ENS Cachan, Université Paris-Saclay

Master report

Conservative extensions of Montague semantics
M2 MPRI: April-August 2017

Supervised by Philippe de Groote, Sémagramme team, Loria, Inria Nancy Grand Est

Mathieu Huot

August 21, 2017

The general context

Computational linguistics is the study of linguistic phenomena from the point of view of computer science. Many approaches have been proposed to tackle the problem. The approach of compositional semantics is that the meaning of a sentence is obtained via the meaning of its subconstituents, the basic constituents being mostly the words, with possibly some extra features which may be understood as side effects from the point of view of the computer scientist, such as interaction with a context, time dependency, etc.

Montague [43, 44] has been one of the major figures to this approach, but what is today known as Montague semantics lacks a modular approach. It means that a model is usually made to deal with a particular problem such as covert movement [47], intentionalisation [12, 8], or temporal modifiers [18], and those models do not necessarily interact well one with another.

This problem in general is an important limitation of the computational models of linguistics that have been proposed so far. De Groote [20] has proposed a model called Abstract Categorial Grammars (ACG), which subsumes many known frameworks [20, 21, 13] such as Tree Adjoining Grammars [24], Rational Transducers and Regular Grammars [20], taking its roots in the simply typed lambda calculus [6].

The research problem

The problem we are interested in is finding a more unifying model, yet close to the intuition and the previous works, which would allow more modularity while remaining in the paradigm of compositional semantics. This problem of modularity is not new but it seems to be difficult to deal with several phenomena in a unified framework, as pointed out in the thesis of Maršík [35]. The solutions that have been proposed so far become very technical as in [37, 18], thus rather unsatisfactory from a unifying perspective.

Shedding a new light on such a problem will help have a better intrinsic understanding of linguistics, develop new implementations and tools dealing with more and more sophisticated phenomena. Finally, our hope is that it contributes to enhance the understanding we may have of how language is treated by human brain. Indeed, recent works such as the work of Dehaene et al. [14] in cognitive science have shown, using the formal theory of language, some properties of language related data management by the brain.

Recent attempts such as Gillibert and Retoré [15], Pollard [46], Lebedeva [31], and Shan [51] have shown that more sophisticated mathematical tools might be useful in this field. We thus continue to pave the way in

the same direction, using classical tools from category theory that have been little known in the computational linguistics community.

Your contribution

Our starting point is de Groote's formalism of ACG [20] for which one major aim was to tighten links between syntax and semantics, and his article on logical relations [19] which aims at showing a way to gain modularity by enriching semantics bit by bit, such that the extension is *conservative*. Still, it lacks a certain degree of generality to be completely satisfactory and we try to address this problem here.

We first generalise ACG in category theory and express logical extensions in our setting. We obtain different ways to express logical relations and for a broader set of examples. Then, we argue that our framework is more modular by showing how to add the framework of Coecke, Grefenstette, and Sadrzadeh [10] of compositional distributional models in our model. This framework was intended to encompass within a compositional formal theory the distributional models (learning methods) used in practice. At last, we show how to add a notion of sub-typing, thus ending with a simple model which is enshrined in category theory, with a great level of modularity, letting the door open for many improvements, on the theory side as well as for applications.

Arguments supporting its validity

Our model is a simple and straightforward extension of de Groote's ACG. Its categorical presentation makes it less ad hoc than previous works and we are able to express some examples from other recent works: monads from Shan [51], distributional models from Coecke, Grefenstette, and Sadrzadeh [10], and the important property of sub-typing using refinement systems from Melliès and Zeilberger [40]. Still, the tools we use are relatively simple and the proofs are straightforward. It may mean that it is a good level of abstraction to work with a syntax-semantics interface.

In addition, there is a willing in the formal computational linguistic community, as can for instance be seen in the theories developed in Kallmeyer and Osswald [25] and Lascarides and Asher [30], to tighten links between syntax and semantics. Hence using the huge amount of work that has be done with Cartesian Closed Categories as semantics for simply-typed calculus seems to be relevant and in the same spirit. In particular, we import techniques well developed in functional programming into formal linguistics and hope to help bridge the gap between the two communities.

Summary and future work

We have provided a new model to deal with both ACG, logical relations, distributional models and sub-typing, allowing many restrictions and possible extensions, in a way we argue to be simple. Category theory seems to be a great tool for its clear point of view on mathematical objects, and might be well-suited for the roots of the formal theory of computational linguistics.

