Logic and linear algebra: an introduction

Daniel Murfet

April 23, 2015

Abstract

We give an introduction to computation and logic tailored for algebraists, with a focus on explaining how to represent programs in the λ-calculus and proofs in linear logic as linear maps between vector spaces. The interesting part of this vector space semantics is based on the cofree cocommutative coalgebra of Sweedler [72] and the recent explicit computations of liftings in [62].

1 Introduction

In mathematical logic the objects of study are *proofs*, just as manifolds, knots or representations are the objects in other disciplines of mathematics. No mathematician needs to be told that a proof may be a complicated object, but it may come as a surprise that some of the techniques invented by logicians to study proofs are closely related to constructions in modern algebra and category theory, such as symmetric monoidal categories and functors between them. For example, in *linear logic* [36] any proof of a formula can be viewed as a decorated graph called a proof-net which is similar in some ways to a bordism or a string diagram, and the semantics of linear logic [59] can be intuitively understood in terms of functors from a symmetric monoidal category of formulas (the objects) and proofs (the morphisms) to other monoidal categories, including the category of vector spaces.[1]

This invites a comparison to the subject of topological field theory [6, 77] which may be described as the study of monoidal functors from categories of bordisms to the category of vector spaces. However, there is a crucial difference which makes the semantics of linear logic more interesting in some ways than representations of a category of bordisms: proofs are *nonlinear* objects, and their semantics is therefore highly infinite. To illustrate how this nonlinearity manifests itself in algebraic semantics, suppose we are given rings A, B and a B-A-bimodule X. A basic operation is the functor

$$M \longmapsto X \otimes_A M \tag{1.1}$$

[1]Unfortunately there does not yet appear to be a formal statement in the literature to the effect that the set of categorical semantics of linear logic is in bijection with functors out of some free category generated by the syntax, we nonetheless feel this is a helpful motivating perspective.

from the category of left A-modules to the category of left B-modules. By analogy with the language of analysis, X is sometimes referred to as a "kernel". This kind of operation on modules is fundamental in the algebra of higher-dimensional topological field theory where defects between field theories are represented by bimodules.

An operation that does not arise as naturally in that context is the following "quadratic" map defined on A-A-bimodules X (that is, in the situation where $A = B$)

$$X \longmapsto X \otimes_A X \,. \tag{1.2}$$

This operation is nonlinear and cannot be represented by tensoring with some bimodule. Nonetheless there *is* a good notion of such nonlinear kernels: they are called *programs*, and the operation above is an incarnation of the Church numeral $\underline{2}$ in the λ-calculus. In this note we will discuss Church numerals and their semantics in a much simpler setting, the vector space semantics of linear logic. Making the above comments about bimodules and programs precise requires the construction of *bicategorical* semantics of linear logic, where the "squaring program" $\underline{2}$ of (1.2) can be interpreted as a kind of bundle over the space parametrising suitable bimodules X, whose fibre over a point $[X]$ is the square $X \otimes_A X$. This idea will be made precise in [**?**, **?**] using matrix factorisations (which are bimodules equipped with an additional operator) and a bicategory which extends the bicategory of Landau-Ginzburg models studied in the context of topological field theory [**?**].

Such novel semantics defined using algebra might be of interest of a proof theorist, but what is there in logic for the algebraist? The author cannot presume to speak for experts in logic, but he can explain the source of his own fascination with the subject. After the work of Turing, Gödel and Church [71] computation has become a fundamental concept in mathematics, and we have a good understanding of which functions are computable. But what kind of thing is the *process* of computation? One answer is that computation is what happens when a Turing machine is iterated, during the β-reduction process of the λ-calculus, or during the cut-elimination process of logic. These models of computation are equivalent, but each is tied to specific syntax. What is the common essence of these processes? This is a question as deep as "What is space?" and it would be absurd to expect conclusive answers [28]. Nonetheless such questions generate interesting mathematics, a recent example being homotopy-type theory [75], and the tools developed in the last few decades of research into higher categories, topological field theory and algebraic geometry offer many interesting avenues of concrete investigation.

Our aim in this note is to give an opinionated introduction to linear logic, highlighting two aspects that we think are of particular interest for algebraists. We hope that this will help provide some background and motivation for the author's papers on the subject [61]. Most of what we have to say is well-known, with the exception of some aspects of the vector space semantics in Section 3.1. There are many aspects of logic and its connections with other subjects that we cannot cover: for a well-written account of analogies between logic, topology and physics we recommend the survey of Baez and Stay [7].

The first aspect of logic that we choose for special emphasis is the already mentioned *nonlinearity* represented in linear logic by the exponential modality, which corresponds in the vector space semantics to forming from a vector space V the universal cocommutative coalgebra $!V$ whose coproduct is the universal way of duplicating elements of V. Through an embedding of a prototypical programming language called the λ-calculus into linear logic we will see how the exponential modality is the aspect of logic responsible for the capability of programs to use their inputs more than once.

The second particularly interesting aspect of logic is *cut-elimination*. This is a fundamental feature of logic going back to Gentzen [34] which may be described (superficially) as the answer to the following question: given two composable functions f, g given by algorithms, what is the simplest form of the algorithm computing $g \circ f$? LINK BACK to earlier comments about what is computation This is linked to deep questions about the nature of computation, as elucidated by Girard [41].

The outline of the article is as follows: in Section 2 we introduce the λ-calculus, intuitionistic logic and linear logic, with an emphasis on the role of duplication. In Section 3 we assign to every proof in intuitionistic linear logic a string diagram and corresponding linear map, and give a detailed example. In Section 4 we turn to cut-elimination, examining a mildly non-trivial example in detail. In Appendix A we examine tangent maps in connection with proofs in linear logic.

Acknowledgements. Thanks to Nils Carqueville, Jesse Burke, and Andante.

Throughout k is an algebraically closed field of characteristic zero.

2 Logic

In this brief introduction to logic we explain what a program is in the λ-calculus, in what sense proofs can be thought of as programs, and how a careful study of duplication in the λ-calculus and logic leads to the refinement by Girard in linear logic. We also introduce our basic example, the Church numeral $\underline{2}$, which in one form or another will occupy our attention for this entire note. For more details see [44, 59] or, for the reader with a lunch break to spare [67].

2.1 The λ-calculus

The theoretical underpinning of programming languages like Lisp [56] is the λ-calculus of Church [24, 69] which captures in a precise mathematical form the concepts of *variables* and *substitution*. The λ-calculus is determined by its terms and a rewrite rule on those terms. The terms are to be thought of as programs, and the rewrite rule as the method of execution of programs. A *term* in the λ-calculus is either one of a countable number of variables $x_0, x_1, x_2, \dots$ or an expression of the type

$$(M\ N) \quad \text{or} \quad (\lambda x\,.\,M) \tag{2.1}$$

where M, N are terms and x is any variable. The terms of the first type are called *function applications* while those of the second type are called *lambda abstractions*. The rewrite rule, called *β-reduction*, is generated by the following basic rewrite rule

$$((\lambda x\,.\,M)\,N) \longrightarrow_\beta M[N/x] \tag{2.2}$$

where M, N are terms, x is a variable, and $M[N/x]$ denotes the term M with all free occurrences of x replaced by N.[2] This rewrite rule generates an equivalence relation on terms called *β-equivalence* which we will write as $M =_\beta N$ [69, §2.5].

We think of the lambda abstraction $(\lambda x\,.\,M)$ as a program with *input* x and *body* M, so that the β-reduction step in (2.2) has the interpretation of our program being fed the input term N which is subsequently bound to x throughout M. A term is *normal* if there are no sub-terms of the type on the left hand side of (2.2). In the λ-calculus *computation* occurs when two terms are coupled by a function application in such a way as to create a term which is not in normal form: then (possibly numerous) β-reductions are performed until a normal form (the output) is reached.[3]

An important example is the following program[4]

$$T := (\lambda y\,.\,(\lambda x\,.\,(y\,(y\ x))))\,.$$

We can understand the "meaning" of this term by seeing what happens under β-reduction when we pair it with other terms. For a term M, we have:

$$(T\,M) = ((\lambda y\,.\,(\lambda x\,.\,(y\,(y\ x))))\,M) \longrightarrow_\beta (\lambda x\,.\,(M\,(M\ x)))\,. \tag{2.3}$$

We see that the result of feeding M to T is itself a program which takes a single input x and returns the resulting of computing $(M\,(M\ x))$. Let's see what happens when we feed this resulting program some other input N

$$((T\,M)\,N) \longrightarrow_\beta ((\lambda x\,.\,(M\,(M\ x)))\,N) \longrightarrow_\beta (M\,(M\ N))\,.$$

This calculation shows how $(T\,M)$ behaves like M^2, in the sense that on an input N it returns M applied twice to N. Obviously we can modify T by nesting more than two function applications, and in this way one defines for each integer $n \geq 0$ a term $\underline{n}$ called

[2]There is a slight subtlety here since we may have to rename variables in order to avoid free variables in N being "captured" as a result of this substitution, see [69, §2.3].

[3]Not every λ-term may be reduced to a normal form by β-reduction because this process does not necessarily terminate. However if it does terminate then the resulting reduced term is canonically associated to the original term; this is the content of Church-Rosser theorem [69, §4.2].

[4]In fact we deal throughout with terms up to an equivalence relation called α-conversion, which ensures that variable names are not important. For example, T is equivalent to $(\lambda z\,.\,(\lambda t\,.\,(z\,(z\ t))))$

the nth *Church numeral* [69, §3.2]. This program has the property that $(\underline{n}\, M)$ behaves like M^n and in particular T is the Church numeral $\underline{2}$.

To put this into a broader context: once the integers have been translated into terms in the λ-calculus it makes sense to say that a program F *computes* a function $f : \mathbb{N} \longrightarrow \mathbb{N}$ if for each $n \geq 0$ we have $(F\ \underline{n}) =_\beta \underline{f(n)}$. A function f is called *computable* if there is a term F in the λ-calculus with this property.

So far we have considered programs only in the *untyped* λ-calculus. The reader familiar with programming will know that functions in commonly used programming languages are typed: for instance, a function computing the nth Fibonacci number in C takes an input of type "int" – the integer n – and returns an output also of type "int". In the *simply-typed* λ-calculus the Church numeral $\underline{2}$ is the same as before, but with type annotations

$$T_{typed} := (\lambda y^{A\to A} \,.\, (\lambda x^A \,.\, (y\,(y\ x))))\,.$$

These annotations indicate that the first input y is restricted to be of the type of a function from A to A (that is, of type $A \to A$) while x is restricted to be of type A. Overall T_{typed} takes a term of type $A \to A$ and returns another term of the same type, so it is a term of type $(A \to A) \to (A \to A)$. For a more complete discussion see [69, §6].

We have now defined the λ-calculus and introduced our basic example of a program, the Church numeral $\underline{2}$. Moreover, we have seen that it is natural think about the "meaning" of programs as *functions* on equivalence classes of terms: for example, if D is the set of β-equivalence classes of terms of type $A \to A$ then there is a function

$$D \longrightarrow D\,, \qquad [M] \mapsto [(T_{typed}\ M)]$$

where $[-]$ denotes β-equivalence classes. This function contains a great deal of information about the term T_{typed}, but this information is not very accessible in light of the fact that we have little handle on the set D. This situation may be compared to the problem of trying to understand a complicated group, and may be approached in the same way: by studying *representations* of programs in the λ-calculus as transformations on mathematical objects. This is referred to as *denotational semantics*, and was initiated as a field by Scott [66] who found a representation of the λ-calculus as continuous maps of certain lattices.

This motivates the following question, which we will keep in mind during the subsequent discussion of linear logic:

Question 2.1. Is there a natural representation of programs in the simply-typed λ-calculus, for example T_{typed}, as linear maps between vector spaces?

Here by a "natural" representation we have in mind something more interesting than linear maps between free vector spaces on sets like D. The obstruction to realising this aim is clear: it would be natural to associate to T_{typed} the map $\alpha \mapsto \alpha^2$ on endomorphisms of some vector space V, but this is not linear. This non-linearity can be traced to the duplication of terms that takes place during β-reduction. To see this, notice how in (2.3) a single occurrence of M on the left becomes a pair of M's on the right, so that during the substitution process the term M is duplicated. This duplication is not linear, at least not in any naive sense, because in general there is no linear map

$$\mathrm{End}_k(V) \longrightarrow \mathrm{End}_k(V) \otimes \mathrm{End}_k(V) \tag{2.4}$$

sending α to $\alpha \otimes \alpha$ for all α, where $\mathrm{End}_k(V)$ denotes the space of linear maps $V \longrightarrow V$. One way to represent T_{typed} as a linear map is to replace $\mathrm{End}_k(V)$ by the universal vector space over it which *is* equipped with a duplication map satisfying some natural axioms; this is the universal cocommutative coalgebra $!\,\mathrm{End}_k(V)$. Using this coalgebra we may associate to T_{typed} a linear map from which the non-linear map $\alpha \mapsto \alpha^2$ may be recovered; see Example 2.4 below and Lemma 3.7.

But we are getting ahead of ourselves. The first step is to reformulate the λ-calculus in terms of classical logic, then we identify the part of the logic that is responsible for this duplication – called the contraction rule – and explain how the contraction rule is refined in linear logic in such a way as to lead naturally to the coalgebra $!\,\mathrm{End}_k(V)$.

