidal graph Γ is the *alphabet*, and the generators of M , with their labelling by φ , correspond to production rules: see Example 7.

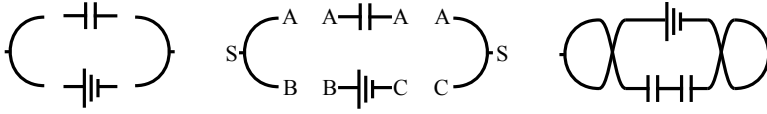
In the classical setting of word languages, a morphism of finite directed *graphs* $G \rightarrow \Sigma$ determines a *regular language* over Σ once we specify initial and final state vertices in G . In a regular monoidal grammar $M \rightarrow \Gamma$, the “vertices” of M are words over S_M , leading to various natural choices of boundary condition (Remark 8). In this paper, we will take initial and final words over S_M^* . Specifying these words defines the symmetric monoidal language of the grammar (Definition 9), and we define the languages arising in this way to be the regular symmetric monoidal languages.

We illustrate these definitions with some pedagogical toy examples. In the remaining sections of this paper, we turn to our extended application in concurrency, and we shall see that Mazurkiewicz trace languages are a natural example of symmetric monoidal languages.

► **Example 6.** Let $\varphi : M \rightarrow \Gamma$ be the regular monoidal grammar where M and Γ have a single sort (\bullet) and no boxes, with $S_\varphi(\bullet) = \bullet$, and initial and final states $n \in S_M^* \cong \mathbb{N}$. Then the symmetric monoidal language of this grammar is the set of *permutations* of n wires: morphisms consisting only of symmetries and identities.

Props have been used to give syntax and semantics for various kinds of *signal flow graph* and *circuit diagrams* [1, 3, 4]. Intuitively, props are well suited for this purpose since wires may freely cross in a circuit.

► **Example 7.** We give a regular monoidal grammar for the syntax of (open) circuits with $n \geq 0$ capacitors in series with a single voltage source (Figure 2). The alphabet Γ has a single sort, and boxes four circuit components (Figure 2, left). The monoidal graph M has four sorts $\{S, A, B, C\}$ and four boxes $s : S \rightarrow AB, c : A \rightarrow A, v : B \rightarrow C, s' : AC \rightarrow S$. S_φ maps the four sorts to the single sort of Γ , and B_φ maps each box to a circuit component. We can draw the grammar $\varphi : M \rightarrow \Gamma$ in a single diagram by drawing the graph for M but replacing each box b with its image under the grammar morphism $B_\varphi(b)$ (Figure 2, centre). The initial and final languages are the single state $\{S\}$. Intuitively, the symmetric monoidal language determined by the grammar is all of the string diagrams $S \rightarrow S$ that can be built using the “sorted” boxes of Γ , then forgetting the sorts.



■ **Figure 2** (Left) The alphabet Γ , giving syntax for circuits. (Centre) A regular monoidal grammar over Γ . (Right) An element of the regular symmetric monoidal language determined by this grammar.

► **Remark 8.** As mentioned above, there are various possible choices for the “initial and final states” of a monoidal grammar. In our previous paper [12], we took the empty word, giving languages of scalar string diagrams (i.e. no “dangling wires”): this neatly generalizes tree grammars. More generally, one can take initial and final *regular languages* of states over S_M , as considered by Bossut [5].

The free prop construction can be used to concisely describe the symmetric monoidal language of a regular monoidal grammar, defining the class of *regular symmetric monoidal languages*:

► **Definition 9.** Let $(\varphi : M \rightarrow \Gamma, I, F)$ be a regular monoidal grammar equipped with regular languages $I, F \subseteq S_M^*$. This determines a symmetric monoidal language by taking the image of the set of morphisms $\bigcup_{i \in I, f \in F} \mathcal{F}_\times M(i, f)$ under $\mathcal{F}_\times \varphi$, giving a set of morphisms in $\mathcal{F}_\times \Gamma$. The languages arising in this way are defined to be the regular symmetric monoidal languages.

In this paper, we will only need the case where I, F consist of single words. The slogan for the general case is that 2-dimensional regular languages have 1-dimensional regular boundaries. In Section 5, we will see that regular symmetric monoidal languages may equivalently be specified by *non-deterministic monoidal automata*.

► **Remark 10.** A regular monoidal grammar determines not only a regular symmetric monoidal language, but also a language in any algebraic structure generated by monoidal graphs, including planar monoidal categories (treated in [12]), and premonoidal categories (which we will use in Section 7). This is analogous to the way in which a finite labelled directed graph may generate both a subset of a free monoid, but also a subset of a free group, by freely adding inverses to the graph. Moreover, many properties of planar regular monoidal languages such as their closure properties proved in [12] only use grammars, and hence the same proofs work for languages in these other algebras.

4 Mazurkiewicz Trace Languages as Symmetric Monoidal Languages

The theory of Mazurkiewicz traces [10, 21, 23] provides a simple but powerful model of concurrent systems. Traces are a generalization of *words* in which specified pairs of letters can commute. If we think of letters as corresponding to atomic *actions*, then commuting letters reflect the *independence* of those particular actions and so their possible concurrent execution: ab is observationally indistinguishable from ba if a and b are independent.

In this section, we show that trace languages are symmetric monoidal languages over monoidal graphs of a particular form that we call *monoidal distributed alphabets*. In Section 5 we introduce *symmetric monoidal automata*, which operationally characterize the regular symmetric monoidal languages. In Section 6 we turn to *asynchronous automata* [32], a well-known model accepting exactly the *recognizable* trace languages, and show that these automata are precisely symmetric monoidal automata over monoidal distributed alphabets.

4.1 Independence and distribution

We recall some definitions from Mazurkiewicz trace theory, before recasting them in terms of monoidal languages. Fix a finite set Σ , an alphabet thought of as a set of atomic actions.

► **Definition 11.** An independence relation on Σ is a symmetric, irreflexive relation I . The induced dependence relation, D_I is the complement of I .

► **Definition 12.** For I an independence relation, let \equiv_I be the least congruence on Σ^* such that $\forall a, b: (a, b) \in I \implies ab \equiv_I ba$. The quotient monoid $\mathcal{T}(\Sigma, I) := \Sigma^*/\equiv_I$ is the trace monoid.

► **Definition 13.** A (Mazurkiewicz) trace language over (Σ, I) is a subset of the trace monoid $\mathcal{T}(\Sigma, I)$.

An element of $\mathcal{T}(\Sigma, I)$ or *trace over* (Σ, I) is thus an equivalence class of words up to commutation of independent letters. A trace language may be thought of as the set of possible observations of a concurrent system's behaviour, in which independent letters stand for actions which may occur concurrently. Independence relations correspond to *distributions*:

► **Definition 14** ([23]). A distribution of an alphabet Σ is a finite tuple of non-empty alphabets $(\Sigma_1, \dots, \Sigma_k)$ such that $\bigcup_{i=1}^k \Sigma_i = \Sigma$.

► **Proposition 15** ([23]). A distribution of Σ corresponds to a function $\text{loc} : \Sigma \rightarrow \mathcal{P}^+(\{1, \dots, k\}) : \sigma \mapsto \{i \mid \sigma \in \Sigma_i\}$.

Such a function gives the set of “locations” of each action $\sigma \in \Sigma$. In terms of concurrency, we can consider this to be a set of memory locations, threads of execution, or runtimes in which σ participates. In particular, every action has a non-empty set of locations.

A well-known construction [23] allows us to move between independence relations and distributions: locations correspond to maximal cliques in the graph of the dependency relation. We recall this construction in the proof of Proposition 16, which refines this correspondence.

Let Ind_Σ be the poset of independence relations on Σ , with order the inclusion of relations. Similarly, define a preorder Dist_Σ on distributions by $(\Sigma_1, \dots, \Sigma_p) \leq (\Sigma'_1, \dots, \Sigma'_q)$ iff for each pair of distinct elements $a, b \in \Sigma$, if there exists $1 \leq j \leq q$ such that Σ'_j contains both a and b , then there exists an Σ_i containing both a and b . Finally, quotient this preorder by taking distributions to be equal up to permutation.

► **Proposition 16.** There is a Galois insertion $\text{Ind}_\Sigma \hookrightarrow \text{Dist}_\Sigma$.

Proof. We construct an injective monotone function $i : \text{Ind}_\Sigma \rightarrow \text{Dist}_\Sigma$. Let an independence relation I over Σ be given, with induced dependence relation D_I . Construct the undirected *dependency graph*: vertices are elements of Σ and there is an edge (a, b) for every $(a, b) \in D_I$. Choose an ordering of maximal cliques of D_I , and define a distributed alphabet by taking Σ_i to be the elements of Σ in the maximal clique i . Different orderings give the same distribution up to permutation, and so the same element of Dist_Σ . This is injective since distinct independence relations induce distinct dependency graphs. It is monotone since if $I \subseteq I'$ then the dependency graph D_I is at least as connected as $D_{I'}$, so if a, b both belong to a maximal clique of $D_{I'}$ then they will both belong to a maximal clique of D_I .

We construct a monotone function $r : \text{Dist}_\Sigma \rightarrow \text{Ind}_\Sigma$. Let $(\Sigma_1, \dots, \Sigma_k)$ be a distribution. Define a relation I by $(a, b) \in I \iff \text{loc}(a) \cap \text{loc}(b) = \emptyset$. This is irreflexive and symmetric, and so an independence relation. r is also clearly well-defined and monotone. Finally it is easy to check that $r \circ i : \text{Ind}_\Sigma \rightarrow \text{Ind}_\Sigma$ is the identity. ◀

Put otherwise, though the same independence relation may be induced by many different distributions, independence relations correspond bijectively with the distributions in the image of $i \circ r$, that is, the distributions obtained via the maximal clique construction.

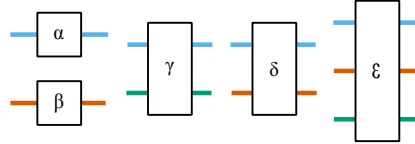
4.2 Symmetric monoidal languages over monoidal distributed alphabets

We now turn to the interpretation of these notions in terms of symmetric monoidal languages. A distribution can be seen as a monoidal graph in which sorts are the locations (runtimes).

► **Definition 17.** *A monoidal distributed alphabet is a finite monoidal graph Γ with the following properties:*

- Γ has set of sorts a finite ordinal $S_\Gamma = \{1 < 2 < \dots < k\}$ for $k \geq 1$,
- sorts $i \in S_\Gamma$ appear in order in the sources and targets of each generator $\gamma \in B_\Gamma$,
- each sort $i \in S_\Gamma$ appears at most once in each source and target,
- for each generator $\gamma \in B_\Gamma$, the sources and targets are non-empty and equal: $s(\gamma) = t(\gamma)$.

In brief, every generator in the alphabet is equipped with some set of runtimes, which serve as its source and target, and the runtimes are conserved. Figure 3 gives an example.



■ **Figure 3** An example of a monoidal distributed alphabet. For example, δ and β are independent but γ and α are not. We use colours for clarity, here blue = 1 < red = 2 < green = 3.

This gives us a way of representing distributions as monoidal graphs and vice-versa, if the graph is a monoidal distributed alphabet. Following Proposition 15, we will use $\text{loc}(\boxed{\gamma})$ to mean the arity (= coarity) of a generator $\boxed{\gamma}$. Since we choose a finite ordinal for the sorts, we have that:

► **Proposition 18.** *Distributed alphabets are in bijection with monoidal distributed alphabets.*

Since the *ordering* of the runtimes is ultimately not relevant to the structure of a trace, we should allow them to freely cross each other in our string diagrams: this is precisely what is enabled by taking the *symmetric* monoidal languages over these alphabets. We also need each runtime to appear once in each element of these languages, so we take the boundaries to be $1 \otimes \dots \otimes n$, which we will write as $\frac{!}{n}$.

► **Definition 19.** *A monoidal trace language is a symmetric monoidal language of the form $L \subseteq \mathcal{F}_\times \Gamma \left(\frac{!}{n}, \frac{!}{n} \right)$ where Γ is a monoidal distributed alphabet.*

Figure 4 gives an example of an element in a monoidal trace language over the monoidal distributed alphabet in Figure 3. We call such morphisms *monoidal traces*, and indeed we shall see below that they are exactly Mazurkiewicz traces. The corresponding string diagram gives an intuitive representation of traces as topological objects.

We now show that monoidal trace languages correspond precisely to Mazurkiewicz trace languages (Theorem 22), by establishing an isomorphism of monoids between trace monoids and monoids of string diagrams generated by monoidal distributed alphabets. Fix a monoidal distributed alphabet Γ . Recall that endomorphism hom-sets in a category are monoids under composition, and that the hom-set $\mathcal{F}_\times \Gamma \left(\frac{!}{n}, \frac{!}{n} \right)$ has elements string diagrams $\frac{!}{n} \rightarrow \frac{!}{n}$ over Γ .

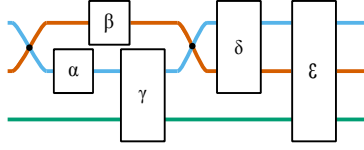


Figure 4 An example of a monoidal trace. β is independent of α and γ , but not δ or ϵ . Thus $\alpha\gamma\beta\delta\epsilon$ and $\beta\alpha\gamma\delta\epsilon$ are two possible serializations of this trace, corresponding to sliding β past α and γ in the string diagram. We use colours for sorts, blue = 1 < red = 2 < green = 3.

► **Lemma 20.** The hom-set $\mathcal{F}_\times\Gamma\left(\begin{smallmatrix} 1 & 1 \\ \vdots & \vdots \\ n & n \end{smallmatrix}\right)$ admits the following presentation as a monoid:

- **Generators:** For each $\boxed{\gamma} \in \Gamma$, the string diagram $N(\gamma) : 1 \otimes \dots \otimes n \rightarrow 1 \otimes \dots \otimes n$ built from symmetries, followed by $\boxed{\gamma}$ tensored with identities, followed by the inverse symmetry. See Figure 5 for an example.
- **Equations:** $N(\alpha) \circledast N(\beta) = N(\beta) \circledast N(\alpha) \iff \text{loc}(\boxed{\alpha}) \cap \text{loc}(\boxed{\beta}) = \emptyset$, where \circledast denotes composition of string diagrams in diagrammatic (left-to-right) order.

Proof. We construct an isomorphism between the monoids. Let $s \in \mathcal{F}_\times\Gamma\left(\begin{smallmatrix} 1 & 1 \\ \vdots & \vdots \\ n & n \end{smallmatrix}\right)$ be a string diagram. We can use interchange (Figure 1) to impose a linear order of generators from left to right in the diagram, e.g. $\boxed{\gamma_1}, \dots, \boxed{\gamma_n}$. This is called putting s in *general position*, by perturbing generators at the same horizontal position [16]. We then split the string diagram into a sequence of slices, each containing one generator. For a slice with right (or left) boundary $\begin{smallmatrix} k_1 \\ \vdots \\ k_n \end{smallmatrix}$, we can use the permutation $\begin{smallmatrix} k_1 \\ \vdots \\ k_n \end{smallmatrix} \rightarrow \begin{smallmatrix} 1 \\ \vdots \\ n \end{smallmatrix}$ followed by its inverse (or vice-versa) to finally obtain s as a sequence $N(\gamma_1) \circledast \dots \circledast N(\gamma_n)$. Any other possible sequence of generators is obtainable by repeatedly interchanging generators: this is possible if and only if their locations are disjoint. Consequently, this defines a function from $\mathcal{F}_\times\Gamma\left(\begin{smallmatrix} 1 & 1 \\ \vdots & \vdots \\ n & n \end{smallmatrix}\right)$ to the monoid presented above. Given that, as argued above, the slicing construction is unique up to interchanging independent generators, this defines a homomorphism. Conversely, given a generator $N(\gamma)$ in the presentation, we map this to the same string diagram in $\mathcal{F}_\times\Gamma\left(\begin{smallmatrix} 1 & 1 \\ \vdots & \vdots \\ n & n \end{smallmatrix}\right)$. Again, it follows from interchange that this extends to a homomorphism, inverse to that above. ◀

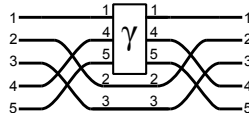


Figure 5 An example of a generator $N(\gamma)$ as in Lemma 20.

We now show that trace monoids are isomorphic to the endomorphism monoids $\mathcal{F}_\times\Gamma\left(\begin{smallmatrix} 1 & 1 \\ \vdots & \vdots \\ n & n \end{smallmatrix}\right)$.

► **Lemma 21.** Let I be an independence relation on an alphabet Σ , and Γ the monoidal distributed alphabet induced by the corresponding distribution (Proposition 18). Then there is an isomorphism of monoids $\mathcal{T}(\Sigma, I) \cong \mathcal{F}_\times\Gamma\left(\begin{smallmatrix} 1 & 1 \\ \vdots & \vdots \\ n & n \end{smallmatrix}\right)$.

Proof. We use the presentation of the endomorphism monoid given in Lemma 20. Define a homomorphism $\alpha : \mathcal{F}_\times\Gamma\left(\begin{smallmatrix} 1 & 1 \\ \vdots & \vdots \\ n & n \end{smallmatrix}\right) \rightarrow \mathcal{T}(\Sigma, I)$ by mapping generators $N(\gamma) \mapsto [\gamma]$. Let $N(\gamma) \circledast N(\gamma') = N(\gamma') \circledast N(\gamma)$, then it follows $[\gamma\gamma'] = [\gamma'\gamma]$ in $\mathcal{T}(\Sigma, I)$, since the former holds iff $\text{loc}(\gamma) \cap \text{loc}(\gamma') = \emptyset$, and so this extends to a homomorphism. Define a homomorphism

$\beta : \mathcal{T}(\Sigma, I) \rightarrow \mathcal{F}_\times \Gamma \left(\begin{smallmatrix} 1 \\ \vdots \\ n \end{smallmatrix}, \begin{smallmatrix} 1 \\ \vdots \\ n \end{smallmatrix} \right)$ by mapping generators $[\gamma] \mapsto N(\gamma)$. $[\gamma\gamma'] = [\gamma'\gamma]$ holds iff $\text{loc}(\gamma) \cap \text{loc}(\gamma') = \emptyset$, iff $\text{loc}(\boxed{\gamma}) \cap \text{loc}(\boxed{\gamma'}) = \emptyset$, iff $N(\gamma) \sharp N(\gamma') = N(\gamma') \sharp N(\gamma)$. Finally it is clear that α and β are inverses, and so witness an isomorphism of monoids. \blacktriangleleft

The following theorem is now immediate: given a monoidal trace language $L \subseteq \mathcal{F}_\times \Gamma \left(\begin{smallmatrix} 1 \\ \vdots \\ n \end{smallmatrix}, \begin{smallmatrix} 1 \\ \vdots \\ n \end{smallmatrix} \right)$ we obtain a trace language $L' \subseteq \mathcal{T}(\Sigma, I)$ using the isomorphism, and vice-versa:

► **Theorem 22.** *Monoidal trace languages are exactly Mazurkiewicz trace languages.*

Lemma 21 also shows that composition of traces corresponds simply to concatenation of the corresponding monoidal traces. Diagrams like Figure 4 are commonplace in the trace literature [11, 32]. Theorem 22 gives a formal basis for these diagrams as elements of symmetric monoidal languages.

► **Remark 23.** The idea of monoidal categories with a runtime is made precise by string diagrams for the *effectful categories* of Román [28]. Free props over monoidal distributed alphabets, considered as monoidal categories with *multiple runtimes* suggest a further generalization of effectful categories, sketched as a setting for concurrency by Jeffrey [14, Section 9.4]. We return to this in Section 7, where effectful (premonoidal) categories will be used to equip a trace language with a new runtime that enforces a strict ordering of events.

5 Symmetric Monoidal Automata

Monoidal automata give an alternative specification of the class of regular monoidal languages: they are analogues of finite-state automata in which transitions have multiple inputs and multiple outputs. Our paper [12] introduced monoidal automata for single-sorted, planar monoidal languages. However, the same data specifies an acceptor for single-sorted symmetric monoidal languages, if we inductively extend to *props*, rather than planar monoidal categories.

In this section we introduce monoidal automata over *multi-sorted* monoidal graphs and show how these recognize (multi-sorted) symmetric monoidal languages. In Section 6, we will see that the asynchronous automata of Zielonka [32] are a natural class of symmetric monoidal automata: those over monoidal distributed alphabets.

► **Definition 24.** *A non-deterministic monoidal semi-automaton is:*

- *an input alphabet, given by a finite monoidal graph Γ ,*
- *an family of non-empty, finite state sets $\{Q_c\}_{c \in S_\Gamma}$ indexed by the sorts of Γ ,*
- *for each $\gamma : c_1 \dots c_n \rightarrow c'_1 \dots c'_m$ in Γ , a transition function $\Delta_\gamma : \prod_{i=1}^n Q_{c_i} \rightarrow \mathcal{P}(\prod_{j=1}^m Q_{c'_j})$.*

As noted in Section 3, there are several candidates for a notion of initial/final state. In the following, we take initial and final words i, f over $\prod Q_c$. A monoidal semi-automaton equipped with initial and final words turns it into a (non-deterministic) *monoidal automaton*.