There are many possible ways to extend our model. One question we would like to explore soon is the possibility to enrich our categories with probabilities, maybe using a monad such as the monad of Giry [16], or maybe as an enrichment [9] of the category to allow more ways of disambiguation. Indeed, there must be a tradeoff between over generating using a formal grammar and disambiguation. The latter is closed to pragmatics and usually obtained via concrete data giving probabilities or clusters of words for instance. Such extensions would open the way for a convenient model to truly gather and unify formal theories dealing with subtle linguistic phenomena and now well-developed learning methods, in compositional and modular ways.

1 Introduction

Consider the canonical example of what is called a donkey sentence:

(1) Every farmer who owns a donkey beats it

It may be interpreted using first-order logic in the two following ways:

$$\forall x \exists y. \Big(\big[FARMER(x) \wedge DONKEY(y) \wedge OWNS(x,y) \big] \rightarrow BEATS(x,y) \Big)$$
$$\exists y \forall x. \Big(\big[FARMER(x) \wedge DONKEY(y) \wedge OWNS(x,y) \big] \rightarrow BEATS(x,y) \Big)$$

The first interpretation says that for any entity that is a farmer, there exists an entity that is a donkey and that is beaten by the given farmer. As for the second interpretation, it says that there is an entity which is a donkey such that any entity being a farmer beats the same donkey. It reveals that while logic is pretty convenient to represent the meaning of a sentence, the donkey sentence has intrinsically several representations. Thus, information from pragmatics is needed to solve the ambiguity and one has to be able to deal with such ambiguities to give a proper treatment of language.

More generally, computational linguistics is about dealing with the understanding of the numerous subtle phenomena occurring in the language. To give but a few examples, one may find dynamic anaphora such as donkey sentences, coercion phenomena such as in the sentence *John read the rumour*, which involves both a physical and an informational aspects as *rumour* is informational but may be written so it may have a physical aspect which is what *read* expects, covert movement such as quantifier raising, and they are all non trivial to tackle.

To deal with such issues, several approaches have been proposed, such as Montague [43], Kanazawa [27], and Martin and Pollard [37], de Groote [12]. It is however not clear which could possibly be best, and even worse it is absolutely not straightforward how to compose several approaches. In a spirit of unification, de Groote developed what he called Abstract Categorial Grammars (ACG) [20]. It roughly consists of a homomorphism interpreting an abstract language — a tectogrammar — such as λ-terms typed over nouns n, noun phrases np, sentences s, into a more concrete one — a phenogrammar — such as λ-terms typed over strings $*$. For instance it could send JOHN : np to $\lambda x : *.\ \textit{John}\ x : * \rightarrow *$.

ACG are a powerful tool in which several known formalisms can be encoded [20, 21, 13], such as Tree Adjoining Grammars from Joshi and Schabes [24], Context Free Grammars, Regular Grammars and Rational Transducers. To go further in a modular approach, one may wonder how to extend an ACG to treat more linguistics phenomena. For instance, if you consider you are given an ACG which does not treat intentionalisation, it is natural to ask whether you could possibly design an ACG dealing with intentionalisation while keeping the previous properties. The new ACG could be seen as a conservative extension of the previous one and a way to express such an extension is proposed by de Groote [19] by means of what is called logical relations.

Learning methods are the ones currently mostly used in applications because they give good and robust results on simple sentences yet lack formal methods to tackle more advanced phenomena. Work has been done to enshrine those methods within a formal framework. For instance, Coecke, Grefenstette, and Sadrzadeh [10] gave a categorical account to make links between Lambek monoids — a categorial framework for syntax analysis — and what is known as distributional methods, to obtain a compositional semantics.

Lastly, there are several works coining dot-types [53, 49] to deal with coercion and sub-typing, as it is an important aspect of natural semantics. Indeed, consider *This book is heavy but uninteresting*. The *book* has

both a physical aspect and an informational one, either being used in a particular situation, but both may happen in a single sentence. A notion of sub-typing and refinement has recently been given a categorical account with what is called refinement systems by Melliès and Zeilberger [40], further developed in Melliès and Zeilberger [39].

