TU Delft

Delft University of Technology

Displayed Monoidal Categories for the Semantics of Linear Logic

Ahrens, Benedikt; Matthes, Ralph; Van Der Weide, Niels; Wullaert, Kobe

DOI
10.1145/3636501.3636956

Publication date
2024

Document Version
Final published version

Published in
CPP 2024 - Proceedings of the 13th ACM SIGPLAN International Conference on Certified Programs and Proofs

Citation (APA)
Ahrens, B., Matthes, R., Van Der Weide, N., & Wullaert, K. (2024). Displayed Monoidal Categories for the Semantics of Linear Logic. In A. Timany, D. Traytel, B. Pientka, & S. Blazy (Eds.), *CPP 2024 - Proceedings of the 13th ACM SIGPLAN International Conference on Certified Programs and Proofs* (pp. 260-273). Association for Computing Machinery (ACM). https://doi.org/10.1145/3636501.3636956

Important note
To cite this publication, please use the final published version (if applicable).
Please check the document version above.

Copyright
Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

Takedown policy
Please contact us and provide details if you believe this document breaches copyrights.
We will remove access to the work immediately and investigate your claim.

This work is downloaded from Delft University of Technology.
For technical reasons the number of authors shown on this cover page is limited to a maximum of 10.

Displayed Monoidal Categories for the Semantics of Linear Logic

Benedikt Ahrens
Delft University of Technology
Netherlands
University of Birmingham
UK
b.p.ahrens@tudelft.nl

Ralph Matthes
IRIT - Université de Toulouse - CNRS - Toulouse INP - UT3
France
ralph.matthes@irit.fr

Niels van der Weide
Radboud University Nijmegen
Netherlands
nweide@cs.ru.nl

Kobe Wullaert
Delft University of Technology
Netherlands
K.F.Wullaert@tudelft.nl

Abstract

We present a formalization of different categorical structures used to interpret linear logic. Our formalization takes place in UniMath, a library of univalent mathematics based on the Coq proof assistant.

All the categorical structures we formalize are based on monoidal categories. As such, one of our contributions is a practical, usable library of formalized results on monoidal categories. Monoidal categories carry a lot of structure, and instances of monoidal categories are often built from complicated mathematical objects. This can cause challenges of scalability, regarding both the vast amount of data to be managed by the user of the library, as well as the time the proof assistant spends on checking code. To enable scalability, and to avoid duplication of computer code in the formalization, we develop "displayed monoidal categories". These gadgets allow for the modular construction of complicated monoidal categories by building them in layers; we demonstrate their use in many examples. Specifically, we define linear-nonlinear categories and construct instances of them via Lafont categories and linear categories.

CCS Concepts: • **Theory of computation → Linear logic**; **Categorical semantics**.

Keywords: linear logic, categorical semantics, monoidal categories, Coq, UniMath

This work is licensed under a Creative Commons Attribution 4.0 International License.

CPP '24, January 15–16, 2024, London, UK

© 2024 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-0488-8/24/01
https://doi.org/10.1145/3636501.3636956

ACM Reference Format:
Benedikt Ahrens, Ralph Matthes, Niels van der Weide, and Kobe Wullaert. 2024. Displayed Monoidal Categories for the Semantics of Linear Logic. In *Proceedings of the 13th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP '24), January 15–16, 2024, London, UK*. ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3636501.3636956

1 Introduction

Linear logic [16] has been an active field of research with a wide variety of applications. Intuitively, linear logic is a form of logic where we see our hypotheses as *resources*, meaning that we must use every assumption exactly once.[1] Among the applications are separation logic [29] and type systems with uniqueness types [6].

To actually apply linear logic, denotational semantics is key. An interpretation using ω-cpos allows one to apply linear logic to programming languages [32]. Linear logic can also be interpreted using game semantics [1, 10] (for instance). Implicit complexity is another application of linear logic [22]. For the purpose of denotational semantics, many different categorical structures for the interpretation of linear logic have been developed, and the relationship between them has been studied, e.g., by De Paiva [14] and Melliès [26, 27].

Monoidal categories play a key role in the categorical semantics of linear logic. To formally develop the categorical semantics of linear logic, one thus needs to develop the theory of monoidal categories in a proof assistant. However, this goal comes with numerous challenges, and one of them lies in constructing examples of monoidal categories. Many categories that one meets in the semantics of linear logic, have rather complicated structure on the objects.

As mentioned before, interesting instances can be drawn from order theory (like ω-cpos), but the abstract notions build a hierarchy of category-theoretic concepts, starting

[1] An "unconsumed" resource has to be managed, and this also counts as "use" in this picture.

with just categories, but extending them to monoidal categories and then refining those into braided, symmetric, and closed monoidal categories, respectively. We are also studying comonoids in those monoidal categories and consider comonads whose coalgebras are again organized into (symmetric) monoidal categories.

A successful formalization of this wealth of variants of concepts needs to avoid copying code. Refinements of a notion have to build on the given notion and only deal with the difference, be it extra data or extra laws. We will follow a layered approach throughout. As an easy example, a monoidal category is obtained by adding to a given category C a monoidal structure M for it. The *notion* of the monoidal structure is used in subsequent notion building. In our hierarchies, we often build a (more complex) category out of the material of a given category C, by adding structure and/or laws to the objects and/or morphisms of C, one of the easiest cases being the notion of a full subcategory by imposing only a condition on the objects. The precise terminology is that of a *displayed category* [3]: it is a description D of data and laws from which the *total category* $\int \mathrm{D}$ is obtained generically, together with a functor back into C, which is why D is called a displayed category *over* C (cf. the leftmost part of Fig. 1).

In the same spirit, there is the notion of displayed bicategory [2] that will not be used in this paper. Let us just say that the bicategory of categories can be refined into a total bicategory of monoidal categories by adding the monoidal structures M to the objects C of that bicategory and adapting the 1-cells and 2-cells accordingly to reflect having monoidal functors and monoidal natural transformations, respectively. The displayed approach capitalizes on the bicategory of categories, and only the added monoidal structure needs verification.

As an original contribution for this paper, we are proposing the notion of a *displayed monoidal category* (in Definition 4.5) that achieves a very different kind of modularization. This time, it is about *building* monoidal structures in layers. Put simply, it integrates the *horizontal extension* of a category C by a monoidal structure M and the *vertical extension* given by a displayed category D over C. It is the description DM in terms of D and M of what has to be added to $\int \mathrm{D}$ to turn it into a monoidal category that "builds on the material" in M, cf. the three right-hand side quarters of Fig. 1. In fact, there is a generic construction (Construction 4.7) of a *total monoidal structure* $\int DM$ from DM, serving as monoidal structure for $\int \mathrm{D}$ (and there is a strict monoidal functor from the obtained monoidal category back into the one on C). This offers a modularization of our constructions since the generic construction is done once and for all and thus profits from M being a monoidal structure for C and D being a displayed category over C. The construction and verification effort is thus limited to the extras described by D, and this only for the monoidal operations.

We apply displayed monoidal categories to the study of the semantics of linear logic. The first categorical model of linear logic is given by Lafont categories [21]. The category of comonoids is key to Lafont categories. However, this notion of model comes with a serious limitation: Girard's key model of coherence spaces is not an instance of a Lafont category. Linear categories [7] overcome this limitation. Instead of looking at the category of comonoids, one looks at the Eilenberg-Moore category of a comonad. Both the monoidal category of comonoids and the (monoidal) Eilenberg-Moore categories can nicely be defined using displayed monoidal categories.

The main notion of model for linear logic used in this paper is given by linear-non linear models [6]. Both Lafont categories and linear categories give rise to such models, and we use this to construct concrete models of linear logic. The leading example of a Lafont category in this paper is the *relational model*, and the monoidal category of pointed posets together with the lifting comonad gives an example of a linear category.

Contributions and Overview. In this paper, we discuss our formalization of monoidal categories and linear logic.

- We start in Section 2 with a formalization of the notion of monoidal category and important examples.
- In Section 3, we formalize the notion of linear-non linear model (Definition 3.5).
- The main technical contribution of this paper, the notion of displayed monoidal category, is introduced in Section 4 (Definition 4.5), and we illustrate this notion with numerous examples. Examples of particular importance to this paper are the Eilenberg-Moore category of a comonad (Example 4.13) and the category of comonoids (Example 4.14).
- Next we formalize a characterization of cartesian monoidal categories in Section 5 (Corollary 5.2).
- Finally, we give concrete models of linear logic in Section 6 (Example 6.5 and Example 6.9).

We discuss related work in Section 7, and we conclude in Section 9.

1.1 Formalization

Our formalization takes place in univalent foundations, specifically, in the library `UniMath` [38] built for the proof assistant Coq [35]. Many constructions and results are independent of univalent foundations and, in particular, of the univalence axiom. The displayed machinery, however, relies crucially on dependent types. In the remainder of this section, we give an overview of the prerequisites from univalent foundations in general and `UniMath` specifically that we use in our formalization.

We write $a = b$ for the type of identifications / equalities / paths from a to b. We do not rely on any inductive types other than the ones specified in the prelude of `UniMath`, such

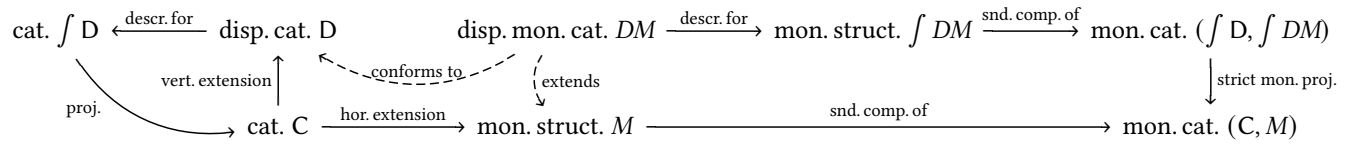


Figure 1. Overview of the concept of displayed monoidal category with parameters C, M and D

as identity types, sum types, natural numbers, and booleans. Indeed, part of the work consists of the construction of certain initial algebras from dependent products, dependent sums, identities, and natural numbers. We use the notions of propositions and sets of Univalent Foundations: a type X is a proposition if $\prod_{x,y:X} x = y$ is inhabited, and a set if the type $x = y$ is a proposition for all $x, y : X$. Hence, despite working in Coq, we do not rely on the universes Prop or Set.

Our main constructions and results, while conveniently expressed using univalent foundations, are not dependent on the full univalence axiom. On top of Martin-Löf type theory, we rely on propositional truncation (for the composition of relations, see Example 2.6) and functional extensionality (for instance, when comparing morphisms in the category of partially ordered sets in Example 2.5).

Propositional truncation in `UniMath` is implemented via an impredicative encoding [4] (for instance, giving details). Such an impredicative encoding requires a form of propositional resizing to avoid raising the universe level of the truncation compared to that of the original type. In order to implement resizing rules (not otherwise available in Coq), `UniMath` uses type-in-type, and hence is in principle inconsistent. We are careful not to use any consequences of type-in-type except for the resulting resizing rules. For the purposes of this paper, a resizing *axiom* as implemented, for instance, in the Coq-HoTT library[2] would work just as well — propositional truncation is only used in the example on relations.

We assume the reader to be familiar with the concepts of category theory as found in the textbook by Mac Lane [23]. A *category* C in `UniMath` is given by a type of objects C_0 and a family of sets $\mathrm{C}(x, y)$ for any $x, y : \mathrm{C}_0$. Following the choice adopted for the `UniMath` library, $x : \mathrm{C}_0$ is abbreviated $x : \mathrm{C}$, $f : \mathrm{C}(x, y)$ is abbreviated $f : x \to y$, and composition is written in "diagrammatic" order, i. e., the composite of $f : x \to y$ and $g : y \to z$ is denoted $f \cdot g : x \to z$.