For classical NFAs, the assignment $a \mapsto \Delta_a$ extends uniquely to a functor $\Sigma^* \rightarrow \text{Rel}$, the inductive extension of the transition structure from letters to words. We can similarly extend monoidal automata to string diagrams. First, we define the codomain prop, $\text{Rel}_{\Gamma, Q}$:

► **Definition 25.** *For a family of sets $\{Q_c\}_{c \in S_\Gamma}$ indexed by the sorts of Γ then $\text{Rel}_{\Gamma, Q}$ is the prop with:*

- *set of objects S_Γ^* ,*
- *morphisms $c_1 \dots c_n \rightarrow c'_1 \dots c'_m$ functions $\prod_{i=1}^n Q_{c_i} \rightarrow \mathcal{P}(\prod_{j=1}^m Q_{c'_j})$,*

- composition is the usual composition of relations, i.e. $f \circ g := \mu \circ \mathcal{P}(g) \circ f$, where μ is the canonical map from sets of subsets to subsets,
- \otimes is given on objects by concatenation,
- and on morphisms $f : \otimes_i c_i \rightarrow \otimes_j c'_j$ and $g : \otimes_k d_k \rightarrow \otimes_l d'_l$ by $f \otimes g := \nabla \circ (f \times g)$, where ∇ sends pairs of subsets to their cartesian product,
- symmetries $\sigma : c_1 c_2 \rightarrow c_2 c_1$ are functions $Q_{c_1} \times Q_{c_2} \rightarrow \mathcal{P}(Q_{c_2} \times Q_{c_1}) : (q, q') \mapsto \{(q', q)\}$.

Note that a non-deterministic monoidal semi-automaton amounts to a morphism of monoidal graphs $\Gamma \rightarrow \mathcal{U}\text{Rel}_{\Gamma, Q}$. The adjunction $\mathcal{F}_X \dashv \mathcal{U}$ implies that there is a unique extension to a strict monoidal functor $\mathcal{F}_X \Gamma \rightarrow \text{Rel}_{\Gamma, Q}$, which we call a non-deterministic symmetric monoidal semi-automaton. This functor maps a string diagram to a relation. When this relation relates the initial word to the final word, the string diagram is *accepted*:

► **Definition 26.** Let $\Delta : \mathcal{F}_X \Gamma \rightarrow \text{Rel}_{\Gamma, Q}$ be a non-deterministic monoidal automaton with initial and final states $i, f \in (\prod_c Q_c)^*$. Then the symmetric monoidal language accepted by Δ is the set of morphisms $\mathcal{L}(\Delta) := \{\alpha \in \mathcal{F}_X \Gamma \mid f \in \Delta(\alpha)(i)\}$.

Intuitively, a run of a symmetric monoidal automaton starts with a *word* of states, whose subwords are modified by transitions corresponding to generators. Identity wires do not modify the states, and symmetries permute adjacent states.

► **Observation 27.** There is an evident correspondence between non-deterministic monoidal automata and regular monoidal grammars. The graphical representation of a grammar (such as Figure 2) makes this most clear: it can also be thought of as the “transition graph” of a non-deterministic monoidal automaton.

► **Remark 28.** We can further abstract our definition of monoidal automaton by noting that $\text{Rel}_{\Gamma, Q}$ is a sub-prop of the Kleisli category of the powerset monad \mathcal{P} , and that this monad could be replaced by another commutative monad [27, Corollary 4.3]. For example, replacing \mathcal{P} with the maybe monad, we obtain deterministic monoidal automata.

6 Asynchronous Automata as Symmetric Monoidal Automata

Asynchronous automata were introduced by Zielonka [32] as a true-concurrent operational model of recognizable trace languages, a well-behaved subclass of trace languages analogous to regular languages. In this section we show they are precisely symmetric monoidal automata over monoidal distributed alphabets, which leads to the following theorem:

► **Theorem 29.** *Recognizable trace languages are exactly regular symmetric monoidal languages over monoidal distributed alphabets.*

We recall the definition of asynchronous automata, before turning to monoidal automata.

► **Definition 30** (Asynchronous automaton [32]). Let $(\Sigma_1, \dots, \Sigma_k)$ be a distribution of an alphabet Σ . For each $1 \leq i \leq k$, let Q_i be a non-empty finite set of states, and for each $\sigma \in \Sigma$ take a transition relation $\Delta_\sigma : \prod_{i \in \text{loc}(\sigma)} Q_i \rightarrow \mathcal{P}(\prod_{i \in \text{loc}(\sigma)} Q_i)$. This defines a global transition relation on the set $Q := \prod_{i=1}^k Q_i$ as follows:

$(q_1, \dots, q_k) \xrightarrow{\sigma} (q'_1, \dots, q'_k) \iff q_i = q'_i \text{ for } i \notin \text{loc}(\sigma) \text{ and } (q'_1, \dots, q'_j) \in \Delta_\sigma(q_{i_1}, \dots, q_{i_j})$
 where $\{i_1, \dots, i_j\} \in \text{loc}(\sigma)$. Finally let $\vec{i} \in Q, F \subseteq Q$ be initial and final words of states.

The global transition relation for σ leaves unchanged those states at locations in the complement of $\text{loc}(\sigma)$, and otherwise acts according to the local transition Δ_σ . An asynchronous automaton has a language over Σ given by the extension of the transition relation

to words. Moreover, asynchronous automata have a language of Mazurkiewicz traces over the distribution of Σ : a trace in $\mathcal{T}(\Sigma, I)$ is accepted when all of its serializations are accepted, which happens when one of its serializations is accepted [32, p. 109]. *Recognizable trace languages* are defined algebraically as those whose syntactic congruence is of finite index [32]. Zielonka’s theorem says that they also have an operational characterization:

► **Theorem 31** (Zielonka [32]). *Asynchronous automata accept precisely the recognizable trace languages.*

Definition 30 closely resembles that of symmetric monoidal automata. Indeed, asynchronous automata are precisely symmetric monoidal automata over monoidal distributed alphabets:

► **Proposition 32.** *For an asynchronous automaton \mathcal{A} , there is a symmetric monoidal automaton over a monoidal distributed alphabet with the same trace language, and vice-versa.*

Proof. An asynchronous automaton with multiple final state words can be normalized to a single final state word in the usual way by introducing a new final state word and modifying transitions appropriately. Then a symmetric monoidal automaton can be constructed by taking the monoidal distributed alphabet associated to the distribution of Σ (Proposition 18), the same transition relations, initial and final state words. We show that the languages coincide. Let $w \in \mathcal{L}(\mathcal{A})$, and consider the corresponding trace $[w]$. Using Lemma 21, we can produce the corresponding monoidal trace. By construction, this is accepted by the symmetric monoidal automaton defined above. The converse is analogous. ◀

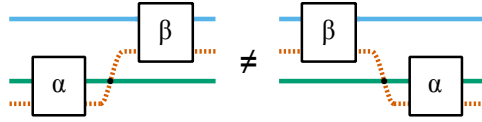
As a corollary, we can invoke Theorem 31 to obtain Theorem 29. In contrast to asynchronous automata, the constructed symmetric monoidal automaton directly accepts traces qua string diagrams, rather than a language of words corresponding to a trace language.

► **Observation 33.** *Jesi, Pighizzini, and Sabadini [15] introduced probabilistic asynchronous automata. Initial and final states, and transition relations are replaced by initial and final distributions, and stochastic transitions. These are precisely what are obtained if the powerset monad in our definition of non-deterministic monoidal automaton (Remark 28) is replaced with the distribution monad [26], whose Kleisli category has morphisms stochastic matrices.*

7 Serialization via Premonoidal Categories

Trace theorists often consider trace languages to be word languages with the property of *trace-closure* with respect to an independence relation [19]: if $u \in L$ and $u \equiv_I v$ then $v \in L$. These languages arise as preimages of trace languages along the quotient map $q_{\Sigma, I} : \Sigma^* \rightarrow \mathcal{T}(\Sigma, I)$. For $L \subseteq \mathcal{T}(\Sigma, I)$ a trace language, $q_{\Sigma, I}^{-1}(L) \subseteq \Sigma^*$ is its *flattening* or *serialization*.

In this section we show that the serialization of monoidal trace languages can be carried out using the algebra and string diagrams of symmetric premonoidal categories. Premonoidal categories are like monoidal categories, except interchange (Figure 1) does not hold in general. The free (symmetric) premonoidal category on a monoidal graph was described using string diagrams by Román [28]. The idea is simple: the string diagrams are the same as for props, but an extra string (the “runtime”) threads through each generator, preventing interchange. Figure 6 shows two premonoidal morphisms $\bullet \otimes \bullet \rightarrow \bullet \otimes \bullet$ that are not equal:



■ **Figure 6** In the free premonoidal category over a monoidal graph, generators are augmented by a string on a new object called the *runtime* (dashed red). This prevents interchange (cf. Figure 1).

In Appendix A, we explain in more detail the construction of the free symmetric premonoidal category $\mathcal{F}_p\Gamma$ on a monoidal graph Γ using string diagrams. In particular, the runtime string appears only once in each string diagram, reflecting that premonoidal categories do not have a tensor product on morphisms. The endomorphism monoid $\mathcal{F}_p\Gamma\left(\begin{smallmatrix} 1 \\ \vdots \\ n \end{smallmatrix}, \begin{smallmatrix} 1 \\ \vdots \\ n \end{smallmatrix}\right)$ is now the free monoid over the boxes of Γ , since the runtime prevents interchange:

► **Proposition 34.** *Let Γ be a monoidal distributed alphabet. Then $\mathcal{F}_p\Gamma\left(\begin{smallmatrix} 1 \\ \vdots \\ n \end{smallmatrix}, \begin{smallmatrix} 1 \\ \vdots \\ n \end{smallmatrix}\right) \cong B_\Gamma^*$, where B_Γ is the set of boxes of Γ .*

Proof. (Sketch) By augmenting the generators of Γ with a new runtime, we create a monoidal distributed alphabet in which every generator depends on every other, that is, the independence relation is empty. Thus the corresponding trace monoid is simply B_Γ^* . From here, we can follow the idea of Lemma 21. ◀

We can define a morphism of monoids $q_\Gamma : \mathcal{F}_p\Gamma\left(\begin{smallmatrix} 1 \\ \vdots \\ n \end{smallmatrix}, \begin{smallmatrix} 1 \\ \vdots \\ n \end{smallmatrix}\right) \rightarrow \mathcal{F}_\times\Gamma\left(\begin{smallmatrix} 1 \\ \vdots \\ n \end{smallmatrix}, \begin{smallmatrix} 1 \\ \vdots \\ n \end{smallmatrix}\right)$ by presenting $\mathcal{F}_p\Gamma\left(\begin{smallmatrix} 1 \\ \vdots \\ n \end{smallmatrix}, \begin{smallmatrix} 1 \\ \vdots \\ n \end{smallmatrix}\right)$ as in Lemma 20, and defining q_Γ on generators by erasing the runtime string. Theorem 35 then follows immediately from the definitions along with Lemma 21 and Proposition 34:

► **Theorem 35.** *For every alphabet B_Γ , the following square of monoid homomorphisms commutes, where q is the quotient monoid homomorphism.*

$$\begin{array}{ccc} B_\Gamma^* & \xrightarrow{q} & \tau(B_\Gamma, I) \\ \cong \downarrow & & \downarrow \cong \\ \mathcal{F}_p\Gamma\left(\begin{smallmatrix} 1 \\ \vdots \\ n \end{smallmatrix}, \begin{smallmatrix} 1 \\ \vdots \\ n \end{smallmatrix}\right) & \xrightarrow{q_\Gamma} & \mathcal{F}_\times\Gamma\left(\begin{smallmatrix} 1 \\ \vdots \\ n \end{smallmatrix}, \begin{smallmatrix} 1 \\ \vdots \\ n \end{smallmatrix}\right) \end{array}$$

As a result, the preimage of a monoidal trace language under the morphism q_Γ corresponds to the serialization of that language.

8 Conclusion

There are several directions in which our theory could be developed. A semi-independence relation drops symmetry from an independence relation: it is simply an irreflexive relation. This gives rise to the theory of semicommutations [7], in which *directed* commutations may occur e.g. $ab \rightarrow ba$, but not vice-versa. This allows for a more fine-grained specification of concurrency. In terms of monoidal languages, it suggests consideration of monoidal distributed alphabets in which the sources and targets of generators may differ.

As noted in Remark 23, our treatment of trace languages suggests a generalization of the notion of *effectful category* [28] (which include premonoidal categories), in which there are *multiple runtimes*. This would enable a semantics for concurrent systems in which we can

consider not only *atomic* actions, but also actions with input and output types. We plan to pursue this axiomatically in future work.

Mazurkiewicz originally introduced traces to give semantics to Petri nets, and showed that this semantics is compositional with respect to *synchronization* of traces [21]. Petri nets have been given semantics in monoidal categories [2, 22], and so the precise relationship of our monoidal formulation of traces to Petri nets remains to be worked out. In particular, this would involve understanding trace synchronization in terms of monoidal categories.

Finally, proofs of Zielonka's theorem (Theorem 31, see [32] for details) remain highly technical, despite several simplifications since Zielonka's version. Investigation of whether the algebra of monoidal categories might yield further simplifications is an intriguing direction.

References

- 1 John C. Baez, Brandon Coya, and Franciscus Rebro. Props in network theory. *Theory and Applications of Categories*, 33(25):727–783, 2018.
- 2 John C Baez, Fabrizio Genovese, Jade Master, and Michael Shulman. Categories of nets. In *2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–13. IEEE, 2021.
- 3 Guillaume Boisseau and Pawel Sobocinski. String diagrammatic electrical circuit theory. *Electronic Proceedings in Theoretical Computer Science*, 372:178–191, 2022.
- 4 Filippo Bonchi, Pawel Sobocinski, and Fabio Zanasi. Full abstraction for signal flow graphs. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '15, page 515–526, New York, NY, USA, 2015. Association for Computing Machinery. doi:10.1145/2676726.2676993.
- 5 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.
- 6 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.
- 7 M Clerbout, M Lattaux, and Y Roos. Semi-commutations. In V Diekert and G Rozenberg, editors, *The Book of Traces*. World Scientific, 1995.
- 8 Bob Coecke, Tobias Fritz, and Robert W. Spekkens. A mathematical theory of resources. *Information and Computation*, 250:59–86, 2016. Quantum Physics and Logic. doi:10.1016/j.ic.2016.02.008.
- 9 Bob Coecke and Aleks Kissinger. *Picturing quantum processes : a first course in quantum theory and diagrammatic reasoning*. Cambridge University Press, 2017.
- 10 V Diekert and G Rozenberg. *The Book of Traces*. World Scientific, 1995. doi:10.1142/2563.
- 11 Volker Diekert and Anca Muscholl. On distributed monitoring of asynchronous systems. In Luke Ong and Ruy de Queiroz, editors, *Logic, Language, Information and Computation*, pages 70–84, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.
- 12 Matthew Earnshaw and Pawel 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. doi:10.4230/LIPIcs.MFCS.2022.44.
- 13 Hoogeboom H J and G Rozenberg. Dependence graphs. In V Diekert and G Rozenberg, editors, *The Book of Traces*. World Scientific, 1995.
- 14 Alan Jeffrey. Premonoidal categories and a graphical view of programs. *Preprint*, 1998.
- 15 S. Jesi, G. Pighizzini, and N. Sabadini. Probabilistic asynchronous automata. *Mathematical systems theory*, 29(1):5–31, Feb 1996. doi:10.1007/BF01201811.
- 16 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.

- 17 C. Krattenthaler. The theory of heaps and the Cartier-Foata monoid. In P. Cartier and D. Foata, editors, *Commutation and Rearrangements*. European Mathematical Information Service, 2006.
- 18 Elena Di Lavore, Giovanni de Felice, and Mario Román. Coinductive streams in monoidal categories, 2022. [arXiv:2212.14494](https://arxiv.org/abs/2212.14494).
- 19 Hendrik Maarand and Tarmo Uustalu. Reordering derivatives of trace closures of regular languages. In *30th International Conference on Concurrency Theory (CONCUR 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.
- 20 Saunders MacLane. Categorical algebra. *Bulletin of the American Mathematical Society*, 71(1):40 – 106, 1965.
- 21 Antoni Mazurkiewicz. Basic notions of trace theory. In J. W. de Bakker, W. P. de Roever, and G. Rozenberg, editors, *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, pages 285–363, Berlin, Heidelberg, 1989. Springer Berlin Heidelberg.
- 22 José Meseguer and Ugo Montanari. Petri nets are monoids. *Information and Computation*, 88(2):105–155, 1990. doi:10.1016/0890-5401(90)90013-8.
- 23 Madhavan Mukund. Automata on distributed alphabets. In *Modern Applications of Automata Theory*, pages 257–288. World Scientific, 2012. doi:10.1142/9789814271059_0009.
- 24 Chad Nester. Concurrent Process Histories and Resource Transducers. *Logical Methods in Computer Science*, Volume 19, Issue 1, January 2023. doi:10.46298/lmcs-19(1:7)2023.
- 25 Dusko Pavlovic. Monoidal computer I: Basic computability by string diagrams. *Information and Computation*, 226:94–116, 2013. Special Issue: Information Security as a Resource. doi:10.1016/j.ic.2013.03.007.
- 26 Paolo Perrone. Distribution monad (nlab entry), 2019. <https://ncatlab.org/nlab/show/distribution+monad>, Last accessed 2023-03-13.
- 27 John Power and Edmund Robinson. Premonoidal categories and notions of computation. *Mathematical Structures in Computer Science*, 7(5), 1997. doi:10.1017/S0960129597002375.
- 28 Mario Román. Promonads and string diagrams for effectful categories. In *ACT '22: Applied Category Theory, Glasgow, United Kingdom, 18 - 22 July, 2022*, 2022. [arXiv:2205.07664](https://arxiv.org/abs/2205.07664), doi:10.48550/arXiv.2205.07664.
- 29 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.
- 30 Gérard Xavier Viennot. Heaps of pieces, I : Basic definitions and combinatorial lemmas. In Gilbert Labelle and Pierre Leroux, editors, *Combinatoire énumérative*, pages 321–350, Berlin, Heidelberg, 1986. Springer Berlin Heidelberg.
- 31 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.
- 32 Wieslaw Zielonka. Notes on finite asynchronous automata. *RAIRO - Theoretical Informatics and Applications - Informatique Théorique et Applications*, 21(2):99–135, 1987.

A Symmetric Strict Premonoidal Categories and Functors

We recall the definitions of (symmetric) strict premonoidal categories and their functors. For more details, see the papers [27, 28].

► **Definition 36.** A strict premonoidal category is a category \mathcal{C} equipped with:

- for each pair of objects $A, B \in \mathcal{C}$ an object $A \otimes B$,
- for each object $A \in \mathcal{C}$ a functor $A \triangleleft -$ whose action on objects sends B to $A \otimes B$,
- for each object $A \in \mathcal{C}$ a functor $- \triangleright A$ whose action on objects sends B to $B \otimes A$, and
- a unit object I ,

such that,

- for each $A \in \mathcal{C}$, strict unitality $I \otimes A = A = A \otimes I$ holds, and

- for each triple $A, B, C \in \mathcal{C}$, strict associativity $A \otimes (B \otimes C) = (A \otimes B) \otimes C$ holds.

The families of functors $A \triangleleft -, - \triangleright A$ are called the *whiskerings* with A : in a premonoidal category we do not have a tensor product of morphisms in general, but we can put an identity on either side of a morphism. A morphism $f : A \rightarrow B \in \mathcal{C}$ is *central* if for every morphism $g : C \rightarrow D$, $(B \triangleleft g) \circ (f \triangleright C) = (f \triangleright C) \circ (A \triangleleft g)$, in other words, f is central if it interchanges with every other morphism g .

► **Definition 37.** A strict premonoidal category is *symmetric* if it is further equipped with a natural isomorphism whose components $c_{A,B} : A \otimes B \rightarrow B \otimes A$ are central and such that $c_{B,A} \circ c_{A,B} = 1_{A \otimes B}$.

► **Definition 38.** A strict premonoidal functor $F : \mathcal{C} \rightarrow \mathcal{D}$ is a functor sending central morphisms to central morphisms and such that $F(I_{\mathcal{C}}) = I_{\mathcal{D}}$, $F(X \otimes Y) = F(X) \otimes F(Y)$.

A.1 String Diagrams for Premonoidal Categories

We recall the construction of the free symmetric strict premonoidal category over a monoidal graph. This is a special case of the construction of free *effectful categories* in [28, Section 2.3].

We first define the *runtime monoidal graph* over a monoidal graph, which augments the generators with a new wire:

► **Definition 39.** Let \mathcal{G} be a monoidal graph. Let R be a sort disjoint from $S_{\mathcal{G}}$. The runtime monoidal graph \mathcal{G}_R has sorts $S_{\mathcal{G}} + \{R\}$ and for each generator $\gamma : S_1 \dots S_n \rightarrow S'_1 \dots S'_m$ in \mathcal{G} a generator $\gamma : RS_1 \dots S_n \rightarrow RS'_1 \dots S'_m$.

Graphically we can depict \mathcal{G}_R as in Figure 7 (right):



■ **Figure 7** Left: A monoidal graph \mathcal{G} . Right: the associated runtime monoidal graph \mathcal{G}_R , where the new sort R is drawn as a dashed string.

► **Definition 40.** The symmetric runtime monoidal category is the free prop $\mathcal{F}_x \mathcal{G}_R$ on \mathcal{G}_R .