2.2 Intuitionistic logic

We present part of the sequent calculus of intuitionistic logic [34]. The logic is defined by its *connectives*, by which propositional atoms $x, y, z, \ldots$ may be combined to form more complicated *formulas*. A *sequent* is an expression of the form

$$A_1, \ldots, A_n \vdash B$$

with formulas $A_1, \ldots, A_n, B$ connected by a *turnstile* $\vdash$. In classical logic, such a sequent would be read as "the conjunction of the A_i implies B" but the interpretation in linear logic is more subtle. A *proof* of a sequent is a series of deductions, beginning from *axioms* of the form $A \vdash A$, which terminates with the given sequent. At each step the form of the deduction must belong to a list of *deduction rules*, which is part of the definition of the logic. Such proofs are often organised into a proof tree, as shown in (2.5) below. This style of formal logic was introduced by Gentzen [34] in order to prove the consistency of Peano arithmetic, and his methods are a fundamental contribution to proof theory.

In propositional intuitionistic logic[5] the connectives are $\vee, \wedge$ and $\Rightarrow$ and examples of formulas include

$$(x \Rightarrow y) \wedge z, \quad ((x \wedge y) \vee z) \Rightarrow y \quad x \vee (y \Rightarrow (z \vee z)), \quad \ldots.$$

Some of the deduction rules of the logic are:

[5]The modifier *propositional* means that we do not include quantifiers $\forall, \exists$ and *intuitionistic* means that sequents are restricted to have only a single formula on the right hand side. This corresponds to the lack of the law of the excluded middle in intuitionistic logic [59, p.14].

$$\frac{}{A \vdash A} \qquad \frac{\Gamma_1 \vdash A \quad \Gamma_2, B \vdash C}{\Gamma_1, \Gamma_2, A \Rightarrow B \vdash C} \Rightarrow L \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B} \Rightarrow R$$

$$\frac{\Gamma \vdash A \quad A \vdash B}{\Gamma \vdash B}\,\text{cut} \qquad \frac{\Gamma, A, A \vdash B}{\Gamma, A \vdash B}\,\text{ctr} \ .$$

Here $\Gamma, \Gamma_1, \Gamma_2$ stand for lists of formulas. The format of a deduction rule is that from a proof of each item in the numerator, for instance $\Gamma_1 \vdash A$ and $\Gamma_2, B \vdash C$, we may generate using the deduction rule a proof of the denominator $\Gamma_1, \Gamma_2, A \Rightarrow B \vdash C$. The horizontal bar is labelled with an abbreviation of the name of the deduction rule. The first rule, called the *axiom rule*, has an empty numerator and thus may be introduced at any time; it represents the principle that from A we may obtain A. The second last rule, the *cut rule*, is modus ponens, while in the last rule, the *contraction rule*, we see the principle that if B may be deduced from two copies of A then it may be deduced from one.

An example of a proof of $\vdash (A \Rightarrow A) \Rightarrow (A \Rightarrow A)$ is

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A \vdash A} \quad \dfrac{\dfrac{}{A \vdash A} \quad \dfrac{}{A \vdash A}}{A, A \Rightarrow A \vdash A} \Rightarrow L}{A, A \Rightarrow A, A \Rightarrow A \vdash A} \Rightarrow L}{A \Rightarrow A, A \Rightarrow A \vdash A \Rightarrow A} \Rightarrow R}{A \Rightarrow A \vdash A \Rightarrow A}\,\text{ctr}}{\vdash (A \Rightarrow A) \Rightarrow (A \Rightarrow A)} \Rightarrow R \qquad (2.5)$$

This depiction is often referred to as a *proof tree*. Obviously this is not the only proof of this sequent, for example we could deduce it in two steps from an axiom rule $A \Rightarrow A \vdash A \Rightarrow A$ followed by the same final step as in (2.5). Usually a sequent has many proofs.

If γ denotes the proof (2.5) taken up to its penultimate step and τ denotes some proof of a sequent $\Gamma \vdash A \Rightarrow A$ then using an instantiation of the cut rule we may generate a new proof of $\Gamma \vdash A \Rightarrow A$ given by the proof tree

$$\frac{\begin{matrix}\tau \\ \vdots \\ \Gamma \vdash A \Rightarrow A\end{matrix} \qquad \begin{matrix}\gamma \\ \vdots \\ A \Rightarrow A \vdash A \Rightarrow A\end{matrix}}{\Gamma \vdash A \Rightarrow A}\,\text{cut} \qquad (2.6)$$

This yields up an interpretation of (2.5) as a *function* which maps proofs of $\Gamma \vdash A \Rightarrow A$ to other proofs of $\Gamma \vdash A \Rightarrow A$. At this point we notice the similarity between γ and the program T_{typed} which maps input terms of type $A \to A$ to output terms of type $A \to A$.

This similiarity is an instance of the Curry-Howard isomorphism [69, §6] which matches *types* in the simply-typed λ-calculus (e.g. $A \to A$) with *formulas* in propositional intuitionistic logic (e.g. $A \Rightarrow A$) and *programs* in the λ-calculus with *proofs* in logic. This correspondence pairs (2.5) and γ with T_{typed}. In light of this equivalence, in the proof γ of $A \Rightarrow A \vdash A \Rightarrow A$ it is natural to view the formula on the left hand side of the turnstile as the "input" and the one on the right hand side as the "output". The contraction rule is therefore responsible for the duplication of inputs and, moreover, since the contraction rule may be used at any time on any formula to the left of the turnstile, the duplication of inputs is unrestricted.[6] Although we will not explain the details, by going a little deeper into the content of the Curry-Howard isomorphism one can see how the use of the contraction rule in the penultimate step of (2.5) matches precisely with the duplication of the input term M that takes place in (2.3).

Here we touch on an important point: the feature of logic which corresponds under the Curry-Howard isomorphism to β-reduction in the λ-calculus. The type of a program does not change under β-reduction, so this must be some kind of transformation of proofs of a given sequent. This transformation is called *cut-elimination*. It consists of a collection of transformations on proofs, each of which, when applied to a proof with cuts, generates a new proof of the same sequent with cuts of lower complexity. Gentzen's main theorem in [34] states that his cut-elimination algorithm produces, in a finite number of steps, a proof without any use of the cut rule at all; such a proof is called *cut-free*. Many important properties of logic such as consistency follow immediately from this fact; for more discussion see [59] and [44, Chapter 13]. We will have more to say about cut-elimination in linear logic in Section 4.

The upshot is that the simply-typed λ-calculus is "the same thing" as propositional intuitionistic logic, and the duplication of terms during substitution in the λ-calculus matches with applications of the contraction rule in logic. Thus, if we want to represent programs as linear maps, it is sufficient to construct a linear representation of *logic*. Moreover, since we have identified duplication of terms as the main obstacle to linearisation of the λ-calculus, a detailed study of the contraction rule appears to be the key. Indeed, this is the insight of Girard with linear logic.

2.3 Linear logic

The linear logic of Girard [36, 59] is a refinement of classical logic in which the contraction rule is restricted to formulas that are prefixed with a new unary connective $!$ called the exponential modality. A hypothesis $!A$ may be used in a proof infinitely many times as in classical logic, but a bare formula A may only be used once[7] (the proof is *linear* in A). The other connectives are also refined, for instance $\Rightarrow$ is refined to a linear implication $\multimap$, with the former being recovered from the latter by the translation $A \Rightarrow B = !A \multimap B$.

We consider propositional intuitionistic linear logic without additives (henceforth re-

[6]The significance of the contraction rule and duplication as the source of infinity in mathematics is beyond the scope of the present note, but see for instance [41, p.78].

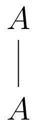
[7]To an algebraist, this may sound as strange as an admonition to only use Nakayama's lemma once a day. After all, isn't the truth endlessly reusable? In reply, we would ask the reader to consider again the interpretation of the proof γ in (2.6) as a function mapping input proofs to output proofs. The act of computing these outputs is a physical process, whether it takes place in a machine or a human mind, and it involves the allocation and deallocation of finite resources. The insight of Girard with linear logic is that far from being implementation or engineering details, these acts of allocation and deallocation are in fact logical, and of fundamental importance.

ferred to simply as *linear logic*). The adjective *intuitionistic* means that the right hand side of a sequent is constrained to have precisely one formula, while *propositional* means that we omit quantifiers.[8] Linear logic has propositional variables $x, y, z, \ldots$, two binary connectives $\multimap$ (linear implication), $\otimes$ (tensor) and a single unary connective $!$ (the exponential). There is a single constant 1. Examples of formulas include

$$(x \multimap y) \otimes z, \quad !(x \multimap 1) \multimap y, \quad !!x \otimes !(y \multimap !z), \quad \ldots .$$

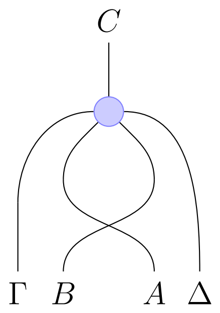
The deduction rules of linear logic are given on the left hand side of the following series of diagrams (2.7) – (2.19). The diagrams on the right should be ignored until Section 3. In all deduction rules, the sets Γ and Δ may be *empty* and, in particular, the promotion rule may be used with an empty premise. In the promotion rule, $!\Gamma$ stands for a list of formulas each of which is preceeded by an exponential modality, for example $!A_1, \ldots, !A_n$.

(Axiom): $$\frac{}{A \vdash A}$$



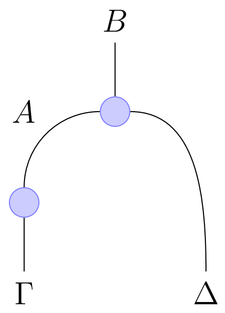
(2.7)

(Exchange): $$\frac{\Gamma, A, B, \Delta \vdash C}{\Gamma, B, A, \Delta \vdash C}$$



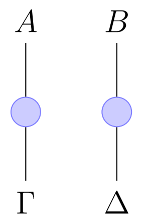
(2.8)

(Cut): $$\frac{\Gamma \vdash A \qquad A, \Delta \vdash B}{\Gamma, \Delta \vdash B}\,\text{cut}$$



(2.9)

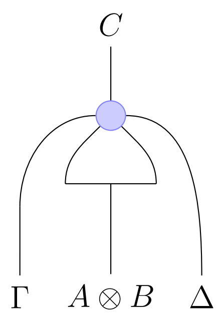
(Right $\otimes$): $$\frac{\Gamma \vdash A \qquad \Delta \vdash B}{\Gamma, \Delta \vdash A \otimes B}\,\otimes\text{-}R$$



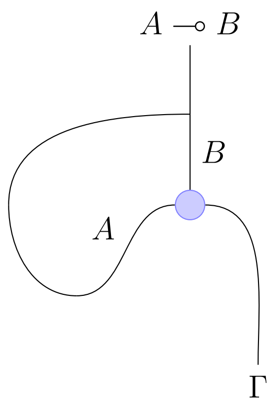
(2.10)

[8]The additives may be included in the obvious way, we omit them simply because our examples do not involve them. At the moment we do not understand quantifiers.

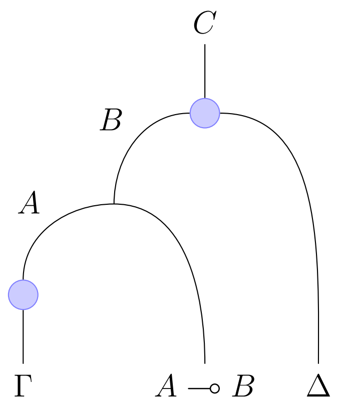
(Left $\otimes$): $\dfrac{\Gamma, A, B \vdash C}{\Gamma, A \otimes B \vdash C}\,\otimes\text{-}L$ (2.11)



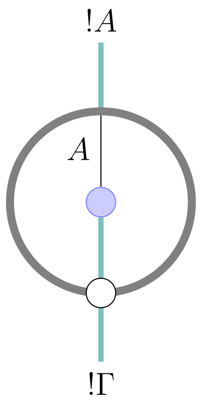
(Right $\multimap$): $\dfrac{A, \Gamma \vdash B}{\Gamma \vdash A \multimap B}\,\multimap R$ (2.12)



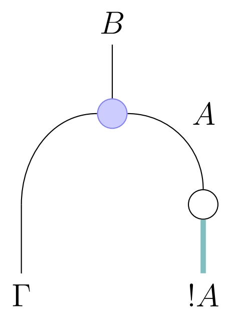
(Left $\multimap$): $\dfrac{\Gamma \vdash A \qquad B, \Delta \vdash C}{\Gamma, A \multimap B, \Delta \vdash C}\,\multimap L$ (2.13)



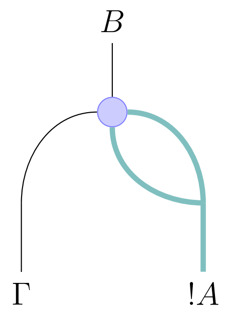
(Promotion): $\dfrac{!\Gamma \vdash A}{!\Gamma \vdash !A}\,\text{prom}$ (2.14)



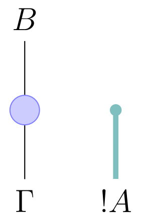
(Dereliction): $\dfrac{\Gamma, A \vdash B}{\Gamma, !A \vdash B}\,\text{der}$ (2.15)



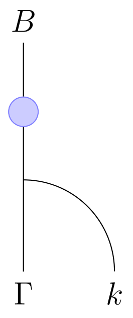
(Contraction): $\dfrac{\Gamma, !A, !A \vdash B}{\Gamma, !A \vdash B}\,\text{ctr}$ (2.16)



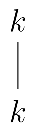
(Weakening): $\dfrac{\Gamma \vdash B}{\Gamma, !A \vdash B}\,\text{weak}$ (2.17)



(Left 1): $\dfrac{\Gamma \vdash A}{\Gamma, 1 \vdash A}\,1\text{-}L$ (2.18)



(Right 1): $\dfrac{}{\vdash 1}\,1\text{-}R$ (2.19)



For the examples in this paper, the relevant deduction rules are the axiom rule, the cut rule, the right and left $\multimap$ introduction rules, the contraction rule and the promotion and dereliction rules. With the exception of the last two, the others have the same form as the rules in classical logic, except with $\multimap$ instead of $\Rightarrow$ and with the contraction rule restricted to formulas $!A$. For more on classical versus linear logic, see Remark 2.5.