The formalization described in this paper has been integrated into `UniMath`, in commit 9e43b0d. Throughout the paper, we are describing the state of the library at that commit, even though the development of `UniMath` and the material discussed here continues. We have put numerous links (in blue colour) to an HTML version of `UniMath` at commit 9e43b0d, hosted on Gitlab. Proof-checking and creation of the HTML documentation can easily be reproduced at home by following the `UniMath` compilation instructions.

[2]See the file implementing this axiom at https://github.com/HoTT/Coq-HoTT/blob/27cf62bd61cb5528b800949f6f995955ab5f33fd/theories/PropResizing/PropResizing.v.

2 Monoidal Categories

In this section, we recall the notion of *monoidal categories*, and several variations of this notion. We also discuss numerous examples of monoidal categories.

Definition 2.1 (monoidal)**.** A **monoidal structure** for a category V consists of:

1. an object $\mathsf{I}_{\mathrm{V}} : \mathrm{V}$, the **unit**;
2. for all objects $x, y : \mathrm{V}$, an object $x \otimes y : \mathrm{V}$, the **tensor of x and y**;
3. for all objects $x : \mathrm{V}$ and morphisms $g : y_1 \to y_2$, a morphism $x \triangleleft g : x \otimes y_1 \to x \otimes y_2$, the **left whiskering with x of g**;
4. for all objects $y : \mathrm{V}$ and morphisms $f : x_1 \to x_2$, a morphism $f \triangleright y : x_1 \otimes y \to x_2 \otimes y$, the **right whiskering with y of f**;
5. for all objects $x : \mathrm{V}$, a morphism $\lambda_x : \mathsf{I}_{\mathrm{V}} \otimes x \to x$ (the family of those morphisms is the **left unitor**) and a morphism $\rho_x : x \otimes \mathsf{I}_{\mathrm{V}} \to x$ (the family of those morphisms is the **right unitor**);
6. for all objects $x, y, z : \mathrm{V}$, a morphism $\alpha_{x,y,z} : (x \otimes y) \otimes z \to x \otimes (y \otimes z)$ (the family of those morphisms is the **associator**);
7. for the same parameters as above, chosen inverses for the unitors and the associator: $\lambda_x^{-1} : x \to \mathsf{I}_{\mathrm{V}} \otimes x$, $\rho_x^{-1} : x \to x \otimes \mathsf{I}_{\mathrm{V}}$ and $\alpha_{x,y,z}^{-1} : x \otimes (y \otimes z) \to (x \otimes y) \otimes z$,

such that the following equations hold (where we omit the index V to the unit), where we first list the laws that concern only the tensor and the whiskerings (that replace the requirement that tensor is a bifunctor in traditional presentations):

8. $x \triangleleft 1_y = 1_{x \otimes y}$;
9. $1_x \triangleright y = 1_{x \otimes y}$;
10. $x \triangleleft (g_1 \cdot g_2) = (x \triangleleft g_1) \cdot (x \triangleleft g_2)$;
11. $(f_1 \cdot f_2) \triangleright y = (f_1 \triangleright y) \cdot (f_2 \triangleright y)$;
12. $(f \triangleright y_1) \cdot (x_2 \triangleleft g) = (x_1 \triangleleft g) \cdot (f \triangleright y_2)$,

followed by the axioms governing the unitors and the associator:

13. for all $f : x \to y$, $(\mathsf{I} \triangleleft f) \cdot \lambda_y = \lambda_x \cdot f$ (naturality of left unitor);
14. $\lambda_x \cdot \lambda_x^{-1} = 1_{\mathsf{I} \otimes x}$ and $\lambda_x^{-1} \cdot \lambda_x = 1_x$;
15. for all $f : x \to y$, $(f \triangleright \mathsf{I}) \cdot \rho_y = \rho_x \cdot f$ (naturality of right unitor);

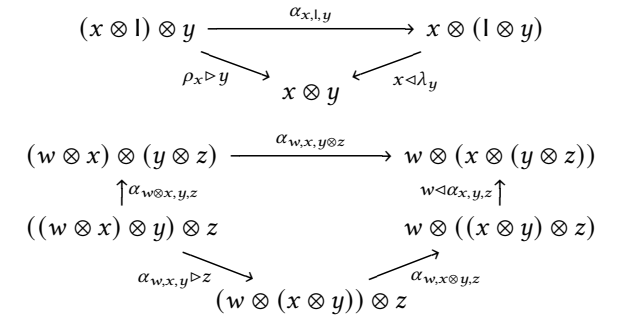


Figure 2. Triangle and pentagon laws of monoidal category in whiskered presentation

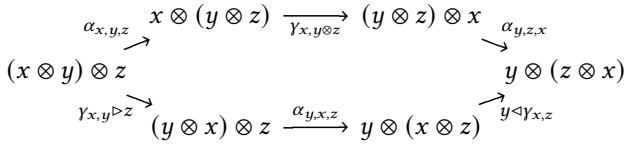


Figure 3. First hexagon law of symmetric monoidal category

16. $\rho_x \cdot \rho_x^{-1} = 1_{x\otimes \mathsf{I}}$ and $\rho_x^{-1} \cdot \rho_x = 1_x$;
17. for all $h : z \to z'$, $\alpha_{x,y,z} \cdot (x \triangleleft (y \triangleleft h)) = ((x \otimes y) \triangleleft h) \cdot \alpha_{x,y,z'}$, for all $f : x \to x'$, $\alpha_{x,y,z} \cdot (f \triangleright (y \otimes z)) = ((f \triangleright y) \triangleright z) \cdot \alpha_{x',y,z}$, for all $g : y \to y'$, $\alpha_{x,y,z} \cdot (x \triangleleft (g \triangleright z)) = ((x \triangleleft g) \triangleright z) \cdot \alpha_{x,y',z}$ (naturality of associator in each of the arguments);
18. $\alpha_{x,y,z} \cdot \alpha^{-1}_{x,y,z} = 1_{(x\otimes y)\otimes z}$ and $\alpha^{-1}_{x,y,z} \cdot \alpha_{x,y,z} = 1_{x\otimes(y\otimes z)}$;
19. $\alpha_{x,\mathsf{I},y} \cdot (x \triangleleft \lambda_y) = \rho_x \triangleright y$ (triangle law); and
20. $(\alpha_{w,x,y} \triangleright z) \cdot \alpha_{w,x\otimes y,z} \cdot (w \triangleleft \alpha_{x,y,z}) = \alpha_{w\otimes x,y,z} \cdot \alpha_{w,x,y\otimes z}$ (pentagon law).

For the latter two laws, see Fig. 2 (a notational variant of the standard textbook definition).

For morphisms $f : x_1 \to x_2$ and $g : y_1 \to y_2$, we define $f \otimes g$ to be $x_1 \triangleleft g \cdot f \triangleright y_2 : x_1 \otimes y_1 \to x_2 \otimes y_2$. A **monoidal category** is a pair (V, M) of a category V and a monoidal structure M for V. We follow common practice and use V also for the monoidal category.

A **symmetric monoidal category** V consists of

1. a monoidal category V;
2. for all objects x and y, a morphism $\gamma_{x,y} : x \otimes y \to y \otimes x$,

such that γ is natural in both arguments, that $\gamma_{x,y}$ and $\gamma_{y,x}$ are inverses to each other, and that the **first hexagon law** holds, that is, the diagram in Fig. 3 commutes. It is well-known that in a symmetric monoidal category, the *second hexagon law* (not shown here) is then derivable (as well as laws concerning the associator and the unitors, for example, $\gamma_{x,\mathsf{I}} \cdot \lambda_x = \rho_x$). A **symmetric monoidal closed category** V consists of a symmetric monoidal category V such that for every $x : \mathsf{V}$, the functor that sends objects $y : \mathsf{V}$ to $x \otimes y$ has a right adjoint.

Remark 2.2. In Definition 2.1 we use a *whiskered style* for presenting monoidal categories: for morphisms, we have a left- and a right-whiskering operation (from which $f \otimes g$ is derived above). This is used "behind the scenes" in Section 4 to make the definition of a displayed monoidal category easier to handle in the implementation.

In the remainder of this section, we look at numerous examples of monoidal categories.

Example 2.3 (cartesian_monoidal)**.** Suppose that C is a category with (chosen) finite products. Then C has a monoidal structure where the tensor operation is given by the binary product and the unit is given by the terminal object.

The monoidal category described in Example 2.3 is of a special kind: it is a *cartesian* monoidal category.

Definition 2.4 (is_cartesian)**.** Let V be a monoidal category. We say that V is **semi-cartesian** if the unit I is a terminal object. Note that whenever V is semi-cartesian, for all objects $x, y : \mathsf{V}$ we have a map $\pi_1 : x \otimes y \to x$ defined as the following composition.

$$x \otimes y \xrightarrow{x \triangleleft t} x \otimes \mathsf{I} \xrightarrow{\rho_x} x$$

Here t is the unique map from y to the object I, which is terminal by assumption. Similarly, we can define a map $\pi_2 : x \otimes y \to y$. We say V is **cartesian** if V is semi-cartesian and if for all $x, y : \mathsf{V}$ the following diagram is a product.

$$x \xleftarrow{\pi_1} x \otimes y \xrightarrow{\pi_2} y$$

However, not every monoidal category is cartesian. One such example is given by the *smash product*.

Example 2.5 (pointed_poset_sym_mon_closed_cat)**.** A **partially ordered set** (poset) is a set P together with a reflexive, anti-symmetric, and transitive relation $\leq_{\mathsf{P}}$. A **pointed poset** is a poset together with an element $\bot_{\mathsf{P}} : \mathsf{P}$ such that for all x we have $\bot_{\mathsf{P}} \leq_{\mathsf{P}} x$. The element $\bot_{\mathsf{P}}$ is called the **bottom element** of P. Note that we have a category Poset_* whose objects are pointed posets, and whose morphisms are monotone maps that preserve the bottom element.

Suppose that we have two pointed posets P_1 and P_2. The smash product of P_1 and P_2 is the quotient of $\mathsf{P}_1 \times \mathsf{P}_2$ where all elements of the shape $(x, \bot_{\mathsf{P}_2})$ and $(\bot_{\mathsf{P}_1}, y)$ are identified. The category Poset_* forms a symmetric monoidal category whose monoidal product is given by the smash product and whose unit is given by the poset of booleans. In addition, Poset_* is monoidal closed: the right adjoint of the smash product is defined by the type of strict monotone functions.

Note that the category of pointed posets has multiple monoidal structures: one can look at the monoidal structure

arising from products (the cartesian one), the monoidal structure arising from coproducts (the cocartesian one), and at the monoidal structure coming from the smash product. The reason why we are interested in the one with the smash product is because we get a nice symmetric monoidal closed category from it.

Example 2.6 (REL_sym_mon_closed_cat)**.** We define the category Rel to be the category whose objects are sets and whose morphisms are relations. The monoidal product in Rel is given by the cartesian product of sets, and this turns Rel into a symmetric monoidal category. The category Rel is also a monoidal closed category: the right adjoint of tensoring is given by the cartesian product of sets as well.

Note that the monoidal product in Rel is taken as the product of sets, and this might suggest that Rel is a cartesian monoidal category. However, this is not the case. In Rel, products and coproducts coincide and they are given by coproducts of sets.

3 Models of Linear Logic

In this paper, we are interested in the use of monoidal categories for the categorical semantics of intuitionistic linear logic. There are three key features to linear logic: the *multiplicative conjunction*, the *linear implication*, and the *bang modality*, where the latter is the *exponential* by which intuitionistic implication can be defined from linear implication. To interpret multiplicative conjunction and the linear implication in categories, we use symmetric monoidal categories. The tensor is used to interpret multiplicative conjunction and the right adjoint of the tensor is used to interpret linear implication.