► **Theorem 41.** The free symmetric strict premonoidal category $\mathcal{F}_p \mathcal{G}$ on a monoidal graph \mathcal{G} has set of objects $S_{\mathcal{G}}$ and a morphism $S_1 \otimes \dots \otimes S_n \rightarrow S'_1 \otimes \dots \otimes S'_m$ is a morphism $R \otimes S_1 \otimes \dots \otimes S_n \rightarrow R \otimes S'_1 \otimes \dots \otimes S'_m$ in the symmetric runtime monoidal category.

Proof. The proof follows [28, Theorem 2.14], in the case where \mathcal{V} is empty, and taking instead the free symmetric strict monoidal category. ◀

In particular note that we no longer have a tensor product of morphisms in $\mathcal{F}_p \mathcal{G}$, since the runtime must appear only once in each domain and codomain, but we do have whiskerings for each object.

Consequently the string diagrams for morphisms $A \rightarrow B$ in $\mathcal{F}_p \mathcal{G}$ are just morphisms $R \otimes A \rightarrow R \otimes B$ in the symmetric runtime monoidal category [28, Corollary 2.15].

Appendix **D**

Paper III

Matthew Earnshaw and Paweł Sobociński. “Regular planar monoidal languages”.
In: *Journal of Logical and Algebraic Methods in Programming* 139 (2024), p. 100963.
ISSN: 2352-2208. DOI: <https://doi.org/10.1016/j.jlamp.2024.100963>

Regular Planar Monoidal Languages^{*}

Matthew Earnshaw¹, Paweł Sobociński¹

^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 [28], rational sequences [1], trees [20, 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, 37].

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 [17], and Thatcher and Wright [36].

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 monoidal categories. Monoids can be seen as *categories* with one object, in which morphisms are the elements of the monoid. 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 [15]. 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 [15]. 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 [21], 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 [39] 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. [32] 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 4.8.

Walters’ note [38] on regular and context-free grammars served as a starting point for our definition of regular monoidal grammar. Rosenthal [31], 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. [18] investigated languages of higher-dimensional 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, 34]. 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 $B_{\mathcal{G}} = \{\gamma, \gamma'\}$, $S_{\mathcal{G}} = \{A, B\}$, $s(\gamma) = AB$, $t(\gamma) = ABA$, $s(\gamma') = A$, $t(\gamma') = BB$:

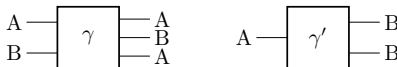


Figure 1: Example of a monoidal graph. Monoidal graphs form the building blocks of string diagrams: a diagrammatic notation for morphisms in monoidal categories.

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 $\gamma : s(\gamma) \rightarrow t(\gamma)$ 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 $\otimes : \mathcal{C} \times \mathcal{C} \rightarrow \mathcal{C}$ (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 $\mathcal{F}\mathcal{G}$ on a monoidal graph \mathcal{G} has monoid of objects $S_{\mathcal{G}}^*$ and morphisms string diagrams inductively defined as in Figure 3.4. The monoidal product is given on objects by concatenation, on diagrams by juxtaposition, and the unit is the empty word.*

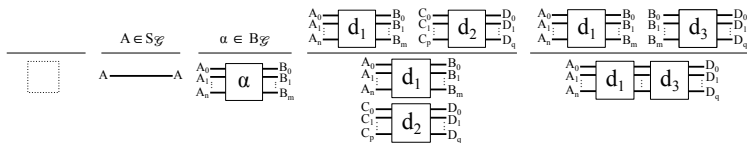


Figure 2: Inductive definition of morphisms in the free pro on a monoidal graph. Left to right: the empty diagram is a diagram; for each sort, the identity string is a diagram; the diagram for every generator α is a diagram; for any two diagrams their vertical juxtaposition is a diagram; and for any two diagrams with matching right and left boundaries, the diagram obtained by joining the matching wires is a 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, 34]. 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} are 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 $\Psi : \mathcal{G}' \rightarrow \mathcal{G}$ of monoidal graphs is a pair of functions $S_{\Psi} : S_{\mathcal{G}} \rightarrow S_{\mathcal{G}'}$, $B_{\Psi} : B_{\mathcal{G}} \rightarrow B_{\mathcal{G}'}$ 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 homomorphism determined by S_{Ψ} .*

Monoidal graphs and their morphisms form a category $\mathbf{MonGraph}$. 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 $F : \mathcal{C} \rightarrow \mathcal{D}$, where \mathcal{C} and \mathcal{D} are monoidal categories, and $F(X \otimes Y) = F(X) \otimes F(Y)$, $F(I_{\mathcal{C}}) = I_{\mathcal{D}}$. A morphism of a pros is simply a strict monoidal functor whose action on objects is determined by a function between their sets of generating sorts.

Any monoidal category has an underlying monoidal graph, given by forgetting composition and monoidal product:

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

There is a “free-forgetful” adjunction $\mathcal{F} \dashv \mathcal{U} : \text{Pro} \rightarrow \text{MonGraph}$ (see for example, [24, 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 $L \subseteq \mathcal{F}\Gamma(0, 0)$ of morphisms with arity and coarity 0 in the free pro category 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 [38], so we briefly explain his elegant idea.

Walters [38] 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 $\phi : G \rightarrow \Sigma$, 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 assignments. On vertices a morphism

$G \rightarrow \Sigma$ 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) appear in the *domain* of a morphism. Roughly speaking, a grammar is *fibred* 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 $\Psi : \mathcal{M} \rightarrow \Gamma$ 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 $\psi : S_{\mathcal{M}} \rightarrow \{\bullet\}$ is unique, and the grammar is determined by its box function $B_{\Psi} : B_{\mathcal{M}} \rightarrow B_{\Gamma}$, 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 or a run over an automaton corresponds to a path in G , and the accepted word is given by the labelling of the path. Formally, 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, $\mathcal{F}\Psi : \mathcal{F}\mathcal{M} \rightarrow \mathcal{F}\Gamma$. We may also refer to these morphisms as grammars.*

Pros are monadic over monoidal graphs: the forgetful functor $\mathcal{U} : \mathbf{Pro} \rightarrow \mathbf{MonGraph}$ has a left adjoint $\mathcal{F} : \mathbf{MonGraph} \rightarrow \mathbf{Pro}$, and \mathbf{Pro} is equivalent to the category of algebras for the induced monad on $\mathbf{MonGraph}$ (see [19, §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 [34]).

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. From another perspective, we can think of a string diagram $s \in \mathcal{F}\mathcal{M}$ as representing a specification for the construction of the string diagram $\mathcal{F}\Psi(s) \in \mathcal{F}\Gamma$ 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’ [38] 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 M is free over a signature containing $1 \rightarrow 1$ generators (Section 7).

A regular monoidal grammar determines a monoidal language as follows:

Definition 4.5. *Given a regular monoidal grammar $\Psi : \mathcal{M} \rightarrow \Gamma$, the image under $\mathcal{F}\Psi$ of the endo-hom-set of the monoidal unit ε in $\mathcal{F}\mathcal{M}$ is a planar 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 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 (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 using the grammar in Figure 3. An example of a morphism in the language defined by this grammar is shown on the right in Figure 3.*

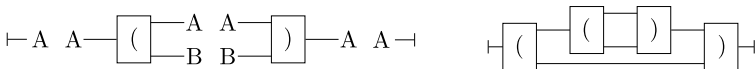


Figure 3: A regular monoidal grammar for balanced parentheses (left), and a morphism in the language (right).

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.

Example 4.7 (Brick walls). A variant on the “brick wall” language introduced by [4] is given by the following grammar (Figure 4). An example of a morphism in the language defined by this grammar is shown on the right in Figure 4.

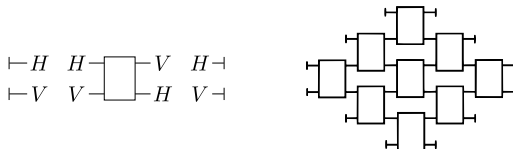


Figure 4: The grammar for the brick wall language (left), and a morphism in the language (right).

In Section 8 we will see how this language of “brick walls” allows us to construct the following example as an intersection of two languages:

Example 4.8 (Sierpiński triangles). *Rothmund, Papadakis, and Winfree [32] use self-assembly of DNA tiles to realize the behaviour of a cellular automaton that computes the Sierpiński triangle fractal, based on the computation of the XOR gate. [32] implicitly depicts a monoidal grammar, and so Sierpiński triangles of arbitrary iteration depth (e.g. Figure 6) are contained in the monoidal language over this grammar (Figure 5).*

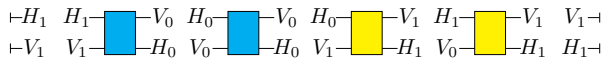


Figure 5: The regular monoidal grammar of Sierpiński triangles.

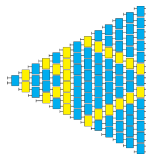


Figure 6: An element of the regular monoidal language of Sierpiński triangles.

Example 4.9. We define a grammar that will serve as a running counterexample in Sections 10 and 11, as it defines a language that cannot be deterministically recognized (Figure 7). In particular it does not satisfy the property of causal closure, a necessary condition for deterministic recognizability which we introduce in Section 10.

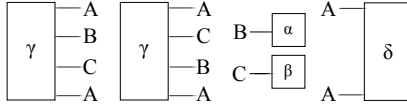


Figure 7: This grammar is “non-deterministic”: there are two possible transitions from the empty word when encountering γ .

The connected string diagrams in this language are exactly two (Figure 8).

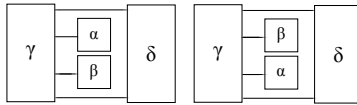


Figure 8: The connected string diagrams in the language of 7.

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 4.10. *If the monoidal graph \mathcal{M} has no edges whose domain is ε and 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 \square). 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 $\boxed{w} \in L$ where \boxed{w} may be factorized (as follows) into $m \geq n$ morphisms with boundaries (k_{i-1}, k_i) of width $1 \leq k_i \leq k$ and such that no $\boxed{w_i}$ is an identity morphism:*

$$\boxed{w_0} \left\{ \begin{array}{c} \vdots \\ k_0 \\ \vdots \\ k_{i-1} \end{array} \right\} \cdots \left\{ \begin{array}{c} \vdots \\ k_i \\ \vdots \\ k_{m-1} \end{array} \right\} \boxed{w_m}$$

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

$$\boxed{w'} \left\{ \begin{array}{c} \vdots \\ \ell \\ \vdots \\ \ell \end{array} \right\} \left\{ \left(\left\{ \begin{array}{c} \vdots \\ \ell \\ \vdots \\ \ell \end{array} \right\} \right)^p \right\} \boxed{w''} \left\{ \begin{array}{c} \vdots \\ \ell \\ \vdots \\ \ell \end{array} \right\} \boxed{w'''} \in L, \forall p \geq 0$$

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

Proof. Let L be the language of the grammar $\varphi : M \rightarrow \Gamma$. 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 $n = \sum_{i=0}^k |V_M|^i$. Let $\boxed{w} \in L$, such that it has a factorization of the form above. Then 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 $w'w''^pw''' \notin L$, 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 $n \geq 2$ strings, Unbraid_n , is a monoidal language defined over a monoidal alphabet with an under-braid, over-braid, and start, end generators with k prongs for $0 \leq k \leq n$. For example, taking $n = 4$ we have the alphabet in Figure 9.*



Figure 9: Monoidal alphabet for Unbraid_4 .

Connected elements of Unbraid_n are defined to be string diagrams with one start and one end generator, between which there is a configuration of n strings which could be “unbraided” to give two parallel strings, by planar isotopy. For example, Figure 10 (left) is an element of Unbraid_n for all $n \geq 2$, but Figure 10 (right) is not.



Figure 10: (Left) Example of an element in Unbraid_n : by it could be untangled by planar deformations. (Right) A string diagram not in Unbraid_n : we cannot uncross the strings using only planar moves.

To prove that Unbraid_n 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:

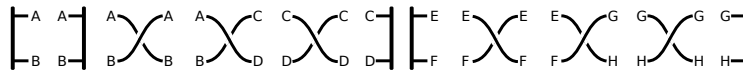


Figure 11: A regular monoidal grammar for the monoidal language $(O^*U^* + U^*O^*)_2$.

Proposition 5.6. *The languages Unbraid_n are not regular monoidal.*

Proof. Consider the intersection $\text{Unbraid}_n \cap (O^*U^* + U^*O^*)_2$. An element of $(O^*U^* + U^*O^*)_2$ is an unbraid just when it comprises n under-braidings followed by n over-braidings, or vice-versa and thus these are the elements of the intersection, which we denote by $(O^nU^n + U^nO^n)_2$. Figure 11 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 $L = (O^nU^n + U^nO^n)_2$ 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 $j - i$ O s, $j - i$ U s, or some number of O s followed by some number of U s. In the first two cases, pumping the section leads to a term with more O s than U s or vice-versa, and in the last case it will no longer be that all O s come before all U s. \square

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 [38] 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 [25], who gave a conceptual proof of the Chomsky-Schützenberger representation theorem using the algebra of multicategories. Earnshaw, Hefford, and Román [14] recently extended this algebra to the setting of monoidal categories, providing a foundation for a notion 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 $Q \xrightarrow{\Delta_a} \mathcal{P}(Q)$. Automata accepting monoidal languages, or

monoidal automata are defined similarly, but the type of the transition function will be different for each generator $\gamma \in \Gamma$. However, our *monoidal automata* do not have initial and final states; string diagrams are simply accepted or rejected depending on their shape. This corresponds to our languages consisting of *scalar* string diagrams; those with no dangling wires. 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.

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

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

For classical NFAs, the assignment $a \mapsto \Delta_a$ extends uniquely to a functor $\Sigma^* \rightarrow \text{Rel}$, 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 \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 , the category of sets and relations, 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 $\mathcal{P}X \times \mathcal{P}Y \rightarrow \mathcal{P}(X \times Y)$ given by the product of subsets), the following lemma gives us the monoidal structure on Rel :

Lemma 6.3 ([29], Corollary 4.3). *Let T be a commutative monad on a symmetric monoidal category \mathcal{C} . Then the Kleisli category $\text{Kl}(T)$ has a canonical monoidal structure, which is given on objects by the monoidal product in \mathcal{C} , and on morphisms $f : X \rightarrow TA, g : Y \rightarrow TB$ 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 ∇ sends a pair (or more generally a tuple) of subsets to a subset of tuples by taking the cartesian product.

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 $n \rightarrow m$ are functions $Q^n \rightarrow \mathcal{P}(Q^m)$,
- composition is Kleisli composition, i.e. the usual composition of relations $f \circ g := \mu \circ \mathcal{P}(g) \circ f$, where μ is the canonical map from sets of subsets to subsets,
- monoidal product of objects given by concatenation, and
- monoidal product of morphisms $f : n \rightarrow m$ and $g : p \rightarrow q$ by $\nabla \circ (f \times g) : n + p \rightarrow m + q$, where ∇ is the monoidal multiplication of \mathcal{P} , i.e. the canonical map from pairs of subsets to their cartesian product.

Remark 6.5. 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. We return to this in Section 10. Stochastic monoidal automata could be obtained by use of the distribution monad [26], 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 $\Gamma \rightarrow \mathcal{U}(\text{Rel}_Q)$. By the adjunction $\mathcal{F} \dashv \mathcal{U}$, such morphisms are in bijection with pro morphisms $\Delta : \mathcal{F}\Gamma \rightarrow \text{Rel}_Q$. We will also refer to the inductive extension Δ as a non-deterministic monoidal automaton, and sometimes write Δ_α for the relation $\Delta(\alpha : n \rightarrow m)$.

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

Definition 6.7. Let $\Delta : \mathcal{F}\Gamma \rightarrow \text{Rel}_Q$ 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 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 $\Psi : \mathcal{M} \rightarrow \Gamma$, define a monoidal automaton with $Q = V_{\mathcal{M}}$, $w(\Delta_\gamma)w' \iff \exists \sigma \in E_\Psi^{-1}(\gamma)$ such that $s(\sigma) = w, t(\sigma) = w'$. Conversely given a monoidal automaton (Q, Δ_Γ) , define a regular monoidal grammar with $V_{\mathcal{M}} = Q$ and take an edge $w \rightarrow w'$ 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 $\Delta_\sigma := \Delta(\sigma, -) : Q \rightarrow \mathcal{P}(Q)$. For \dashv take the transition function $Q \rightarrow \mathcal{P}(Q^0)$ to be the characteristic function of $F \subseteq Q$, sending elements of F to $\{\bullet\}$ and to \emptyset otherwise, and for \vdash take the function $Q^0 \rightarrow \mathcal{P}(Q)$ 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 $\{\bullet\} \rightarrow \mathcal{P}(Q)$. We can always “normalize” such an automaton into an equivalent NFA with one initial state (see [33, §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

Classical non-deterministic finite-state 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

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 $\Psi : \mathcal{M} \rightarrow \Gamma, \Psi' : \mathcal{M}' \rightarrow \Gamma$ respectively. Define the grammar $\Psi + \Psi' : \mathcal{M} + \mathcal{M}' \rightarrow \Gamma$, where $E_{\mathcal{M} + \mathcal{M}'} := E_{\mathcal{M}} + E_{\mathcal{M}'}, V_{\mathcal{M} + \mathcal{M}'} := V_{\mathcal{M}} + V_{\mathcal{M}'}$, and $E_{\Psi + \Psi'} := [\Psi, \Psi']$ (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. \square

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 $\mathcal{L}(\Delta \times \Delta') = L \cap L'$. \square

Remark 8.3. *The Sierpiński triangle language (Example 4.8) is the intersection of the brick wall language (Example 4.7) and an “XOR gate” language: this explains the origin of the states in the grammar shown in Example 4.8.*

Lemma 8.4 (Closure under monoidal product and factors). *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 $(\mathcal{F}h)L$ is a regular monoidal language over Γ' .*

Proof. Let L be given by the regular monoidal grammar $\Psi : \mathcal{M} \rightarrow \Gamma$, that is $L = \mathcal{F}\Psi[\mathcal{F}\mathcal{M}(\varepsilon, \varepsilon)]$. Consider the grammar given by the composite $h \circ \Psi : \mathcal{M} \rightarrow \Gamma'$. Since \mathcal{F} is a functor we have: $\mathcal{F}(h \circ \Psi)[\mathcal{F}\mathcal{M}(\varepsilon, \varepsilon)] = (\mathcal{F}h \circ \mathcal{F}\Psi)[\mathcal{F}\mathcal{M}(\varepsilon, \varepsilon)] = (\mathcal{F}h)L$, thus $h \circ \Psi$ is a grammar for $(\mathcal{F}h)L$. \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 , $(\mathcal{F}h)^{-1}(L)$ is a regular monoidal language over Γ' .*

Proof. Let $(Q, \{\Delta_\gamma\}_{\gamma \in \Gamma})$ be an automaton recognizing L with inductive extension $\Delta : \mathcal{F}\Gamma \rightarrow \text{Rel}_Q$. Consider the automaton given by the composite $\Delta \circ \mathcal{F}h : \mathcal{F}\Gamma' \rightarrow \text{Rel}_Q$. We have $\mathcal{L}(\Delta \circ \mathcal{F}h) = (\mathcal{F}h)^{-1}(\mathcal{L}(\Delta)) = (\mathcal{F}h)^{-1}(L)$, 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) , where $n, m \geq 0$, is a scalar string diagram with a hole (Figure 13) with zero or more additional strings exiting the first box and entering the second (indicated by ellipses in Figure 13).*

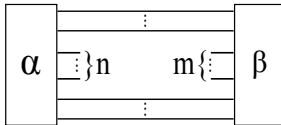


Figure 13: A context of capacity (n, m) . α and β stand for arbitrary string diagrams with arity and coarity 0, respectively.

Contexts are not themselves string diagrams, but they can be given a precise semantics as morphisms in the produoidal category of contexts of over the pro

$\mathcal{F}\Gamma$ [14]. It will suffice here to consider them according to the more informal definition above.

Given a context of capacity (n, m) , we can fill the hole with a string diagram $\gamma : n \rightarrow m$. Write $C[\gamma]$ for the resulting string diagram. 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 $\mathcal{F}\Gamma(n, m)$. Then $\gamma \equiv_L \delta$ whenever $C[\gamma] \in L \iff C[\delta] \in L$, for all contexts C of capacity (n, m) .*

Definition 9.3. *The syntactic pro of a monoidal language L is the quotient pro $\mathcal{F}\Gamma/\equiv_L$. The quotient functor $S_L : \mathcal{F}\Gamma \rightarrow \mathcal{F}\Gamma/\equiv_L$ 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 $\alpha \equiv_L \square$, since the empty diagram is in every language and if C is a context of capacity $(0, 0)$ distinguishing α and \square , then we have a contradiction by Lemma 8.4. So $\alpha \in S_L^{-1}(\square)$, and conversely. \square

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 $\mathcal{F}\Gamma/\equiv_L$ 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 $\Delta : \mathcal{F}\Gamma \rightarrow \text{Rel}_Q$. 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 $\mathcal{F}\Gamma/\sim \rightarrow \mathcal{F}\Gamma/\equiv_L$ 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. \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 [20]. Therefore we cannot expect to determinize non-deterministic monoidal automata. However, we have already seen monoidal languages that are deterministically recognizable: Examples 4.7 and 4.8, 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 $Q + \{\perp\}$.*

Recall the definition of the category Par_Q from Remark 6.5. Then as in Observation 6.6, such assignments $\gamma \mapsto \delta_\gamma$ uniquely extend to morphisms of pros $\delta : \mathcal{F}\Gamma \rightarrow \text{Par}_Q$, and we will also refer to such functors as deterministic monoidal automata. δ maps scalar string diagrams to one of the two functions $Q^0 \rightarrow Q_\perp^0$, and we use this to define the language of the automaton:

Definition 10.2. *Let $\delta : \mathcal{F}\Gamma \rightarrow \text{Par}_Q$ 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 [20]. 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.