It is standard practice to use the words *formula* and *type* interchangeably in systems like linear logic, and this is especially convenient when a formula denotes a "data type" like integers or lists, as in the following example.

Example 2.2. For any type A the type of *integers on A* is

$$\mathbf{int}_A = !(A \multimap A) \multimap (A \multimap A)\,. \tag{2.20}$$

The proof of the linear logic version of the Church numeral $\underline{2}$ is

$$\begin{prooftree}
\AxiomC{}
\UnaryInfC{$A \vdash A$}
\AxiomC{}
\UnaryInfC{$A \vdash A$}
\AxiomC{}
\UnaryInfC{$A \vdash A$}
\RightLabel{$\multimap L$}
\BinaryInfC{$A, A \multimap A \vdash A$}
\RightLabel{$\multimap L$}
\BinaryInfC{$A, A \multimap A, A \multimap A \vdash A$}
\RightLabel{$\multimap R$}
\UnaryInfC{$A \multimap A, A \multimap A \vdash A \multimap A$}
\doubleLine
\RightLabel{der}
\UnaryInfC{$!(A \multimap A), !(A \multimap A), \vdash A \multimap A$}
\RightLabel{ctr}
\UnaryInfC{$!(A \multimap A) \vdash A \multimap A$}
\RightLabel{$\multimap R$}
\UnaryInfC{$\vdash \mathbf{int}_A$}
\end{prooftree} \tag{2.21}$$

A doubled horizontal line stands for repeated applications of a deduction rule (in this case, dereliction is applied twice). Although the proof obviously has a structure similar to (2.5) notice that there is a conversion between $A \multimap A$ and its infinite version $!(A \multimap A)$, where the duplication occurs. For each integer $n \geq 0$ there is a proof $\underline{n}$ of $\mathbf{int}_A$ constructed along similar lines, see [36, §5.3.2] and [25, §3.1].

As in intuitionistic logic, cut-elimination in linear logic generates an equivalence relation on proofs of a given sequent, with a unique cut-free proof in each equivalence class. This equivalence relation plays a role similar to diffeomorphism of bordisms in topological field theory, but is more complicated because of the presence of the exponential modality, as for example the diagrammatic transformations in Section 4 below demonstrate.

A *categorical semantics* of linear logic [59, 18] assigns to each type A an object $[\![A]\!]$ of some category and to each proof of $\Gamma \vdash A$ a morphism $[\![\Gamma]\!] \longrightarrow [\![A]\!]$ in such a way that two proofs equivalent under cut-elimination are assigned the same morphism; these objects and morphisms are called *denotations*. The connectives of linear logic become structure on the category of denotations, and compatibility with cut-elimination imposes identities relating these structures to one another. The upshot is that to define a categorical semantics the category of denotations must be a closed symmetric monoidal category equipped with a comonad, which is used to model the exponential modality [59, §7]. This is a refinement of the equivalence between simply-typed λ-calculus and cartesian closed categories due to Lambek and Scott [54]. The first semantics of linear logic were the coherence spaces of Girard [36, §3] which are a refined form of Scott's model of the λ-calculus. Models of full linear logic with negation involve the $\star$-autonomous categories of Barr [9, 10, 11] and the extension to include quantifiers involves indexed monoidal categories [68].

In this paper denotations all take place in the category $\mathcal{V}$ of k-vector spaces. We explain the denotations of types now, and leave the denotation of proofs to the next section. To this end, for a vector space V let $!V$ denote the cofree cocommutative coalgebra generated by V. We will discuss the explicit form of this coalgebra in the next section; for the moment it is enough to know that it exists and is determined up to unique isomorphism.

Definition 2.3. The *denotation* $[\![A]\!]$ of a type A is defined inductively as follows:

- The propositional variables $x, y, z, \dots$ are assigned chosen finite-dimensional vector spaces $[\![x]\!], [\![y]\!], [\![z]\!], \dots$;
- $[\![1]\!] = k$;
- $[\![A \otimes B]\!] = [\![A]\!] \otimes [\![B]\!]$;
- $[\![A \multimap B]\!] = [\![A]\!] \multimap [\![B]\!]$ which is notation for $\operatorname{Hom}_k([\![A]\!], [\![B]\!])$;
- $[\![!A]\!] = ![\![A]\!]$.

The denotation of a group of formulas $\Gamma = A_1, \dots, A_n$ is their tensor product

$$[\![\Gamma]\!] = [\![A_1]\!] \otimes \cdots \otimes [\![A_n]\!] \,.$$

If Γ is empty then $[\![\Gamma]\!] = k$.

We continue with the Church numeral $\underline{2}$ as our motivating example:

Example 2.4. Let A be a type whose denotation is $V = [\![A]\!]$. Then from (2.20),

$$[\![\mathbf{int}_A]\!] = [\![!(A \multimap A) \multimap (A \multimap A)]\!] = \operatorname{Hom}_k(!\operatorname{End}_k(V), \operatorname{End}_k(V)) \,.$$

Skipping ahead a little, the denotation of the proof $\underline{2}$ of $\vdash \mathbf{int}_A$ will be a morphism

$$[\![\underline{2}]\!] : k \longrightarrow [\![\mathbf{int}_A]\!] = \operatorname{Hom}_k(!\operatorname{End}_k(V), \operatorname{End}_k(V)) \,, \tag{2.22}$$

or equivalently, a linear map $!\operatorname{End}_k(V) \longrightarrow \operatorname{End}_k(V)$. What is this linear map? It turns out (see Example 3.5 below for details) that it is the composite

$$!\operatorname{End}_k(V) \xrightarrow{\Delta} !\operatorname{End}_k(V) \otimes !\operatorname{End}_k(V) \xrightarrow{d \otimes d} \operatorname{End}_k(V)^{\otimes 2} \xrightarrow{-\circ-} \operatorname{End}_k(V) \tag{2.23}$$

where Δ is the coproduct, d is the universal map, and the last map is the composition. How to reconcile this linear map with the corresponding program in the λ-calculus, which has the meaning "square the input function"? As we will explain below, for $\alpha \in \operatorname{End}_k(V)$ there is a naturally associated element $|o\rangle_\alpha \in !\operatorname{End}_k(V)$ with the property that

$$\Delta|o\rangle_\alpha = |o\rangle_\alpha \otimes |o\rangle_\alpha, \qquad d|o\rangle_\alpha = \alpha \,.$$

Then $[\![\underline{2}]\!]$ maps this element to

$$|o\rangle_\alpha \longmapsto |o\rangle_\alpha \otimes |o\rangle_\alpha \longmapsto \alpha \otimes \alpha \longmapsto \alpha \circ \alpha \,. \tag{2.24}$$

This demonstrates how the coalgebra $!\operatorname{End}_k(V)$ may be used to encode nonlinear maps, such as squaring an endomorphism.

Remark 2.5. We have already mentioned the equivalence of the simply-typed λ-calculus and propositional intuitionistic logic under the Curry-Howard isomorphism [69, §6.5]. There is an embedding of propositional intuitionistic logic into propositional intuitionistic *linear* logic [36, §5.1] making use of the additive connectives of linear logic which we have omitted in the above. This means that every program in the simply-typed λ-calculus may be assigned a proof in linear logic (with additives) in such a way that β-reduction in the λ-calculus corresponds to cut-elimination in linear logic. For more on linear logic proofs as computer programs, see [53, 1, 14].

For example, if A, B denote types in propositional intuitionistic logic, and A°, B° the corresponding types in linear logic, then $(A \Rightarrow B)^\circ := (!A^\circ) \multimap B^\circ$ where for atoms A we declare $A^\circ = A$. There is a corresponding translation of proofs of $\vdash A$ to proofs of $\vdash A^\circ$. The Church numeral $\underline{2}$ is a λ-term of type $(A \to A) \to (A \to A)$ which corresponds to a proof in intuitionistic logic of the sequent $\vdash (A \Rightarrow A) \Rightarrow (A \Rightarrow A)$. If A is atomic, the translation of this type to linear logic is $\mathbf{int}'_A = !(!A \multimap A) \multimap (!A \multimap A)$. Thus a Church numeral in the λ-calculus determines a proof of $\vdash \mathbf{int}'_A$ not $\vdash \mathbf{int}_A$ under this translation. However, this translation is not necessarily the most useful or economical one because of the over-use of exponentials [36, §5.3]. For reasons of clarity we follow standard practice in using the "linear logic version" of the Church numeral $\underline{2}$ in Example 2.2 above, rather than the literal translation.

3 Diagrams and denotations

In the previous section we introduced the connectives and deduction rules of linear logic, and we associated to each type A a vector space $[\![A]\!]$. In this section we complete the construction of the vector space semantics of linear logic by assigning to each proof π of a sequent $\Gamma \vdash B$ a linear map $[\![\Gamma]\!] \longrightarrow [\![B]\!]$. We have already sketched how this assignment works in the case of the Church numeral $\underline{2}$ in Example 2.4.

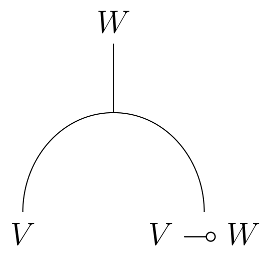
This is almost completely formal: the category of k-vector spaces has all the properties of a category of proof denotations described earlier, including the comonad ! given by taking cofree cocommutative coalgebras, so it is automatic that semantics of intuitionistic linear logic may be constructed within $\mathcal{V}$. However, to explicitly calculate the denotations of proofs we need formulas for promotion and dereliction which are not automatic: in fact, this paper seems to be the first time they have been written down. This is not as surprising as it might seem: although studying semantics of *linear* logic using vector spaces is an natural thing to do, research has focused on full linear logic with negation. This is more complicated than the intuitionistic case, because types must be interpreted by self-dual objects and this leads to topological vector spaces [9, 10, 16, 17, 43, 30, 31]. However these more sophisticated models have the disadvantage that it is inconvenient to write down denotations of proofs, which is why for this introduction we stick to the intuitionistic case. See Remark 3.9 for more on the existing literature.

Our assignment of linear maps to proofs will be presented using string diagrams. One style of string diagrams, called *proof-nets*, were introduced by Girard and have been fundamental to linear logic since the beginning of the subject [36]. However proof-nets are designed for full linear logic and this does not match a semantics involving infinite-dimensional vector spaces. Instead we use a style of diagrams which is standard in category theory, following Joyal and Street [49, 50, 55, 52]. Our recommended reference for this approach to proof-nets is Melliès [59, 58].

Let $\mathcal{V}$ denote the category of k-vector spaces (not necessarily finite dimensional). Then $\mathcal{V}$ is symmetric monoidal and for each object V the functor $V \otimes -$ has a right adjoint

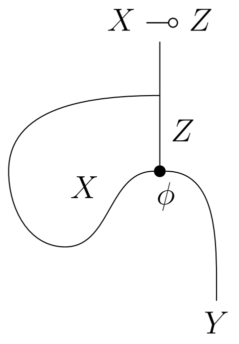
$$V \multimap - := \operatorname{Hom}_k(V, -)\,.$$

In addition to the usual diagrammatics of a symmetric monoidal category, we draw the counit $V \otimes (V \multimap W) \longrightarrow W$ as



(3.1)

The adjoint $Y \longrightarrow X \multimap Z$ of a morphism $\phi : X \otimes Y \longrightarrow Z$ is depicted as follows:[9]



(3.2)

Next we present the categorical construct corresponding to the exponential modality in terms of an adjunction, following Benton [13], see also [59, §7]. Let $\mathcal{C}$ denote the category of counital, coassociative, cocommutative coalgebras in $\mathcal{V}$. In this paper whenever we say *coalgebra* we mean an object of $\mathcal{C}$. This is a symmetric monoidal category in which the tensor product (inherited from $\mathcal{V}$) is cartesian, see [72, Theorem 6.4.5], [8] and [59, §6.5].