However, as is well-known, the interpretation of the bang modality is more subtle. There are several formalisms to interpret the bang modality, among which are *linear-non linear models*, *Lafont categories*, and *linear categories*, all of which we formalized in `UniMath`. In this section, we review linear-non linear models [6], which we use as the main framework for the categorical semantics of linear logic. This is because the two others give rise to two families of examples of linear-non linear models, cf. Construction 6.4 and Construction 6.8 in Section 6.

The key notion for defining linear-non linear models is that of *symmetric monoidal adjunction*. In order to define those, we first need to recall some other well-known concepts for monoidal categories that we present in our whiskered format, but which are mathematically equivalent to the textbook notions.

Definition 3.1 (fmonoidal_lax)**.** Let V_1 and V_2 be monoidal categories and let $\mathsf{F} : \mathsf{V}_1 \to \mathsf{V}_2$ be a functor. A **lax monoidal structure** on F consists of

1. a morphism $\mathsf{unit}_\mathsf{F} : \mathsf{I}_{\mathsf{V}_2} \to \mathsf{F}\,\mathsf{I}_{\mathsf{V}_1}$;

$$\begin{array}{ccc} \mathsf{I}_{\mathsf{V}_2} \otimes \mathsf{F}\,x & \xrightarrow{\mathsf{unit}_\mathsf{F} \triangleright \mathsf{F}\,x} & \mathsf{F}\,\mathsf{I}_{\mathsf{V}_1} \otimes \mathsf{F}\,x \\ {\scriptstyle \lambda_{\mathsf{F}\,x}}\downarrow & & \downarrow{\scriptstyle \mathsf{tens}_\mathsf{F}(\mathsf{I}_{\mathsf{V}_1}, x)} \\ \mathsf{F}\,x & \xleftarrow{\mathsf{F}\,\lambda_{\mathsf{F}\,x}} & \mathsf{F}\,(\mathsf{I}_{\mathsf{V}_1} \otimes x) \end{array}$$

$$\begin{array}{ccc} \mathsf{F}\,x \otimes \mathsf{I}_{\mathsf{V}_2} & \xrightarrow{\mathsf{F}\,x \triangleleft \mathsf{unit}_\mathsf{F}} & \mathsf{F}\,x \otimes \mathsf{F}\,\mathsf{I}_{\mathsf{V}_1} \\ {\scriptstyle \rho_{\mathsf{F}\,x}}\downarrow & & \downarrow{\scriptstyle \mathsf{tens}_\mathsf{F}(x, \mathsf{I}_{\mathsf{V}_1})} \\ \mathsf{F}\,x & \xleftarrow{\mathsf{F}\,\rho_{\mathsf{F}\,x}} & \mathsf{F}\,(x \otimes \mathsf{I}_{\mathsf{V}_1}) \end{array}$$

$$\begin{array}{ccc} (\mathsf{F}\,x \otimes \mathsf{F}\,y) \otimes \mathsf{F}\,z & \xrightarrow{\alpha_{\mathsf{F}\,x, \mathsf{F}\,y, \mathsf{F}\,z}} & \mathsf{F}\,x \otimes (\mathsf{F}\,y \otimes \mathsf{F}\,z) \\ {\scriptstyle \mathsf{tens}_\mathsf{F}(x,y) \triangleright \mathsf{F}\,z}\downarrow & & \downarrow{\scriptstyle \mathsf{F}\,x \triangleleft \mathsf{tens}_\mathsf{F}(y,z)} \\ \mathsf{F}\,(x \otimes y) \otimes \mathsf{F}\,z & & \mathsf{F}\,x \otimes \mathsf{F}\,(y \otimes z) \\ {\scriptstyle \mathsf{tens}_\mathsf{F}(x \otimes y, z)}\downarrow & & \downarrow{\scriptstyle \mathsf{tens}_\mathsf{F}(x, y \otimes z)} \\ \mathsf{F}\,((x \otimes y) \otimes z) & \xrightarrow{\mathsf{F}\,\alpha_{x,y,z}} & \mathsf{F}\,(x \otimes (y \otimes z)) \end{array}$$

$$\begin{array}{ccc} \mathsf{F}\,x_1 \otimes \mathsf{F}\,y & \xrightarrow{\mathsf{tens}_\mathsf{F}(x_1, y)} & \mathsf{F}\,(x_1 \otimes y) \\ {\scriptstyle \mathsf{F}\,f \triangleright \mathsf{F}\,y}\downarrow & & \downarrow{\scriptstyle \mathsf{F}\,(f \triangleright y)} \\ \mathsf{F}\,x_2 \otimes \mathsf{F}\,y & \xrightarrow{\mathsf{tens}_\mathsf{F}(x_2, y)} & \mathsf{F}\,(x_2 \otimes y) \end{array}$$

$$\begin{array}{ccc} \mathsf{F}\,x \otimes \mathsf{F}\,y_1 & \xrightarrow{\mathsf{tens}_\mathsf{F}(x, y_1)} & \mathsf{F}\,(x \otimes y_1) \\ {\scriptstyle \mathsf{F}\,x \triangleleft \mathsf{F}\,g}\downarrow & & \downarrow{\scriptstyle \mathsf{F}\,(x \triangleleft g)} \\ \mathsf{F}\,x \otimes \mathsf{F}\,y_2 & \xrightarrow{\mathsf{tens}_\mathsf{F}(x, y_2)} & \mathsf{F}\,(x \otimes y_2) \end{array}$$

Figure 4. Laws governing a lax monoidal functor

2. for all objects $x, y : \mathsf{V}_1$, a morphism $\mathsf{tens}_\mathsf{F}(x, y) : \mathsf{F}\,x \otimes \mathsf{F}\,y \to \mathsf{F}\,(x \otimes y)$;

such that the diagrams in Fig. 4 commute for any x, x_1, x_2, y, y_1, y_2, $z : \mathsf{C}$ (the first two express naturality of tens_F). A **lax monoidal functor** consists of a functor $\mathsf{F} : \mathsf{V}_1 \to \mathsf{V}_2$ together with a lax monoidal structure on it. We usually just write F to indicate a lax monoidal functor. A lax monoidal functor F is called **strong** if unit_F and $\mathsf{tens}_\mathsf{F}(x, y)$ are isomorphisms for all x and y. If they are even dependently equal to identity morphisms, then F is called **strict**.

If V_1 and V_2 are symmetric monoidal categories with symmetric braidings γ^1 and γ^2, then a lax monoidal functor is called **symmetric** if the following diagram commutes:

$$\begin{array}{ccc} \mathsf{F}\,x \otimes \mathsf{F}\,y & \xrightarrow{\gamma^2_{\mathsf{F}\,x, \mathsf{F}\,y}} & \mathsf{F}\,y \otimes \mathsf{F}\,x \\ {\scriptstyle \mathsf{tens}_\mathsf{F}(x,y)}\downarrow & & \downarrow{\scriptstyle \mathsf{tens}_\mathsf{F}(y,x)} \\ \mathsf{F}\,(x \otimes y) & \xrightarrow{\mathsf{F}\,\gamma^1_{x,y}} & \mathsf{F}\,(y \otimes x) \end{array}$$

Note that in the literature on monoidal categories, there also is the notion of oplax monoidal functors. These are dual to lax monoidal functors: we require unit_F and $\mathsf{tens}_\mathsf{F}(x, y)$ to be in the reverse direction. However, in this paper, we are only concerned with lax and strong monoidal functors.

An example of a lax monoidal functor is given by the lift operation on pointed posets. This operation adds a new bottom element to a pointed poset; that is, a new element that is smaller than all the other elements.

Example 3.2 (lax_monoidal_lift_poset_comonad)**.** We have a lax monoidal functor $\mathsf{Lift} : \mathsf{Poset}_* \to \mathsf{Poset}_*$. It sends every poset P to the poset $\mathsf{Lift}(P)$ whose carrier is the type $P + \mathbf{1}$. We denote the added element by $\bot$. The order of $\mathsf{Lift}(P)$ is given by $x \leq_{\mathsf{Lift}(P)} y$ if either $x \leq_P y$ or if $x = \bot$

Note that in Example 3.2 we use pointed posets instead of arbitrary posets. This is actually a meaningful difference: the monoidal structure on the category of pointed posets is given by the smash product, whereas the monoidal structure on the category of posets is given by the cartesian product. Next we recall the notion of *monoidal transformations*.

Definition 3.3 (is_mon_nat_trans)**.** Suppose that we have lax monoidal functors $F, G : V_1 \to V_2$. A **monoidal natural transformation** θ consists of a natural transformation $\theta : F \Rightarrow G$ such that the following diagrams commute:

$$\begin{array}{ccc} & I & \\ {\scriptstyle \mathsf{unit}_F}\swarrow & & \searrow{\scriptstyle \mathsf{unit}_G} \\ F\,I & \xrightarrow{\theta(I)} & G\,I \end{array}$$

$$\begin{array}{ccc} F\,x \otimes F\,y & \xrightarrow{\theta(x)\otimes\theta(y)} & G\,x \otimes G\,y \\ {\scriptstyle \mathsf{tens}_F(x,y)}\downarrow & & \downarrow{\scriptstyle \mathsf{tens}_G(x,y)} \\ F\,(x \otimes y) & \xrightarrow{\theta(x\otimes y)} & G\,(x \otimes y) \end{array}$$

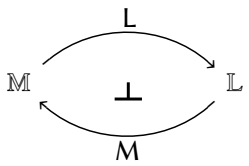
With the notion of monoidal transformation in place, we can define *monoidal adjunctions*.

Definition 3.4 (is_sym_monoidal_adjunction)**.** A **monoidal adjunction** between monoidal categories V_1 and V_2 consists of an adjunction $L \overset{\varepsilon}{\underset{\eta}{\dashv}} R$ such that L and R are lax monoidal functors and its unit η and its counit ε are monoidal transformations.

A monoidal adjunction is called a **symmetric monoidal adjunction** if L and R are symmetric lax monoidal functors.

Now we present the key notion of this section: *linear-non linear models*.

Definition 3.5 ([6], linear_non_linear_model)**.** A **linear-non linear model** consists of a symmetric monoidal adjunction $L \overset{\varepsilon}{\underset{\eta}{\dashv}} M$ where $L : \mathbb{M} \to \mathbb{L}$ such that $\mathbb{M}$ is a cartesian monoidal category and $\mathbb{L}$ is a symmetric monoidal closed category. This is partly visualized by



As we announced in the introduction to this section, the multiplicative conjunction and linear implication can now be interpreted as the tensor product and the right adjoint of the tensor product in $\mathbb{L}$.

To interpret the bang modality, we need to exploit the symmetric monoidal adjunction. The main point is that this modality is interpreted as a *symmetric monoidal comonad* on $\mathbb{L}$. Recall that a **comonad** F on a category C consists of an endofunctor $F : C \to C$ and natural transformations $\varepsilon_F : F \Rightarrow \mathsf{id}$ (the *counit*) and $\delta_F : F \Rightarrow F \cdot F$ (the *comultiplication*) such that the following diagrams commute for any $x : C$.

$$\begin{array}{ccccc} & & F\,x & & \\ & {\scriptstyle 1_{F\,x}}\swarrow & \downarrow{\scriptstyle \delta_F(x)} & \searrow{\scriptstyle 1_{F\,x}} & \\ F\,x & \xleftarrow[\varepsilon_F(F\,x)]{} & F\,(F\,x) & \xrightarrow[F\,(\varepsilon_F(x))]{} & F\,x \end{array}$$

$$\begin{array}{ccc} F\,x & \xrightarrow{\delta_F(x)} & F\,(F\,x) \\ {\scriptstyle \delta_F(x)}\downarrow & & \downarrow{\scriptstyle F\,(\delta_F(x))} \\ F\,(F\,x) & \xrightarrow{\delta_F(F\,x)} & F\,(F\,(F\,x)) \end{array}$$

Definition 3.6 (symmetric_monoidal_comonad)**.** A **symmetric monoidal comonad** consists of a comonad F such that F is a symmetric lax monoidal functor and ε_F and δ_F are monoidal transformations.