Definition 10.3 ([9]). *A cartesian restriction prop is a prop in which every object is equipped with a commutative comonoid structure (with the counit depicted by \dashv , comultiplication by \dashv , and symmetry by \times) that is coherent, and for which the comultiplication is natural (see Appendix B for details).*

Definition 10.4. *The free cartesian restriction prop on a monoidal graph \mathcal{M} , denoted $\mathcal{F}_\perp \mathcal{M}$ 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 \dashv on X given by the function $X \rightarrow \{\bullet, \perp\}$ sending every element to \bullet , and \dashv 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, $\bar{\delta} : \mathcal{F}_\downarrow\Gamma \rightarrow \text{Par}_Q$, and these have an 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 $\mathcal{F}\Gamma$ and $\mathcal{F}_\downarrow\Gamma$:

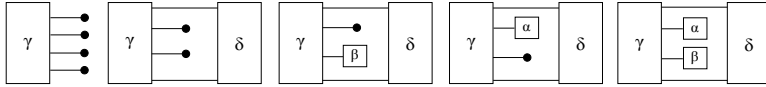
Lemma 10.6. *If (Q, δ_Γ) is a deterministic monoidal automaton, then δ factors through $\bar{\delta}$ as $\delta = \bar{\delta} \circ \mathcal{H}_\Gamma$, where $\mathcal{H}_\Gamma : \mathcal{F}\Gamma \rightarrow \mathcal{F}_\downarrow\Gamma$ sends morphisms to their equivalence class in $\mathcal{F}_\downarrow\Gamma$.*

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

Recall that any restriction category is poset-enriched: $f \leq g$ 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 $f \leq g \iff f \otimes g = f$. Now we can define causal histories:

Definition 10.7. *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 $\mathcal{H}_\Gamma(\gamma) \leq h$ 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 $\mathcal{H}_\Gamma(L)^\dagger$, 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 4.9:



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

Proof. Since $\delta = \bar{\delta} \circ \mathcal{H}_\Gamma$, if δ accepts γ , then $\bar{\delta}$ accepts $\mathcal{H}_\Gamma(\gamma)$. Let h be a causal history of γ . Then $\bar{\delta}(\mathcal{H}_\Gamma(\gamma)) = \bar{\delta}(h \otimes \mathcal{H}_\Gamma(\gamma)) = \bar{\delta}(h) \otimes \bar{\delta}(\mathcal{H}_\Gamma(\gamma))$. But then $\bar{\delta}$ 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 $\otimes ch(L)$ denote the closure of the set of causal histories of L under monoidal product. Then the causal closure $cl(L)$ of L is $\mathcal{H}_\Gamma^{-1}(\otimes ch(L))$. A monoidal language is causally closed if it is equal to its causal closure.*

To illustrate causal closure, consider Figure 14, which shows part of the derivation of a morphism in the causal closure of the language of Example 4.9.

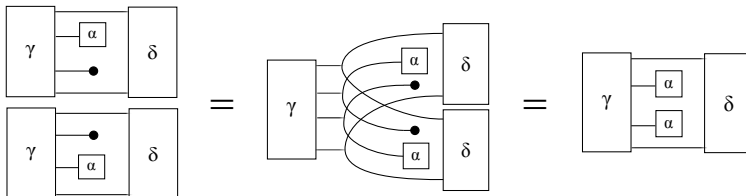


Figure 14: These string diagrams are equal in the equational theory of cartesian restriction categories. On the left, we have the monoidal product of two causal histories of elements of the language from Example 4.9. This determines a string diagram in the image of \mathcal{H}_Γ (i.e. expressible without using the cartesian restriction structure), which is an element of the causal closure of the language.

The leftmost diagram in Figure 14 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 $\dashv\lrcorner$ (for γ), then unitality (twice), then naturality of $\dashv\lrcorner$ (for δ). The rightmost form of the diagram exhibits this morphism as being in the image of \mathcal{H}_Γ , and its preimage under \mathcal{H}_Γ is the same diagram in $\mathcal{F}\Gamma$. 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 $\delta : \mathcal{F}\Gamma \rightarrow \text{Par}_Q$. We have $\delta = \bar{\delta} \circ \mathcal{H}_\Gamma$ and from Lemma 10.8 that $\bar{\delta}$ 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 $\mathcal{F}\Gamma/\equiv_L$ has a cartesian restriction prop structure. We exhibit a pro morphism $\mathcal{F}\Gamma/\equiv_L \xrightarrow{\phi} \text{Par}_Q$ such that $\mathcal{F}\Gamma \xrightarrow{S_L} \mathcal{F}\Gamma/\equiv_L \xrightarrow{\phi} \text{Par}_Q$ is a deterministic monoidal automaton accepting exactly L .

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

$$\left(\begin{array}{c} \boxed{\alpha_1} \\ \vdots \\ \boxed{\alpha_n} \end{array} \right) \xrightarrow{\phi([\beta])} \left(\begin{array}{c} \boxed{\alpha_1} \\ \vdots \\ \boxed{\alpha_n} \end{array} \text{---} \boxed{\beta} \begin{array}{c} \text{---} \bullet \\ \text{---} \bullet \\ \text{---} \bullet \\ \vdots \\ \text{---} \bullet \end{array} \right), \dots, \left(\begin{array}{c} \boxed{\alpha_1} \\ \vdots \\ \boxed{\alpha_n} \end{array} \text{---} \boxed{\beta} \begin{array}{c} \text{---} \bullet \\ \text{---} \bullet \\ \text{---} \bullet \\ \vdots \\ \text{---} \bullet \end{array} \right)$$

When $m = 0$, let $\phi([\beta])([\alpha_1], \dots, [\alpha_n]) = \bullet$, if $[\beta \circ (\alpha_1 \otimes \dots \otimes \alpha_n)] = \boxed{}$, and $\phi([\beta])([\alpha_1], \dots, [\alpha_n]) = \perp$ otherwise. The proof that this defines a morphism of pros is an exercise in diagrammatic reasoning using the equational theory of cartesian restriction categories and can be found in Appendix C. To see that this automaton accepts exactly L , let $\alpha \in \mathcal{L}(\phi \circ S_L)$, then by definition we must have $S_L(\alpha) = \boxed{}$, and so $\alpha \in L$ (by Lemma 9.5). Conversely let $\alpha \in L$, then $S_L(\alpha) = \boxed{}$ and by definition $\phi(\boxed{})(\bullet) = \bullet$, so $\alpha \in \mathcal{L}(\phi \circ S_L)$. 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 $\Gamma = \{ \bullet\text{---}, \text{---}\bullet \}$, having one connected component: $\bullet\text{---}\bullet$. The syntactic pro of this language has a cartesian restriction prop structure, with the counit given by the equivalence class $[-\bullet]$, comultiplication by $[\text{---}\bullet]$, and symmetry by $[\text{---}\bullet\text{---}\bullet]$. It is clear that $\mathcal{F}\Gamma/\equiv_L(0, 1)$ 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 [20, §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 4.6) and Sierpiński triangles (Example 4.8), 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 $\text{Rel} \rightarrow \text{Set}$,

sending sets to their powersets and relations to their corresponding functions, right adjoint to the inclusion $\text{Set} \hookrightarrow \text{Rel}$. 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 $\Delta : Q^n \rightarrow \mathcal{P}(Q^m)$ is convex if there is a morphism Δ^* such that the following square commutes:*

$$\begin{array}{ccc} (\mathcal{P}Q)^n & \xrightarrow{\Delta^*} & (\mathcal{P}Q)^m \\ \nabla_{\mathcal{P}} \downarrow & & \downarrow \nabla_{\mathcal{P}} \\ \mathcal{P}(Q^n) & \xrightarrow{\Delta^\#} & \mathcal{P}(Q^m) \end{array}$$

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

Example 11.2. *The relation $\Delta_\gamma : Q^0 \rightarrow \mathcal{P}(Q^4)$ induced by the grammar in Example 4.9 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 $\text{CRel}_Q \hookrightarrow \text{Rel}_Q$.*

Proof. See Appendix C. □

Definition 11.4. *An automaton $\Delta : \mathcal{F}\Gamma \rightarrow \text{Rel}_Q$ 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 (\emptyset in Rel_Q , but \perp in $\text{Par}_{\mathcal{P}^+(Q)}$):

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

$$\begin{array}{ccc} \Delta_\alpha : Q^n \rightarrow \mathcal{P}(Q^m) & & \\ \downarrow & & \\ \mathcal{P}^+(Q)^n \xrightarrow{\eta^n} (\perp \mathcal{P}^+(Q))^n \xrightarrow{\Delta_\alpha^*} (\perp \mathcal{P}^+(Q))^m \xrightarrow{\nabla_\perp} \perp(\mathcal{P}^+(Q)^m) \end{array}$$

where we use elide the isomorphisms $\mathcal{P}(Q)^n \cong (\perp \mathcal{P}^+(Q))^n$. \perp is the maybe monad, η is the unit of this monad, and ∇_\perp is its monoidal multiplication with respect to the cartesian product, sending a tuple \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 $\Delta : \mathcal{F}\Gamma \rightarrow \mathbf{CRel}_Q$ is now just given by post-composition with the functor $\mathcal{D}_Q : \mathbf{CRel}_Q \rightarrow \mathbf{Par}_{\mathcal{D}^+}(Q)$. We show that this preserves the language:

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

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

Example 11.7. *Non-deterministic monoidal automata over bottom-up tree monoidal alphabets (Definition 7.3) are convex, with $\Delta^* := \Delta^\# \circ \nabla_{\mathcal{D}}$. 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 $\nabla_{\mathcal{D}}$, 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 [27] 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 [35]; lifting this to the setting of monoidal categories is an 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 [16], 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 [30] in which an ordering on generators is defined. 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 be 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 [14]. Following Melliès and Zeilberger [25], 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.

References

- [1] Jean Berstel and C Reutenauer. *Rational series and their languages*. Eatsc 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/CB09780511977619.
- [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] M. Earnshaw, M. Román, and J. Hefford. The produoidal algebra of process decomposition. *Preprint*, 2023.
- [15] 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.
- [16] Matthew Earnshaw and Paweł Sobociński. String diagrammatic trace theory. Submitted, 2023.
- [17] 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.
- [18] 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.

- [19] 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.
- [20] Ferenc Gécseg and Magnus Steinby. Tree automata, 2015. doi:10.48550/ARXIV.1509.06233.
- [21] T. Heindel. A Myhill-Nerode theorem beyond trees and forests via finite syntactic categories internal to monoids. *Preprint*, 2017.
- [22] Alan Jeffrey. Premonoidal categories and flow graphs. *Electronic Notes in Theoretical Computer Science*, 10:51, 1998. HOOTS II, Second Workshop on Higher-Order Operational Techniques in Semantics. doi:10.1016/S1571-0661(05)80688-7.
- [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] Jade Master. Petri nets based on lawvere theories. *Mathematical Structures in Computer Science*, 30(7):833–864, 2020. doi:10.1017/S0960129520000262.
- [25] 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>.
- [26] Paolo Perrone. Distribution monad (nlab entry), 2019. <https://ncatlab.org/nlab/show/distribution+monad>, Last accessed 2023-03-13.
- [27] 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.
- [28] Jean-Éric Pin and Dominique Perrin. *Infinite Words - Automata, Semigroups, Logic and Games*. Pure and Applied Mathematics (Amsterdam). Elsevier, 2004.
- [29] John Power and Edmund Robinson. Premonoidal categories and notions of computation. *Mathematical Structures in Computer Science*, 7(5), 1997. doi:10.1017/S0960129597002375.
- [30] 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.

- [31] Kimmo I. Rosenthal. Quantaloids, enriched categories and automata theory. *Applied Categorical Structures*, 3(3):279–301, 1995. doi:10.1007/bf00878445.
- [32] Paul W. K Rothmund, Nick Papadakis, and Erik Winfree. Algorithmic self-assembly of DNA Sierpinski triangles. *PLOS Biology*, 2(12), 12 2004. doi:10.1371/journal.pbio.0020424.
- [33] Jacques Sakarovitch. *Elements of automata theory*. Cambridge University Press, Cambridge New York, 2009.
- [34] 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.
- [35] Howard Straubing. *Finite automata, formal logic, and circuit complexity*. Progress in Theoretical Computer Science. Birkhauser Verlag AG, Basel, Switzerland, December 1993.
- [36] 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.
- [37] 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.
- [38] 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.
- [39] 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 \mathcal{C} is an equivalence relation $f \sim g$ on pairs of parallel morphisms $f, g : x \rightarrow x'$ 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 \mathcal{C} , we can define the quotient monoidal category \mathcal{C}/\sim as the category with objects those of \mathcal{C} , and homsets $(\mathcal{C}/\sim)(x, x') := \mathcal{C}(x, x')/\sim$, with composition and monoidal product defined in the obvious way. The quotient functor $\mathcal{C} \rightarrow \mathcal{C}/\sim$ is monoidal, full and bijective on objects (and strict when \mathcal{C} is strict). When \mathcal{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 \bullet for the counit of the comonoid on an arbitrary object, \lrcorner for the comultiplication of the comonoid on an arbitrary object, \smile 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):

$$\begin{array}{c} \lrcorner \\ \bullet \end{array} = \begin{array}{c} \lrcorner \\ \bullet \end{array} \quad \lrcorner = \lrcorner \smile \quad \bullet = \bullet \begin{array}{c} \bullet \\ \lrcorner \end{array} \quad (\text{CCM})$$

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:

$$x \otimes y \begin{array}{c} \lrcorner \\ \bullet \end{array} \begin{array}{c} x \otimes y \\ x \otimes y \end{array} = \begin{array}{c} y \\ \lrcorner \\ x \end{array} \begin{array}{c} y \\ x \\ y \\ x \end{array} \quad x \otimes y \bullet = \begin{array}{c} y \\ x \end{array} \bullet \quad (\text{coherent})$$

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

$$x \begin{array}{c} \lrcorner \\ \lrcorner \end{array} \begin{array}{c} y \\ y \end{array} = x \begin{array}{c} \lrcorner \\ \lrcorner \end{array} \begin{array}{c} y \\ y \end{array} \quad (\text{natural})$$

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 $\Delta_\alpha : Q^a \rightarrow \mathcal{P}(Q^b)$, $\Delta_\beta : Q^b \rightarrow \mathcal{P}(Q^c)$, and take $(\Delta_\beta \diamond \Delta_\alpha)^* = \Delta_\beta^* \circ \Delta_\alpha^*$, where \diamond is composition in $\text{Kl}(\mathcal{P})$. Consider the following diagram:

$$\begin{array}{ccccc}
(\mathcal{P}Q)^a & \xrightarrow{\Delta_\alpha^*} & (\mathcal{P}Q)^b & \xrightarrow{\Delta_\beta^*} & (\mathcal{P}Q)^c \\
\downarrow \nabla & & \downarrow \nabla & & \downarrow \nabla \\
\mathcal{P}(Q^a) & \xrightarrow{\Delta_\alpha^\#} & \mathcal{P}(Q^b) & \xrightarrow{\Delta_\beta^\#} & \mathcal{P}(Q^c) \\
& \searrow \mathcal{P}(\Delta_\alpha) & \nearrow \mu & \searrow \mathcal{P}(\Delta_\beta) & \nearrow \mu \\
& & \mathcal{P}^2(Q^b) & & \mathcal{P}^2(Q^c) \\
& & \searrow \mathcal{P}^2(\Delta_\beta) & \nearrow \mathcal{P}(\mu) & \\
& & & & \mathcal{P}^3(Q^c)
\end{array}$$

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:

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

by naturality of μ ,

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

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

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

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

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

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

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

$$\begin{array}{ccc}
 (\perp \mathcal{P}^+(Q))^n & & \\
 \nabla_{\perp} \downarrow & \searrow \nabla_{\mathcal{P}} & \\
 \perp(\mathcal{P}^+(Q))^n & \xrightarrow{\perp \nabla_{\mathcal{P}^+}} & (\perp \mathcal{P}^+)(Q^n)
 \end{array}$$

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

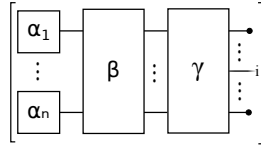
Proof of Lemma 11.5. We first show that the action on morphisms is well-defined. It suffices to show that if Δ_{α}^* , $\hat{\Delta}_{\alpha}^*$ are distinct witnesses to the convexity of Δ_{α} , then $\nabla_{\perp} \circ \Delta_{\alpha}^* = \nabla_{\perp} \circ \hat{\Delta}_{\alpha}^*$. From Lemma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

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 $\Delta_{\alpha} : Q^a \rightarrow \mathcal{P}(Q^b)$, $\Delta_{\beta} : Q^b \rightarrow \mathcal{P}(Q^c)$. We require $\mathcal{D}_Q(\Delta_{\beta} \diamond \Delta_{\alpha}) = \mathcal{D}_Q(\Delta_{\beta}) \circ \mathcal{D}_Q(\Delta_{\alpha})$. 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**.

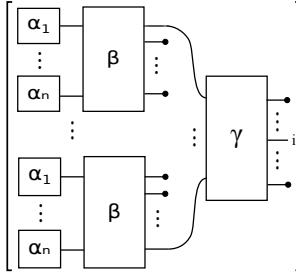
$$\begin{array}{ccc}
 (\perp \mathcal{P}^+(Q))^b & \xrightarrow{\eta^b} & (\perp \perp \mathcal{P}^+(Q))^b \\
 \nabla_{\perp} \downarrow & \searrow \eta & \downarrow \nabla_{\perp} \\
 \perp \mathcal{P}^+(Q)^b & \xrightarrow{\perp \eta^b} & \perp(\perp \mathcal{P}^+(Q))^b
 \end{array}$$

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

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 $\phi([\gamma] \circ [\beta])([\alpha_1], \dots, [\alpha_n])$ is the equivalence class:

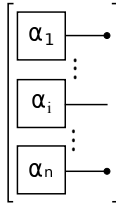


where the i^{th} output of γ is dangling on the right. The i^{th} component of $(\phi([\gamma]) \circ \phi([\beta]))([\alpha_1], \dots, [\alpha_n])$ is the equivalence class:

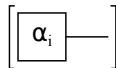


But since $\mathcal{F}\Gamma/\equiv_L$ is a cartesian restriction prop, the representatives of these equivalence classes are the same diagram (by repeated applications of the naturality of $\dashv\lrcorner$ and unitality). Hence this is the same equivalence class, and ϕ preserves composition.

For identities, the i^{th} component of $\phi([\text{id}_n])([\alpha_1], \dots, [\alpha_n])$ 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

Appendix **E**

Paper IV

Matthew Earnshaw and Mario Román. “Context-Free Languages of String Diagrams”. In: *28th International Conference on Foundations of Software Science and Computation Structures*. To appear. 2025

Context-Free Languages of String Diagrams

Matt Earnshaw¹ and Mario Román^{1,2}

¹ Department of Software Science, Tallinn University of Technology

² Department of Computer Science, University of Oxford

Abstract. We introduce context-free languages of morphisms in monoidal categories, extending recent work on the categorification of context-free languages, and regular languages of string diagrams. Context-free languages of string diagrams include classical context-free languages of words, trees, and hypergraphs, when instantiated over appropriate monoidal categories. We prove a representation theorem for context-free languages of string diagrams: every such language arises as the image under a monoidal functor of a regular language of string diagrams.

1 Introduction

Monoids are the classical algebraic home of formal languages, and a long line of research beginning in the 60s has sought to extend the tools and concepts of language theory to other algebraic structures, such as trees [23,44], traces [14], hypergraphs [12,25,45], models of algebraic theories [19,49], algebras for monads [3,4], and categories [2,50].

Categories are “monoids with many objects”, and passing from the theory of context-free languages in monoids to the theory of context-free languages in categories has been the subject of recent work by Melliès and Zeilberger [34,35]. This novel structural point of view suggests a natural generalization to categories with additional structure. Here, we pursue this idea for *monoidal categories*. On the one hand, strict monoidal categories are *two-dimensional monoids*, and so a natural step from their one-dimensional counterpart. On the other hand, they have a natural graphical syntax, *string diagrams*, providing a fresh approach to languages of graphs.

A vast literature has explored language theory in various algebras of graphs, culminating in the celebrated results of Courcelle [12]. Our point of departure is the claim that many graphical notions can be naturally viewed as *morphisms in monoidal categories*; that is, monoidal categories provide a suitable algebraic framework for graphical formal languages. This manuscript pursues this idea in the context of recent work in the foundations of language theory which takes a structural approach to *context-freeness*. Ultimately, this line of work seeks to unify the various generalizations of context-free languages, and identify reusable tools for reasoning about them.

1.1 Languages of string diagrams

Monoidal categories have an intuitive, sound and complete graphical syntax: *string diagrams*. String diagrams resemble graphical languages commonly found in engineering and science, and indeed, they allow us to reason about Markov kernels [22], linear algebra [6], or quantum processes [1]. In computer science, they provide foundations for visual programming [28,30].