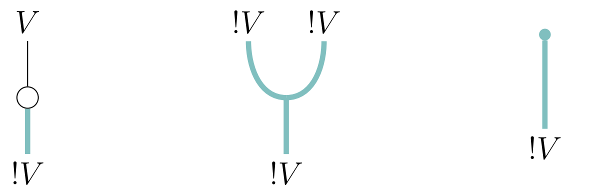
[9]This is somewhat against the spirit of the diagrammatic calculus, since the loop labelled X is not "real" and is only meant as a "picture" to be placed at a vertex between a strand labelled Y and a strand labelled $X \multimap Z$. This should not cause confusion, because we will never manipulate this strand on its own. The idea is that if X were a finite-dimensional vector space, so that $X \multimap Z \cong X^\vee \otimes Z$, the above diagram would be absolutely valid, and we persist with the same notation even when X is not dualisable. In our judgement the clarity achieved by this slight cheat justifies a little valour in the face of correctness.

By results of Sweedler [72, Chapter 6] the forgetful functor $L : \mathcal{C} \longrightarrow \mathcal{V}$ has a right adjoint R and we set $! = L \circ R$, as in the following diagram:[10]

$$\begin{tikzcd} \mathcal{C} \arrow[r, shift left, "L"] & \mathcal{V} \arrow[l, shift left, "R"] \end{tikzcd} \qquad ! = L \circ R\,.$$

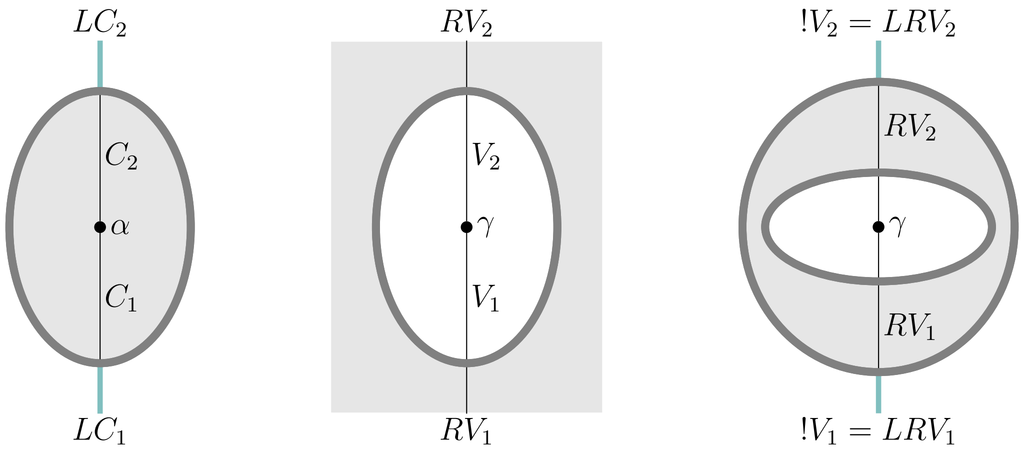
Both L and its adjoint R are monoidal functors.

For each V there is a coalgebra $!V$ and a counit of adjunction $d : !V \longrightarrow V$. Since this map will end up being the interpretation of the dereliction rule in linear logic, we refer to it as the *dereliction map*. In string diagrams it is represented by an empty circle. Although it is purely decorative, it is convenient to represent coalgebras in string diagrams drawn in $\mathcal{V}$ by thick lines, so that for $!V$ the dereliction, coproduct and counit are drawn respectively as follows:



(3.3)

In this paper our string diagrams involve both $\mathcal{V}$ and $\mathcal{C}$ and our convention is that white regions represent $\mathcal{V}$ and gray regions stand for $\mathcal{C}$. A standard way of representing monoidal functors between monoidal categories is using coloured regions [59, §5.7]. The image under L of a morphism $\alpha : C_1 \longrightarrow C_2$ in $\mathcal{C}$ is drawn as a vertex in a grey region embedded into a white region. The image of a morphism $\gamma : V_1 \longrightarrow V_2$ under R is drawn using a white region embedded in a gray plane. For example, the diagrams representing $L(\alpha)$, $R(\gamma)$ and $!\gamma = LR(\gamma)$ are respectively



[10]The existence of a right adjoint to the forgetful functor can also be seen to hold more generally as a consequence of the adjoint functor theorem [8].

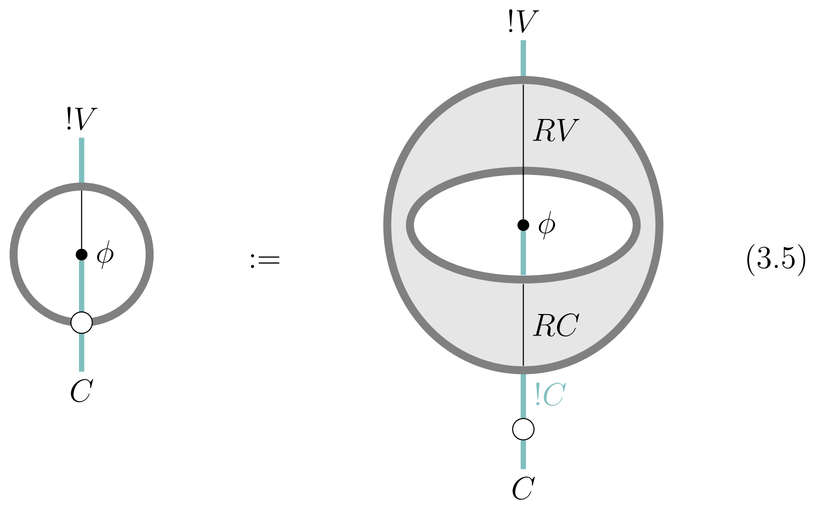
The adjunction between R and L means that for any coalgebra C and linear map $\phi : C \longrightarrow V$ there is a unique morphism of coalgebras $\Phi : C \longrightarrow\, !V$ making

$$\begin{tikzcd} C \arrow[r, "\phi"] \arrow[dr, "\Phi"'] & V \\ & !V \arrow[u, "d"'] \end{tikzcd} \tag{3.4}$$

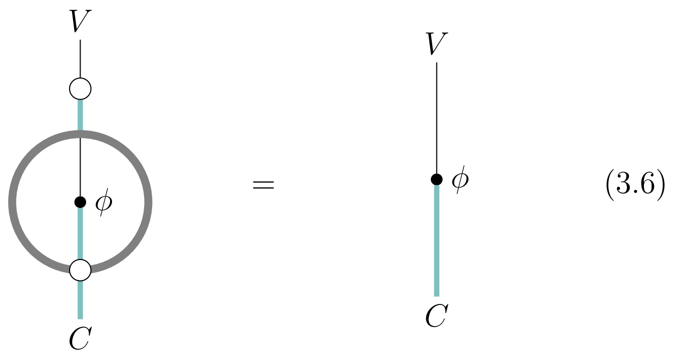
commute. The lifting Φ may be constructed as the unit followed by $!\phi$,

$$\Phi := \; C \longrightarrow\, !C \xrightarrow{\;!\phi\;}\, !V$$

and since we use an empty circle to denote the unit $C \longrightarrow\, !C$, this has the diagrammatic representation given on the right hand side of the following diagram. The left hand side is a convenient abbreviation for this morphism, that is, for the lifting Φ:



We follow the logic literature in referring to the grey circle denoting the induced map Φ as a *promotion box*. Commutativity of (3.4) is expressed by the identity



The coalgebra $!V$ and the dereliction map $!V \longrightarrow V$ satisfy a universal property and are therefore unique up to isomorphism. However, to actually understand the denotations of proofs, we will need the more explicit construction which follows from the work of Sweedler [72] and is spelt out in [62]. If V is finite-dimensional then

$$!V = \bigoplus_{P \in V} \mathrm{Sym}_P(V) \tag{3.7}$$

where $\mathrm{Sym}_P(V) = \mathrm{Sym}(V)$ is the symmetric coalgebra. If $e_1, \ldots, e_n$ is a basis for V then as a vector space $\mathrm{Sym}(V) \cong k[e_1, \ldots, e_n]$. The notational convention in [62] is to denote, for elements $\nu_1, \ldots, \nu_s \in V$, the corresponding tensor in $\mathrm{Sym}_P(V)$ using kets

$$|\nu_1, \ldots, \nu_s\rangle_P := \nu_1 \otimes \cdots \otimes \nu_s \in \mathrm{Sym}_P(V)\,. \tag{3.8}$$

And in particular, the identity element of $\mathrm{Sym}_P(V)$ is denoted by a vacuum vector

$$|o\rangle_P := 1 \in \mathrm{Sym}_P(V)\,. \tag{3.9}$$

We remark that if $\nu = 0$ then $|\nu\rangle_P = 0$ is the zero vector, which is distinct from $|o\rangle_P = 1$. We keep in mind that (3.9) denotes the case $s = 0$ of (3.8) and to avoid unwieldy notation we sometimes write $\nu_1 \otimes \cdots \otimes \nu_s \cdot |o\rangle_P$ for $|\nu_1, \ldots, \nu_s\rangle_P$. With this notation the universal map $d : !V \longrightarrow V$ is defined by

$$d|o\rangle_P = P, \quad d|\nu\rangle_P = \nu, \quad d|\nu_1, \ldots, \nu_s\rangle_P = 0 \quad s > 1\,.$$

The coproduct on $!V$ is defined by

$$\Delta|\nu_1, \ldots, \nu_s\rangle_P = \sum_{I \subseteq \{1, \ldots, s\}} |\nu_I\rangle_P \otimes |\nu_{I^c}\rangle_P \tag{3.10}$$

where I ranges over all subsets including the empty set, for a subset $I = \{i_1, \ldots, i_p\}$ we denote by ν_I the sequence $\nu_{i_1}, \ldots, \nu_{i_p}$, and I^c is the complement of I. In particular

$$\Delta|o\rangle_P = |o\rangle_P \otimes |o\rangle_P\,.$$

The counit $!V \longrightarrow k$ is defined by $|o\rangle_P \mapsto 1$ and $|\nu_1, \ldots, \nu_s\rangle_P \mapsto 0$ for $s > 0$.

When V is infinite-dimensional we may define $!V$ as the direct limit over the coalgebras $!W$ for finite-dimensional subspaces $W \subseteq V$. These are sub-coalgebras of $!V$, and we may therefore use the same notation as in (3.8) to denote arbitrary elements of $!V$. Moreover the coproduct, counit and universal map d are given by the same formulas; see [62, §2.1]. A proof of the fact that the map $d : !V \longrightarrow V$ described above is universal among linear maps to V from cocommutative coalgebras is given in [62, Theorem 2.18], but as has been mentioned this is originally due to Sweedler [72], see [62, Appendix B].

To construct semantics of linear logic we also need an explicit description of liftings, as given by the next theorem which is [62, Theorem 2.20]. For a set X the set of partitions of X is denoted $\mathcal{P}_X$.

Theorem 3.1. *Let W, V be vector spaces and $\phi : !W \longrightarrow V$ a linear map. The unique lifting to a morphism of coalgebras $\Phi : !W \longrightarrow !V$ is given by*

$$\Phi|\nu_1, \ldots, \nu_s\rangle_P = \sum_{C \in \mathcal{P}_{\{1,\ldots,s\}}} \phi|\nu_{C_1}\rangle_P \otimes \cdots \otimes \phi|\nu_{C_l}\rangle_P \cdot |o\rangle_Q \tag{3.11}$$

for $P, \nu_1, \ldots, \nu_s \in W$, where $Q = \phi|o\rangle_P$ and l denotes the length of the partition C.

Example 3.2. The simplest example of a coalgebra is the field k. Any $P \in V$ determines a linear map $k \longrightarrow V$ whose lifting to a morphism of coalgebras $k \longrightarrow !V$ sends $1 \in k$ to the vacuum $|o\rangle_P$, as shown in the commutative diagram

$$\begin{tikzcd} k \arrow[r, "P"] \arrow[dr, dotted, "|o\rangle_P"'] & V \\ & !V \arrow[u, "d"'] \end{tikzcd} \quad . \tag{3.12}$$

Such liftings arise from promotions with empty premises, e.g. the proof

$$\dfrac{\dfrac{\overline{A \vdash A}}{\vdash A \multimap A}\multimap R}{\vdash !(A \multimap A)}\text{prom}$$

Incidentally, this explains why $[\![!A]\!] = \mathrm{Sym}([\![A]\!])$ does not lead to semantics of linear logic, since the denotation of the above proof is a morphism of coalgebras $k \longrightarrow !\,\mathrm{End}_k([\![A]\!])$ whose composition with dereliction yields the map $k \longrightarrow \mathrm{End}_k([\![A]\!])$ sending $1 \in k$ to the identity. But this map does not admit a lifting into the symmetric coalgebra, because it produces an infinite sum. However the symmetric coalgebra *is* universal in a restricted sense and is (confusingly) sometimes also called a cofree coalgebra; see [74, §4]. For further discussion of the symmetric coalgebra in the context of linear logic see [19, 60].

3.1 The vector space semantics

Recall from Definition 2.3 the definition of $[\![A]\!]$ for each type A.

Definition 3.3. The *denotation* $[\![\pi]\!]$ of a proof π of $\Gamma \vdash B$ is a linear map $[\![\Gamma]\!] \longrightarrow [\![B]\!]$ defined by inductively assigning a string diagram to each proof tree; by the basic results of the diagrammatic calculus [49] this diagram unambiguously denotes a linear map. The inductive construction is described by the second column in (2.7) – (2.17).