To get familiar with symmetric monoidal comonads, let us look at two examples.

Example 3.7 (lift_poset_symmetric_monoidal_comonad)**.** The lax monoidal functor Lift has the structure of a symmetric monoidal comonad, on the monoidal category Poset_*.

In addition, every symmetric monoidal adjunction gives rise to a symmetric monoidal comonad.

Problem 3.8. Given a symmetric monoidal adjunction $L \overset{\varepsilon}{\underset{\eta}{\dashv}} R$ where $L : V_1 \to V_2$, to construct a symmetric monoidal comonad $\mathsf{AdjToCmd}(L \overset{\varepsilon}{\underset{\eta}{\dashv}} R)$ on V_2.

Construction 3.9 (for Problem 3.8; sym_monoidal_adjunction_to_sym_monoidal_cmd)**.** The adjunction $L \overset{\varepsilon}{\underset{\eta}{\dashv}} R$ gives rise to a comonad V_2: the endofunctor of this comonad is given by $R \cdot L$, the counit of the comonad is given by ε, and the comultiplication is given by composition and whiskering using ε and η. To establish that this comonad is symmetric monoidal, we only need to check that the defined functor and natural transformations are (symmetric) lax monoidal. This follows from the fact that (symmetric) lax monoidal functors and transformations are closed under composition and whiskering. □

If we have a linear-non linear model, then from Construction 3.9 we get a comonad on $\mathbb{L}$. Since $\mathbb{M}$ is cartesian, the weakening and contraction rules in linear logic follow.

In the remainder of the paper, our goal is to construct examples of linear-non linear models. To do so, we take three steps.

1. We construct the monoidal categories $\mathbb{M}$ and $\mathbb{L}$. In the examples that we are interested in, $\mathbb{M}$ is either the category of commutative comonoids in some monoidal category or the Eilenberg-Moore category of some comonad. In Section 4, we discuss how to construct such categories using *displayed monoidal categories*.

2. We prove that $\mathbb{M}$ is cartesian. To do so, we use a general theorem that we discuss in Section 5.
3. We construct a symmetric monoidal adjunction between $\mathbb{M}$ and $\mathbb{L}$. To check whether an adjunction is symmetric monoidal, it suffices to check whether the left adjoint is strong. We prove this in Section 6.

4 Displayed Monoidal Categories

In this section, we devise a modular way of constructing monoidal categories. As explained in the introduction, we introduce displayed monoidal categories for this purpose. These need the concept of displayed categories [3] that we recall here for self-containedness.

4.1 The Theory

Definition 4.1 ([3, Definition 3.1], disp_cat)**.** Given a category C, a **displayed category** D **over** C consists of:

1. for each object $x : \mathsf{C}$, a type D_x of **objects over** x;
2. for each morphism $f : x \to y$ in C, $\bar{x} : \mathsf{D}_x$ and $\bar{y} : \mathsf{D}_y$, a set of **morphisms from** $\bar{x}$ **to** $\bar{y}$ **over** f, with membership of $\bar{f}$ in this set denoted as $\bar{f} : \bar{x} \to_f \bar{y}$;
3. for each $x : \mathsf{C}$ and $\bar{x} : \mathsf{D}_x$, an identity morphism $1_{\bar{x}} : \bar{x} \to_{1_x} \bar{x}$;
4. for all morphisms $f : x \to y$ and $g : y \to z$ in C and objects $\bar{x} : \mathsf{D}_x$, $\bar{y} : \mathsf{D}_y$ and $\bar{z} : \mathsf{D}_z$, a composition morphism $\bar{f} \cdot \bar{g} : \bar{x} \to_{f \cdot g} \bar{z}$ for $\bar{f} : \bar{x} \to_f \bar{y}$ and $\bar{g} : \bar{y} \to_g \bar{z}$,

such that, for all suitably typed inputs, we have:

5. $\bar{f} \cdot 1_{\bar{y}} =_* \bar{f}$,
6. $1_{\bar{x}} \cdot \bar{f} =_* \bar{f}$,
7. $\bar{f} \cdot (\bar{g} \cdot \bar{h}) =_* (\bar{f} \cdot \bar{g}) \cdot \bar{h}$.

Remark 4.2. These axioms are all *dependent* equalities, over equalities of morphisms in C. For instance, if $\bar{f} : \bar{x} \to_f \bar{y}$, then $\bar{f} \cdot 1_{\bar{y}} : \bar{x} \to_{f \cdot 1_y} \bar{y}$, so the displayed right unit axiom $\bar{f} \cdot 1_{\bar{y}} =_* \bar{f}$ is over the ordinary right unit axiom $f \cdot 1_y = f$ of C. Since we generally assume hom-sets for our categories C, the dependent equality does not depend on how that base equality is proven. As is done in [3], we therefore omit the precise base equalities as indices to = and indicate their presence by $*$.

Definition 4.3 ([3, Definition 3.2], total_category, pr1_category)**.** Given a displayed category D over category C, we define

1. the **total category of** D, written $\int \mathsf{D}$, whose objects are pairs $(x, \bar{x})$ with $x : \mathsf{C}$ and $x : \mathsf{D}_x$ and whose morphisms from $(x, \bar{x})$ to $(y, \bar{y})$ are pairs $(f, \bar{f})$ with $f : x \to y$ and $\bar{f} : \bar{x} \to_f \bar{y}$. (We omit identities and composition, as well as proofs of the axioms since they are straightforwardly obtained from C and D);
2. the evident forgetful functor from $\int \mathsf{D}$ to C that simply takes the first projection, both on objects and morphisms.

There is also the notion of displayed functor between displayed categories D_1 over C_1 and D_2 over C_2 over a functor $F : \mathsf{C}_1 \to \mathsf{C}_2$ [3, Definition 3.11]. With the whiskered style of the tensor operation in a monoidal structure comes the need for a dedicated definition of whiskered displayed bifunctors; the analogue of the tensor in a displayed monoidal category is the specific case where the source displayed categories and the target displayed category coincide. For the sake of brevity, we only detail the definition of displayed tensor in the paper.

Definition 4.4 (disp_bifunctor)**.** Given a displayed category D over category V and a tensor operation on V given by Item 2, Item 3 and Item 4 of Definition 2.1, a **displayed tensor for** D is given by

1. for all $x, y : \mathsf{V}$ and $\bar{x} : \mathsf{D}_x$ and $\bar{y} : \mathsf{D}_y$, a displayed object $\bar{x} \mathbin{\bar{\otimes}} \bar{y} : \mathsf{D}_{x \otimes y}$;
2. for all $x, y_1, y_2 : \mathsf{V}$, $\bar{x} \in \mathsf{D}_x$, $\bar{y}_1 \in \mathsf{D}_{y_1}$, $\bar{y}_2 \in \mathsf{D}_{y_2}$, $g : y_1 \to y_2$ and $\bar{g} : \bar{y}_1 \to_g \bar{y}_2$, a displayed morphism $\bar{x} \mathbin{\bar{\triangleleft}} \bar{g} : \bar{x} \mathbin{\bar{\otimes}} \bar{y}_1 \to_{x \triangleleft g} \bar{x} \mathbin{\bar{\otimes}} \bar{y}_2$;
3. for all $x_1, x_2, y : \mathsf{V}$, $\bar{x}_1 \in \mathsf{D}_{x_1}$, $\bar{x}_2 \in \mathsf{D}_{x_2}$, $\bar{y} \in \mathsf{D}_y$, $f : x_1 \to x_2$ and $\bar{f} : \bar{x}_1 \to_f \bar{x}_2$, a displayed morphism $\bar{f} \mathbin{\bar{\triangleright}} \bar{y} : \bar{x}_1 \mathbin{\bar{\otimes}} \bar{y} \to_{f \triangleright y} \bar{x}_2 \mathbin{\bar{\otimes}} \bar{y}$,

such that, for all suitably typed inputs, we have:

4. $\bar{x} \mathbin{\bar{\triangleleft}} 1_{\bar{y}} =_* 1_{\bar{x} \mathbin{\bar{\otimes}} \bar{y}}$;
5. $1_{\bar{x}} \mathbin{\bar{\triangleright}} \bar{y} =_* 1_{\bar{x} \mathbin{\bar{\otimes}} \bar{y}}$;
6. $\bar{x} \mathbin{\bar{\triangleleft}} (\bar{g}_1 \cdot \bar{g}_2) =_* (\bar{x} \mathbin{\bar{\triangleleft}} \bar{g}_1) \cdot (\bar{x} \mathbin{\bar{\triangleleft}} \bar{g}_2)$;
7. $(\bar{f}_1 \cdot \bar{f}_2) \mathbin{\bar{\triangleright}} \bar{y} =_* (\bar{f}_1 \mathbin{\bar{\triangleright}} \bar{y}) \cdot (\bar{f}_2 \mathbin{\bar{\triangleright}} \bar{y})$;
8. $(\bar{f} \mathbin{\bar{\triangleright}} \bar{y}_1) \cdot (\bar{x}_2 \mathbin{\bar{\triangleleft}} \bar{g}) =_* (\bar{x}_1 \mathbin{\bar{\triangleleft}} \bar{g}) \cdot (\bar{f} \mathbin{\bar{\triangleright}} \bar{y}_2)$.

These dependent equalities are in lockstep with the laws governing tensor and whiskerings in V and are to be understood over those laws.

Definition 4.5 (disp_monoidal)**.** Given a monoidal category (V, M) and a displayed category D over V, a **displayed monoidal category** ***DM*** **for** V, M **and** D consists of

1. a displayed object $\bar{I}_{DM} : \mathsf{D}_{\mathsf{I}_\mathsf{V}}$ (henceforth written without the index);
2. a displayed tensor for D over the tensor in M;
3. for all $x : \mathsf{V}$, $\bar{x} : \mathsf{D}_x$, displayed morphisms
$$\bar{\lambda}_{\bar{x}} : \bar{I} \mathbin{\bar{\otimes}} \bar{x} \to_{\lambda_x} \bar{x}$$
and
$$\bar{\rho}_{\bar{x}} : \bar{x} \mathbin{\bar{\otimes}} \bar{I} \to_{\rho_x} \bar{x};$$
4. for all $x, y, z : \mathsf{V}$, $\bar{x} : \mathsf{D}_x$, $\bar{y} : \mathsf{D}_y$ and $\bar{z} : \mathsf{D}_z$, a displayed morphism $\bar{\alpha}_{\bar{x},\bar{y},\bar{z}} : (\bar{x} \mathbin{\bar{\otimes}} \bar{y}) \mathbin{\bar{\otimes}} \bar{z} \to_{\alpha_{x,y,z}} \bar{x} \mathbin{\bar{\otimes}} (\bar{y} \mathbin{\bar{\otimes}} \bar{z})$;
5. for the same parameters as above, chosen displayed inverses (over the inverse base morphisms) for the two preceding items: $\bar{\lambda}^{-1}_{\bar{x}} : \bar{x} \to_{\lambda_x^{-1}} \bar{I} \mathbin{\bar{\otimes}} \bar{x}$, $\bar{\rho}^{-1}_{\bar{x}} : \bar{x} \to_{\rho_x^{-1}} \bar{x} \mathbin{\bar{\otimes}} \bar{I}$, and $\bar{\alpha}^{-1}_{\bar{x},\bar{y},\bar{z}} : \bar{x} \mathbin{\bar{\otimes}} (\bar{y} \mathbin{\bar{\otimes}} \bar{z}) \to_{\alpha^{-1}_{x,y,z}} (\bar{x} \mathbin{\bar{\otimes}} \bar{y}) \mathbin{\bar{\otimes}} \bar{z}$,

such that the thirteen families of dependent equations hold that follow in lockstep the equations in Item 13 to Item 20 of Definition 2.1, precisely after the model of the laws in Definition 4.4. Here we only detail the last two of them:

6. $\bar{\alpha}_{\bar{x},\bar{I},\bar{y}} \cdot (\bar{x} \mathbin{\bar{\triangleleft}} \bar{\lambda}_{\bar{y}}) =_* \bar{\rho}_{\bar{x}} \mathbin{\bar{\triangleright}} \bar{y}$;
7. $(\bar{\alpha}_{\bar{w},\bar{x},\bar{y}} \mathbin{\bar{\triangleright}} \bar{z}) \cdot \bar{\alpha}_{\bar{w},\bar{x}\bar{\otimes}\bar{y},\bar{z}} \cdot (\bar{w} \mathbin{\bar{\triangleleft}} \bar{\alpha}_{\bar{x},\bar{y},\bar{z}}) =_* \bar{\alpha}_{\bar{w}\bar{\otimes}\bar{x},\bar{y},\bar{z}} \cdot \bar{\alpha}_{\bar{w},\bar{x},\bar{y}\bar{\otimes}\bar{z}}$.