The use of string diagrams as a syntax in these various domains suggests the need for a corresponding theory of string diagrams as a formal language. This is one aim of recent work on languages of string diagrams or *monoidal languages*, such as that elaborated by Sobociński and the first author [16,17], who introduced the class of *regular monoidal languages*. A monoidal language in this sense is simply a subset of morphisms in a strict monoidal category, just as a classical formal language is a subset of a monoid. In this work, we introduce a natural class of *context-free* monoidal languages, which capture various extended notions of context-free language found in the computer science literature.

1.2 Context-free languages over categories

Our main point of reference in this paper is the recent work of Melliès and Zeilberger [34,35]. This work is a thoroughgoing refashioning of the theory of context-free languages from a “fibrational” point of view. Melliès and Zeilberger demonstrate that it is natural and fruitful to consider context-free languages over arbitrary *categories*. They introduce an adjunction between *splicing* (introducing gaps or contexts in terms) and *contouring* (linearizing derivation trees), and use it to give a novel conceptual proof of the Chomsky-Schützenberger representation theorem: every context-free language is the image of the intersection of a regular language and the language of balanced parentheses [9].

Melliès and Zeilberger provide an ample supply of examples of context-free languages in categories, such as context-free languages of runs over an automaton, languages with an explicit end-of-input marker, multiple context-free grammars [46] and a grammar of series-parallel graphs. However, it is less clear how notions such as context-free grammars of trees and hypergraphs fit into this framework. In this paper, we show how this can be accomplished by adapting the machinery of Melliès and Zeilberger to the wider setting of monoidal categories and their string diagrams. This generalization is non-trivial, and sheds light on the intriguing differences between languages of string diagrams and classical languages. In particular, our two-dimensional version of the Chomsky-Schützenberger representation theorem says that every context-free language of string diagrams is the image under a monoidal functor of a regular language of string diagrams: no intersection of context-free and regular languages is necessary.

Related work Bruggink and König have investigated recognizable languages of morphisms in a category using a notion of automaton functor [8], which is similar to our notion of non-deterministic monoidal automaton. Similar ideas have also been investigated by Colcombet and Petrisan [11]. Griffing has also introduced

a notion of recognizable set of morphisms in a category [24]. These works deal with languages over categories, rather than monoidal categories.

The representation of context-free grammars as certain morphisms of multi-graphs was introduced by Walters in a short paper [51]. A similar type-theoretical version of this idea was also introduced by De Groote [13]. As discussed more extensively above, this idea was taken up and substantially refined by Melliès and Zeilberger, first in a conference paper [34] and later in an extended version [35].

A different notion of context-free families of string diagrams has been introduced by Zamdzhiev [52]. There, string diagrams are defined combinatorially as *string graphs*, and context-free families are then generated by B-edNCE graph grammars [45]. Though similar, the resulting notion is not directly comparable to ours. Here, we use the native algebra of monoidal categories and their multicategories of contexts to define and investigate languages.

Finally, Heindel’s abstract [26] claims a proof of a Chomsky-Schützenberger theorem for morphisms in symmetric monoidal categories, but the work described in this abstract was never published. Our development is quite different from that outlined in Heindel’s abstract. We prove a stronger representation theorem that does not require an intersection of languages; we work without the assumption of symmetry; and we generalize the categorical machinery of Melliès and Zeilberger.

Contributions We introduce context-free languages of string diagrams (Definition 4.7) and show that they include a wide variety of examples in the computer science literature including context-free languages of trees and hypergraphs. We introduce the category of raw optics (Definition 5.1) over a monoidal category, and its left adjoint, the optical contour (Definition 5.2, Theorem 5.1). We use this machinery to prove a representation theorem for context-free monoidal languages (Theorem 6.1), relating them to previous work on regular monoidal languages.

2 Preliminaries

In this paper, we define context-free grammars as particular *morphisms of multi-graphs*. This point of view, while perhaps unfamiliar, is simple and powerful. It suggests natural generalizations of context-free grammars, such as we will pursue in the main body of the paper, and new conceptual tools for reasoning about them. This idea is not original to us; its roots go back to Walters [51], with recent refinement and extension by Melliès and Zeilberger [34,35].

2.1 Context-free languages in free monoids and other categories

We introduce the definition of context-free grammars as morphisms of certain *multigraphs*. Multigraphs (or *species* in the work of Melliès and Zeilberger [35]) are a kind of graph in which edges have a *list* of sources and single target. It is often helpful to think of a multigraph as a *signature*, specifying a set of typed

operations. Note that this is a different use of the term *multigraph* from that specifying graphs allowing multiple parallel edges.

Definition 2.1. A multigraph M is a set S of sorts, and sets $M(X_1, \dots, X_n; Y)$ of generating operations (or multimorphisms), for each pair of a list of sorts X_1, \dots, X_n and a sort Y . A multigraph is finite if sorts and operations are finite sets. A morphism of multigraphs is given by a function f on sorts and functions $M(X_1, \dots, X_n; Y) \rightarrow M(fX_1, \dots, fX_n; fY)$ between sets of operations.

Multigraphs freely generate multicategories, also known as *operads* (though this term sometimes refers only to the single-sorted, symmetric case). See Leinster [33] for a comprehensive reference on multicategories. The free multicategory $\mathcal{F}_{\nabla}M$ over a multigraph M has as multimorphisms $\mathcal{F}_{\nabla}M(X_1, \dots, X_n; Y)$ the “trees” rooted at Y , with open leaves X_1, \dots, X_n , that one can build by “plugging together” operations in M . We call closed trees, i.e. nullary multimorphisms $d \in \mathcal{F}_{\nabla}M(; Y)$, *derivations*. Every multicategory \mathbb{M} has an underlying multigraph, denoted $|\mathbb{M}|$, given by forgetting identities and composition.

Every rule in a context-free grammar is of the form $R \rightarrow w_1 R_1 \dots R_{n-1} w_n$, where R, R_i are non-terminals, and w_i are (possibly empty) words over an alphabet Σ . The insight of Melliès and Zeilberger [35] is that this data may be arranged as an operation $R_1, \dots, R_n \rightarrow R$ in a multigraph over an n -ary operation $w_1 - \dots - w_n$ called a *spliced word*: a word with n gaps, as in Figure 1. We introduce the multicategory of *spliced arrows in a category*.

Definition 2.2 (Melliès and Zeilberger [35]). The multicategory of spliced arrows, \mathcal{WC} , over a category \mathbb{C} , contains, as objects, pairs of objects of \mathbb{C} , denoted as $\frac{A}{B}$. Its multimorphisms are morphisms of the original category, but with n “gaps” or “holes”, into which other morphisms (with holes) may be spliced. More precisely, the multimorphisms of \mathcal{WC} are given by:

$$\mathcal{WC}\left(\frac{A_1}{B_1}, \dots, \frac{A_n}{B_n}, \frac{X}{Y}\right) := \mathbb{C}(X; A_1) \times \prod_{i=1}^{n-1} \mathbb{C}(B_i; A_{i+1}) \times \mathbb{C}(B_n; Y).$$

By convention, nullary multimorphisms are morphisms of \mathbb{C} , that is $\mathcal{WC}(\frac{X}{Y}) := \mathbb{C}(X; Y)$. The identity is given by a pair of identities of the original category, multicategorical composition is derived from the composition of the original category.

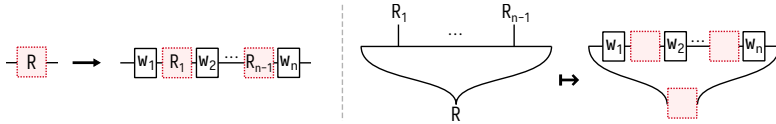


Fig. 1: (Left) Generic form of a context-free rule. (Right) Context-free rules as a morphism of multigraphs into spliced arrows; here, spliced arrows in a monoid.

We can now present a context-free grammar in terms of a morphism of multigraphs from a multigraph of non-terminals to the underlying multigraph of spliced arrows, as in Figure 1.

Definition 2.3 (Melliès and Zeilberger [35]). *A context-free grammar of morphisms in a category \mathbb{C} is a morphism of multigraphs $G \rightarrow |\mathscr{W}\mathbb{C}|$ and a sort S in G (the start symbol).*

By the free-forgetful adjunction between multicategories and multigraphs, morphisms $\phi : G \rightarrow |\mathscr{W}\mathbb{C}|$ and morphisms of multicategories (or multifunctors) $\hat{\phi} : \mathcal{F}_{\nabla}G \rightarrow \mathscr{W}\mathbb{C}$ are in bijection. This allows for a slick definition of the language of a grammar.

Definition 2.4 (Melliès and Zeilberger [35]). *Let $\mathcal{G} = (\phi : G \rightarrow |\mathscr{W}\mathbb{C}|, S)$ be a context-free grammar of morphisms in \mathbb{C} . The language of \mathcal{G} is given by the image of the set of derivations $\mathcal{F}_{\nabla}G(; S)$ under the multifunctor $\hat{\phi}$.*

When \mathbb{C} is a finitely generated free monoid considered as a one-object category, then context-free grammars over \mathbb{C} correspond precisely to the classical context-free grammars.

An important realization of Melliès and Zeilberger is that the operation of forming the multicategory of spliced arrows in \mathbb{C} has a left adjoint. That is, every multicategory gives rise to a category called the *contour* of \mathbb{M} , and this contouring operation is left adjoint to splicing. We refer to their paper for more details [35, Section 3.2]. Contours give a conceptual replacement for Dyck languages in the classical theory of context-free languages: they linearize derivation trees.

In Section 5, we define a new contour of multicategories which we call the *optical contour*; we shall use it to prove a representation theorem for languages of string diagrams (Theorem 6.1), inspired by generalized Chomsky-Schützenberger representation theorem proved by Melliès and Zeilberger.

2.2 Monoidal categories, their string diagrams and languages

In this paper, we will mostly be concerned with monoidal categories presented by generators and equations between the string diagrams built from these generators. Generators are given by *polygraphs*.

Definition 2.5. *A polygraph Γ is a set S_{Γ} of sorts, and sets $\Gamma(X_1 \otimes \dots \otimes X_n; Y_1 \otimes \dots \otimes Y_m)$ of generators for every pair of lists X_i, Y_j of sorts. A polygraph is finite if sorts and generators are finite sets. A morphism of polygraphs is a function f on sorts and functions $\Gamma(X_1 \otimes \dots \otimes X_n; Y_1 \otimes \dots \otimes Y_m) \rightarrow \Gamma(fX_1 \otimes \dots \otimes fX_n; fY_1 \otimes \dots \otimes fY_m)$ between generators.*

For a generator γ of arity $X_1 \otimes \dots \otimes X_n$ and coarity $Y_1 \otimes \dots \otimes Y_m$ we write $\gamma : X_1 \otimes \dots \otimes X_n \rightarrow Y_1 \otimes \dots \otimes Y_m$. When S is single-sorted, we use natural numbers for the arities and coarities; this case will cover most of the examples in the following. We depict generators as boxes with strings on the left and right for their arities and coarities (Figure 2).



Fig. 2: The free strict monoidal category over a polygraph Γ has set of objects S_Γ^* and morphisms string diagrams given inductively over the generators of Γ as above quotiented by the equivalence relation generated by planar isotopy of diagrams, keeping left and right boundaries fixed. The leftmost rule denotes the empty diagram. We use colours to indicate sorts.

Proposition 2.1. *String diagrams with generators in a polygraph construct a monoidal category (Figure 2). The monoidal category of string diagrams over a polygraph is the free strict monoidal category over the polygraph [29,47]. Every monoidal category is equivalent to a strict one. In particular, string diagrams are sound and complete for monoidal categories.*

We shall need to impose equations between string diagrams, such as in defining symmetric monoidal categories, cartesian monoidal categories and hypergraph categories. To this end, we introduce the following notion of presentation.

Definition 2.6. *A finite presentation of a strict monoidal category consists of a finite polygraph of generators, \mathcal{P} , and a finite polygraph of equations, \mathcal{E} , with projections for the two sides of each equation, $l, r: \mathcal{E} \rightarrow |\mathcal{F}_\otimes \mathcal{P}|$. The strict monoidal category presented by $(\mathcal{P}, \mathcal{E}, l, r)$, is defined as the free strict monoidal category generated by \mathcal{P} and quotiented by the equations in \mathcal{E} ; in other words, the equalizer of the two projections $l^*, r^*: \mathcal{F}_\otimes \mathcal{E} \rightarrow \mathcal{F}_\otimes \mathcal{P}$.*

For the soundness and completeness of string diagrams, see Joyal and Street [29]. For a survey of string diagrams for monoidal categories, see Selinger [47].

Definition 2.7. *A monoidal language or language of string diagrams is a subset of morphisms in a strict monoidal category.*

3 Regular Monoidal Languages

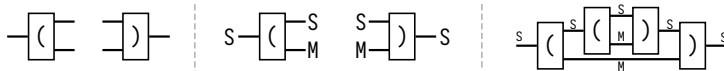
Before introducing context-free monoidal languages, we introduce the *regular* case, which shall play an important role in Section 6. Regular monoidal languages were introduced by Sobociński and the first author [16,17], following earlier work of Bossut and Heindel [7,27]. They are defined by a simple automaton model, reminiscent of tree automata. In a regular monoidal language, the *alphabet* is given by a finite polygraph.

Definition 3.1. *A non-deterministic monoidal automaton comprises: a finite polygraph Γ (the alphabet); a finite set of states Q ; for each generator $\gamma: n \rightarrow m$ in Γ , a transition function $\Delta_\gamma: Q^n \rightarrow \mathcal{P}(Q^m)$; and initial and final state vectors $i, f \in Q^*$.*

Example 3.1. Classical non-deterministic finite state automata arise as monoidal automata over single-sorted polygraphs in which every generator has arity and coarity 1. Bottom-up regular tree automata [23] arise precisely from monoidal automata over single-sorted polygraphs in which every generator has coarity 1 and arbitrary arity, with initial state the empty word and final state a singleton.

A finite state automaton over an alphabet Σ accepts elements of the free monoid Σ^* . A monoidal automaton over a polygraph Γ accepts morphisms in the free monoidal category $\mathcal{F}_\otimes \Gamma$ over Γ . Let us see some examples before giving the formal definition of the accepted language. We depict the transitions of a monoidal automaton as elements of a polygraph with strings labelled by states, and generators labelled by the corresponding element of Γ .

Example 3.2. Consider the following polygraph containing generators (left, below) for an opening and closing parenthesis, and the monoidal automaton over this polygraph with $Q = \{S, M\}$, $i = f = S$, and transitions shown below, centre. An accepting run over this automaton is shown below, right. The string diagram accepted by this run is what we obtain by erasing the states from this picture.



It is clear that the language accepted by this automaton is exactly the “balanced parentheses”, but note that this is not a language of *words*, since we use an extra string to keep track of opening and closing parentheses. This principle will play an important role in our representation theorem in Section 6. Roughly speaking, this extra wire arises from the optical contour of a string language of balanced parentheses.

Example 3.3. In the field of DNA computing, Rothmund, Papadakis and Winfree demonstrated self-assembly of Sierpiński triangles from *DNA tiles* [43]. Sobociński and the first author [16] showed how to recast the tile model as a regular monoidal language over a polygraph containing two tile generators (white and grey), along with start and end generators, as in Figure 3. Note that the start (end) generators have arity (coarity) 0, and hence effect a transition from (to) the empty word of states.

Transitions of a non-deterministic monoidal automaton over Γ extend inductively to string diagrams in $\mathcal{F}_\otimes \Gamma$, giving functions $\hat{\delta}_{n,m} : Q^n \times \mathcal{F}_\otimes \Gamma(n, m) \rightarrow \mathcal{P}(Q^m)$ (Definition F.1).

Definition 3.2. A string diagram $s : n \rightarrow m$ in the free monoidal category $\mathcal{F}_\otimes \Gamma$ over a polygraph Γ is in the language of a non-deterministic monoidal automaton $(\{\Delta_\gamma\}_{\gamma \in \Gamma}, i, f)$ if and only if the run over s reaches the final state, $f \in \hat{\delta}_{n,m}(i, s)$.

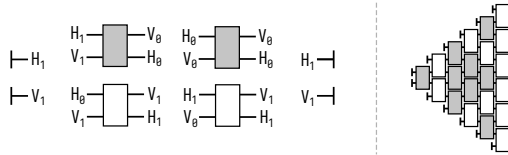


Fig. 3: Transitions for the Sierpiński monoidal automaton (left) and an element of the language (right). The initial and final states are the empty word.

Definition 3.3. A monoidal language L is a regular monoidal language if and only if there exists a non-deterministic monoidal automaton accepting L .

The data of a monoidal automaton is equivalent to a morphism of finite polygraphs, which we call a *regular monoidal grammar*, following Walters' [51] use of the term *grammar* when data is presented as *fibred* over an alphabet, and automata when the alphabet *indexes* transitions as in Definition 3.1. We shall use this convenient presentation in the following.

Definition 3.4. A regular monoidal grammar is a morphism of finite polygraphs $\psi : \mathbb{Q} \rightarrow \Gamma$, equipped with finite initial and final sorts $i, f \in S_{\mathbb{Q}}^*$. The morphisms in $\mathcal{F}_{\otimes} \mathbb{Q}(i, f)$ are derivations in the grammar, and their image under the free monoidal functor $\mathcal{F}_{\otimes} \psi$ is the language of the grammar; a subset of morphisms in $\mathcal{F}_{\otimes} \Gamma$.

Proposition 3.1. For every non-deterministic monoidal automaton there is a regular monoidal grammar with the same language, and vice-versa.

Proof (Sketch). See Appendix, Proposition 3.1.

Not every monoidal language is a regular monoidal language. The following is an example.

Proposition 3.2. Let Γ be the polygraph containing two generators: one for “over-braiding” $\overline{\times}$ and one for “under-braiding” $\underline{\times}$. The language of unbraids on two strings over Γ , i.e. diagrams equivalent under planar isotopy to untangled strings, is not a regular monoidal language.

Proof (Proof sketch). We can use the pumping lemma for regular monoidal languages (Appendix, Lemma B.1), with $k = 2$. The argument is analogous to that for classical languages of balanced parentheses: every over-braiding or under-braiding must be eventually balanced with its opposite.

In the next section, we introduce *context-free monoidal languages* and we shall see that unbraids fall in this class. In Section 6, we prove a surprising representation theorem: every context-free monoidal language is the image under a monoidal functor of a regular monoidal language.

Remark 3.1. As defined, regular monoidal languages are subsets of *free* strict monoidal categories: we shall need only this case in order to prove our main theorem. Context-free monoidal languages will be defined over *arbitrary* strict monoidal categories, so this raises the question of extending the regular case to monoidal categories that are not free. We suggest this can be done by a generalization of Melliès and Zeilberger’s definition of *finite-state automata over a category* as finitary *unique lifting of factorizations* functors [35, Section 2].

4 Context-Free Monoidal Languages

We now turn our attention to context-free grammars over monoidal categories. The multicategory of spliced arrows is defined for any category. However, for categories equipped with a monoidal structure, it is natural to consider more general kinds of holes than allowed by the spliced arrows construction (Figure 4). Rather than tuples of disjoint pieces, we should allow the possibility that a hole can be surrounded by strings. The necessity of considering these more general holes is forced upon us by various examples that could not be captured using spliced arrows (e.g. Examples 4.2 and 4.4). Proofs omitted from this section may be found in Appendix G.

4.1 The symmetric multicategory of diagram contexts

Context-free monoidal grammars should contain productions from a variable to an incomplete diagram containing multiple variables or “holes”. This section constructs diagram contexts over an arbitrary polygraph. Diagram contexts represent the incomplete derivation of a monoidal term: as such, they consist of string diagrams over which we add “holes”. We shall notate these holes in string diagrams as pink boxes (e.g. Figure 4).

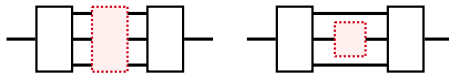


Fig. 4: (Left) A spliced arrow is a tuple of morphisms. (Right) In a monoidal category, there is the possibility of more general holes, which do not split a morphism into disjoint pieces.

Substituting another diagram context inside a hole induces a symmetric multicategorical structure on the diagrams: symmetry means that we do not distinguish the specific order in which the holes appear. This allows us to avoid declaring a particular ordering of holes when defining a context-free monoidal grammar. We could achieve this by introducing a rule that allows us to permute contexts. However, this breaks the correspondence between terms and derivations. Instead, we shall use shufflings, inspired by the work of Shulman [48].

Definition 4.1. A shuffling of two lists, $\Psi \in \text{Shuf}(\Gamma, \Delta)$ is any list Ψ that contains the elements of both Γ and Δ in any order but preserving the relative orders of Γ and Δ .

For instance, if $\Gamma = [\underline{x}, \underline{y}, \underline{z}]$ and $\Delta = [\underline{u}, \underline{v}]$, a shuffling is $\Psi = [\underline{x}, \underline{u}, \underline{y}, \underline{z}, \underline{v}]$, but not $[\underline{y}, \underline{u}, \underline{z}, \underline{x}, \underline{v}]$. The theory of diagram contexts will introduce a shuffling every time it mixes two contexts: this way, if a term was derived by combining two contexts, we can always reorder these contexts however we want. For instance, the term $[\underline{u}, \underline{v}] \vdash [\underline{u}] \mathbin{\text{\textcircled{;}}} [\underline{v}]$ was derived from composing the axioms $[\underline{u}] \vdash [\underline{u}]$ and $[\underline{v}] \vdash [\underline{v}]$; by choosing a different shuffling, we can also derive the term $[\underline{v}, \underline{u}] \vdash [\underline{u}] \mathbin{\text{\textcircled{;}}} [\underline{v}]$. Let us now formally introduce the theory.