Trying to generalise ACG and develop ideas from category theory in computational linguistics is not new. For instance Kiselyov [28] used a more general notion than monads, called applicative functors, introduced by McBride and Paterson [38]. De Groote and Maarek [22] give type theoretic extensions of ACG such as pattern matching. Shan [52] tries to show the usefulness of monads as advocated by Moggi [41] in programming languages to explain some linguistic phenomena. It is further developed in his thesis [51]. In the same vein, Maršík [35] uses monads for linguistic purposes. Pollard [46] involves ideas from topos theory to solve a linguistic problem.

To sum up there is a paradigm underlying those works tightening links between syntax and semantics at a higher level of abstraction to have a greater level of modularity. Categorical semantics might well help us develop that paradigm and our work is a step toward that claim.

The report is organised as follows. First, we present the different basics of the theories we use to build our categorical model. More precisely, we present ACG, then logical relations, distributional models, and finally refinement type systems in the categorical sense of Melliès and Zeilberger [40]. Next, we present our results. These follow the same outline as the background section, progressively incorporating the notions for our use. This is concluded with a summary of the results and a few guidelines for further work. Finally, the appendix contains some classical category theory used but not recalled in the main document, most of the proofs, and lastly more directions for categorical investigations of computational linguistics.

Familiarity with simply-typed λ-calculus and basic category theory is assumed, such as categories, functors, natural transformations, Cartesian closed categories, the extended Curry-Howard isomorphism including category theory, free object, universal property, monomorphism and tensorial strength. See Barendregt et al. [6] for simply-typed λ-calculus and Lambek and Scott [29] for the previous notions on category theory.

2 Background

2.1 Abstract Categorial Grammars

This formalism is introduced by de Groote [20]. We start by giving a few definitions and examples. We will be intensively working with λ-calculus and thus recall some important definitions for complexity issues, which are an important matter in computational linguistics.

Definition 2.1 (Implicative types)**.** *Given a set of atomic types A, the set $\mathscr{T}(A)$ of implicative types is defined inductively as follows:*

- $A \subset \mathscr{T}(A)$*;*
- *if* $\alpha, \beta \in \mathscr{T}(A)$ *then* $\alpha \multimap \beta \in \mathscr{T}(A)$.

Definition 2.2 (Order of a type)**.** *The order of a type is given inductively as follows, where $\multimap$ is the arrow type, commonly used in the linear setting:*

- $ord(\alpha) = 1$ *for any atomic type* α*;*
- $ord(\alpha \multimap \beta) = max\{ord(\alpha) + 1, ord(\beta)\}$

Example 2.3.
- The curried version $\hat{f}$ of a function f taking n arguments of atomic types and returning a term of atomic type has type $\hat{f} : \alpha_1 \multimap \alpha_2 \multimap \ldots \multimap \alpha_n \multimap \beta$ and is of order 2;
- A function f mapping a function to something of atomic type has type $f : (A \multimap B) \multimap \beta$ and is of order at least 3.

ACG are based on the following definition, roughly describing a λ-calculus with constants generated by some atomic types.

Definition 2.4 (Higher-order signature)**.** *A higher-order signature is a triple $\langle A, C, \tau \rangle$ defined as follows:*

- *A is a finite set of atomic types*
- *C is a finite set of constants*
- *$\tau : C \to \mathscr{T}(A)$ is a type assignment that maps each constant from C to an implicative type built upon A*

We denote by $\Lambda(\Sigma)$ the set of simply-typed lambda terms built upon the higher order signature Σ, *i.e.* a simply-typed lambda-calculus enriched with some typed constants, and for any type α, we denote by $\Lambda^{\alpha}(\Sigma)$ those terms of type α.

Example 2.5. Given atomic types n for nouns, np for noun phrases and s for sentences, consider the following:

- $A = \{n, np, s\}$, $C = \{\textsc{John}, \textsc{Mary}, \textsc{love}, \textsc{dog}\}$ and $\tau(\textsc{John}) = np, \tau(\textsc{Mary}) = np, \tau(\textsc{love}) = np \multimap np \multimap s, \tau(\textsc{dog}) = n$;
- $A = \{e, t\}$ where t is for truth values and e for entities. $C = \{he, really, fail\}$ and $\tau(he) = e$, $\tau(really) = t \multimap t, \tau(fail) = e \multimap e \multimap t$.