Problem 4.6. Given a displayed monoidal category DM for V, M and D, construct a monoidal structure $\int DM$ for $\int \mathsf{D}$ and a strong monoidal forgetful functor from $(\int \mathsf{D}, \int DM)$ to (V, M).

Construction 4.7 (for Problem 4.6; total_monoidal, projection_fmonoidal)**.** Definition 4.3 restricts the design space to one single solution: the first components of the unit, the tensor (objects), the whiskerings, the unitors and the associator come from M, and the respective second components come from DM. All the laws are then dependent equalities of pairs, where the first components are dealt with by the axioms of M, and the second components can *then* be dealt with by the laws of DM. A lax monoidal functor F based on $\mathsf{F} := \pi_1^{\mathsf{D}}$ is obtained with $\mathrm{unit}_{\mathsf{F}}$ and $\mathrm{tens}_{\mathsf{F}}((x,\bar{x}),(y,\bar{y}))$ suitable identity morphisms, whence this is a strict monoidal functor and in particular a strong monoidal functor. □

The terminology of "displayed monoidal category" is now justified as being the "material" to obtain a monoidal category from the total category of a displayed category by using a given monoidal structure on its base. By itself, a displayed monoidal category is neither a displayed category nor a (monoidal) category.

Remark 4.8. The whole point of identifying the notion of displayed monoidal category is to avoid having to give this "boilerplate code" in the construction of monoidal categories on total categories $\int \mathsf{D}$. For example, when D is *locally propositional* (that is, when the displayed morphisms from $\bar{x}$ to $\bar{y}$ over f always form a proposition), all the laws of a displayed tensor and a displayed monoidal category trivially hold, cf. make_disp_monoidal_locally_prop. However, Construction 4.7 does not become trivial in this case since the laws of M still enter the verification.

The modularization continues with the notion of *displayed symmetric monoidal category*, which we describe more compactly since it follows the same pattern of lifting the further horizontal extension according to a given vertical extension described by a displayed category.

Definition 4.9 (disp_symmetric)**.** Let (V, M) and D be given as in Definition 4.5, and let DM be a displayed monoidal category for these parameters. Assume the additional structure of a symmetric monoidal category for V (the symmetric braiding γ and its laws). A **displayed symmetric monoidal category *DS*** is given by

1. for all $x, y : \mathsf{V}$ and $\bar{x} : \mathsf{D}_x$ and $\bar{y} : \mathsf{D}_y$, a displayed morphism $\bar{\gamma}_{\bar{x},\bar{y}} : \bar{x} \mathbin{\bar{\otimes}} \bar{y} \to_{\gamma_{x,y}} \bar{y} \mathbin{\bar{\otimes}} \bar{x}$;

such that the families of dependent equations hold that follow in lockstep the equations expressing the naturality of γ, and the first hexagon law, of which we detail the latter:

2. $\bar{\alpha}_{\bar{x},\bar{y},\bar{z}} \cdot \bar{\gamma}_{\bar{x},\bar{y}\bar{\otimes}\bar{z}} \cdot \bar{\alpha}_{\bar{y},\bar{z},\bar{x}} =_* (\bar{\gamma}_{\bar{x},\bar{y}} \mathbin{\bar{\triangleright}} \bar{z}) \cdot \bar{\alpha}_{\bar{y},\bar{x},\bar{z}} \cdot (\bar{y} \mathbin{\bar{\triangleleft}} \bar{\gamma}_{\bar{x},\bar{z}})$

Given a displayed symmetric monoidal category DS with the parameters given above, we readily construct the **total symmetric structure** $\int DS$ that, together with the total monoidal structure $\int DM$, turns the total category $\int \mathsf{D}$ into a symmetric monoidal category, the **total symmetric monoidal category** for the given parameters. The strong monoidal functor described in Construction 4.7 is immediately seen to be symmetric as well. For the formalization, see total_symmetric and projection_is_symmetric.

4.2 The Examples

Example 4.10 (disp_monoidal_fullsub, disp_symmetric_fullsub)**.** Given a monoidal category V and a family $(P_x)_{x:\mathsf{V}}$ of types (indexed over the objects of V), there is the locally propositional displayed category D, with $\mathsf{D}_x := P_x$ and the unit type for all sets of displayed morphisms, which determines identity and composition in a trivial way. $\int \mathsf{D}$ is then (a notational variant of) the full subcategory induced by $(P_x)_{x:\mathsf{V}}$. The monoidal structure of V comes into play when asking for $u : P_{\mathsf{I}}$ and a family $(t_{x,y})_{x,y:\mathsf{V}}$ of functions $t_{x,y} : P_x \to P_y \to P_{x\otimes y}$. A displayed monoidal category is then constructed with the displayed tensor given by the $(t_{x,y})_{x,y:\mathsf{V}}$, and with $\bar{I} := u$. All the other data are displayed morphisms, hence trivially determined, and the displayed laws are trivial. Provided that V even has a symmetric structure, a symmetric displayed monoidal category is then obtained trivially as well, with $\bar{\gamma}_{\bar{x},\bar{y}}$ the inhabitant of the unit type. Thus, as the total structure, we get the symmetric monoidal full subcategory induced by $(P_x)_{x:\mathsf{V}}$, u and $(t_{x,y})_{x,y:\mathsf{V}}$.

Example 4.11 (dialgebra_monoidal)**.** Let V_1 and V_2 be monoidal categories, F a strong monoidal functor and G a lax monoidal functor, both from V_1 to V_2. For simplicity of our formalization, F is required to be strong, however it could be relaxed to oplax. We construct the category of monoidal dialgebras as a locally propositional displayed category D over V_1 as follows: for $x : \mathsf{V}_1$, the type D_x is the type of morphisms in V_2 from $\mathsf{F}\,x$ to $\mathsf{G}\,x$. The set of displayed morphisms from $\bar{x} : \mathsf{F}\,x \to \mathsf{G}\,x$ to $\bar{y} : \mathsf{F}\,y \to \mathsf{G}\,y$ over $f : x \to y$ is the proposition that the following diagram commutes:

$$\begin{array}{ccc} \mathsf{F}\,x & \xrightarrow{\mathsf{F}\,f} & \mathsf{F}\,y \\ {\scriptstyle \bar{x}}\downarrow & & \downarrow{\scriptstyle \bar{y}} \\ \mathsf{G}\,x & \xrightarrow{\mathsf{G}\,f} & \mathsf{G}\,y \end{array}$$

The constructions $1_{\bar{x}}$ and $\bar{f} \cdot \bar{g}$ for D are nothing but the obvious proofs of commutation of the diagrams:

$$\begin{array}{ccc} \mathsf{F}\,x & \xrightarrow{\mathsf{F}\,1_x} & \mathsf{F}\,x \\ {\scriptstyle \bar{x}}\downarrow & & \downarrow{\scriptstyle \bar{x}} \\ \mathsf{G}\,x & \xrightarrow{\mathsf{G}\,1_x} & \mathsf{G}\,x \end{array} \quad \text{and} \quad \begin{array}{ccc} \mathsf{F}\,x & \xrightarrow{\mathsf{F}\,(f\cdot g)} & \mathsf{F}\,z \\ {\scriptstyle \bar{x}}\downarrow & & \downarrow{\scriptstyle \bar{z}} \\ \mathsf{G}\,x & \xrightarrow{\mathsf{G}\,(f\cdot g)} & \mathsf{G}\,z \end{array}$$

(For the second diagram, this is the horizontal pasting of the two input diagrams for $\bar{f}$ and $\bar{g}$.) The laws all hold trivially

because of local propositionality. Notice that D is *groupoidal* (groupoidal_disp_cat): If $f : x \to y$ is invertible, then so is the only possible $\bar{f} : \bar{x} \to_f \bar{y}$ – the diagram for $\bar{f}$ becomes one for its inverse by interchanging $\bar{x}$ and $\bar{y}$ and by replacing f with its inverse. $\int \mathsf{D}$ is then the **category of dialgebras** for F and G (not yet exploiting monoidal structure) – a common generalization of F-algebras and their algebra homomorphisms (with G the identity) and G-coalgebras and their coalgebra homomorphisms (with F the identity). $\int \mathsf{D}$ has a canonical monoidal structure, induced by those of V_1 and V_2, and this is seen through a displayed monoidal category for V_1 and D. Since D is locally propositional, the laws for the displayed tensor and the displayed monoidal category are all trivial, but the definition of the displayed morphisms in the construction involves proving those diagrams to commute. However, the inverses of the unitors and the associator are obtained generically from D being groupoidal, making the overall construction less burdensome. Here, we only show the definition of $\bar{I}$ and $\bar{x} \bar{\otimes} \bar{y}$ (from the defined tensor action $\bar{x} \otimes \bar{y}$ in V_2), given by the commuting diagrams

$$\begin{array}{ccc} \mathsf{F}\,\mathsf{I}_{\mathsf{V}_1} & \xrightarrow{\mathrm{unit}_{\mathsf{F}}^{-1}} & \\ \downarrow \bar{I} & & \mathsf{I}_{\mathsf{V}_2} \\ \mathsf{G}\,\mathsf{I}_{\mathsf{V}_1} & \xrightarrow{\mathrm{unit}_{\mathsf{G}}} & \end{array} \quad \text{and} \quad \begin{array}{ccc} \mathsf{F}\,(x \otimes y) & \xrightarrow{\mathrm{tens}_{\mathsf{F}}(x,y)^{-1}} & \mathsf{F}\,x \otimes \mathsf{F}\,y \\ \downarrow \bar{x} \bar{\otimes} \bar{y} & & \downarrow \bar{x} \otimes \bar{y} \\ \mathsf{G}\,(x \otimes y) & \xleftarrow{\mathrm{tens}_{\mathsf{G}}(x,y)} & \mathsf{G}\,x \otimes \mathsf{G}\,y \end{array}$$

If V_1 and V_2 and F and G are even symmetric monoidal (categories resp. functors), then a displayed symmetric monoidal category is easily obtained. Thanks to local propositionality of D, only the diagram corresponding to the existence of $\bar{\gamma}_{\bar{x},\bar{y}} : \bar{x} \bar{\otimes} \bar{y} \to_{\gamma_{x,y}} \bar{y} \bar{\otimes} \bar{x}$ has to be shown to commute (which only uses that F and G are symmetric monoidal and that γ is natural).