Definition 4.2. The theory of diagram contexts \mathbb{P} over a polygraph, \mathcal{P} , is described by the following logic. This logic contains objects $(A, B, C, \dots \in \mathcal{P}_{\text{obj}}^*)$ that consist of lists of types of the polygraph, $X, Y, Z, \dots \in \mathcal{P}_{\text{obj}}$; it also contains contexts $(\Gamma, \Delta, \Psi, \dots)$ that consist of lists of pairs of objects. Apart from the single variables (x, y, z, \dots) and the generators of the polygraph (f, g, h, \dots) ; we consider fully formed terms (t_1, t_2, \dots) .

IDENTITY	GENERATOR	HOLE
$\frac{}{\vdash \text{id} : \overline{X}}$	$\frac{}{\vdash f : \overline{X_1, \dots, X_n} / \overline{Y_1, \dots, Y_m}}}$	$\frac{}{[\underline{x}] : \overline{A} \vdash [\underline{x}] : \overline{A}}$
SEQUENTIAL		
$\frac{}{\Gamma \vdash t_1 : \overline{A} / \overline{B}}$	$\frac{}{\Delta \vdash t_2 : \overline{B} / \overline{C}}$	$\Phi \in \text{Shuf}(\Gamma; \Delta)$
$\frac{}{\Phi \vdash t_1 \mathbin{\text{\textcircled{;}}} t_2 : \overline{A} / \overline{C}}$		
PARALLEL		
$\frac{}{\Gamma \vdash t_1 : \overline{A_1} / \overline{B_1}}$	$\frac{}{\Delta \vdash t_2 : \overline{A_2} / \overline{B_2}}$	$\Phi \in \text{Shuf}(\Gamma; \Delta)$
$\frac{}{\Phi \vdash t_1 \otimes t_2 : \overline{A_1 \text{\textcircled{+}} A_2} / \overline{B_1 \text{\textcircled{+}} B_2}}$		

Where $\text{\textcircled{+}}$ denotes the concatenation of lists. Every term in a given context has a unique derivation. We consider terms up to α -equivalence and we impose the following equations over the terms whenever they are constructed over the same context: $(t_1 \mathbin{\text{\textcircled{;}}} t_2) \mathbin{\text{\textcircled{;}}} t_3 = t_1 \mathbin{\text{\textcircled{;}}} (t_2 \mathbin{\text{\textcircled{;}}} t_3)$; $t \mathbin{\text{\textcircled{;}}} \text{id} = t$; $t_1 \otimes (t_2 \otimes t_3) = (t_1 \otimes t_2) \otimes t_3$; $(t_1 \mathbin{\text{\textcircled{;}}} t_2) \otimes (t_3 \mathbin{\text{\textcircled{;}}} t_4) = (t_1 \otimes t_3) \mathbin{\text{\textcircled{;}}} (t_2 \otimes t_4)$.

Proposition 4.1. The multicategory of derivable sequents in the theory of diagram contexts is symmetric. In logical terms, exchange is admissible in the theory of diagram contexts: whenever we can prove that a diagram context exists under certain context Γ , we can prove that it exists under a permutation of Γ .

Proposition 4.2. Derivable sequents in the theory of diagram contexts over a polygraph \mathcal{P} form the free strict monoidal category over the polygraph extended with special ‘‘hole’’ generators, $\mathcal{P} + \{h_{A,B} : A \rightarrow B \mid A, B \in \mathcal{P}_{\text{obj}}^*\}$. Derivable

sequents over the empty context form the free strict monoidal category over the polygraph \mathcal{P} . Moreover, there exists a symmetric multifunctor

$$i: [\mathcal{F}_\otimes \mathcal{P}] \rightarrow [\mathcal{P}]$$

interpreting each monoidal term as its derivable sequent.

Remark 4.1. Various notions of “holes in a monoidal category” exist in the literature, under names such as *optics*, *contexts*, or *wiring diagrams* [38,40]. Hefford and the authors [41,15] gave a universal characterization of the *produoidal* category of optics over a monoidal category. This produoidal structure is useful for describing *decompositions* of diagrams. The above logic generates a multicategory similar to the operad of directed, acyclic *wiring diagrams* introduced by Patterson, Spivak and Vagner [38]; whose operations are generic morphism shapes, rather than holes in a specific monoidal category.

Definition 4.3. A symmetric multigraph is a multigraph G equipped with bijections $\sigma^*: G(X_1, \dots, X_n; Y) \cong G(X_{\sigma(1)}, \dots, X_{\sigma(n)}; Y)$ for every list X_1, \dots, X_n of sorts and every permutation σ , satisfying $(\sigma \cdot \tau)^* = \sigma^* \circ \tau^*$ and $\text{id}^* = \text{id}$. A morphism of symmetric multigraphs is a morphism of multigraphs which commutes with the bijections.

Definition 4.4. Every multigraph, M , freely induces a symmetric multigraph, $\text{clique}(M)$, with the same objects and, for each $f \in M(X_1, \dots, X_n; Y)$, a clique of elements

$$f_\sigma \in \text{clique}(M)(X_{\sigma(1)}, \dots, X_{\sigma(n)}; Y),$$

connected by symmetries, meaning that $\sigma^*(f_\tau) = f_{\sigma \cdot \tau}$. This is the left adjoint to the inclusion of symmetric multigraphs into multigraphs.

Remark 4.2. Given any symmetric multigraph G , finding a multigraph M whose clique recovers it, $\text{clique}(M) = G$, amounts to choosing a representative for each one of the cliques of the multigraph. Any symmetric multigraph can be (non-uniquely) recovered in this way: for each multimorphism $f \in G(X_1, \dots, X_n; Y)$, we can consider its orbit under the action of the symmetric group, $\text{orb}(f) = \{\sigma^*(f) \mid \sigma \in S_n\}$ – the orbits of different elements may coincide, but each element does have one – and picking an element g_o for each orbit, $o \in \{\text{orb}(f) \mid f \in G\}$, recovers a multigraph giving rise to the original symmetric multigraph.

Definition 4.5. The theory of diagram contexts over a finitely presented monoidal category, $(\mathcal{P}, \mathcal{E}, l, r)$ (Definition 2.6), is the theory of diagram contexts over its generators, quotiented by its equations; in other words, it is the equalizer of the two projections of each equation, interpreted as derivable sequents $([l^*] \circ i), ([r^*] \circ i): [\mathcal{E}] \rightarrow [\mathcal{P}]$.

Proposition 4.3. The formation of diagram contexts in a monoidal category or polygraph extends to functors $[\]: \text{MonCat} \rightarrow \text{MultiCat}$ and $[\]: \text{PolyGraph} \rightarrow \text{MultiGraph}$, which moreover commute with the free multicategory \mathcal{F}_∇ and free monoidal category functors \mathcal{F}_\otimes .

At this point, the reader may doubt that the formation of diagram contexts has a left adjoint similar to the contour functor for spliced arrows. Indeed, in order to recover a left adjoint, we shall need to introduce another multicategory of diagrams which we call raw optics. This technical device will allow us to prove our main theorem (Theorem 6.1). However, let us first see the definition of context-free monoidal grammar, and some examples.

4.2 Context-Free Monoidal Grammars

We now have the ingredients for our central definition. A context-free monoidal grammar specifies a language of string diagrams by a collection of rewrites between *diagram contexts*, where the non-terminals of a context-free grammar are now (labelled) *holes* in a diagram (e.g. Figure 8). Our definition is entirely analogous to Definition 2.3, but using our new symmetric multicategory of diagram contexts in a monoidal category, instead of spliced arrows.

Definition 4.6. *A context-free monoidal grammar over a strict monoidal category (\mathbb{C}, \otimes, I) is a morphism of symmetric multigraphs $\Psi : \mathcal{G} \rightarrow \mathbb{C}$, into the underlying multigraph of diagram contexts in \mathbb{C} , where \mathcal{G} is finite, and a start sort $S_{X,Y} \in \Psi^{-1}(X)$.*

We shall use the notation $S \sqsubset \frac{A}{B}$ to indicate that $\Psi(S) = \frac{A}{B}$, following the convention in the literature [35]. A morphism of symmetric multigraphs $\Psi : \mathcal{G} \rightarrow \mathbb{C}$ defining a grammar uniquely determines, via the free-forgetful adjunction, a symmetric multifunctor $\hat{\Psi} : \mathcal{F}_{\nabla} \mathcal{G} \rightarrow \mathbb{C}$, mapping (closed) derivations to morphisms of \mathbb{C} . The language of a grammar is then defined analogously to Definition 2.4:

Definition 4.7. *Let $(\Psi : \mathcal{G} \rightarrow \mathbb{C}, S \sqsubset \frac{A}{B})$ be a context-free monoidal grammar. The language of Ψ is the set of morphisms in $\mathbb{C}(A; B)$ given by the image under $\hat{\Psi}$ of the set of derivations $\mathcal{F}_{\nabla} \mathcal{G}(\cdot; S)$. A set of morphisms L in \mathbb{C} is a context-free monoidal language if and only if there exists a context-free monoidal grammar whose language is L .*

Example 4.1 (Classical context-free languages). Every context-free monoidal grammar of the following form is equivalent to a classical context-free grammar of words. Let Γ be a single-sorted finite polygraph whose generators are all of arity and coarity 1. Then context-free monoidal grammars over $\mathcal{F}_{\otimes} \Gamma$ with a start symbol $\phi(S) \sqsubset \frac{1}{1}$ are context-free grammars of words over Γ . Figure 5 gives the classical example of balanced parentheses. Similarly, every context-free grammar of words may be encoded as a context-free monoidal grammar in this way.

Example 4.2 (Context-free tree grammars). Context-free tree grammars [23,44] are defined over *ranked alphabets* of terminals and non-terminals, which amount to polygraphs in which the generators have arbitrary arity (the rank) and coarity 1. Productions have the form $A(x_1, \dots, x_m) \rightarrow t$ where the left hand side is a non-terminal of rank m whose frontier is labelled by the variables x_i in order, and

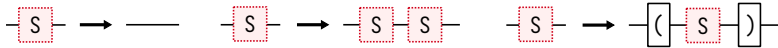


Fig. 5: Balanced parentheses as a context-free monoidal grammar.

whose right hand side is a tree t built from terminals and non-terminals, and whose frontier is labelled by variables from the set $\{x_1, \dots, x_m\}$. Note that t may use the variables non-linearly.

For example, let S be a non-terminal with coarity 0, A a non-terminal with coarity 2, f a terminal of coarity 2, and x a terminal of coarity 0 (a leaf). Then a possible rule over these generators is $A(x_1, x_2) \rightarrow f(x_1, A(x_1, x_2))$, where x_1 appears non-linearly. In order to allow such non-linear use of variables in a context-free monoidal grammar, we can consider the free *cartesian* category over Γ . In terms of string diagrams, this amounts to introducing new generators for copying ($\dashv\sqsubset$) and deleting variables ($\dashv\sqsupset$), satisfying some equations which we recall in Appendix D.

Let Γ be a polygraph in which generators have arbitrary arity, and coarity 1, as above. Context-free monoidal grammars over the free cartesian category on Γ , with a start symbol $S \sqsubset \begin{smallmatrix} 0 \\ 1 \end{smallmatrix}$ are context-free tree grammars. In Figure 6 we extend the above data to a full example. Note that by allowing start symbols $S \sqsubset \begin{smallmatrix} 0 \\ n \end{smallmatrix}$, we can produce forests of n trees.

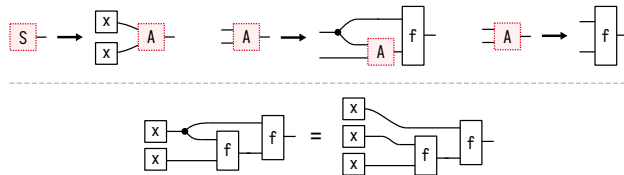


Fig. 6: Example of a context-free tree grammar as a context-free monoidal grammar. The string diagrams at the bottom are *equal* in the free *cartesian* category over the polygraph of terminals.

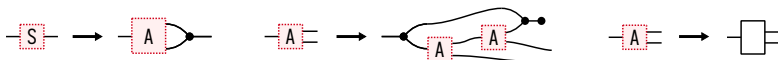


Fig. 7: A hypergraph grammar for simple *control flow graphs* with branching and looping, as a context-free monoidal grammar. Based on Habel [25, Example 3.3].

Example 4.3 (Hyperedge-replacement grammars). Hyperedge-replacement (HR) grammars are a kind of *context-free graph grammar* [20]. We consider HR grammars in *normal form* in the sense of Habel [25, Theorem 4.1]. A production

$N \rightarrow R$ of an HR grammar has N a non-terminal with arity and coarity, and R a hypergraph with the same arity and coarity³, whose hyperedges are labelled by some finite set of terminals and non-terminals. Just as trees are morphisms in free *cartesian* monoidal categories (Example 4.2), hypergraphs are the morphisms of monoidal categories equipped with extra structure, known as *hypergraph categories* [42,5,21]. Generators in a polygraph are exactly directed hyperedges. The extra structure in a hypergraph category, which we recall in Appendix E, amounts to a combinatorial encoding of patterns of wiring between nodes.

Let Γ be a polygraph of terminal hyperedges, G a multigraph of non-terminal rules, and $S \in G$ a start symbol. Then context-free monoidal grammars $(G \rightarrow |\text{Hyp}[\Gamma]|, S)$ over the free hypergraph category on Γ are exactly hyperedge replacement grammars over Γ (e.g. Figure 7). A hole in a morphism in $\text{Hyp}[\Gamma]$ is a placeholder for an (n, m) hyperedge, the grammar labels these holes by non-terminals, and composition corresponds to hyperedge replacement.

Example 4.4 (Unbraids). We return to the language of unbraids suggested in Proposition 3.2. Take the grammar over the over- and under-braiding polygraph depicted in Figure 8, with start symbol $S \sqsubset \frac{2}{2}$. The language of this grammar consists of unbraids on two strings.

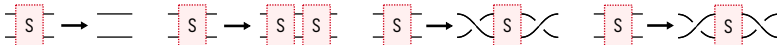


Fig. 8: A context-free monoidal grammar of unbraids, with start symbol S .

Let us record some basic closure properties of context-free monoidal languages.

Proposition 4.4. *Context-free monoidal languages over \mathbb{C} with start symbol $S \sqsubset \frac{X}{Y}$ are closed under union. The underlying morphism is given by the copairing, and start symbols can be unified by introducing a fresh symbol and productions where necessary, as in the classical case. Context-free monoidal languages are also closed under images of strict monoidal functors: the underlying morphism is given by postcomposition.*

5 Optical Contour of a Multicategory

An important realization of Mellies and Zeilberger is that the formation of spliced arrows in a category has a left adjoint, which they call the *contour* of a multicategory [35, Section 3.2]. This adjunction is a key conceptual tool in their generalized version of the Chomsky-Schützenberger representation theorem, and is closely linked to the notion of *item* in LR parsing [35]. In this section, we present a similar adjunction for the monoidal setting. However, it is not clear that the formation of diagram contexts has a left adjoint. We must therefore first conduct a dissection of diagram contexts into *raw optics*.

³ A *multi-pointed hypergraph* in Habel’s terminology.

5.1 The multicategory of raw optics

A raw optic is a tuple of morphisms obtained by cutting a diagram context into a sequence of disjoint pieces. The term *optics* refers to a notion closely related to diagram contexts, which are defined exactly as a quotient of raw optics [10,39]. In Section 5 we shall see that raw optics has a left adjoint, the optical contour, and this will be enough to prove our representation theorem (Theorem 6.1).

Definition 5.1. *The multicategory of raw optics over a strict monoidal category \mathbb{C} , denoted $\text{ROpt}[\mathbb{C}]$, is defined to have, as objects, pairs $\frac{A}{B}$ of objects of \mathbb{C} , and, as its set of multimorphisms, $\text{ROpt}[\mathbb{C}]_{(B_1, \dots, B_n; \frac{A}{T})}^{(A_1, \dots, A_n)}$, the following set, where we write AB for $A \otimes B$,*

$$\sum_{M_i, N_i \in \mathbb{C}} \mathbb{C}(S; M_1 A_1 N_1) \times \prod_{i=1}^{n-1} \mathbb{C}(M_i B_i N_i; M_{i+1} A_{i+1} N_{i+1}) \times \mathbb{C}(M_n B_n N_n; T).$$

As a special case, $\text{ROpt}[\mathbb{C}]_{(\frac{S}{T})} := \mathbb{C}(S; T)$. In other words, a multimorphism, from $\frac{A_1}{B_1}, \dots, \frac{A_n}{B_n}$ to $\frac{S}{T}$, consists of two families of objects, M_1, \dots, M_n and N_1, \dots, N_n , and a family of functions, (f_0, \dots, f_n) , with types $f_0: S \rightarrow M_1 \otimes A_1 \otimes N_1$; with $f_i: M_i \otimes B_i \otimes N_i \rightarrow M_{i+1} \otimes A_{i+1} \otimes N_{i+1}$; for each $1 \leq i \leq n-1$; and $f_n: M_n \otimes B_n \otimes N_n \rightarrow T$. In the special nullary case, we have a single morphism $f_0: S \rightarrow T$.

Identities are given by pairs $(\text{id}_A, \text{id}_B)$. Given two raw optics $f = (f_0, \dots, f_n)$ and $g = (g_0, \dots, g_m)$, their composition is defined by

$$f \circledast_i g := (g_0, \dots, g_i \circledast (\text{id} \otimes f_0 \otimes \text{id}), \dots, \text{id} \otimes f_i \otimes \text{id}, \dots, (\text{id} \otimes f_n \otimes \text{id}) \circledast_i g_{i+1}, \dots, g_m).$$

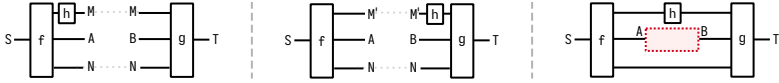


Fig. 9: Two raw optics (left, centre) in $\text{ROpt}[\mathbb{C}]_{(\frac{A}{B}; \frac{S}{T})}$ which quotient to the same diagram context. Note that a raw optic is not the same as a spliced arrow: the types M, N must match.

Every raw optic can be glued into a diagram context, as illustrated in Figure 9. More precisely, we have the following result.

Proposition 5.1. *There is an identity on objects multifunctor $q: \text{ROpt}[\mathbb{C}] \rightarrow \mathbb{C}$ mapping each raw optic to its corresponding diagram context. Equivalently, there is an identity on objects symmetric multifunctor $q^*: \text{clique}(\text{ROpt}[\mathbb{C}]) \rightarrow \mathbb{C}$; this symmetric multifunctor is full.*

Proposition 5.2. *The construction of raw optics extends to a functor $\text{ROpt}: \text{MonCat} \rightarrow \text{MultiCat}$ between the categories of strict monoidal categories and strict monoidal functors, and multicategories and multifunctors.*

Remark 5.1. We could have defined context-free monoidal grammars as morphisms into raw optics, rather than diagram contexts, but this would require an arbitrary choice of raw optic for each rule, as in Figure 9. In particular, this would force us to choose a particular ordering of the holes, since raw optics do not form a *symmetric* multicategory. On the other hand, that such a choice exists will be needed to prove our representation theorem (Section 6).

5.2 Optical contour

We now introduce the left adjoint to the formation of raw optics, which we call the *optical contour* of a multicategory. The difference from the contour of Section 2 is that additional objects M_i, N_i are introduced which keep track of strings that might surround holes. This gives rise to a strict monoidal category.

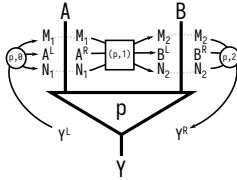


Fig. 10: A multimorphism $p \in \mathbb{M}(A, B; Y)$ and its three sectors given by optical contour: $(p, 0) : Y^L \rightarrow M_1 \otimes A^L \otimes N_1$, $(p, 1) : M_1 \otimes A^R \otimes N_1 \rightarrow M_2 \otimes B^L \otimes N_2$, $(p, 2) : M_2 \otimes B^R \otimes N_2 \rightarrow Y^R$.

Definition 5.2. Let \mathbb{M} be a multicategory. Its optical contour, \mathbb{CM} , is the strict monoidal category presented by a polygraph whose generators are given by taking contours of multimorphisms in \mathbb{M} . Each multimorphism gives rise to a set of generators for the monoidal category \mathbb{CM} – its set of sectors, as in Figure 10.