Definition 2.6 (Lexicon)**.** *Given two higher-order signatures $\Sigma_1 = \langle A_1, C_1, \tau_1 \rangle$ and $\Sigma_2 = \langle A_2, C_2, \tau_2 \rangle$, a lexicon $\mathscr{L} : \Sigma_1 \to \Sigma_2$ is defined to be a pair $\langle F, G \rangle$ such that:*

- *$F : A_1 \to \mathscr{T}(A_2)$ is a function that interprets atomic types of Σ_1 as types built on A_2;*
- *$G : C_1 \to \Lambda(\Sigma_2)$ is a function that interprets the constants of Σ_1 as lambda terms over Σ_2;*
- *interpretation functions are compatible with typing,* i.e. *for any $c \in C_1$, the following judgement is derivable:*

$$\vdash_{\Sigma_2} G(c) : \widehat{F}(\tau_1(c))$$

where $\widehat{F}$ is the unique homomorphic extension of F.

The intuition is that a lexicon is a homomorphism between simply-typed λ-calculi that interprets constants of its domain as closed terms of its codomain. Those definitions meet in the following:

Definition 2.7 (ACG)**.** *An ACG is a quadruple $\langle \Sigma_1, \Sigma_2, \mathscr{L}, s \rangle$, where*

- *Σ_1, Σ_2 are linear higher-order signatures, called the abstract vocabulary and the object vocabulary, respectively;*
- *$\mathscr{L} : \Sigma_1 \to \Sigma_2$ is a lexicon;*
- *s is a type of the abstract vocabulary, called distinguished type of the grammar*

ACG are to be seen as an semantics-syntax interface where roughly speaking the abstract vocabulary plays the role of the semantics while the object vocabulary plays the role of syntax. It is actually more complex than that as lexicons compose.

Example 2.8. Suppose you are given a binary associative (infix) operator $+$ to represent concatenation of strings. As ACG compose, one may anotice that by interpreting $string$ as $(* \multimap *)$, the concatenation operator is actually definable using λ-terms. Then, an example is given as follows:

Σ_1:

$$\begin{aligned} n, np, s &: type \\ J &: np \\ U &: n \\ A &: n \multimap ((np \multimap s) \multimap s) \\ S &: ((np \multimap s) \multimap s) \multimap (np \multimap s) \end{aligned}$$

Σ_2:

$$\begin{aligned} string &: type \\ /a/ &: string \\ /John/ &: string \\ /seeks/ &: string \\ /unicorn/ &: string \end{aligned}$$

$\mathscr{L} : \Sigma_1 \to \Sigma_2$:

$$\begin{aligned} n, np, s &:= string \\ J &:= /John/ \\ U &:= /unicorn/ \\ A &:= \lambda x.\lambda p.p(/a/ + x) \\ S &:= \lambda p.\lambda x.p(\lambda y.x + /seeks/ + y) \end{aligned}$$

As said in the introduction, ACG can encode several other formalisms such as Context Free Grammars (CFG), Rational Transducers [20] and Tree Adjoining Grammars (TAG) [21]. It shows that ACG are relevant for linguistic purposes and they yield links between the different formalisms. Hence they are a step forward to tackle the problem of finding a unifying theory to deal with linguistic phenomena. The order of an ACG is the maximum order of the constants in its abstract vocabulary. Second-order ACG are important as they may be parsed in polynomial time — this problem is equivalent to the membership problem — and all the different encodings from above may be done using second-order ACG.

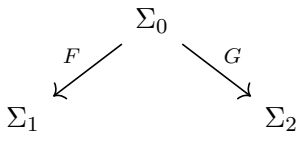
Remark 2.9. The condition on the object vocabulary can be relaxed from linear types to almost linear types, where only atomic types are allowed to be used non-linearly. Doing so, the newly-obtained ACG still enjoy many nice properties, such as polynomial time complexity for second-order ACG [54, 27].

2.2 Logical relations

In this section, higher-order signatures and lexicons are not restricted to a linear setting. This section explains the basic theory of logical relations, which comes from the article of de Groote [19]. The basic setting here is a pair of lexicons and we will try to see how to see one as the conservative extension of the other one, *i.e.* that they are related by what is called a logical relation.