Example 4.12 (monoidal_pointed_objects)**.** Given a monoidal category V, the category I/V of *slices* under the unit of V is obtained as $\int \mathsf{D}$, for D the locally propositional displayed category over V, defined with D_x the morphisms in V from I to x, and with the set of displayed morphisms from $\bar{x} : \mathsf{I} \to x$ to $\bar{y} : \mathsf{I} \to y$ over $f : x \to y$ the proposition that $\bar{x} \cdot f = \bar{y}$. Mathematically, this is nothing but an instance of the previous example, with V_1 and V_2 set to V, F set to constantly I (which is strongly monoidal) and G set to the identity on V. We therefore get a monoidal structure on $\int \mathsf{D}$, and we call it the monoidal category of **monoidal-pointed objects** (the name comes from considering $\bar{x}$ as a *monoidal point*, since its source is the unit and not necessarily a terminal object). As an instance of this construction, we get the monoidal category of pointed endofunctors on a category C, where we start with the endofunctor category $[\mathsf{C}, \mathsf{C}]$ that is monoidal with $\otimes$ given by composition (which is another natural example of a non-cartesian monoidal category) and the identity on C as unit.

Example 4.13 (sym_monoidal_cat_co_eilenberg_moore)**.** Let V be a symmetric monoidal category and $(\mathsf{F}, \varepsilon_{\mathsf{F}}, \delta_{\mathsf{F}})$ be a symmetric monoidal comonad. We construct the symmetric monoidal **Eilenberg-Moore category** of this comonad using our displayed technology, combining Example 4.10 and Example 4.11. The objects of the category to be constructed are coalgebras for $(\mathsf{F}, \varepsilon_{\mathsf{F}}, \delta_{\mathsf{F}})$, that is, dialgebras for the identity and F that are compatible with ε_{F} and δ_{F}. For a coalgebra (x, h) with $h : x \to \mathsf{F}x$, this asks for the commutation of the following diagrams:

$$\begin{array}{ccc} x & \xrightarrow{h} & \mathsf{F}\,x \\ & {\scriptstyle 1_x} \searrow & \downarrow \varepsilon_{\mathsf{F}}(x) \\ & & x \end{array} \quad \text{and} \quad \begin{array}{ccc} x & \xrightarrow{h} & \mathsf{F}x \\ \downarrow h & & \downarrow \delta_{\mathsf{F}}(x) \\ \mathsf{F}\,x & \xrightarrow{\mathsf{F}h} & \mathsf{F}\,(\mathsf{F}\,x) \end{array}$$

We use Example 4.10 with the symmetric monoidal category of dialgebras of Example 4.11 for the identity and F as parameter V and with $P_{(x,h)}$ the proposition expressing the commutation of the above diagrams. We are thus left with providing u and $t_{(x,h),(y,k)}$ in the notation of that example, with $k : y \to \mathsf{F}y$. They are not data but just proofs of commutation of diagrams (for t depending on assumed commuting diagrams), for which the assumption that we started with a monoidal comonad is crucial. We denote the resulting displayed category and total category by $\mathsf{EM}_{\mathsf{d}}(\mathsf{F})$ and $\mathsf{EM}(\mathsf{F})$, respectively. We denote the (strict monoidal) forgetful functor from $\mathsf{EM}(\mathsf{F})$ to V by $\mathsf{U}^{\mathsf{EM}(\mathsf{F})}$.

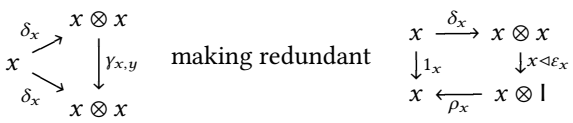
Example 4.14. Given a symmetric monoidal category V, we consider the **monoidal category of comonoids** that are internal to V (we need symmetry of V in our construction). Its objects are triples $(x, \varepsilon_x, \delta_x)$ with counit $\varepsilon_x : x \to \mathsf{I}$ and comultiplication $\delta_x : x \to x \otimes x$, provided commutation of the following diagrams:

$$\begin{array}{ccccc} & & x \otimes x & \xrightarrow{\delta_x \triangleright x} & (x \otimes x) \otimes x \\ & \overset{\delta_x}{\nearrow} & & & \downarrow \alpha_{x,x,x} \\ x & & & & \\ & \underset{\delta_x}{\searrow} & x \otimes x & \xrightarrow{x \triangleright \delta_x} & x \otimes (x \otimes x) \end{array} \quad \text{and} \quad \begin{array}{ccccc} x \otimes x & \xleftarrow{\delta_x} & x & \xrightarrow{\delta_x} & x \otimes x \\ \downarrow \varepsilon_x \triangleright x & & \downarrow 1_x & & x \triangleleft \varepsilon_x \downarrow \\ \mathsf{I} \otimes x & \xrightarrow[\lambda_x]{} & x & \xleftarrow[\rho_x]{} & x \otimes \mathsf{I} \end{array}$$

The morphisms of this category are meant to follow from first principles, as dictated by a dialgebraic view of counit and comultiplication. This profits again from the notion of displayed category. We consider the displayed categories D_1 and D_2 over V that are the displayed categories of dialgebras for the (strong monoidal) identity functor as first argument in both cases. For D_1, the second argument is the strong monoidal diagonal functor (an object x is mapped to $x \otimes x$, and already the data for a lax monoidal functor requires symmetry of V). For D_2, the second argument is the strong monoidal functor that is constantly the unit of V (see Example 4.12). We use the generic construction of a directed product of two displayed categories (not shown here) to combine D_1 and D_2 into a locally propositional displayed category D over V, and $\int \mathsf{D}$ then has as objects the triples $(x, \varepsilon_x, \delta_x)$, but not yet the commutation of the diagrams. Using Example 4.10, we get a locally propositional displayed category D′ (see comonoid_laws_disp_cat) over $\int \mathsf{D}$ that embodies

those laws. Example 4.11 gives us displayed monoidal categories for D_1 and D_2, and there is a generic construction of a directed product of two displayed monoidal categories (not shown here) in case of local propositionality of the given displayed categories, hence we get a displayed monoidal category for D. To get a displayed monoidal category for D′, Example 4.10 asks us to provide "operations" u and t, but they are nothing but requirements about the propagation of the comonoid laws through the monoidal structure obtained for $\int D$. For a more versatile use, we can repackage the obtained nested structures so that they are displayed over the original V and not over $\int D$. This uses a Σ-*construction* (corresponding to rearranging the nesting of Σ-types) that is generically available for locally propositional displayed categories and provides a suitable displayed monoidal category although local propositionality is typically lost then. The displayed monoidal category thus obtained is formalized as disp_monoidal_comonoids.

We also consider the **symmetric monoidal category of commutative comonoids**, internal to V. A commutative comonoid is a comonoid whose comultiplication is compatible with the symmetric braiding of V, expressed in the following diagram.



The category of commutative comonoids Ccom(V) is the full subcategory induced by this extra property, while the redundancy of the mentioned counit law is only embodied in a convenience constructor in our formalization (make_commutative_comonoid). Using the same generic tools as for the comonoids, we obtain a displayed monoidal category $\mathrm{Ccom}_d(V)$, then a displayed symmetric monoidal category, thus the announced symmetric monoidal category of commutative comonoids (symmetric_cat_commutative_comonoids). We denote the (strict monoidal) forgetful functor from Ccom(V) to V by U_V^{Ccom}. At no point is the reasoning specific to comonoids repeated in this process, so there is no duplication of code dealing with the base comonoid structure.

Note that in the same way, we can construct the category of monoids.

5 Characterizing Cartesian Monoidal Categories

In this section, we show that cartesian monoidal categories can be classified using the existence of commutative comonoids. We also use that result to show that the category of commutative comonoids is cartesian.

Theorem 5.1 ([27, Prop. 16], monoidal_is_cartesian_from_comonoid)**.** *Let* V *be a monoidal category. The monoidal structure is cartesian if for every object* $x : V$*, a comonoid structure* $(\varepsilon_x, \delta_x)$ *on* x *is given, such that*

1. *every morphism is a comonoid homomorphism;*
2. $\varepsilon_I = 1_I$*, that is, the counit of the monoidal unit is the identity;*
3. *for every* x *and* $y : V$*, the following diagram commutes:*

$$\begin{array}{ccc} x \otimes y & \xrightarrow{\delta_{x\otimes y}} & (x \otimes y) \otimes (x \otimes y) \\ {\scriptstyle 1_{x\otimes y}}\downarrow & & \downarrow{\scriptstyle (1_x \otimes \varepsilon_y)\otimes(\varepsilon_x \otimes 1_y)} \\ x \otimes y & \xleftarrow{\rho_x \otimes \lambda_y} & (x \otimes I) \otimes (I \otimes y) \end{array}$$

As already remarked by Melliès [27], the coassociativity law is not used. Furthermore, Melliès [27] (equivalently) replaces assumption 1 by requiring that the counit and comultiplication are part of a natural transformation.

Provided that V is symmetric monoidal, assumptions 2 and 3 in Theorem 5.1 can be rephrased by requiring that the natural transformations are monoidal:

Corollary 5.2 ([27, Cor. 17], symm_monoidal_is_cartesian_from_comonoid)**.** *Let* V *be a symmetric monoidal category. The monoidal structure is cartesian if for every object* $x : V$*, a comonoid structure* $(\varepsilon_x, \delta_x)$ *on* x *is given, such that*

1. *every morphism is a comonoid homomorphism;*
2. $\varepsilon_I = 1_I$*, that is, the counit of the monoidal unit is the identity;*
3. *for every* x *and* $y : V$*, the following diagram commutes:*

$$\begin{array}{ccc} x \otimes y & \xrightarrow{\delta_{x\otimes y}} & (x \otimes y) \otimes (x \otimes y) \\ {\scriptstyle \delta_x \otimes \delta_y}\downarrow & \nearrow_{\mathrm{inner\_swap}_{x,x,y,y}} & \\ (x \otimes x) \otimes (y \otimes y) & & \end{array}$$

where $\mathrm{inner\_swap}_{x,y,z,w}$ *of type* $(x \otimes y) \otimes (z \otimes w) \to (x \otimes z) \otimes (y \otimes w)$ *is defined in terms of the symmetric braiding.*

Because commutative comonoids (internal to a symmetric monoidal category) are equivalently comonoids in the (symmetric) monoidal category of comonoids, we conclude:

Corollary 5.3 ([27, Cor. 18], cartesian_monoidal_cat_of_comm_comonoids)**.** *Every (symmetric) monoidal category of commutative comonoids, internal to a symmetric monoidal category, is cartesian.*

6 Constructing Models of Linear Logic

The final step that we take to construct models of linear logic, is constructing a symmetric monoidal adjunction. Fortunately, there is a simple way to check whether an adjunction is symmetric monoidal: it suffices to show that the left adjoint is a strong monoidal functor.

Proposition 6.1 ([26, Lemma 13], sym_monoidal_adjunction_from_strong)**.** *Suppose that we have symmetric monoidal*

categories V_1 *and* V_2 *and an adjunction* $\mathsf{L} \dashv \mathsf{R}$ *where* $\mathsf{L} : \mathsf{V}_1 \to \mathsf{V}_2$ *is a lax symmetric monoidal functor. If the adjunction* $\mathsf{L} \dashv \mathsf{R}$ *is symmetric monoidal, then* L *is strong monoidal. Conversely, if* L *is strong monoidal, then* $\mathsf{L} \dashv \mathsf{R}$ *is symmetric monoidal.*

In the literature, this proposition is usually phrased as a logical equivalence. However, we refrain from stating it this way. The reason for that is that in the framework of homotopy type theory, a logical equivalence is translated as an equivalence between two propositions (i. e., two types for which all elements are equal). However, the type that an adjunction $\mathsf{L} \dashv \mathsf{R}$ is symmetric monoidal, is not a proposition in general, because there could be multiple proofs that a functor is symmetric monoidal.