Explicitly, for each object $A \in \mathbb{M}$, the optical contour \mathbb{CM} contains a left polarized, A^L , and a right polarized, A^R , version of the object. Additionally, for each multimorphism $f \in \mathbb{M}(X_1, \dots, X_n; Y)$, there exists a family of objects $M_1^f, \dots, M_n^f, N_1^f, \dots, N_n^f$, whose superscripts we omit when they are clear from context. The morphisms are given by the following generators. For each $f \in \mathbb{M}(X_1, \dots, X_n; Y)$, we consider the following $n + 1$ generators:

$$\begin{aligned} (f, 0) &: Y^L \rightarrow M_1^f \otimes X_1^L \otimes N_1^f, \\ (f, i) &: M_i^f \otimes X_i^R \otimes N_i^f \rightarrow M_{i+1}^f \otimes X_{i+1}^L \otimes N_{i+1}^f, \text{ for } 1 \leq i \leq n-1, \text{ and} \\ (f, n) &: M_n^f \otimes X_n^R \otimes N_n^f \rightarrow Y^R. \end{aligned}$$

In particular, for a nullary multimorphism $f \in \mathbb{M}(; Y)$, we consider a generator $(f, 0) : Y^L \rightarrow Y^R$. Further, we ask for the following equations which ensure that the optical contour preserves identities and composition: for all $x \in \mathbb{M}$,

$(\text{id}_X, 0) = \text{id}_{X^L}, (\text{id}_X, 1) = \text{id}_{X^R}$ with $M_1^{\text{id}_X} = N_1^{\text{id}_X} = I$; and given any $f \in \mathbb{M}(X_1, \dots, X_n; Y_i)$ and $g \in \mathbb{M}(Y_1, \dots, Y_m; Z)$,

$$(f \circledast_i g, j) = \begin{cases} (g, j) & j < i, \text{ with } M_j^{f \circledast_i g} = M_j^g, N_j^{f \circledast_i g} = N_j^g \\ (g, i) \circledast (\text{id} \otimes (f, 0) \otimes \text{id}) & j = i, \text{ with } M_i^{f \circledast_i g} = M_j^g \otimes M_0^f \\ \text{id}_{M_i^g} \otimes (f, j-i) \otimes \text{id}_{N_j^g} & i < j < i+n, \text{ with } M_j^{f \circledast_i g} = M_i^g \otimes M_{j-i}^f \\ (\text{id} \otimes (f, n) \otimes \text{id}) \circledast (g, i+1) & j = i+n+1, \text{ with } M_j^{f \circledast_i g} = M_i^g \otimes M_n^f \\ (g, j-n) & j > i+n+1 \text{ with } M_j^{f \circledast_i g} = M_{j-n}^g. \end{cases}$$

In particular, when $f \in \mathbb{M}(\cdot; Y_i)$ is nullary, $(f \circledast_i g, 0) = g_i \circledast f_0 \circledast g_{i+1}$.

Theorem 5.1. *Optical contour is left adjoint to raw optics; there exists an adjunction $(\mathcal{C} \dashv \text{ROpt}) : \text{MonCat} \rightarrow \text{MultiCat}$.*

Proof. See Appendix C.

6 A Monoidal Representation Theorem

The Chomsky-Schützenberger representation theorem says that every context-free language can be obtained as the image under a homomorphism of the intersection of a Dyck language and a regular language [9]. Melliès and Zeilberger [34] use their splicing-contour adjunction to give a novel proof of this theorem for context-free languages in categories: the classical version is recovered when the category is a free monoid. The role of the Dyck language, providing linearizations of derivation trees, is taken over by *contours* of derivations.

Monoidal categories provide a more striking case: the Dyck language is not needed because the information that parentheses encode can be carried instead by tensor products. In this section, we show that a regular monoidal language of optical contours is sufficient to reconstruct the original language. Theorem 6.1 states that *every context-free monoidal language is the image under a monoidal functor of a regular monoidal language*.

Our strategy will be to first choose a factoring of a grammar into raw optics, then use the optical contour/raw optics adjunction to produce the required monoidal functor. We must first establish that such a factoring exists. Omitted proofs may be found in Appendix I.

Lemma 6.1. *Any morphism of symmetric multigraphs underlying a context-free monoidal grammar, $\phi : G \rightarrow |\mathbb{C}|$, factors (non-uniquely) through the quotienting of raw optics (Proposition 5.1); meaning that there exists some multigraph G' satisfying $G = \text{clique}(G')$, and some morphism $\phi_r : G' \rightarrow |\text{ROpt}[\mathbb{C}]|$, such that $\phi = \text{clique}(\phi_r) \circledast q^*$.*

Call the factor $\phi_r : G' \rightarrow |\text{ROpt}[\mathbb{C}]|$ a *raw representative* of ϕ . It amounts to choosing a fixed ordering of the holes in a diagram context for each rule in the grammar, and a particular splicing into a raw optic.

Lemma 6.2. *Let $\mathcal{G} = (\phi, S)$ be a context-free monoidal grammar. Then the language of any raw representative ϕ_r of ϕ (with start symbol S) equals the language of \mathcal{G} . That is, $\phi_r[\mathcal{F}_\nabla G'(; S)] = \phi[\mathcal{F}_\nabla G(; S)]$.*

Lemma 6.3. *A raw representative $\phi_r : G' \rightarrow |\mathbf{ROpt}[\mathbb{C}]|$ uniquely determines a strict monoidal functor $I_\phi : \mathcal{F}_\otimes(\mathcal{C}G') \rightarrow \mathbb{C}$.*

We shall see that this monoidal functor maps the following regular monoidal language over $\mathcal{C}G$ to the language of the original context-free monoidal grammar.

Definition 6.1. *Let $\mathcal{G} = (\phi : G \rightarrow |\mathbb{C}|, S)$ be a context-free monoidal grammar, and ϕ_r a raw representative with domain G' . Define a regular representative of \mathcal{G} to be the regular monoidal grammar $\mathcal{R} = (\text{id} : \mathcal{C}G' \rightarrow \mathcal{C}G', S^L, S^R)$ over optical contours of G' whose morphism of polygraphs is the identity.*

Lemma 6.4. *Given a multigraph G , there is a bijection between derivations rooted at a sort S and optical contours from S^L to S^R , i.e. $\mathcal{F}_\nabla G(; S) \cong \mathcal{F}_\otimes(\mathcal{C}G)(S^L; S^R)$.*

Theorem 6.1. *The language of a context-free monoidal grammar $\mathcal{G} = (\phi : G \rightarrow |\mathbb{C}|, S)$ equals the image of a regular representative under the monoidal functor I_ϕ of Lemma 6.3.*

Theorem 6.1 is at first quite surprising, since in comparison with the usual Chomsky-Schützenberger theorem and its generalization [35], one might expect to see an *intersection* of a regular monoidal language and a context-free monoidal language. Instead, this theorem tells us that regular monoidal languages are powerful enough to encode context-free monoidal languages, even while the latter is strictly more expressive than the former. Just as a context-free grammar suffices to specify a programming language which may encode instructions for arbitrary computations, regular monoidal languages can specify arbitrary context-free monoidal languages, with a monoidal functor effecting the “compilation”.

7 Conclusion

There are still many avenues to explore in this structural approach to context-free languages. One obvious direction is to investigate a notion of pushdown automaton for context-free monoidal languages. In fact, it still remains to be elaborated how pushdown automata emerge for context-free languages over plain categories. Following the general principle of *parsing as a lifting problem* [35], and the duality of grammars (fibered) and automata (indexed) may provide some clue to characterizing such automata by a universal property.

The study of languages and the dependence relations that diagram contexts naturally present may be useful to the study of complexity in monoidal categories, such as the notion of “monoidal width” proposed by Di Lavore and Sobociński [31,32]. Conversely, measures of monoidal complexity may inform the cost of parsing different terms.

Finally, different types of string diagram exist for a variety of widely applied categorical structures beyond monoidal categories, such as double categories [37]. There are many opportunities to extend the general principle elaborated here to a notion of context-free language in these structures.

References

1. Samson Abramsky and Bob Coecke. Categorical quantum mechanics. In Kurt Engesser, Dov M. Gabbay, and Daniel Lehmann, editors, *Handbook of Quantum Logic and Quantum Structures*, pages 261–323. Elsevier, Amsterdam, 2009. URL: <https://www.sciencedirect.com/science/article/pii/B9780444528698500104>, doi: 10.1016/B978-0-444-52869-8.50010-4.
2. Jirí Adámek, Robert S.R. Myers, Henning Urbat, and Stefan Milius. Varieties of languages in a category. In *2015 30th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 414–425, 2015. doi:10.1109/LICS.2015.46.
3. Achim Blumensath. Algebraic language theory for Eilenberg–Moore algebras. *Logical Methods in Computer Science*, 17, 2021.
4. 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.
5. Filippo Bonchi, Fabio Gadducci, Aleks Kissinger, Pawel Sobocinski, and Fabio Zanasi. String diagram rewrite theory I: Rewriting with Frobenius structure. *J. ACM*, 69(2), mar 2022. doi:10.1145/3502719.
6. Filippo Bonchi, Robin Piedeleu, Pawel Sobocinski, and Fabio Zanasi. Graphical affine algebra. In *34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019, Vancouver, BC, Canada, June 24-27, 2019*, pages 1–12. IEEE, 2019. doi:10.1109/LICS.2019.8785877.
7. 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.
8. H.J. Sander Bruggink and Barbara König. Recognizable languages of arrows and cospans. *Mathematical Structures in Computer Science*, 28(8):1290–1332, 2018.
9. Noam Chomsky and Marcel-Paul Schützenberger. The algebraic theory of context-free languages. In P. Braffort and D. Hirschberg, editors, *Computer Programming and Formal Systems*, volume 35 of *Studies in Logic and the Foundations of Mathematics*, pages 118–161. Elsevier, 1963. URL: <https://www.sciencedirect.com/science/article/pii/S0049237X08720238>, doi:10.1016/S0049-237X(08)72023-8.
10. Bryce Clarke, Derek Elkins, Jeremy Gibbons, Fosco Loregiàn, Bartosz Milewski, Emily Pillmore, and Mario Román. Profunctor optics, a categorical update. *CoRR*, abs/2001.07488, 2020. URL: <https://arxiv.org/abs/2001.07488>, arXiv:2001.07488.
11. Thomas Colcombet and Daniela Petrisan. Automata minimization: a functorial approach. *Log. Methods Comput. Sci.*, 16(1), 2020.
12. 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/CB09780511977619.
13. Philippe de Groote. Towards abstract categorial grammars. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics*, pages

- 252–259, Toulouse, France, July 2001. Association for Computational Linguistics. URL: <https://aclanthology.org/P01-1033>, doi:10.3115/1073012.1073045.
14. Volker Diekert and Grzegorz Rozenberg. *The Book of Traces*. World Scientific, 1995. doi:10.1142/2563.
 15. Matthew Earnshaw, James Hefford, and Mario Román. The Produoidal Algebra of Process Decomposition. In Aniello Murano and Alexandra Silva, editors, *32nd EACSL Annual Conference on Computer Science Logic (CSL 2024)*, volume 288 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 25:1–25:19, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. URL: <https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.CSL.2024.25>, doi:10.4230/LIPIcs.CSL.2024.25.
 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. Matthew Earnshaw and Paweł Sobociński. Regular planar monoidal languages. *Journal of Logical and Algebraic Methods in Programming*, 2024. In Press.
 19. 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.
 20. Joost Engelfriet. *Context-Free Graph Grammars*, pages 125–213. Springer Berlin Heidelberg, Berlin, Heidelberg, 1997. doi:10.1007/978-3-642-59126-6_3.
 21. Brendan Fong and David I. Spivak. Hypergraph categories. *Journal of Pure and Applied Algebra*, 223(11):4746–4777, 2019. URL: <https://www.sciencedirect.com/science/article/pii/S0022404919300489>, doi:10.1016/j.jpaa.2019.02.014.
 22. Tobias Fritz. A synthetic approach to Markov kernels, conditional independence, and theorems on sufficient statistics. *CoRR*, abs/1908.07021, 2019. URL: <http://arxiv.org/abs/1908.07021>, arXiv:1908.07021.
 23. Ferenc Gécseg and Magnus Steinby. *Tree Languages*, page 1–68. Springer-Verlag, Berlin, Heidelberg, 1997.
 24. Gary Griffing. Composition-representative subsets. *Theory and Applications of Categories*, 11(19):420–437, 2003.
 25. Annegret Habel. *Hyperedge Replacement: Grammars and Languages*, volume 643 of *Lecture Notes in Computer Science*. Springer, 1992. URL: <https://doi.org/10.1007/BFb0013875>, doi:10.1007/BFb0013875.
 26. Tobias Heindel. The Chomsky-Schützenberger theorem with circuit diagrams in the role of words. *Abstract*, 2017.
 27. Tobias Heindel. A Myhill-Nerode theorem beyond trees and forests via finite syntactic categories internal to monoids. *Preprint*, 2017.
 28. Alan Jeffrey. Premonoidal categories and a graphical view of programs. *Preprint, Dec*, pages 80688–7, 1997.

29. André Joyal and Ross Street. The geometry of tensor calculus, I. *Advances in Mathematics*, 88(1):55–112, 1991. URL: <https://www.sciencedirect.com/science/article/pii/000187089190003P>, doi:10.1016/0001-8708(91)90003-P.
30. Mohammad Amin Kuhail, Shahbano Farooq, Rawad Hammad, and Mohammed Bahja. Characterizing visual programming approaches for end-user developers: A systematic review. *IEEE Access*, 9:14181–14202, 2021.
31. Elena Di Lavore. *Monoidal Width*. PhD thesis, Tallinn University of Technology, 2023.
32. Elena Di Lavore and Pawel Sobocinski. Monoidal width. *Log. Methods Comput. Sci.*, 19(3), 2023. URL: [https://doi.org/10.46298/lmcs-19\(3:15\)2023](https://doi.org/10.46298/lmcs-19(3:15)2023), doi:10.46298/LMCS-19(3:15)2023.
33. Tom Leinster. *Higher Operads, Higher Categories*. London Mathematical Society Lecture Note Series. Cambridge University Press, 2004. doi:10.1017/CB09780511525896.
34. Paul-André Mellies 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*, 2022.
35. Paul-André Mellies and Noam Zeilberger. The categorical contours of the Chomsky-Schützenberger representation theorem. Preprint, December 2023. URL: <https://hal.science/hal-04399404>.
36. Paul-André Mellies. Categorical semantics of linear logic. In *Interactive models of computation and program behaviour, panoramas et synthèses*, volume 27, pages 1–196. Société Mathématique de France, 2009.
37. David Jaz Myers. String diagrams for double categories and equipments, 2018. [arXiv:1612.02762](https://arxiv.org/abs/1612.02762).
38. Evan Patterson, David I. Spivak, and Dmitry Vagner. Wiring diagrams as normal forms for computing in symmetric monoidal categories. *Electronic Proceedings in Theoretical Computer Science*, 333:49–64, February 2021. URL: <http://dx.doi.org/10.4204/EPTCS.333.4>, doi:10.4204/eptcs.333.4.
39. Mitchell Riley. Categories of Optics. *arXiv preprint arXiv:1809.00738*, 2018.
40. Mario Román. Open diagrams via coend calculus. *Electronic Proceedings in Theoretical Computer Science*, 333:65–78, Feb 2021. URL: <http://dx.doi.org/10.4204/EPTCS.333.5>, doi:10.4204/eptcs.333.5.
41. Mario Román. *Monoidal Context Theory*. PhD thesis, Tallinn University of Technology, 2023.
42. Robert Rosebrugh, Nicoletta Sabadini, and Robert F.C. Walters. Generic commutative separable algebras and cospans of graphs. *Theory and applications of categories*, 15:164–177, 2005.
43. Paul W. K. Rothmund, Nick Papadakis, and Erik Winfree. Algorithmic self-assembly of DNA Sierpinski triangles. *PLOS Biology*, 2(12), 12 2004. doi:10.1371/journal.pbio.0020424.
44. William C. Rounds. Context-free grammars on trees. In *Proceedings of the First Annual ACM Symposium on Theory of Computing*, STOC '69, page 143–148, New York, NY, USA, 1969. Association for Computing Machinery. doi:10.1145/800169.805428.
45. Grzegorz Rozenberg. *Handbook Of Graph Grammars And Computing By Graph Transformation, Vol 1: Foundations*. World Scientific Publishing Company, 1997. URL: <https://books.google.ee/books?id=KwbtCgAAQBAJ>.
46. Hiroyuki Seki, Takashi Matsumura, Mamoru Fujii, and Tadao Kasami. On multiple context-free grammars. *Theoretical Computer Science*, 88(2):191–

- 229, 1991. URL: <https://www.sciencedirect.com/science/article/pii/030439759190374B>, doi:10.1016/0304-3975(91)90374-B.
47. Peter Selinger. A survey of graphical languages for monoidal categories. In Bob Coecke, editor, *New Structures for Physics*, pages 289–355. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. doi:10.1007/978-3-642-12821-9_4.
48. Michael Shulman. Categorical logic from a categorical point of view. *Available on the web*, 2016. URL: <https://mikesulman.github.io/catlog/catlog.pdf>.
49. James 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.
50. Bret Tilson. Categories as algebra: An essential ingredient in the theory of monoids. *Journal of Pure and Applied Algebra*, 48(1):83–198, 1987. URL: <https://www.sciencedirect.com/science/article/pii/0022404987901083>, doi:10.1016/0022-4049(87)90108-3.
51. Robert 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.
52. Vladimir Zamdzhiev. *Rewriting Context-free Families of String Diagrams*. PhD thesis, University of Oxford, 2016.

A Monoidal Categories

Definition A.1. A strict monoidal category \mathbb{C} consists of a monoid of objects, or resources, $(\mathbb{C}_{obj}, \otimes, I)$, and a collection of morphisms, or processes, $\mathbb{C}(X; Y)$, indexed by an input $X \in \mathbb{C}_{obj}$ and an output $Y \in \mathbb{C}_{obj}$. A strict monoidal category is endowed with operations for the sequential and parallel composition of processes, respectively

$$\begin{aligned} (\circlearrowleft): \mathbb{C}(X; Y) \times \mathbb{C}(Y; Z) &\rightarrow \mathbb{C}(X; Z), \\ (\otimes): \mathbb{C}(X; Y) \times \mathbb{C}(X'; Y') &\rightarrow \mathbb{C}(X \otimes X'; Y \otimes Y'), \end{aligned}$$

and a family of identity morphisms, $id_X \in \mathbb{C}(X; X)$. Strict monoidal categories must satisfy the following axioms.

1. Sequencing is unital, $f \circlearrowleft id_Y = f$ and $id_X \circlearrowleft f = f$.
2. Sequencing is associative, $f \circlearrowleft (g \circlearrowleft h) = (f \circlearrowleft g) \circlearrowleft h$.
3. Tensoring is unital, $f \otimes id_I = f$ and $id_I \otimes f = f$.
4. Tensoring is associative, $f \otimes (g \otimes h) = (f \otimes g) \otimes h$.
5. Tensoring and identities interchange, $id_A \otimes id_B = id_{A \otimes B}$.
6. Tensoring and sequencing interchange,

$$(f \circlearrowleft g) \otimes (f' \circlearrowleft g') = (f \otimes f') \circlearrowleft (g \otimes g').$$

Remark A.1. We take this particular formulation of the definition, slightly different from that found in most references, from the thesis of Román [41].

Definition A.2. A symmetric strict monoidal category is a monoidal category equipped with a natural family of isomorphisms $\sigma_{X, Y} : X \otimes Y \rightarrow Y \otimes X$ for every pair of objects X, Y . We can extend string diagrams to express this structure, by allowing strings to cross without tangling. That is, we introduce components (below, left) for every pair of sorts, and equations (below, right) expressing that these are natural isomorphisms. Adding this structure to the free monoidal category over a polygraph presents the free symmetric monoidal category over that polygraph.



Definition A.3. A strict monoidal functor, $F: \mathbb{C} \rightarrow \mathbb{D}$, is a monoid homomorphism between their objects, $F_{obj}: \mathbb{C}_{obj} \rightarrow \mathbb{D}_{obj}$, and an assignment of morphisms $f \in \mathbb{C}(X; Y)$ to morphisms $F(f) \in \mathbb{D}(FX; FY)$. A functor must preserve sequential composition, $F(f \circlearrowleft g) = F(f) \circlearrowleft F(g)$; parallel composition, $F(f \otimes g) = F(f) \otimes F(g)$; and identities, $F(id) = id$. Strict monoidal categories with strict monoidal functors form a category, MonCat .

B Pumping lemma for regular monoidal languages

Lemma B.1 ([18]). *Let L be a regular monoidal language. Then $\forall k \in \mathbb{N}^+, \exists n$ such that for any $s \in L$ where s may be factorized into $m \geq n$ non-identity morphisms $s = s_0 \circ \dots \circ s_i \circ \dots \circ s_m$ where $s_i : k_i \rightarrow k_{i+1}$, with $1 \leq k_i \leq k$, there exists i, j, ℓ such that $k_i = k_j = \ell$ and $s' \circ (s'')^a \circ s''' \in L$ for all $a \geq 0$, where $s' = s_0 \circ \dots \circ s_i$, $s'' = s_{i+1} \circ \dots \circ s_j$, and $s''' = s_{j+1} \circ \dots \circ s_m$.*

Proof. Let L be the language of a grammar $(\phi : M \rightarrow \Gamma, I, F)$. If L has a finite number of connected string diagrams, then for any k take n be longer than the longest factorization over all diagrams in L , then the lemma holds vacuously. Otherwise let k be given, then take $n = \sum_{i=0}^k |S_M|^i$. Let $s \in L$, such that it has a factorization of the form above. Then by the pigeonhole principle, we will have i, j, ℓ as required in the lemma.

Lemma B.2 (Contrapositive form). *Let L be a language and suppose that $\exists k \in \mathbb{N}^+$ such that $\forall n$ there exists a morphism $w \in L$ that factorizes as in Lemma B.1 and for all i, j, ℓ such that $k_i = k_j = \ell$, there exists an a such that the pumped morphism $w'w''^aw''' \notin L$, then L is not regular monoidal.*

Remark B.1. This reduces to the pumping lemma for words and trees, taking $k = 1$.