In each rule we assume a morphism has already been assigned to each of the sequents in the numerator of the deduction rule. These inputs are represented by blue circles in the diagram, which computes the morphism to be assigned to the denominator. To simplify the appearance of diagrams, we adopt the convention that a strand labelled by a type A represents a strand labelled by the denotation $[\![A]\!]$. In particular, a strand labelled with a sequence $\Gamma = A_1, \ldots, A_n$ represents a strand labelled by $[\![A_1]\!] \otimes \cdots \otimes [\![A_n]\!]$.

Some comments:

- The diagram for the axiom rule (2.7) depicts the identity of $[\![A]\!]$.
- The diagram for the exchange rule (2.8) uses the symmetry $[\![B]\!] \otimes [\![A]\!] \longrightarrow [\![A]\!] \otimes [\![B]\!]$.
- The diagram for the cut rule (2.9) depicts the composition of the two inputs.
- The right tensor rule (2.10) depicts the tensor product of the two given morphisms, while the left tensor rule (2.11) depicts the identity, viewed as a morphism between two strands labelled $[\![A]\!]$ and $[\![B]\!]$ and a single strand labelled $[\![A]\!] \otimes [\![B]\!]$.
- The diagram for the right $\multimap$ rule (2.12) denotes the adjoint of the input morphism, as explained in (3.2), while the left $\multimap$ rule (2.13) uses the composition map of (3.1).
- The diagram for the promotion rule (2.14) depicts the lifting of the input to a morphism of coalgebras, as explained in (3.5).
- The diagram for the dereliction rule (2.15) depicts the composition of the input with the universal map out of the coalgebra $!V$. The notation for this map, and the maps in the contraction (2.16) and weakening rules (2.17) are as described in (3.3).

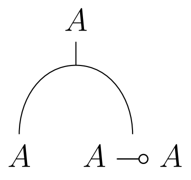
Remark 3.4. For this to be a valid semantics, two proofs related by cut-elimination must be assigned the same morphism. This is a consequence of the general considerations in [59, §7]. More precisely, $\mathcal{V}$ is a Lafont category [59, §7.2] and in the terminology of *loc.cit.* the adjunction between $\mathcal{V}$ and $\mathcal{C}$ is a linear-nonlinear adjunction giving rise to a model of intuitionistic linear logic. For an explanation of how the structure of a symmetric monoidal category extrudes itself from the cut-elimination transformations, see [59, §2].

Given the embedding of the simply-typed λ-calculus into linear logic, recalled in Remark 2.5, and the above construction of an interpretation of linear logic in the category $\mathcal{V}$ of k-vector spaces, we are finally in a position to answer Question 2.1 in the positive. With patience, the reader may use the above to translate any program into a linear map. To explain how this works in practice, we go through the details of assigning a diagram and the corresponding linear map to the proof tree (2.21) of the Church numeral $\underline{2}$.

Example 3.5. We convert the proof tree (2.21) to a diagram in stages, beginning with the leaves. Each stage is depicted in three columns: in the first is a partial proof tree, in the second is the diagram assigned to it by Definition 3.3, and in the third is the explicit linear map which is the value of the diagram.

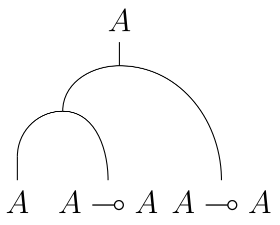
Recall that a strand labelled A actually stands for the vector space $V = [\![A]\!]$, so for instance the first diagram denotes a linear map $V \otimes \mathrm{End}_k(V) \longrightarrow V$:

$$\dfrac{\dfrac{}{A \vdash A} \qquad \dfrac{}{A \vdash A}}{A, A \multimap A \vdash A} \multimap L$$



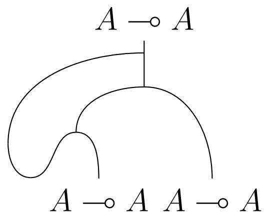
$$a \otimes \alpha \mapsto \alpha(a)$$

$$\dfrac{\dfrac{}{A \vdash A} \qquad \dfrac{\dfrac{}{A \vdash A} \quad \dfrac{}{A \vdash A}}{A, A \multimap A \vdash A}\,{\scriptstyle \multimap L}}{A, A \multimap A, A \multimap A \vdash A}\,{\scriptstyle \multimap L}$$



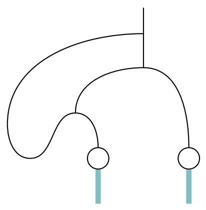
$$a \otimes \alpha \otimes \beta \mapsto \beta(\alpha(a))$$

$$\dfrac{\dfrac{\dfrac{}{A \vdash A} \qquad \dfrac{\dfrac{}{A \vdash A} \quad \dfrac{}{A \vdash A}}{A, A \multimap A \vdash A}\,{\scriptstyle \multimap L}}{A, A \multimap A, A \multimap A \vdash A}\,{\scriptstyle \multimap L}}{A \multimap A, A \multimap A \vdash A \multimap A}\,{\scriptstyle \multimap R}$$



$$\alpha \otimes \beta \mapsto \beta \circ \alpha$$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{A \vdash A} \qquad \dfrac{\dfrac{}{A \vdash A} \quad \dfrac{}{A \vdash A}}{A, A \multimap A \vdash A}\,{\scriptstyle \multimap L}}{A, A \multimap A, A \multimap A \vdash A}\,{\scriptstyle \multimap L}}{A \multimap A, A \multimap A \vdash A \multimap A}\,{\scriptstyle \multimap R}}{!(A \multimap A), !(A \multimap A) \vdash A \multimap A}\,{\scriptstyle \mathrm{der}} \tag{3.13}$$

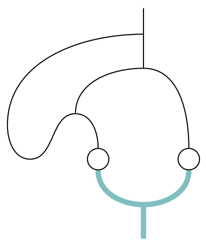


The map $! \operatorname{End}_k(V) \otimes ! \operatorname{End}_k(V) \longrightarrow \operatorname{End}_k(V)$ in (3.13) is zero on $|\nu_1, \ldots, \nu_s\rangle_\alpha \otimes |\mu_1, \ldots, \mu_t\rangle_\beta$ for $\alpha, \beta \in \operatorname{End}_k(V)$ unless $s, t \le 1$, and in those cases it is given by

$$\begin{aligned}
|o\rangle_\alpha \otimes |o\rangle_\beta &\longmapsto \beta \circ \alpha \\
|\nu\rangle_\alpha \otimes |o\rangle_\beta &\longmapsto \beta \circ \nu \\
|o\rangle_\alpha \otimes |\mu\rangle_\beta &\longmapsto \mu \circ \alpha \\
|\nu\rangle_\alpha \otimes |\mu\rangle_\beta &\longmapsto \mu \circ \nu\,.
\end{aligned}$$

The next deduction rule in $\underline{2}$ is a contraction:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A \vdash A} \qquad \dfrac{\dfrac{}{A \vdash A} \quad \dfrac{}{A \vdash A}}{A, A \multimap A \vdash A}\,{\scriptstyle \multimap L}}{A, A \multimap A, A \multimap A \vdash A}\,{\scriptstyle \multimap L}}{A \multimap A, A \multimap A \vdash A \multimap A}\,{\scriptstyle \multimap R}}{!(A \multimap A), !(A \multimap A) \vdash A \multimap A}\,{\scriptstyle \mathrm{der}}}{!(A \multimap A) \vdash A \multimap A}\,{\scriptstyle \mathrm{ctr}} \tag{3.14}$$



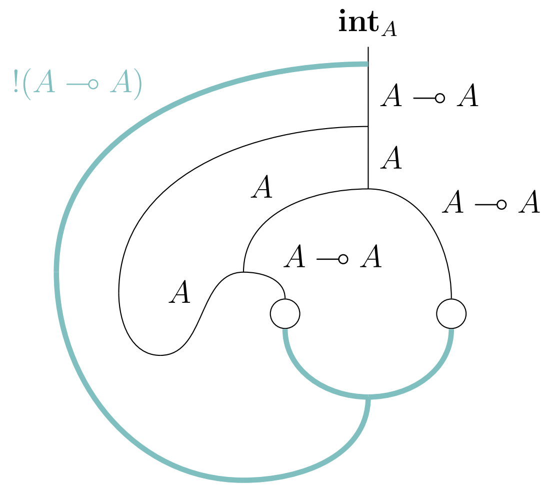
The denotation of this map is the composition $\phi = [\![\underline{2}]\!] : ! \operatorname{End}_k(V) \longrightarrow \operatorname{End}_k(V)$ in (2.23). We may compute using the above that

$$\phi|o\rangle_\alpha = \alpha^2, \qquad \phi|\nu\rangle_\alpha = \{\nu, \alpha\}\,, \qquad \phi|\nu\mu\rangle_\alpha = \{\nu, \mu\}\,. \tag{3.15}$$

For example

$$|\nu\rangle_\alpha \longmapsto |\nu\rangle_\alpha \otimes |o\rangle_\alpha + |o\rangle_\alpha \otimes |\nu\rangle_\alpha \longmapsto \alpha \circ \nu + \nu \circ \alpha = \{\nu, \alpha\}\,.$$

The final step in the proof of $\underline{2}$ consists of moving the $!(A \multimap A)$ to the right side of the sequent, which yields the final diagram:



(3.16)

The denotation of this morphism is the map in (2.22). The reader might like to compare this style of diagram for $\underline{2}$ to the corresponding proof-net in [36, §5.3.2].

In Example 2.4 we sketched how to recover the function $\alpha \mapsto \alpha^2$ from the denotation of the Church numeral $\underline{2}$, but now we can put this on a firmer footing. The translation of the λ-calculus into linear logic encourages us to think of a proof π of a sequent $!B \vdash C$ as a program whose input of type B may be used multiple times. There is *a priori* no linear map $[\![B]\!] \longrightarrow [\![C]\!]$ associated to π but there is a function $[\![\pi]\!]_{nl}$ defined on $P \in [\![B]\!]$ by lifting to $![\![B]\!]$ and then applying $[\![\pi]\!]$:

$$\begin{tikzcd} k \arrow[r, "P"] \arrow[dr, dotted, "|o\rangle_P"'] & {[\![B]\!]} & {[\![C]\!]}\ . \\ & {![\![B]\!]} \arrow[u, "d"] \arrow[ur, "{[\![\pi]\!]}"'] & \end{tikzcd} \tag{3.17}$$

That is,

Definition 3.6. The function $[\![\pi]\!]_{nl} : [\![B]\!] \longrightarrow [\![C]\!]$ is defined by $[\![\pi]\!]_{nl}(P) = [\![\pi]\!]|o\rangle_P$.

The discussion above shows that, with $V = [\![A]\!]$,

Lemma 3.7. $[\![\underline{2}]\!]_{nl} : \mathrm{End}_k(V) \longrightarrow \mathrm{End}_k(V)$ *is the map* $\alpha \mapsto \alpha^2$.

This completes our explanation of how to represent the Church numeral $\underline{2}$ as a linear map (modulo the evasion discussed in Remark 3.9). From this presentation we see clearly that the non-linearity of the map $\alpha \mapsto \alpha^2$ is concentrated, in fact, not in the duplication step but in the "promotion" step (3.17) where the input vector α is turned into a vacuum $|o\rangle_\alpha \in \,!\operatorname{End}_k(V)$. The promotion step is non-linear since $|o\rangle_\alpha + |o\rangle_\beta$ is not a morphism of coalgebras and thus cannot be equal to $|o\rangle_{\alpha+\beta}$. After this step, the duplication, dereliction and composition shown in (2.24) are all linear. Finally, when $k = \mathbb{C}$ it is interesting to compare $[\![2]\!]_{nl}$ with the map $\alpha \mapsto \alpha^2$ of smooth manifolds $\operatorname{End}_k(V)$, see Appendix A.

Example 3.8. Let $\underline{2}'$ denote the proof of $!(A \multimap A) \vdash A \multimap A$ in (3.14). As discussed in Example 2.4, we may confuse the denotation of $\underline{2}$ and $\underline{2}'$. Applied to $\underline{2}'$ the promotion rule generates a new proof,

$$\cfrac{\begin{array}{c}\underline{2}'\\ \vdots \\ !(A \multimap A) \vdash A \multimap A\end{array}}{!(A \multimap A) \vdash \,!(A \multimap A)}\ \text{prom}$$

which we denote $\operatorname{prom}(\underline{2}')$. By definition the denotation $\Phi := [\![\operatorname{prom}(\underline{2}')]\!]$ of this proof is the unique morphism of coalgebras

$$\Phi : \,!\operatorname{End}_k(V) \longrightarrow \,!\operatorname{End}_k(V)$$

with the property that $d \circ \Phi = \phi$, where $\phi = [\![\underline{2}]\!]$ is as in (2.23). From (3.15) and Theorem 3.1 we compute that, for example

$$\begin{aligned}
\Phi|o\rangle_\alpha &= |o\rangle_{\alpha^2}\,,\\
\Phi|\nu\rangle_\alpha &= \{\nu,\alpha\} \cdot |o\rangle_{\alpha^2},\\
\Phi|\nu\mu\rangle_\alpha &= \big(\{\nu,\mu\} + \{\nu,\alpha\} \otimes \{\mu,\alpha\}\big) \cdot |o\rangle_{\alpha^2}\,,\\
\Phi|\nu\mu\theta\rangle_\alpha &= \big(\{\nu,\mu\} \otimes \{\theta,\alpha\} + \{\theta,\mu\} \otimes \{\nu,\alpha\} + \{\nu,\theta\} \otimes \{\mu,\alpha\}\\
&\quad + \{\nu,\alpha\} \otimes \{\mu,\alpha\} \otimes \{\theta,\alpha\}\big) \cdot |o\rangle_{\alpha^2}\,.
\end{aligned}$$

Note that the commutators, e.g. $\{\nu,\alpha\}$ are defined using the product internal to $\operatorname{End}_k(V)$, whereas inside the bracket in the last two lines, the terms $\{\nu,\alpha\}$ and $\{\mu,\alpha\}$ are multiplied in the algebra $\operatorname{Sym}(\operatorname{End}_k(V))$ before being made to act on the vacuum.