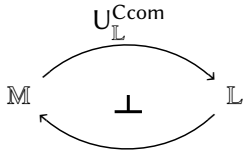
6.1 Lafont Categories

We look at two methods to construct examples of linear-non linear models. The first one we look at, is the notion of *Lafont category* [14, Definition 2]. In a Lafont category, the cartesian category $\mathbb{M}$ is the category of commutative comonoids in some symmetric monoidal closed category, and the bang modality is given by the free comonoid construction. Note that by Corollary 5.3, the category of commutative comonoids is always cartesian. As such, to show that this gives rise to a linear-non linear model, it suffices to shows that the forgetful functor has a right adjoint.

Definition 6.2 (lafont_category)**.** A **Lafont category** consists of a symmetric monoidal closed category $\mathbb{L}$ such that $\mathsf{U}_{\mathbb{L}}^{\mathsf{Ccom}}$ has a right adjoint.

Problem 6.3. Given a Lafont category, to construct an LNL model.

Construction 6.4 (for Problem 6.3; linear_non_linear_model_from_lafont_category)**.** Suppose that we have a Lafont category $\mathbb{L}$. Then we have the following linear-non linear model



Note that $\mathsf{Ccom}(\mathbb{L})$ is cartesian by Corollary 5.3. In addition, the adjunction is symmetric monoidal by Proposition 6.1, because the functor $\mathsf{U}_{\mathbb{L}}^{\mathsf{Ccom}}$ is strong. □

Examples of Lafont categories are difficult to find in practice [14]. However, one example is the *relational model of linear logic*.

Example 6.5 (relational_model)**.** The symmetric monoidal closed category Rel (Example 2.6) is a Lafont category. To establish that, we need to construct a right adjoint of $\mathsf{U}_{\mathsf{Rel}}^{\mathsf{Ccom}}$. The main idea behind the construction of this right adjoint, is that commutative commonoids in Rel are the same as commutative monoids of sets. As such, the action of the right adjoint on some set X is given by the set of finite multisets of X [21]. Note that the finite multiset construction gives a left adjoint to the forgetful functor from commutative monoids to sets, and this determines the action on morphisms.

6.2 Linear Categories

Another method to construct linear-non linear models, is via *linear categories*. In a linear category, the category $\mathbb{M}$ is the category of coalgebras for some comonad over a symmetric monoidal closed category. Note that this directly gives us the desired adjunction: the adjunction coming from the Eilenberg-Moore category is symmetric monoidal, because the left adjoint is a strong monoidal functor. However, the Eilenberg-Moore category for a comonad is not necessarily cartesian. The extra requirements of a linear category guarantee that the Eilenberg-Moore category is cartesian.

Definition 6.6 ([7, Def. 3], linear_category)**.** A **linear category** consists of

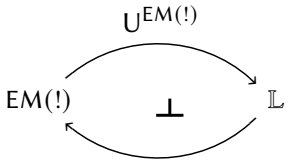
1. a symmetric monoidal closed category $\mathbb{L}$;
2. a symmetric monoidal comonad ! on $\mathbb{L}$ (recall that its comultiplication is called $\delta_!$);
3. for every object $x : \mathbb{L}$, morphisms $\mathsf{e}_x : !x \to \mathsf{I}$ and $\mathsf{d}_x : !x \to !x \otimes !x$.

such that the following holds for every object x.

4. the data $(!x, \mathsf{e}_x, \mathsf{d}_x)$ forms a commutative comonoid;
5. e_x and d_x are natural transformations;
6. e_x is a coalgebra morphism from $\delta_!(x)$ to $\mathsf{unit}_!$;
7. d_x is a coalgebra morphism from $\delta_!(x)$ to $\delta_!(x) \otimes \delta_!(x) \cdot \mathsf{tens}_!(x, x)$;
8. $\delta_!(x)$ is a morphism of comonoids from $(!x, \mathsf{e}_x, \mathsf{d}_x)$ to $(!!x, \mathsf{e}_{!x}, \mathsf{d}_{!x})$.

Problem 6.7. Given a linear category, to construct a linear-non linear model.

Construction 6.8 (for Problem 6.7; linear_to_lnl)**.** We construct a linear-non linear model as follows.



The adjunction is symmetric monoidal by Proposition 6.1, because the left adjoint $\mathsf{U}^{\mathsf{EM}(!)}$ is a strong monoidal functor. The main work lies in establishing that $\mathsf{EM}(!)$ is a cartesian monoidal category. This follows from the extra conditions of linear categories and Corollary 5.2. □

One example of a linear category is given by the lifting operation on pointed posets. This example is used to model linear features of functional programming languages [6].

Example 6.9 (lifting_linear_category)**.** In Example 3.7, we constructed the lifting comonad on pointed posets. This gives rise to a linear category. One can directly use the universal property of the smash product to construct the required morphisms and verify the laws.

7 Related Work

In this section, we discuss related work, categorized into the two topics *formalization* (in Section 7.1) and *semantics of linear logic* (in Section 7.2).

7.1 Formalization of Monoidal Categories

There are numerous formalizations of monoidal categories available. In the HoTT library, there is a formalized definition of monoidal category [5, 17, 35], and that definition has been formalized in 1lab [36] (dedicated to cubical methods in homotopy type theory, based on Agda [28]) as well.

The coherence theorem is one of the corner stones in the theory of monoidal categories. This theorem has been formalized in multiple libraries, such as Isabelle [34, 39], Mathlib (Lean) [24] and Agda [18]. However, these formalizations only consider monoidal categories, and not *symmetric* monoidal categories. Piceghello studied the coherence theorem for both monoidal and symmetric monoidal categories [31], and formalized these theorems using the HoTT library [5].

Three of the present authors [40] formalize univalent monoidal categories using the UniMath library [38], and they show that the bicategory of univalent monoidal categories is univalent. They also construct a Rezk completion for monoidal categories, which associates, to any monoidal category, a univalent one, in a universal way. The same three of the present authors also used monoidal categories in their formalization of non-wellfounded syntax [25]. The present paper makes use of the formalization of monoidal categories in whiskered style that was done for that work.

Hu and Carette [18] build a library on category theory in the proof assistant Agda, based on the concept of "E-categories" [30]. This library contains definitions of different variants of monoidal categories. In particular, it contains the definition of $*$-autonomous category, which is used for models of *classical* linear logic. Choudhury and Fiore [13] give a formalization of finite multisets in cubical Agda [37], and they discuss applications to linear logic. However, their formalization includes neither monoidal categories nor general categorical frameworks for the semantics of linear logic.

7.2 Semantics of Linear Logic

The semantics of linear logic has been an active topic of research since its inception [16]. Several overviews of the semantics of linear logic have been written, in particular by De Paiva [14] and Melliès [26, 27].

In this paper, we consider three notions of models of linear logic. These are linear-non linear-models, introduced by Benton [6], Lafont categories, introduced by Lafont [21], and linear categories, introduced by Benton, Bierman, De Paiva, and Hyland [8, 9]. Many examples of models are given in the literature [11, 19] (for example). One notion of model that we do not consider in this paper is that of a *Seely category* [9, 33]. Note that in many papers on linear logic, $*$-autonomous categories are considered, which model classical linear logic.

8 Summary of Formalization Choices

In this section, we briefly summarize the choices we have made in the formalization.

Firstly, monoidal categories are formalized in whiskered and curried style — see also Remark 2.2. In particular, working with a curried tensor operation is much easier than working with an uncurried one taking pairs as arguments: Coq seems to have a lot of trouble synthesizing implicit arguments when a pair is expected, even when the synthesis of the individual components is trivial.

Secondly, we use displayed technology to avoid the duplication of code. Indeed, displayed (monoidal) categories express exactly the *difference* between a mathematical structure and an "extended" mathematical structure with more data or properties.

Thirdly, we constructed the monoidal categories that are used in Section 6 in a different way compared to mathematical texts. The category of comonoids (Example 4.14) and the Eilenberg-Moore category (Example 4.13) for a comonad are usually constructed in one step by specifying the objects and morphisms. However, we refrained from doing so: instead we built these structures step by step. More concretely, we first defined displayed categories for the relevant operations, and then we took a full subcategory to guarantee that the necessary laws are satisfied as well. The reason for doing so is that it allows us to reuse code: both the category of comonoids and the Eilenberg-Moore category were constructed as a full subcategory of the category of dialgebras (Example 4.11). As such, we did not need to define a monoidal structure specific to either of these categories, but instead, we reused results about these general constructions.

Finally, the way we phrased the definition of linear categories (Definition 6.6) in our formalization is slightly different from mathematical texts, since we use an unfolded definition of linear categories. This difference is rather minor, and both would give usable definitions. However, we had a slight preference for the unfolded version. We had two reasons for doing so. First, the unfolded definition clearly distinguishes data and properties. Second, it is slightly more convenient to phrase and use. More specifically, every linear category has a counit e_{-}, which is a natural transformation to the constant functor. If one were to use the standard definition of natural transformation, each naturality condition would have an extra identity in it coming from the action of

the constant functor. The more compact definition also requires more careful management of the coalgebra structures, since one needs to express that both e_x and d_x are coalgebra morphisms.

9 Conclusion and Perspectives

We have presented a formalization effort for the categorical semantics of linear logic that calls for a layered approach to dealing with a variety of structures based on monoidal categories. Taking seriously this challenge for proof engineering when it comes to formalization on a computer, we developed the notion of a displayed monoidal category and deployed a sufficiently rich library to put this concept to use for a carefully chosen fragment of linear logic. We limited ourselves to the propositional fragment, with only multiplicative conjunction, linear implication and the "of course" exponential, denoted "!" (and called bang) that provides controlled forms of weakening and contraction. This allowed us to profit from the abstraction obtained through the notion of linear-non linear model [6], while already requiring a certain depth of our stacked categorical structures. We also formalized that the linear-non linear models encompass other important classes of categorical models of linear logic. As compared to Melliès [27], we deliberately left out the Seely categories and thus did not formalize its dualizing object $\bot$ that is needed for the interpretation of classical linear logic. Again in contrast to Melliès [27], we did not embrace the higher categorical perspective (of 2-categories or bicategories). We could have done that since the UniMath library already has a substantial formalization of bicategories [2] (and also the bicategory of monoidal categories [40]). For the present paper, we considered that a dependency on bicategories would have asked for too many prerequisites. Our displayed monoidal categories have all their interest already in the present setting, and they already rely essentially on dependent types in the host language.

The entire formalization took place within the UniMath library that is a Coq library. This means in particular that all constructions and proofs can be checked automatically by a small kernel – much less than the entire Coq system that helps in doing those constructions and proofs.

The deliberate limitations of our work can easily be translated into perspectives for future work. A natural step would be to formalize the Seely categories. A desirable step would be to connect our formalized categorical models with formalized syntax of linear logic and thus formally verify that cut elimination does not change the denotation in the categorical models. We could be inspired by formalized meta-theory of linear logics [12] in a different theorem prover (Abella) or try to build on the Coq library Yalla by Olivier Laurent[3] for the syntactic aspects. A larger project that could capitalize on recent research would be the formalization of categorical semantics for least and greatest fixed points on the level of the types / formulas of (propositional) linear logic [15, 20].

[3] https://perso.ens-lyon.fr/olivier.laurent/yalla/, available on GitHub

Acknowledgments

We gratefully acknowledge the work by the Coq development team in providing the Coq proof assistant and surrounding infrastructure, as well as their support in keeping UniMath compatible with Coq. We also thank the anonymous referees of CPP 2024 for their helpful comments. This research was supported by the NWO project "The Power of Equality" OCENW.M20.380, which is financed by the Dutch Research Council (NWO).

References

[1] Samson Abramsky and Guy McCusker. 1999. Game Semantics. In *Computational Logic*, Ulrich Berger and Helmut Schwichtenberg (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 1–55.