Definition 2.10 (Span)**.** *A span is a* 5*-tuple* $\langle \Sigma_0, \Sigma_1, \Sigma_2, F, G \rangle$ *where:*

- Σ_0, Σ_1 *and* Σ_2 *are higher-order signatures;*
- $F : \Sigma_0 \to \Sigma_1$ *is a lexicon;*
- $G : \Sigma_0 \to \Sigma_2$ *is a lexicon.*



Definition 2.11 (Logical relation)**.** *Given a span* $\langle \Sigma_0, \Sigma_1, \Sigma_2, F, G \rangle$*, a logical relation is a family of binary relations,* $R = \{R_\alpha\}_{\alpha \in \mathscr{T}(A_0)}$*, where* A_0 *is the set of atomic types of the signature* Σ_0*, such that:*

- $R_\alpha \subset \Lambda^{F\alpha}(\Sigma_1) \times \Lambda^{G\alpha}(\Sigma_2)$*, for* $\alpha \in \mathscr{T}(A_0)$*;*
- $t_1 R_{\alpha \to \beta} t_2$ *iff* $\forall u_1 \in \Lambda^{F\alpha}(\Sigma_1), \forall u_2 \in \Lambda^{G\alpha}(\Sigma_2), u_1 R_\alpha u_2 \Rightarrow (t_1 u_1)\ R_\beta\ (t_2 u_2)$.

We will be only interested in logical relations that are $\beta\eta$-closed, *i.e.* those that satisfy the following:

Definition 2.12 ($\beta\eta$-closed logical relation)**.** *A logical relation* R *on a span* $\langle \Sigma_0, \Sigma_1, \Sigma_2, F, G \rangle$ *is* $\beta\eta$*-closed if for every* $\alpha \in \mathscr{T}(A_0)$*, every* $t_1, u_1 \in \Lambda^{F\alpha}(\Sigma_1)$ *such that* $t_1 =_{\beta\eta} u_1$*, and every* $t_2, u_2 \in \Lambda^{G\alpha}(\Sigma_2)$ *such that* $t_2 =_{\beta\eta} u_2$*, if* $t_1 R_\alpha t_2$ *then* $u_1 R_\alpha u_2$*.*

Remark 2.13. It actually suffices to require the condition for every atomic type $\alpha \in A_0$ [19].

The main proposition proved in the article of de Groote [19] is stated as follows:

Theorem 2.14 (Fundamental theorem of logical relations)**.** *Given a span $\langle \Sigma_0, \Sigma_1, \Sigma_2, F, G\rangle$ and a $\beta\eta$-closed logical relation R on this span such that $F(c)R_{\tau_0(c)}G(c)$, for every $c \in C_0$. Then, for every closed term $t \in \Lambda^\alpha(\Sigma_0)$ we have $F(t)R_\alpha G(t)$.* □

Having this theorem in mind, let's suppose we have a current interpretation. We want to ensure that a new interpretation is valid if and only if the current one is. In order to ensure this property, we construct a procedure that allows to transform terms such that the original term and its transform are logically related. To do this, we define a family of embedding functions, $\{\mathbb{E}_\alpha\}_{\alpha\in\mathscr{T}(A_0)}$, and a family of projection functions, $\{\mathbb{P}_\alpha\}_{\alpha\in\mathscr{T}(A_0)}$, that will relate the current and new interpretations.

Suppose you are given a type transformer T coming with three operations of type schemes as follows:

$$\begin{aligned} U &: \alpha \to T\alpha \\ \cdot &: T(\alpha \to \beta) \to T\alpha \to T\beta \\ C &: (\alpha \to T\beta) \to T(\alpha \to \beta) \end{aligned}$$

satisfying the following laws:

$$\begin{aligned} (Uf)\cdot(Ua) &= U(fa) \\ C(\lambda x.U(fx)) &= Uf \end{aligned}$$

Finally, suppose you are given the embedding $\mathbb{E}$ and the projection $\mathbb{P}$ at atomic types such that for any atomic type α and every term $t : \alpha$ we have:

$$\mathbb{P}_\alpha(\mathbb{E}_\alpha(Ut)) = Ut$$

Then, at functional type, the embedding and projections functions are given by means of the following equations:

$$\begin{aligned} \mathbb{E}_{\alpha\to\beta}t &= \lambda x.\mathbb{E}_\beta(t\cdot(\mathbb{P}_\alpha x)) \\ \mathbb{P}_{\alpha\to\beta}t &= C\Big(\lambda x.\mathbb{P}_\beta\Big(t(\mathbb{E}_\alpha(Ux))\Big)\Big) \end{aligned}$$

The final type transformer $\mathcal{T}$ is then given on terms by:

$$\mathcal{T}t = \mathbb{E}_\alpha(Ut)$$

It can then be shown that this leads to a logical relation. Thus, we can apply the previous machinery to important examples from linguistics. The three motivating examples in de Groote's article are the following:

Example 2.15 (Intensionalisation)**.** The idea is to allow constants to depend on a set of possible worlds, modelled by a new constant ι and a set s. Let $\Sigma_0 = \langle\{e, t\}, C_0, \tau_0\rangle$, and consider a set $C_r \subset C_0$ corresponding to rigid constants, *i.e.* those which do not vary from one possible world to another. We construct $\Sigma_1 = \langle\{e, t, s\}, C_0 \bigcup\{\iota\}, \tau_1\rangle$ and $\Sigma_2 = \langle\{e, t, s\}, C_0, \tau_2\rangle$ with:

$$\tau_1(c) = \begin{cases} s & \text{if } c = \iota \\ \tau_0(c) & \text{if } c \in C_r \\ (s \to \tau_0(c)) & \text{if } c \notin C_r \end{cases} \qquad \tau_2(c) = \begin{cases} \tau_0(c) & \text{if } c \in C_r \\ (s \to \tau_0(c)) & \text{if } c \notin C_r \end{cases}$$

Then, we define two morphisms $(-)^* : \Sigma_0 \to \Sigma_1, (\bar{-}) : \Sigma_0 \to \Sigma_2$ as follows:

$$\begin{aligned} e^* &= e & \bar{e} &= s \to e \\ t^* &= t & \bar{t} &= s \to t \\ c^* &= \begin{cases} c & \text{if } c \in C_r \\ (c\,\iota) & \text{if } c \notin C_r \end{cases} & \bar{c} &= \mathbb{E}_{\tau_0(c)}(Uc^*) \end{aligned}$$

where the primitives that are needed are defined as follows:

$$\begin{aligned} T\alpha &= s \to \alpha & \mathbb{E}_e t &= t \\ Ut &= \lambda i.t[\textsc{i} := i] & \mathbb{E}_t t &= t \\ t \cdot u &= \lambda i.ti(ui) & \mathbb{P}_e t &= t \\ Ct &= \lambda ix.txi & \mathbb{P}_t t &= t \end{aligned}$$

Example 2.16 (Dynamisation)**.** The idea is to allow a sentence s of type t of truth values to depend on its left context $\gamma_1 : c$ and of its right context $\gamma_2 : c \to t$. The right context is close to what is called continuation in functional programming. Indeed, it is a term that needs its left context to be evaluated. Thus dynamic propositions are seen as terms of type $c \to (c \to t) \to t$ where c is the type of the discourse context (roughly what has been read so far), and $(c \to t)$ is the type of discourse continuations (what remains to be read). Here again, we consider as given a signature $\Sigma_0 = \langle \{e, t\}, C_0, \tau_0 \rangle$. Then, we take Σ_1 to be Σ_0, and $(-)^*$ to be the identity. As for Σ_2, it consists of Σ_0 enriched with a set C_d of dynamic primitives. Accordingly, we have $\Sigma_2 = \langle \{e, t, c\}, C_0 \bigcup C_d, \tau_2 \rangle$ where $\tau_2(c) = \tau_0(c)$ for every $c \in C_0$, and we leave C_d unspecified but for the existence of a constant nil $\in C_d$. $\bar{(-)} : \Sigma_0 \to \Sigma_2$ is given by $\bar{e} = e$, $\bar{t} = c \to (c \to t) \to t$, $\bar{c} = \mathbb{E}_{\tau_0(c)}(Uc^*)$, where the needed primitives are defined to be:

$$\begin{aligned} T\alpha &= c \to \alpha & \mathbb{E}_e t &= t \text{ nil} \\ Ut &= \lambda c.t & \mathbb{E}_t t &= \lambda ck.(tc) \wedge (kc) \\ t \cdot u &= \lambda c.tc(uc) & \mathbb{P}_e t &= \lambda c.t \\ Ct &= \lambda cx.txc & \mathbb{P}_t t &= \lambda c.tc(\lambda c.\text{true}) \end{aligned}$$

It is to be noted, however, that we need to treat logical connectives separately and need to be given special translations. For instance dynamic conjunction is given by $\bar{\wedge} = \lambda abek.ae(\lambda e.bek)$.