Remark 3.9. As we have already mentioned, there are numerous other semantics of linear logic defined using topological vector spaces. The reader curious about how these vector space semantics are related to the "relational" style of semantics such as coherence spaces should consult Ehrhard's paper on finiteness spaces [30].

Indeed one can generate numerous examples of vector space semantics by looking at comonads on the category $\mathcal{V}$ defined by truncations on the coalgebras $!V$. For example, given a vector space V, let $!_0 V$ denote the subspace of $!V$ generated by the vacua $|o\rangle_P$. This is the free space on the underlying *set* of V, and it is a subcoalgebra of $!V$ given as a coproduct of trivial coalgebras. It is easy to see that this defines an appropriate comonad and gives rise to a semantics of linear logic [73, §4.3]. However the semantics defined using $!V$ is more interesting, because universality allows us to mix in arbitrary coalgebras C. In Appendix A we examine the simplest example where C is the dual of the algebra $k[t]/t^2$ and relate this to tangent vectors.

4 Cut-elimination

We have now introduced the sequent calculus of intuitionistic linear logic, and seen how to represent linear logic in vector spaces. But so far we have only talked about denotations of types and proofs which, to borrow an insightful analogy from [41, §III], together play a role analogous to that of statics within classical mechanics. The *dynamical* part of linear logic is a set of rewrite rules on proofs in the sequent calculus, which together define the possible "interactions" between proofs and the results of these interactions. The full set of rewrite rules is given in [59, Section 3] and in the alternative language of proof-nets in [36, §4], [63, p.18]. Rather than give the full set of rewrite rules which would involve us in various subtleties, we present a worked example involving our favourite proof $\underline{2}$. By presenting both the proof transformations and associated transformations of string diagrams we hope the reader will be able to grasp the core ideas.

A proof in linear logic is *cut-free* if it contains no occurrences of the cut rule. For each sequent $\Gamma \vdash A$ there is an equivalence relation on the set of proofs of the sequent, generated by a series of proof transformations that are together called *cut-elimination*. Each of these transformations generates a proof that is "closer" to being cut-free, according to a certain measure of cut complexity.

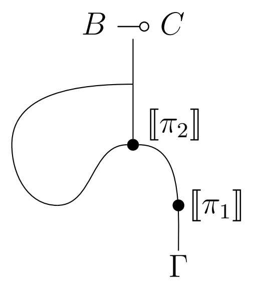
Theorem 4.1 (Girard)**.** *Every proof in linear logic may be related via cut-elimination to a unique cut-free proof of the same sequent.*

Proof. See [44, Chapter 13] for a sketch of Gentzen's cut-elimination in classical logic, and [59, §3] for the case of intuitionistic linear logic. □

Given a proof π the unique cut-free proof in the same equivalence class is called the *cut-free normalisation* of π. Before the main example, we examine two proof transformations from the list in [59, Section 3] which will be needed.

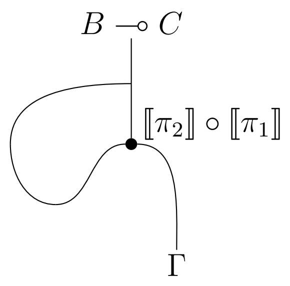
Example 4.2. The cut-elimination transformation [59, §3.11.10] tells us that

$$\dfrac{\begin{matrix}\pi_1\\ \vdots\\ \Gamma\vdash A\end{matrix}\qquad \dfrac{\begin{matrix}\pi_2\\ \vdots\\ B, A\vdash C\end{matrix}}{A\vdash B\multimap C}{\scriptstyle \multimap R}}{\Gamma\vdash B\multimap C}{\scriptstyle \mathrm{cut}} \tag{4.1}$$



is transformed to the proof

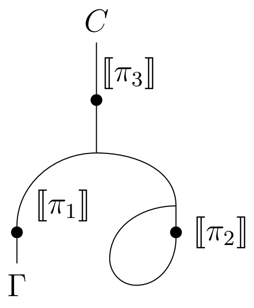
$$\dfrac{\dfrac{\begin{matrix}\pi_1\\ \vdots\\ \Gamma\vdash A\end{matrix}\qquad \begin{matrix}\pi_2\\ \vdots\\ B, A\vdash C\end{matrix}}{B,\Gamma\vdash C}{\scriptstyle \mathrm{cut}}}{\Gamma\vdash B\multimap C}{\scriptstyle \multimap R} \tag{4.2}$$



and thus the two corresponding diagrams are equal. In this case, the equality generated by cut-elimination expresses the fact that the Hom-tensor adjunction is natural.

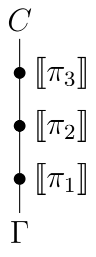
The transformation [59, §3.8.2] tells us that

$$\dfrac{\dfrac{\begin{matrix}\pi_2\\ \vdots\\ A\vdash B\end{matrix}}{\vdash A\multimap B}{\scriptstyle \multimap R}\qquad \dfrac{\begin{matrix}\pi_1\\ \vdots\\ \Gamma\vdash A\end{matrix}\quad \begin{matrix}\pi_3\\ \vdots\\ B\vdash C\end{matrix}}{\Gamma, A\multimap B\vdash C}{\scriptstyle \multimap L}}{\Gamma\vdash C}{\scriptstyle \mathrm{cut}} \tag{4.3}$$



may be transformed to

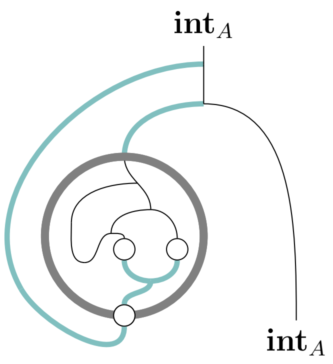
$$\dfrac{\dfrac{\begin{matrix}\pi_1\\ \vdots\\ \Gamma\vdash A\end{matrix}\quad \begin{matrix}\pi_2\\ \vdots\\ A\vdash B\end{matrix}}{\Gamma\vdash B}{\scriptstyle \mathrm{cut}}\qquad \begin{matrix}\pi_3\\ \vdots\\ B\vdash C\end{matrix}}{\Gamma\vdash C}{\scriptstyle \mathrm{cut}} \tag{4.4}$$



which is again an obvious property of the Hom-tensor adjunction.

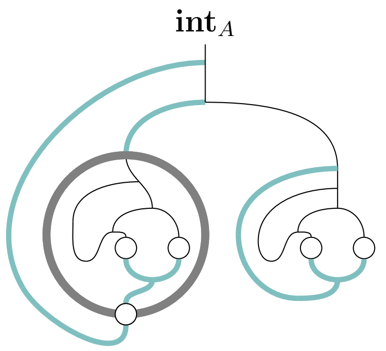
The following proof $\underline{\mathrm{mult}}_2$ represents multiplication by 2 on A-integers:

$$\dfrac{\dfrac{\dfrac{\begin{matrix}\underline{2}'\\ \vdots\\ !(A\multimap A)\vdash A\multimap A\end{matrix}}{!(A\multimap A)\vdash !(A\multimap A)}\,\text{prom}\qquad\dfrac{}{A\multimap A\vdash A\multimap A}}{!(A\multimap A),\mathbf{int}_A\vdash A\multimap A}\,{\multimap}\,L}{\mathbf{int}_A\vdash\mathbf{int}_A}\,{\multimap}\,R$$



where $\underline{2}'$ is the proof in (3.14). We feed $\underline{2}$ as input to $\underline{\mathrm{mult}_2}$ by cutting:

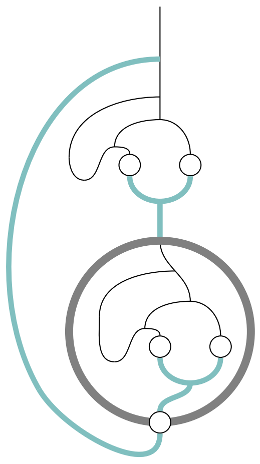
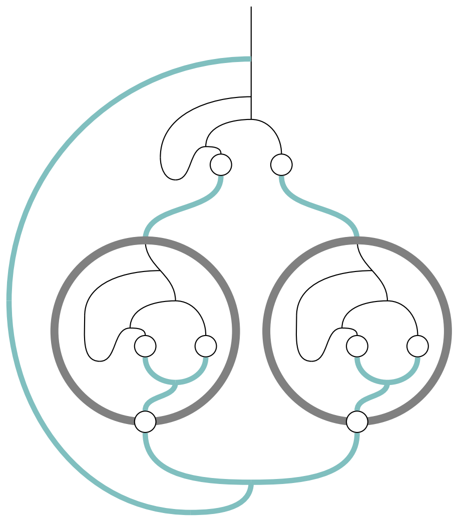
$$\dfrac{\begin{matrix}\underline{2}\\ \vdots\\ \vdash\mathbf{int}_A\end{matrix}\qquad\begin{matrix}\underline{\mathrm{mult}_2}\\ \vdots\\ \mathbf{int}_A\vdash\mathbf{int}_A\end{matrix}}{\vdash\mathbf{int}_A}\,\text{cut}$$



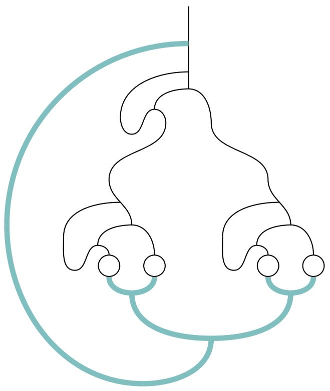
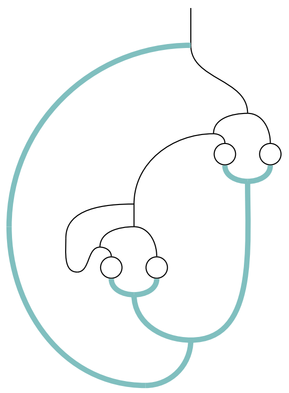
(4.5)

We denote this cut of the two proofs by $\underline{\mathrm{mult}_2}\,|\,\underline{2}$. Not surprisingly, the cut-free normalisation of $\underline{\mathrm{mult}_2}\,|\,\underline{2}$ is the Church numeral $\underline{4}$. Each of the proof transformations generated by the cut-elimination algorithm applied to $\underline{\mathrm{mult}_2}\,|\,\underline{2}$ (see below) yields a new proof with the same denotation, and this sequence of proofs represents a particular sequence of manipulations of the string diagram.

We now enumerate these diagrammatic transformations. From (4.5) the first step is to use naturality of the Hom-tensor adjunction, as in the manipulation from (4.1) to (4.2). Then we are in the position of (4.3), with π_2 a part of $\underline{2}$ and π_1 the promoted Church numeral. The manipulation from (4.3) to (4.4) is to take the left leg and feed it as an input to the right leg. This yields the first equality below. The second equality follows from the fact that a promotion box represents a morphism of coalgebras, and thus can be commuted past the coproduct whereby it is duplicated:

(4.5) =

=

(4.6)

At this point the promotions cancel with the derelictions by the identity (3.6), "releasing" the pair of Church numerals contained in the promotion boxes. This yields the first equality below, while the second is an application of the general form of the identity represented by the transformation of diagrams in (4.3) – (4.4):

(4.6) =

=


This last diagram is the denotation of $\underline{4}$, so we conclude that (at least at the level of the denotations) the output of the program $\underline{\mathrm{mult}_2}$ on the input $\underline{2}$ is $\underline{4}$.

We examine the beginning of the cut-elimination process applied to the proof (4.5). Our reference for cut-elimination is Melliès [59, §3.3]. We encourage the reader to put the following series of proof trees side-by-side with the evolving diagrams in the above to see the correspondence between cut-elimination and diagram manipulation.