C Optical contour-splice adjunction

Theorem 5.1. *Optical contour is left adjoint to raw optics; there exists an adjunction $(\mathcal{C} \dashv \mathbf{ROpt}) : \mathbf{MonCat} \rightarrow \mathbf{MultiCat}$.*

Proof. Let \mathbb{C} be a strict monoidal category and let \mathbb{M} be a multicategory. We need first to prove that the two constructions involved, \mathcal{C} and \mathbf{ROpt} , are indeed functors – this proof, although tedious, proceeds as expected and we prefer to omit it here.

We will show that there is a bijection between strict monoidal functors $\mathcal{C}\mathbb{M} \rightarrow \mathcal{C}$ and multifunctors $\mathbb{M} \rightarrow \mathbf{ROpt}[\mathcal{C}]$.

- The objects of $\mathbf{ROpt}[\mathcal{C}]$ are pairs of objects. Mapping an object of the multicategory $X \in \mathbb{M}$ to a pair of objects is the same as mapping two objects, X^L and X^R , to the objects of the category \mathcal{C} .
- Mapping a multimorphism $f \in \mathbb{M}(X_1, \dots, X_n; Y)$ to the multicategory of raw optics consists of choosing a family of functions (f_0, \dots, f_n) together with two families of objects M_1, \dots, M_n and N_1, \dots, N_n . This is the same choice we need to map each one of the components of the contour of $f \in \mathbb{M}(X_1, \dots, X_n; Y)$ to that exact family of functions.

That is, we have only checked that, by construction, the maps out of the contour correspond with multifunctors to raw optics. The adjunction remains conceptually interesting because it links two concepts that have different conceptual interpretations, even if it can be reduced to note that one has been defined as the adjoint to the other.

D Cartesian monoidal categories

The free *cartesian category* over a polygraph may be presented using string diagrams. As with symmetric monoidal categories, we add some new generators and equations, to the effect that every object is equipped with a natural and uniform counital comagma structure. That is, in addition to symmetric structure, we add the following generators and equations.

This structure must be uniform, in the sense that the structure on tensor products is given by tensor products of structure. See [47, Section 4.1] for more details.

Remark D.1. In most sources, cartesian categories are presented in terms of the presence of cocommutative comonoid structure. However, it is folklore that counital comagmas suffice [36, p. 122] [41].

E Hypergraph categories

The free *hypergraph category* over a polygraph may be presented using string diagrams. As with cartesian monoidal categories, we add some new generators and equations. This extra structure amounts to equipping every object with the structure of a special commutative Frobenius algebra. That is, in addition to symmetry we ask for the following generators and equations:

Moreover, this structure must be uniform in the sense that the structure on tensor products of objects is given by tensor products of the structure, see [21] for more details.

The morphisms of the free hypergraph category over a polygraph are in bijection with multi-pointed hypergraphs in the sense of Habel [25, Definition 1.3],

that is, hypergraphs with “open” source and target boundaries [5]. Intuitively, string diagrams in a hypergraph category are hypergraphs.

F Details for Section 3

Proposition 3.1. *For every non-deterministic monoidal automaton there is a regular monoidal grammar with the same language, and vice-versa.*

Proof (Proof sketch). From the transitions $\Delta_\gamma \subseteq Q^n \times Q^m$ of a monoidal automaton, we can build a polygraph \mathbb{Q} by taking a generator $\gamma_i : q_1 \otimes \dots \otimes q_n \rightarrow q'_1 \otimes \dots \otimes q'_m$ for each $((q_1, \dots, q_n), (q'_1, \dots, q'_m)) \in \Delta_\gamma$. The morphism of polygraphs $\psi : \mathbb{Q} \rightarrow \Gamma$ simply maps γ_i to γ . The reverse is analogous.

Definition F.1. *Given a non-deterministic monoidal automaton over a polygraph Γ , we inductively define transition functions $\hat{\delta}_{n,m} : Q^n \times \mathcal{F}_\otimes \Gamma(n, m) \rightarrow \mathcal{P}(Q^m)$ over string diagrams in the free monoidal category over Γ with arity n and coarity m as follows:*

- for a generator $\gamma \in \Gamma$, $\hat{\delta}_{n,m}(q, \gamma) := \delta_{n,m}(\gamma)$,
- for identities, $\hat{\delta}_{n,n}(q, \text{id}) := \{q\}$,
- for a tensor product $s_1 \otimes s_2$, where $s_1 : n_1 \rightarrow m_1, s_2 : n_2 \rightarrow m_2$ with $n = n_1 + n_2, m = m_1 + m_2$ and $q = q_1 ++ q_2$, $\hat{\delta}_{n,m}(q, s_1 \otimes s_2) := \{p ++ p' \mid p \in \hat{\delta}_{n_1, m_1}(q_1, s_1), p' \in \hat{\delta}_{n_2, m_2}(q_2, s_2)\}$,
- for a composite $s; s'$, where $s : n \rightarrow p, s' : p \rightarrow m$, $\hat{\delta}_{n,m}(q, s; s') := \{\hat{\delta}_{p,m}(q', s') \mid q' \in \hat{\delta}_{n,m}(q, s)\}$.

G Proofs omitted from Section 4

Proposition 4.1. *The multicategory of derivable sequents in the theory of diagram contexts is symmetric. In logical terms, exchange is admissible in the theory of diagram contexts: whenever we can prove that a diagram context exists under certain context Γ , we can prove that it exists under a permutation of Γ .*

Proof. Assume we derived a term $\Gamma, \boxed{u}, \boxed{v}, \Delta \vdash t$; let us show we could also derive $\Gamma, \boxed{v}, \boxed{u}, \Delta \vdash t$. We proceed by structural induction, recurring to the first term where the variables \boxed{u} and \boxed{v} appeared at two sides of the rule: this rule must have been of the form $t_1 \ddagger t_2$ or $t_1 \otimes t_2$ for $\Gamma_1, \boxed{u}, \Delta_1 \vdash t_1$ and $\Gamma_2, \boxed{v}, \Delta_2 \vdash t_2$, where $\Gamma \in \text{Shuf}(\Gamma_1; \Gamma_2)$ and $\Delta \in \text{Shuf}(\Delta_1; \Delta_2)$. In that case, we can deduce that $\Gamma, \boxed{v}, \boxed{u}, \Delta \in \text{Shuf}(\Gamma_1, \boxed{u}, \Delta_1; \Gamma_2, \boxed{v}, \Delta_2)$; as a consequence, $\Gamma, \boxed{v}, \boxed{u}, \Delta \vdash t$ can be derived.

Proposition 4.2. *Derivable sequents in the theory of diagram contexts over a polygraph \mathcal{P} form the free strict monoidal category over the polygraph extended with special “hole” generators, $\mathcal{P} + \{h_{A,B} : A \rightarrow B \mid A, B \in \mathcal{P}_{obj}^*\}$. Derivable*

sequents over the empty context form the free strict monoidal category over the polygraph \mathcal{P} . Moreover, there exists a symmetric multifunctor

$$i: |\mathcal{F}_\otimes \mathcal{P}| \rightarrow |\mathcal{P}|$$

interpreting each monoidal term as its derivable sequent.

Proof. We proceed by structural induction. We first note that the three nullary rules of the logic correspond to terms of the free strict monoidal category over the polygraph $\mathcal{P} + \{h_{A,B} \mid A, B \in \mathcal{P}_{obj}^*\}$. The first corresponds to identities, the second corresponds to generators, and the third, when employed with types A and B , corresponds to the additional generator $h_{A,B}$. We then note that the two binary rules correspond to sequential and parallel composition, thus obtaining the classical algebraic theory of monoidal terms over the polygraph $\mathcal{P} + \{h_{A,B} \mid A, B \in \mathcal{P}_{obj}^*\}$.

Quotienting over the equations of monoidal categories, as we do when we impose the equations of the theory of diagram contexts, recovers the free strict monoidal category: in a tautological sense, the free strict monoidal category is precisely the one generated by the operations of a monoidal category quotiented by the axioms of a monoidal category. This contrasts sharply with a much more interesting description of the free strict monoidal category: that using string diagrams. As both are exhibited as satisfying the same universal property, they are necessarily equivalent.

As a particular case, a derivable sequent over the empty context must, by structural induction, avoid any use of the holes. As a consequence of the previous reasoning, it is generated from the polygraph \mathcal{P} and it must be a morphism of the free strict monoidal category.

Finally, the symmetric multifunctor can be described by structural induction: it preserves identities, holes, sequential and parallel compositions, and it sends each monoidal term with no holes $h \in |\mathcal{F}_\otimes \mathcal{P}|$ to its derivation under the empty context, $h \in |\mathcal{P}|$.

H Details from Section 5

Proposition H.1. *The following square of adjunctions commutes.*

$$\begin{array}{ccc}
 \text{PolyGraph} & \begin{array}{c} \xleftarrow{C} \\ \perp \\ \xrightarrow{\text{ROpt}} \end{array} & \text{MultiGraph} \\
 \mathcal{F}_\otimes \downarrow \dashv \uparrow U & & \mathcal{F}_\vee \downarrow \dashv \uparrow U \\
 \text{MonCat} & \begin{array}{c} \xleftarrow{C} \\ \perp \\ \xrightarrow{\text{ROpt}} \end{array} & \text{MultiCat}
 \end{array}$$

Proof. This follows by unwinding definitions.

I Proofs omitted from Section 6

Lemma 6.1. *Any morphism of symmetric multigraphs underlying a context-free monoidal grammar, $\phi : G \rightarrow \boxed{\mathbb{C}}$, factors (non-uniquely) through the quotienting of raw optics (Proposition 5.1); meaning that there exists some multigraph G' satisfying $G = \text{clique}(G')$, and some morphism $\phi_r : G' \rightarrow |\text{ROpt}[\mathbb{C}]|$, such that $\phi = \text{clique}(\phi_r) \circledast q^*$.*

Proof. This is a consequence of the fact that q^* is full. Given any diagram context, we argue that we can obtain a (non-unique) diagram context of the form of a raw optic

$$t_1 \circledast (\text{id}_{M_1} \otimes \boxed{x_1} \otimes \text{id}_{N_1}) \circledast t_2 \circledast (\text{id}_{M_2} \otimes \boxed{x_2} \otimes \text{id}_{N_2}) \circledast \dots \circledast (\text{id}_{M_n} \otimes \boxed{x_n} \otimes \text{id}_{N_n}) \circledast t_{n+1}.$$

Indeed, by structural induction, if the diagram is formed by a hole or a generator, it can be put in raw optic form by adding identities; if the diagram is a composition, we can put both factors in raw optic form and check that their composition is again in raw optic form; if the diagram is a tensoring of two diagrams in raw optic form, we can always apply the interchange law and note that whiskering a raw optic by an object returns again a raw optic.

It is the case that every map $G \rightarrow \text{clique}(H)$ arises as a map $G' \rightarrow H$ for some multigraph G' such that $G = \text{clique}(G')$. Combining both facts, we obtain the desired result.

Lemma 6.2. *Let $\mathcal{G} = (\phi, S)$ be a context-free monoidal grammar. Then the language of any raw representative ϕ_r of ϕ (with start symbol S) equals the language of \mathcal{G} . That is, $\phi_r[\mathcal{F}_\nabla G'(; S)] = \phi[\mathcal{F}_\nabla G(; S)]$.*

Proof (Proof sketch). A raw representative amounts to choosing a specific ordering of the holes and generators in a diagram context. By definition (Lemma 6.1), these quotient to the original diagram contexts. In particular, closed derivations quotient to the same element of \mathbb{C} .

Lemma 6.3. *A raw representative $\phi_r : G' \rightarrow |\text{ROpt}[\mathbb{C}]|$ uniquely determines a strict monoidal functor $I_\phi : \mathcal{F}_\otimes(\mathcal{C}G') \rightarrow \mathbb{C}$.*

Proof. Using the free-forgetful adjunction, the raw representative, ϕ_r , determines a unique multifunctor $\mathcal{F}_\nabla G' \rightarrow \text{ROpt}[\mathbb{C}]$. Using the adjunction of Theorem 5.1, this in turn determines a unique monoidal functor $\mathcal{C}(\mathcal{F}_\nabla G') \rightarrow \mathbb{C}$. Finally, using the commutativity of \mathcal{C} with \mathcal{F}_∇ (Proposition H.1), we obtain $I_\phi : \mathcal{F}_\otimes(\mathcal{C}G') \rightarrow \mathbb{C}$. Explicitly, the action of I_ϕ on generators is given by: $A^L \mapsto \pi_1(\phi_r(A))$, $A^R \mapsto \pi_2(\phi_r(A))$, $(f, i) \mapsto \pi_i(\phi_r(f))$, where π are projections.

Lemma 6.4. *Given a multigraph G , there is a bijection between derivations rooted at a sort S and optical contours from S^L to S^R , i.e. $\mathcal{F}_\nabla G(; S) \cong \mathcal{F}_\otimes(\mathcal{C}G)(S^L; S^R)$.*

Proof. Let $d \in \mathcal{F}_\nabla G(; S)$ be a derivation. We define a family of functions $\{C_X : \mathcal{F}_\nabla G(; X) \rightarrow \mathcal{F}_\otimes(\mathcal{C}G)(X^L, X^R)\}_{X \in G}$ by structural recursion. There are two

cases: if d is a generating operation $d \in G(; S)$, then $C_S(d) := (d, 0) : S^L \rightarrow S^R$. Otherwise, d is a composite $(p_1, \dots, p_n) \circledast g$ where $g \in G(X_1, \dots, X_n; S)$ is a generating operation and $p_i \in \mathcal{F}_\nabla G(; X_i)$, in which case $C_S(d) := (g, 0) \circledast C_{X_1}(p_1) \circledast (g, 1) \circledast \dots \circledast C_{X_n}(p_n) \circledast (g, n)$.

We define functions C_S^{-1} right to left in a similar fashion. Let $c \in \mathcal{F}_\otimes(\mathcal{C}G)(S^L; S^R)$ be an optical contour. If $c = (c', 0)$ is a generating sector then $C_S^{-1}(c) := c'$. Otherwise c is a composite $((g, 0) : S^L \rightarrow M_1 \otimes X_1^L \otimes N_1) \circledast c_1 \circledast ((g, 1) : M_1 \otimes X_1^R \otimes N_1 \rightarrow M_2 \otimes X_2^L \otimes N_2) \circledast \dots \circledast c_n \circledast ((g, n) : M_1 \otimes X_1^R \otimes N_1 \rightarrow S^R)$ where (g, i) are generating sectors and $c_i \in \mathcal{F}_\otimes(\mathcal{C}G)(X_i^L, X_i^R)$, in which case $C_S^{-1}(c) := (C_{X_1}^{-1}(c_1), \dots, C_{X_n}^{-1}(c_n)) \circledast g$. It is clear that these functions are mutually inverse and hence form a bijection.

Theorem 6.1. *The language of a context-free monoidal grammar $\mathcal{G} = (\phi : G \rightarrow \mathbb{C}, S)$ equals the image of a regular representative under the monoidal functor I_ϕ of Lemma 6.3.*

Proof. By Lemma 6.2, the languages $L(\mathcal{G})$ and $L((\phi_r, S))$ are equal for any raw representative ϕ_r of ϕ , where $L((\phi_r, S)) = \phi_r[\mathcal{F}_\nabla G'(; S)]$. It therefore suffices to show that $\phi_r[\mathcal{F}_\nabla G'(; S)] = I_\phi[\mathcal{F}_\otimes(\mathcal{C}G')(S^L; S^R)]$. We show the inclusion left to right. Let $d \in \mathcal{F}_\nabla G'(; S)$ be a derivation, and let $C_S(d)$ be the corresponding optical contour given by Lemma 6.4. Then by the definition of I_ϕ (Lemma 6.3) and C_S , we have $I_\phi(C_S(d)) = \phi_r(d)$. We show the inclusion right to left. Let $g \in \mathcal{F}_\otimes(\mathcal{C}G')(S^L; S^R)$ be a contour from S^L to S^R , and let $C_S^{-1}(g)$ be the corresponding derivation given by Lemma 6.4. Then just as before we have $\phi_r(C_S^{-1}(g)) = I_\phi(g)$.

Rights statement: For the purpose of Open Access the Author has applied a Creative Commons-BY public copyright license to any Author Accepted Manuscript version arising from this submission.

Curriculum Vitae

Personal Data

- Name: Matthew David Earnshaw
- 27th September 1992
- Nationality: British

Contact details

- Email: matt@earnshaw.org.uk

Employment History

- 2020-2025 - Tallinn University of Technology, Department of Software Science, Early Stage Researcher
- 2016-2020 - Senior Software Developer, Silver Lab, Department of Neuroscience, University College London

Defended theses

- MSci, 2015, supervisor Mitesh Patel, *Angular analysis of the $B^0 \rightarrow K^{*0} \mu^+ \mu^-$ decay at LHCb*, Imperial College London

Education

- 2020-2025 - Tallinn University of Technology, Computer Science (PhD studies)
- 2011-2015 - Imperial College London, Physics (MSci)

Scientific activities

Papers

- Matthew Earnshaw and Paweł Sobociński. “Regular Monoidal Languages”. In: *47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022)*. Ed. by Stefan Szeider, Robert Ganian and Alexandra Silva. Vol. 241. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022, 44:1–44:14. ISBN: 978-3-95977-256-3. DOI: 10.4230/LIPIcs.MFCS.2022.44
- Matthew Earnshaw and Paweł Sobociński. “String Diagrammatic Trace Theory”. In: *48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023)*. Ed. by Jérôme Leroux, Sylvain Lombardy

and David Peleg. Vol. 272. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023, 43:1–43:15. ISBN: 978-3-95977-292-1. DOI: 10.4230/LIPIcs.MFCS.2023.43

- Matthew Earnshaw and Paweł Sobociński. “Regular planar monoidal languages”. In: *Journal of Logical and Algebraic Methods in Programming* 139 (2024), p. 100963. ISSN: 2352-2208. DOI: <https://doi.org/10.1016/j.jlamp.2024.100963>
- Matt Earnshaw, James Hefford and Mario Román. “The Produoidal Algebra of Process Decomposition”. In: *32nd EACSL Annual Conference on Computer Science Logic (CSL 2024)*. Ed. by Aniello Murano and Alexandra Silva. Vol. 288. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024, 10:1–10:18. ISBN: 978-3-95977-264-8. DOI: 10.4230/LIPIcs.CSL.2023.10
- Matthew Earnshaw and Mario Román. “Context-Free Languages of String Diagrams”. In: *28th International Conference on Foundations of Software Science and Computation Structures*. To appear. 2025

Presentations

- In December 2024 at the *University of Tartu* Computer Science Seminar, I spoke about monoidal automata and string diagrams for effectful categories.
- In July 2024 at the *Structure Meets Power* workshop (Tallinn), I spoke about the preprint *Context-Free Languages of String Diagrams*.
- In June 2024 at the *Applied Category Theory* conference (Oxford), I spoke about the produoidal structure of optics over a monoidal category.
- In March 2024 at the *TSEM Seminar* (Tallinn), I spoke about monoidal automata and string diagrams for effectful categories.
- In February 2024 at the *Estonian-Latvian Theory Days* (Randivaljä), I spoke about the preprint *Context-Free Languages of String Diagrams*.
- In November 2023 at the *Nordic Workshop on Programming Theory* (Västerås), I spoke about presentations of premonoidal categories by devices.
- In October 2023 at the online *ipomset project seminar*, I spoke about monoidal languages.
- In September 2023 at the *Mathematical Foundations of Computer Science* conference (Bordeaux), I spoke about the paper *String-Diagrammatic Trace Theory*.

- In July 2023 at the *Bob Walters Memorial Meeting* (Tallinn), I spoke about enrichment in bicategories.
- In March 2023 at the *Lawvere Memorial Meeting* (Tallinn), I spoke about natural structure and the dialectic of concept formation.
- In November 2022 at the *Nordic Workshop on Programming Theory* workshop (Bergen), I spoke about the paper *Regular Monoidal Languages*.
- In September 2022 at the *SYCO 9* workshop (Como), I spoke about the paper *Regular Monoidal Languages*.
- In August 2022 at the *Mathematical Foundations of Computer Science* conference (Vienna), I spoke about the paper *Regular Monoidal Languages*.

Elulookirjeldus

Isikuandmed

- Nimi: Matthew David Earnshaw
- Sünniaeg: 27. september 1992
- Sünnikoht: Hertfordshire, UK
- Rahvus: Inglise

Kontaktandmed

- E-post: matt@earnshaw.org.uk

Teenistuskäik

- 2020-2025 - Tallinna Tehnikaülikool, Tarkvarateaduse instituut, doktorant-nooremteadur
- 2016-2020 - Vanem-tarkvaraarendaja, Silver Lab, Neuroteaduste osakond, University College London

Kaitstud lõputööd

- Magistrikraad, 2015, (juh) Mitesh Patel, *Angular analysis of the $B^0 \rightarrow K^{*0} \mu^+ \mu^-$ decay at LHCb*, Imperial College London

Hariduskäik

- 2020-2025 - Tallinna Tehnikaülikool, Tarkvarateadus (doktoriõpe)
- 2011-2015 - Imperial College London, Füüsika (MSci)

Teadustegevus

Teadusartiklite ja konverentsiteeside loetelu on toodud ingliskeelse elulookirjelduse juures.

ISSN 2585-6901 (PDF)
ISBN 978-9916-80-243-4 (PDF)