[2] Benedikt Ahrens, Dan Frumin, Marco Maggesi, Niccolò Veltri, and Niels van der Weide. 2021. Bicategories in univalent foundations. *Math. Struct. Comput. Sci.* 31, 10 (2021), 1232–1269. https://doi.org/10.1017/S0960129522000032

[3] Benedikt Ahrens and Peter LeFanu Lumsdaine. 2019. Displayed Categories. *Log. Methods Comput. Sci.* 15, 1 (2019). https://doi.org/10.23638/LMCS-15(1:20)2019

[4] Steve Awodey, Jonas Frey, and Sam Speight. 2018. Impredicative Encodings of (Higher) Inductive Types. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, Anuj Dawar and Erich Grädel (Eds.). ACM, 76–85. https://doi.org/10.1145/3209108.3209130

[5] Andrej Bauer, Jason Gross, Peter LeFanu Lumsdaine, Michael Shulman, Matthieu Sozeau, and Bas Spitters. 2017. The HoTT library: a formalization of homotopy type theory in Coq. In *Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs, CPP 2017, Paris, France, January 16-17, 2017*, Yves Bertot and Viktor Vafeiadis (Eds.). ACM, 164–172. https://doi.org/10.1145/3018610.3018615

[6] P. N. Benton. 1994. A Mixed Linear and Non-Linear Logic: Proofs, Terms and Models (Extended Abstract). In *Computer Science Logic, 8th International Workshop, CSL '94, Kazimierz, Poland, September 25-30, 1994, Selected Papers (Lecture Notes in Computer Science, Vol. 933)*, Leszek Pacholski and Jerzy Tiuryn (Eds.). Springer, 121–135. https://doi.org/10.1007/BFb0022251

[7] P. N. Benton, Gavin M. Bierman, Valeria de Paiva, and Martin Hyland. 1992. Linear Lambda-Calculus and Categorial Models Revisited. In *Computer Science Logic, 6th Workshop, CSL '92, San Miniato, Italy, September 28 - October 2, 1992, Selected Papers (Lecture Notes in Computer Science, Vol. 702)*, Egon Börger, Gerhard Jäger, Hans Kleine Büning, Simone Martini, and Michael M. Richter (Eds.). Springer, 61–84. https://doi.org/10.1007/3-540-56992-8_6

[8] P. N. Benton, Gavin M. Bierman, Valeria de Paiva, and Martin Hyland. 1993. A Term Calculus for Intuitionistic Linear Logic. In *Typed Lambda Calculi and Applications, International Conference on Typed Lambda Calculi and Applications, TLCA '93, Utrecht, The Netherlands, March 16-18, 1993, Proceedings (Lecture Notes in Computer Science, Vol. 664)*, Marc Bezem and Jan Friso Groote (Eds.). Springer, 75–90. https://doi.org/10.1007/BFb0037099

[9] Gavin M. Bierman. 1995. What is a Categorical Model of Intuitionistic Linear Logic?. In *Typed Lambda Calculi and Applications, Second International Conference on Typed Lambda Calculi and Applications, TLCA '95, Edinburgh, UK, April 10-12, 1995, Proceedings (Lecture Notes in Computer Science, Vol. 902)*, Mariangiola Dezani-Ciancaglini and Gordon D. Plotkin (Eds.). Springer, 78–93. https://doi.org/10.1007/BFb0014046

[10] Ana C. Calderon and Guy McCusker. 2010. Understanding Game Semantics Through Coherence Spaces. In *Proceedings of the 26th Conference on the Mathematical Foundations of Programming Semantics, MFPS 2010, Ottawa, Ontario, Canada, May 6-10, 2010 (Electronic Notes in Theoretical Computer Science, Vol. 265)*, Michael W. Mislove and Peter Selinger (Eds.). Elsevier, 231–244. https://doi.org/10.1016/j.entcs.2010.08.014

[11] Alberto Carraro, Thomas Ehrhard, and Antonino Salibra. 2010. Exponentials with Infinite Multiplicities. In *Computer Science Logic, 24th International Workshop, CSL 2010, 19th Annual Conference of the EACSL, Brno, Czech Republic, August 23-27, 2010. Proceedings (Lecture Notes in Computer Science, Vol. 6247)*, Anuj Dawar and Helmut Veith (Eds.). Springer, 170–184. https://doi.org/10.1007/978-3-642-15205-4_16

[12] Kaustuv Chaudhuri, Leonardo Lima, and Giselle Reis. 2019. Formalized meta-theory of sequent calculi for linear logics. *Theor. Comput. Sci.* 781 (2019), 24–38. https://doi.org/10.1016/j.tcs.2019.02.023

[13] Vikraman Choudhury and Marcelo Fiore. 2023. Free Commutative Monoids in Homotopy Type Theory. *Electronic Notes in Theoretical Informatics and Computer Science* 1 (2023). https://doi.org/10.46298/entics.10492

[14] Valeria de Paiva. 2014. *Categorical Semantics of Linear Logic for All.* Springer Netherlands, Dordrecht, 181–192. https://doi.org/10.1007/978-94-007-7548-0_9

[15] Thomas Ehrhard and Farzad Jafarrahmani. 2021. Categorical models of Linear Logic with fixed points of formulas. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021.* IEEE, 1–13. https://doi.org/10.1109/LICS52264.2021.9470664

[16] Jean-Yves Girard. 1987. Linear Logic. *Theor. Comput. Sci.* 50 (1987), 1–102. https://doi.org/10.1016/0304-3975(87)90045-4

[17] Jason Gross, Adam Chlipala, and David I. Spivak. 2014. Experience Implementing a Performant Category-Theory Library in Coq. In *Interactive Theorem Proving - 5th International Conference, ITP 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings (Lecture Notes in Computer Science, Vol. 8558)*, Gerwin Klein and Ruben Gamboa (Eds.). Springer, 275–291. https://doi.org/10.1007/978-3-319-08970-6_18

[18] Jason Z. S. Hu and Jacques Carette. 2021. Formalizing category theory in Agda. In *CPP '21: 10th ACM SIGPLAN International Conference on Certified Programs and Proofs, Virtual Event, Denmark, January 17-19, 2021*, Catalin Hritcu and Andrei Popescu (Eds.). ACM, 327–342. https://doi.org/10.1145/3437992.3439922

[19] Martin Hyland and Andrea Schalk. 2003. Glueing and orthogonality for models of linear logic. *Theor. Comput. Sci.* 294, 1/2 (2003), 183–231. https://doi.org/10.1016/S0304-3975(01)00241-9

[20] Farzad Jafarrahmani. 2023. *Fixpoints of Types in Linear Logic from a Curry-Howard-Lambek Perspective.* PhD thesis. Université Paris Cité, France. https://hal.science/tel-04295098v1

[21] Yves Lafont. 1988. The Linear Abstract Machine. *Theor. Comput. Sci.* 59 (1988), 157–180. https://doi.org/10.1016/0304-3975(88)90100-4

[22] Ugo Dal Lago and Martin Hofmann. 2011. Realizability models and implicit complexity. *Theor. Comput. Sci.* 412, 20 (2011), 2029–2047. https://doi.org/10.1016/J.TCS.2010.12.025

[23] Saunders Mac Lane. 1998. *Categories for the Working Mathematician* (second ed.). Graduate Texts in Mathematics, Vol. 5. Springer-Verlag, New York. xii+314 pages.

[24] The mathlib Community. 2020. The Lean mathematical library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2020, New Orleans, LA, USA, January 20-21, 2020*, Jasmin Blanchette and Catalin Hritcu (Eds.). ACM, 367–381. https://doi.org/10.1145/3372885.3373824

[25] Ralph Matthes, Kobe Wullaert, and Benedikt Ahrens. 2023. Substitution for Non-Wellfounded Syntax with Binders through Monoidal Categories. *CoRR* abs/2308.05485 (2023). https://doi.org/10.48550/ARXIV.2308.05485

[26] Paul-André Melliès. 2003. Categorical models of linear logic revisited. (Oct. 2003). https://hal.science/hal-00154229 working paper or preprint.

[27] Paul-André Melliès. 2009. *Categorical Semantics of Linear Logic.* Panorama & Synthèses, Vol. 27. Société Mathématique de France. https://www.irif.fr/~mellies/papers/panorama-submitted.pdf

[28] Ulf Norell. 2008. Dependently Typed Programming in Agda. In *Advanced Functional Programming, 6th International School, AFP 2008, Heijen, The Netherlands, May 2008, Revised Lectures (Lecture Notes in Computer Science, Vol. 5832)*, Pieter W. M. Koopman, Rinus Plasmeijer, and S. Doaitse Swierstra (Eds.). Springer, 230–266. https://doi.org/10.1007/978-3-642-04652-0_5

[29] Peter W. O'Hearn. 2003. On bunched typing. *J. Funct. Program.* 13, 4 (2003), 747–796. https://doi.org/10.1017/S0956796802004495

[30] Erik Palmgren. 2017. On Equality of Objects in Categories in Constructive Type Theory. In *23rd International Conference on Types for Proofs and Programs, TYPES 2017, May 29-June 1, 2017, Budapest, Hungary (LIPIcs, Vol. 104)*, Andreas Abel, Fredrik Nordvall Forsberg, and Ambrus Kaposi (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 7:1–7:7. https://doi.org/10.4230/LIPIcs.TYPES.2017.7

[31] Stefano Piceghello. 2021. *Coherence for Monoidal and Symmetric Monoidal Groupoids in Homotopy Type Theory.* The University of Bergen. https://bora.uib.no/bora-xmlui/handle/11250/2830640

[32] Gordon D. Plotkin. 1993. Type Theory and Recursion (Extended Abstract). In *Proceedings of the Eighth Annual Symposium on Logic in Computer Science (LICS '93), Montreal, Canada, June 19-23, 1993.* IEEE Computer Society, 374. https://doi.org/10.1109/LICS.1993.287571

[33] R. A. G. Seely. 1989. Linear logic, $*$-autonomous categories and cofree coalgebras. In *Categories in computer science and logic (Boulder, CO, 1987).* Contemp. Math., Vol. 92. Amer. Math. Soc., Providence, RI, 371–382. https://doi.org/10.1090/conm/092/1003210

[34] Eugene W. Stark. 2017. Monoidal Categories. *Arch. Formal Proofs* 2017 (2017). https://www.isa-afp.org/entries/MonoidalCategory.shtml

[35] The Coq Development Team. 2022. The Coq Proof Assistant. https://doi.org/10.5281/zenodo.7313584

[36] The 1Lab Development Team. [n. d.]. The 1Lab. https://1lab.dev

[37] Andrea Vezzosi, Anders Mörtberg, and Andreas Abel. 2019. Cubical agda: a dependently typed programming language with univalence and higher inductive types. *Proc. ACM Program. Lang.* 3, ICFP (2019), 87:1–87:29. https://doi.org/10.1145/3341691

[38] Vladimir Voevodsky, Benedikt Ahrens, Daniel Grayson, et al. [n. d.]. UniMath — a computer-checked library of univalent mathematics. Available at http://unimath.org. https://doi.org/10.5281/zenodo.7848572

[39] Makarius Wenzel, Lawrence C. Paulson, and Tobias Nipkow. 2008. The Isabelle Framework. In *Theorem Proving in Higher Order Logics, 21st International Conference, TPHOLs 2008, Montreal, Canada, August 18-21, 2008. Proceedings (Lecture Notes in Computer Science, Vol. 5170)*, Otmane Aït Mohamed, César A. Muñoz, and Sofiène Tahar (Eds.). Springer, 33–38. https://doi.org/10.1007/978-3-540-71067-7_7

[40] Kobe Wullaert, Ralph Matthes, and Benedikt Ahrens. 2022. Univalent Monoidal Categories. In *28th International Conference on Types for Proofs and Programs, TYPES 2022, June 20-25, 2022, LS2N, University of Nantes, France (LIPIcs, Vol. 269)*, Delia Kesner and Pierre-Marie Pédrot (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 15:1–15:21. https://doi.org/10.4230/LIPIcs.TYPES.2022.15

Received 2023-09-19; accepted 2023-11-25