To begin, we expose the first layer of structure within $\underline{\mathrm{mult}_2}$ to obtain

$$\frac{\begin{array}{c}\underline{2}\\ \vdots \\ \vdash \mathbf{int}_A\end{array} \qquad \dfrac{\begin{array}{c}\underline{\mathrm{mult}'_2}\\ \vdots \\ !(A \multimap A), \mathbf{int}_A \vdash A \multimap A\end{array}}{\mathbf{int}_A \vdash \mathbf{int}_A}\,{\scriptstyle \multimap R}}{\vdash \mathbf{int}_A}\,{\scriptstyle \mathrm{cut}} \tag{4.7}$$

where $\underline{\mathrm{mult}'_2}$ indicates the "remainder" of the proof $\underline{\mathrm{mult}_2}$. For a cut against a proof whose last deduction rule is a right introduction rule for $\multimap$, the cut elimination procedure [59, §3.11.10] prescribes that (4.7) be transformed to

$$\frac{\dfrac{\begin{array}{c}\underline{2}\\ \vdots \\ \vdash \mathbf{int}_A\end{array} \qquad \begin{array}{c}\underline{\mathrm{mult}'_2}\\ \vdots \\ !(A \multimap A), \mathbf{int}_A \vdash A \multimap A\end{array}}{!(A \multimap A) \vdash A \multimap A}\,{\scriptstyle \mathrm{cut}}}{\vdash \mathbf{int}_A}\,{\scriptstyle \multimap R} \tag{4.8}$$

If we fill in the content of $\underline{\mathrm{mult}'_2}$, this proof may be depicted as follows:

$$\frac{\dfrac{\dfrac{\begin{array}{c}\underline{2'}\\ \vdots \\ !(A \multimap A) \vdash A \multimap A\end{array}}{\vdash \mathbf{int}_A} \qquad \dfrac{\dfrac{\begin{array}{c}\underline{2'}\\ \vdots \\ !(A \multimap A) \vdash A \multimap A\end{array}}{!(A \multimap A) \vdash !(A \multimap A)}\,{\scriptstyle \mathrm{prom}} \qquad \dfrac{}{A \multimap A \vdash A \multimap A}}{!(A \multimap A), \mathbf{int}_A \vdash A \multimap A}\,{\scriptstyle \multimap L}}{!(A \multimap A) \vdash A \multimap A}\,{\scriptstyle \mathrm{cut}}}{\vdash \mathbf{int}_A}\,{\scriptstyle \multimap R} \tag{4.9}$$

The next cut-elimination step [59, §3.8.2] transforms this proof to

$$\frac{\dfrac{\dfrac{\dfrac{\begin{array}{c}\underline{2'}\\ \vdots \\ !(A \multimap A) \vdash A \multimap A\end{array}}{!(A \multimap A) \vdash !(A \multimap A)}\,{\scriptstyle \mathrm{prom}} \qquad \begin{array}{c}\underline{2'}\\ \vdots \\ !(A \multimap A) \vdash A \multimap A\end{array}}{!(A \multimap A) \vdash A \multimap A}\,{\scriptstyle \mathrm{cut}} \qquad \dfrac{}{A \multimap A \vdash A \multimap A}}{!(A \multimap A) \vdash A \multimap A}\,{\scriptstyle \mathrm{cut}}}{\vdash \mathbf{int}_A}\,{\scriptstyle \multimap R} \tag{4.10}$$

As may be expected, cutting against an axiom rule does nothing, so this is equivalent to

$$\dfrac{\dfrac{\dfrac{\begin{matrix}\underline{2}'\\ \vdots\\ !(A\multimap A)\vdash A\multimap A\end{matrix}}{!(A\multimap A)\vdash\, !(A\multimap A)}\,{\scriptstyle\mathrm{prom}}\qquad \dfrac{\begin{matrix}\underline{2}''\\ \vdots\\ !(A\multimap A),!(A\multimap A)\vdash A\multimap A\end{matrix}}{!(A\multimap A)\vdash A\multimap A}\,{\scriptstyle\mathrm{ctr}}}{!(A\multimap A)\vdash A\multimap A}\,{\scriptstyle\mathrm{cut}}}{\vdash \mathbf{int}_A}\,{\scriptstyle\multimap R}$$

where $\underline{2}''$ is a sub-proof of $\underline{2}$. Here is the important step: cut-elimination replaces a cut of a promotion against a contraction by a pair of promotions [59, §3.9.3]. This step corresponds to the doubling of the promotion box in (4.6)

$$\dfrac{\dfrac{\dfrac{\dfrac{\begin{matrix}\underline{2}'\\ \vdots\\ !(A\multimap A)\vdash A\multimap A\end{matrix}}{!(A\multimap A)\vdash\, !(A\multimap A)}\qquad \dfrac{\dfrac{\begin{matrix}\underline{2}'\\ \vdots\\ !(A\multimap A)\vdash A\multimap A\end{matrix}}{!(A\multimap A)\vdash\, !(A\multimap A)}\qquad \begin{matrix}\underline{2}''\\ \vdots\\ !(A\multimap A),!(A\multimap A)\vdash A\multimap A\end{matrix}}{!(A\multimap A),!(A\multimap A)\vdash A\multimap A}\,{\scriptstyle\mathrm{cut}}}{!(A\multimap A),!(A\multimap A)\vdash A\multimap A}\,{\scriptstyle\mathrm{cut}}}{!(A\multimap A)\vdash A\multimap A}\,{\scriptstyle\mathrm{ctr}}}{\vdash \mathbf{int}_A}\,{\scriptstyle\multimap R}$$

We only sketch the rest of the cut-elimination process: next, the dereliction in $\underline{2}''$ will be annihilate with the promotions in the two copies of $\underline{2}'$ according to [59, §3.9.1]. Then there are numerous eliminations involving the right and left $\multimap$ introduction rules.

5 The geometry of interaction

One interesting aspect of linear logic is Girard's program to study the semantics of the cut-elimination process; see [41, §III] and [38, 39, 40]. The purpose of this section is to very briefly explain his idea. As mentioned in the Introduction, the three most common views on what the dynamical process of computation "is" are the execution of a Turing machine, β-reduction in the λ-calculus, and cut-elimination in sequent calculus. At a first approach we notice that common to all three of these models of computation is a tension between the *implicit* and the *explicit*. We use the computation of Section 4 to explain.

Consider the proofs $\underline{\mathrm{mult}}_2$ and $\underline{2}$ and their cut $\underline{\mathrm{mult}}_2\,|\,\underline{2}$. The latter is equivalent under cut-elimination to the cut-free proof $\underline{4}$. Since this answer is derived from a deterministic algorithm – cut-elimination – the knowledge is certainly *implicit* in the proof $\underline{\mathrm{mult}}_2\,|\,\underline{2}$. But some work was necessary to convert this implicit truth into *explicit* truth. Although this example is a trivial one, the reader can easily imagine a similar calculation whose answer is not as apparent as $2\times 2 = 4$ – or, put aside arithmetic and note that the answer $\underline{4}$ is not obvious from a glance at the diagram (4.5). This conversion of implicit truth to explicit truth, or *explicitation*, is a fundamental aspect of computation.

However, this explicitation process is missing from most mathematical models of computation, which assign to the syntactical gadgets of Turing machines, λ-calculus or logic mathematical objects and transformations between them. To elaborate: suppose that in linear logic that we have proofs π of $A \vdash B$ and ρ of $B \vdash C$, and let $\rho\,|\,\pi$ denote the cut of one against the other, as displayed in the following proof tree:

$$\begin{array}{cc} \pi & \rho \\ \vdots & \vdots \end{array} \qquad (5.1)$$
$$\dfrac{A \vdash B \qquad B \vdash C}{A \vdash C}\,\text{cut}$$

Let $\widetilde{\rho\,|\,\pi}$ denote the cut-free proof of $A \vdash C$ produced from $\rho\,|\,\pi$ by the cut-elimination process. We regard this as the output of the program ρ computed on the input π. In the vector space semantics there is a commutative diagram of linear maps

$$\begin{tikzcd} & \llbracket B \rrbracket \arrow[dr, "\llbracket \rho \rrbracket"] & \\ \llbracket A \rrbracket \arrow[ur, "\llbracket \pi \rrbracket"] \arrow[rr, "\llbracket \rho\,|\,\pi \rrbracket = \llbracket \widetilde{\rho\,|\,\pi} \rrbracket"'] & & \llbracket C \rrbracket \end{tikzcd}$$

On this level, the calculation of output from input is a one-step affair $\llbracket \pi \rrbracket \mapsto \llbracket \rho \rrbracket \circ \llbracket \pi \rrbracket$. This may be contrasted with the syntax, where two steps are involved

$$\pi \longmapsto \rho\,|\,\pi \longmapsto \widetilde{\rho\,|\,\pi}\,. \qquad (5.2)$$

The point is that this second step, cut-elimination, is completely invisible in the semantics since both $\rho\,|\,\pi$ and its normalisation have the same denotation – by construction. The explicitation that happens in the syntax is absent in the semantics.

Girard proposed [41] that we should look instead for semantics in which the denotations of $\rho\,|\,\pi$ and $\widetilde{\rho\,|\,\pi}$ are distinct and there are "dynamics" which generate the latter from the former. He refers to the field of study of such dynamics as the *geometry of interaction*.[11] The first example of such a semantics constructed by Girard in [38] has been influential, although arguably it is still a bit mysterious. Since the λ-calculus may be translated into intuitionistic logic, and from there into intuitionistic linear logic, any semantics of linear logic yields a method for the execution of programs in the λ-calculus. One practical application of Girard's geometry of interaction model of linear logic is that it yields a method of executing programs in which the elementary reduction steps are local – that is, they do not dependent on global coordination [26, 27]. This is closely related to famous work of Lamping on optimal reduction in the λ-calculus [45].

[11]The categorically minded reader will detect the hint of higher-categories: it would be natural to expect that the denotations of a proof and its cut-free normalisation should be 1-morphisms connected by some structure on the level of 2-morphisms which models cut-elimination.

A Tangents and proofs

Example A.1. Let $\mathcal{T}$ denote the dual of the finite-dimensional algebra $k[t]/(t^2)$. It has a k-basis $1 = 1^*$ and $\varepsilon = t^*$ and coproduct Δ and counit u defined by

$$\Delta(1) = 1 \otimes 1, \quad \Delta(\varepsilon) = 1 \otimes \varepsilon + \varepsilon \otimes 1, \quad u(1) = 1, \quad u(\varepsilon) = 0\,.$$

Recall that a tangent vector at a point x on a scheme X is a morphism $\operatorname{Spec}(k[t]/t^2) \longrightarrow X$ sending the closed point to x. Given a finite-dimensional vector space V and $R = \operatorname{Sym}(V^*)$ with $X = \operatorname{Spec}(R)$, this is equivalent to a morphism of k-algebras

$$\varphi : \operatorname{Sym}(V^*) \longrightarrow k[t]/t^2$$

with $\varphi^{-1}((t)) = x$. Such a morphism of algebras is determined by its restriction to V^*, which as a linear map $\varphi|_{V^*} : V^* \longrightarrow k[t]/t^2$ corresponds to a pair of elements (P, Q) of V, where $\varphi(\tau) = \tau(P) \cdot 1 + \tau(Q) \cdot t$. Then φ sends a polynomial f to

$$\varphi(f) = f(P) \cdot 1 + \partial_Q(f)|_P \cdot t\,.$$

The map $\varphi|_{V^*}$ is also determined by its dual, which is a linear map $\phi : \mathcal{T} \longrightarrow V$. By the universal property, this lifts to a morphism of coalgebras $\Phi : \mathcal{T} \longrightarrow {!V}$. If ϕ is determined by a pair of points $(P, Q) \in V^{\oplus 2}$ as above, then it may checked directly that

$$\Phi(1) = |o\rangle_P, \qquad \Phi(\varepsilon) = |Q\rangle_P$$

is a morphism of coalgebras lifting ϕ.

Motivated by this example, we make a preliminary investigation into tangent vectors at proof denotations. Let A, B be types with finite-dimensional denotations $[\![A]\!], [\![B]\!]$.

Definition A.2. Given a proof π of $\vdash A$ a *tangent vector* at π is a morphism of coalgebras $\theta : \mathcal{T} \longrightarrow {![\![A]\!]}$ with the property that $\theta(1) = |o\rangle_{[\![\pi]\!]}$, or equivalently that the diagram

$$\begin{tikzcd} k \arrow[r, "{[\![\pi]\!]}"] \arrow[d, "1"'] & {[\![A]\!]} \\ \mathcal{T} \arrow[r, "\theta"'] & {![\![A]\!]} \arrow[u, "d"'] \end{tikzcd} \tag{A.1}$$

commutes. The space of tangent vectors at π is denoted T_π.

It follows from Example A.1 that there is a linear isomorphism

$$[\![A]\!] \longrightarrow T_\pi$$

sending $Q \in [\![A]\!]$ to the coalgebra morphism θ with $\theta(1) = |o\rangle_{[\![\pi]\!]}$ and $\theta(\varepsilon) = |Q\rangle_{[\![\pi]\!]}$.

Note that the denotation of a program not only maps inputs to outputs (if we identify inputs and outputs with vacuum vectors) but also tangent vectors to tangent vectors. To wit, if ρ is a proof of a sequent $!A \vdash B$ with denotation $\lambda : ![\![A]\!] \longrightarrow [\![B]\!]$, then composing a tangent vector θ at a proof π of $\vdash A$ with the lifting Λ of λ leads to a tangent vector at the cut of ρ against the promotion of π. That is, the linear map

$$\mathcal{T} \xrightarrow{\ \theta\ } ![\![A]\!] \xrightarrow{\ \Lambda\ } ![\![B]\!] \tag{A.2}$$

is a tangent vector at the following proof, which we denote $\rho\,|\,\pi$

$$\dfrac{\dfrac{\begin{matrix}\pi\\ \vdots\\ \vdash A\end{matrix}}{\vdash !A}\,\text{prom} \qquad \begin{matrix}\rho\\ \vdots\\ !A\vdash B\end{matrix}}{\vdash B}\,\text{cut}$$

By Theorem 3.1 the linear map of tangent spaces induced in this way by ρ is

$$\begin{gathered}[\![A]\!] \cong T_\pi \longrightarrow T_{\rho\,|\,\pi} \cong [\![B]\!] \\ Q \longmapsto \lambda|Q\rangle_{[\![\pi]\!]}\end{gathered} \tag{A.3}$$

When ρ computes a smooth map of differentiable manifolds, this map can be compared with an actual map of tangent spaces. We examine $\rho = \underline{2}$ below. It would be interesting to understand these maps in more complicated examples; this seems to be related to the differential λ-calculus [32, 33], but we have not tried to work out the precise connection.

Example A.3. When $k = \mathbb{C}$ and $Z = [\![\underline{2}]\!]_{nl}$ we have by Lemma 3.7

$$Z : M_n(\mathbb{C}) \longrightarrow M_n(\mathbb{C}), \qquad Z(\alpha) = \alpha^2\,.$$

The tangent map of the smooth map of manifolds Z at α is $(Z_*)_\alpha(\nu) = \{\nu, \alpha\}$. When α is the denotation of some proof π of $\vdash A \multimap A$ this agrees with the tangent map assigned in (A.3) to the proof $\underline{2}$ at π, using (3.15).

References

[1] S. Abramsky, *Computational interpretations of linear logic*, Theoretical Computer Science, 1993.

[2] S. Abramsky, *Retracting some paths in process algebra*, In CONCUR 96, Springer Lecture Notes in Computer Science **1119**, 1–17, 1996.

[3] S. Abramsky, *Geometry of Interaction and linear combinatory algebras*, Mathematical Structures in Computer Science, **12**, 625–665, 2002.

[4] S. Abramsky and R. Jagadeesan, *New foundations for the Geometry of Interaction*, Information and Computation **111** (1), 53–119, 1994.

[5] M. Anel, A. Joyal, *Sweedler theory of (co)algebras and the bar-cobar constructions*, [arXiv:1309.6952]

[6] M. Atiyah, *Topological quantum field theories*, Publications Mathématique de l'IHÉS 68, 175–186, 1989.

[7] J. Baez and M. Stay, *Physics, topology, logic and computation: a Rosetta stone*, in B. Coecke (ed.) New Structures for Physics, Lecture Notes in Physics 813, Springer, Berlin, 95–174, 2011

[8] M. Barr, *Coalgebras over a commutative ring*, Journal of Algebra 32, 600–610, 1974.

[9] ———, *$\star$-autonomous categories*, Number 752 in Lecture Notes in Mathematics. Springer-Verlag, 1979.

[10] ———, *Accessible categories and models of linear logic*, Journal of Pure and Applied Algebra, 69(3):219–232, 1990.

[11] ———, ?-autonomous categories and linear logic, Mathematical Structures in Computer Science, 1(2):159–178, 1991.

[12] ———, *The Chu construction: history of an idea*, Theory and Applications of Categories, Vol. 17, No. 1, 10–16, 2006.

[13] N. Benton, *A mixed linear and non-linear logic; proofs, terms and models*, in Proceedings of Computer Science Logic 94, vol. 933 of Lecture Notes in Computer Science, Verlag, 1995.

[14] N. Benton, G. Bierman, V. de Paiva and M. Hyland, *Term assignment for intuitionistic linear logic*, Technical report 262, Computer Laboratory, University of Cambridge, 1992.

[15] R. Block, P. Leroux, *Generalized dual coalgebras of algebras, with applications to cofree coalgebras*, J. Pure Appl. Algebra 36, no. 1, 15–21, 1985.

[16] R. Blute, *Hopf algebras and linear logic*, Mathematical Structures in Computer Science, 6(2):189–217, 1996.

[17] R. Blute and P. Scott, *Linear Laüchli semantics*, Annals of Pure and Applied Logic, 77:101–142, 1996.

[18] ———, *Category theory for linear logicians*, Linear Logic in Computer Science 316: 3–65, 2004.

[19] R. Blute, P. Panangaden, R. Seely, *Fock space: a model of linear exponential types*, in: Proc. Ninth Conf. on Mathematical Foundations of Programming Semantics, Lecture Notes in Computer Science, Vol. 802, Springer, Berlin, 1–25, 1994.

[20] R. Bott and L.W. Tu, *Differential forms in Algebraic Topology*, Graduate Texts in Mathematics, **82**, Springer, 1982.

[21] A. Căldăraru and S. Willerton, *The Mukai pairing, I: a categorical approach*, New York Journal of Mathematics **16**, 61–98, 2010 [arXiv:0707.2052].

[22] N. Carqueville and D. Murfet, *Adjunctions and defects in Landau-Ginzburg models*, [arXiv:1208.1481].

[23] N. Carqueville and I. Runkel, *On the monoidal structure of matrix bi-factorisations*, J. Phys. A: Math. Theor. **43** 275–401, 2010 [arXiv:0909.4381].

[24] A. Church, *The Calculi of Lambda-conversion*, Princeton University Press, Princeton, N. J. 1941.

[25] V. Danos and J.-B. Joinet, *Linear logic and elementary time*, Information and Computation 183, 123–127, 2003.

[26] V. Danos and L. Regnier, *Local and Asynchronous beta-reduction (an analysis of Girard's execution formula)* in: Springer Lecture Notes in Computer Science **8**, 296–306, 1993.

[27] V. Danos and L. Regnier, *Proof-nets and the Hilbert space*, in (Girard *et. al.* 1995), 307–328, 1995.

[28] P. J. Denning, *Ubiquity symposium "What is computation?"*: opening statement, Ubiquity 2010. Available on the Ubiquity website.

[29] T. Dyckerhoff and D. Murfet, *Pushing forward matrix factorisations*, Duke Math. J. Volume 162, Number 7 1249–1311, 2013 [arXiv:1102.2957].

[30] T. Ehrhard, *Finiteness spaces*, Math. Structures Comput. Sci. 15 (4) 615–646, 2005.

[31] ———, *On Köthe sequence spaces and linear logic*, Mathematical Structures in Computer Science 12.05, 579–623, 2002.

[32] T. Ehrhard and L. Regnier, *The differential lambda-calculus*, Theoretical Computer Science 309.1: 1–41, 2003.

[33] ———, *Differential interaction nets*, Theoretical Computer Science 364.2: 166–195, 2006.

[34] G. Gentzen, *The Collected Papers of Gerhard Gentzen*, (Ed. M. E. Szabo), Amsterdam, Netherlands: North-Holland, 1969.

[35] E. Getzler, P. Goerss, *A model category structure for differential graded coalgebras*, preprint, 1999.

[36] J.-Y. Girard, *Linear Logic*, Theoretical Computer Science 50 (1), 1–102, 1987.

[37] ———, *Normal functors, power series and the λ-calculus* Annals of Pure and Applied Logic, 37: 129–177, 1988.

[38] ———, *Geometry of Interaction I: Iinterpretation of System F*, in Logic Colloquium '88, ed. R. Ferro, et al. North-Holland, 221–260, 1988.

[39] ———, *Geometry of Interaction II: Deadlock-free Algorithms*, COLOG-88, Springer Lecture Notes in Computer Science **417**, 76–93, 1988.

[40] ———, *Geometry of Interaction III: Accommodating the Additives*, in (Girard *et al.* 1995), pp.1–42.

[41] ———, *Towards a geometry of interaction*, In J. W. Gray and A. Scedrov, editors, Categories in Computer Science and Logic, volume 92 of Contemporary Mathematics, 69–108, AMS, 1989.

[42] ———, *Light linear logic*, Information and Computation 14, 1995.

[43] ———, *Coherent Banach spaces: a continuous denotational semantics*, Theoretical Computer Science, 227: 275–297, 1999.

[44] J.-Y. Girard, Y. Lafont, and P. Taylor, *Proofs and Types*, Cambridge Tracts in Theoretical Computer Science 7 ,Cambridge University Press, 1989.

[45] G. Gontheir, M. Abadi and J.-J. Lévy, *The geometry of optimal lambda reduction*, in 9th Annual IEEE Symp. on Logic in Computer Science (LICS), 15–26, 1992.

[46] E. Haghverdi and P. Scott, *Geometry of Interaction and the dynamics of prood reduction: a tutorial*, in New Structures for Physics, Lecture notes in Physics **813**, 357–417, 2011.

[47] H. Hazewinkel, *Cofree coalgebras and multivariable recursiveness*, J. Pure Appl. Algebra 183, no. 1–3, 61–103, 2003.

[48] M. Hyland and A. Schalk, *Glueing and orthogonality for models of linear logic*, Theoretical Computer Science, 294: 183–231, 2003.

[49] A. Joyal and R. Street, *The geometry of tensor calculus I*, Advances in Math. **88**, 55–112, 1991.

[50] A. Joyal and R. Street, *The geometry of tensor calculus II*, draft available at http://maths.mq.edu.au/~street/GTCII.pdf

[51] A. Joyal, R. Street and D. Verity, *Traced monoidal categories*, Math. Proc. Camb. Phil. Soc. 119, 447–468, 1996.

[52] M. Khovanov, *Categorifications from planar diagrammatics*, Japanese J. of Mathematics **5**, 153–181, 2010 [arXiv:1008.5084].

[53] Y. Lafont, *The Linear Abstract Machine*, Theoretical Computer Science, 59 (1,2):157–180, 1988.

[54] J. Lambek and P. J. Scott, *Introduction to higher order categorical logic*, Cambridge Studies in Advanced Mathematics, vol. 7, Cambridge University Press, Cambridge, 1986.

[55] A. D. Lauda, *An introduction to diagrammatic algebra and categorified quantum* $\mathfrak{sl}_2$, Bulletin of the Institute of Mathematics Academia Sinica (New Series), Vol. **7**, No. 2, 165–270, 2012 [arXiv:1106.2128].

[56] J. McCarthy, *Recursive functions of symbolic expressions and their computation by machine, Part I.*, Communications of the ACM 3.4: 184–195, 1960.

[57] D. McNamee, *On the mathematical structure of topological defects in Landau-Ginzburg models*, MSc Thesis, Trinity College Dublin, 2009.

[58] P.-A. Melliès, *Functorial boxes in string diagrams*, In Z. Ésik, editor, Computer Science Logic, volume 4207 of Lecture Notes in Computer Science, pages 1–30, Springer Berlin / Heidelberg, 2006.

[59] P-A. Melliès, *Categorical semantics of linear logic*, in : Interactive models of computation and program behaviour, Panoramas et Synthèses 27, Société Mathématique de France, 2009.

[60] P.-A. Melliès, N. Tabareau, C. Tasson, *An explicit formula for the free exponential modality of linear logic*, in: 36th International Colloquium on Automata, Languages and Programming, July 2009, Rhodes, Greece, 2009.

[61] D. Murfet, *Computing with cut systems*, [arXiv:1402.4541].

[62] D. Murfet, *On Sweedler's cofree cocommutative coalgebra*, [arXiv:1406.5749].

[63] M. Pagani and L. Tortora de Falco.*Strong normalization property for second order linear logic*, Theoretical Computer Science 411.2 (2010): 410–444.

[64] A. Polishchuk and A. Vaintrob, *Chern characters and Hirzebruch-Riemann-Roch formula for matrix factorizations*, Duke Mathematical Journal 161.10: 1863–1926, 2012 [arXiv:1002.2116].

[65] U. Schreiber, *Quantization via Linear homotopy types*, [arXiv:1402.7041].

[66] D. Scott, *Data types as lattices*, SIAM Journal of computing, 5:522–587, 1976.

[67] ———, *The Lambda calculus, then and now*, available on YouTube with lecture notes, 2012.

[68] R. Seely, *Linear logic, star-autonomous categories and cofree coalgebras*, Applications of categories in logic and computer science, Contemporary Mathematics, 92, 1989.

[69] P. Selinger, *Lecture notes on the Lambda calculus*, [arXiv:0804.3434].

[70] M. Shirahata, *Geometry of Interaction explained*, available online.

[71] R. I. Soare, *Computability and Incomputability*, in CiE 2007: Computation and Logic in the Real World, LNCS 4497, Springer, 705–715.

[72] M. Sweedler, *Hopf Algebras*, W. A. Benjamin, New York, 1969.

[73] B. Valiron and S. Zdancewic, *Finite vector spaces as model of simply-typed lambda-calculi*, [arXiv:1406.1310].

[74] D. Quillen, *Rational homotopy theory*, The Annals of Mathematics, Second Series, Vol. 90, No. 2, 205–295, 1969.

[75] The Univalent Foundations Program, *Homotopy Type Theory: Univalent Foundations of Mathematics*, Institute for Advanced Study (Princeton), 2013.

[76] Y. Yoshino, *Cohen-Macaulay modules over Cohen-Macaulay rings*, London Mathematical Society Lecture Note Series, vol. 146, Cambridge University Press, Cambridge, 1990.

[77] E. Witten, *Topological quantum field theory*, *Communications in Mathematical Physics*, 117 (3), 353–386, 1988.

Department of Mathematics, University of Southern California
E-mail address: `murfet@usc.edu`