Example 2.17 (Type raising)**.** We may consider Montague's type-raising as follows. Basic types of Σ_0 are $\{n, np, s\}$ and those of Σ_1 are $\{e, t\}$. Σ_0 specifies the syntax of a fragment of natural language that does not contain quantified noun phrases. Σ_1 specifies the object language in which a Montagovian interpretation of this fragment is given, and $(-)^*$ corresponds to this interpretation. Σ_2 is the same as Σ_1, while $\bar{(-)}$ interprets noun-phrases as type-raised entities:

$$\begin{aligned} n^* &= e \to t & \bar{n} &= e \to t \\ np^* &= e & \bar{np} &= (e \to t) \to t \\ s^* &= t & \bar{s} &= t \\ & & \bar{c} &= \mathbb{E}_{\tau_0(c)}(Uc^*) \end{aligned}$$

with the following primitives:

$$\begin{aligned} T\alpha &= (\alpha \to t) \to t & \mathbb{E}_n t &= \lambda x.t(\lambda k.kx) & \mathbb{P}_n t &= \lambda k.kt \\ Ut &= \lambda k.kt & \mathbb{E}_{np} t &= t & \mathbb{P}_{np} t &= t \\ t \cdot u &= \lambda k.t(\lambda x.u(\lambda y.k(xy))) & \mathbb{E}_s t &= t(\lambda x.x) & \mathbb{P}_s t &= \lambda k.kt \\ Ct &= \lambda k.k(\lambda x.L(tx)) & & & & \end{aligned}$$

where L is a function of type $((\alpha \to t) \to t) \to \alpha$ such that $L(\lambda k.kt) = t$, which is not λ-definable at all types but exists at the semantic level, thus the need to add constants with proper reduction rules.

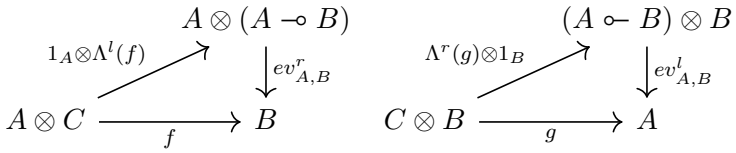
2.3 Distributional models

We recall one of the frameworks whose aim is to unify formal linguistics and learning methods in a simple, nice and formal model. This framework has been successfully used in linguistics and is developed by Coecke,

Grefenstette, and Sadrzadeh [10], improved in Preller [48]. It is not based on ACG but on what is called a Lambek monoid, which is seen in category theory as follows:

Definition 2.18 (Lambek monoid)**.** *A Lambek monoid is a monoidal bi-closed category. More specifically, it is a category with a monoidal tensor $\otimes$ and a unit I, such that for all pairs of objects A, B of the category, there exists a pair of objects $A \multimap B, A \multimapinv B$ and a pair of morphisms $ev^l_{A,B} : A \otimes (A \multimap B) \to B$ and $ev^r_{A,B} : (A \multimapinv B) \otimes B \to A$.*

These morphisms are referred to as left and right evaluations. They are such that for every pair of arrows $f : (A \otimes C) \to B$ and $g : (C \otimes B) \to A$ there exist two unique morphisms $\Lambda^l(f) : C \to A \multimap B$ and $\Lambda^r(g) : C \to A \multimapinv B$, called left and right currying, making the following diagrams commute:



There are several classical coherence conditions that are required. See Mac Lane [33] for instance.

Example 2.19. Consider the free monoidal bi-closed category with basic types n, np, s, and the following constants : $John : np, loves : (np \multimap s) \multimapinv np, Mary : np$. Then, *John loves Mary* is obtained by the evaluation of $John \otimes loves \otimes Mary$. Note that the tensor is not symmetric and $Mary \otimes loves \otimes John$ would lead to a different sentence, in that case simply *Mary loves John*. Finally, $loves \otimes Mary \otimes John$ can not be evaluated into something of type s and is not a correct sentence.

Essentially, distributional models are working as follows. Consider a word, say *cat*. Count the occurrences of the words close to it in different corpora. For instance you may you obtain 137 occurrences of *cute*, 120 occurrences of *fluffy*, 140 occurrences of *feed*, etc. Those data are collected in an array a. More precisely, consider a — huge — finite number of words as an orthogonal basis of vectors of a finite dimensional vector space over the reals. Then, the array a gives you a vector in this space.

If the vectors can be normalised, then you may compute the cosine of two vectors a, b simply as the scalar product. This cosine gives you a way to compare how close the meanings of the words considered are. It makes sense in that we may roughly admit that a semantics of a term is given by its use in the different possible contexts. For instance, *king* and *queen* are pretty close, and similarly for *dog* and *cat*.

We recall that the category $\mathcal{FV}ect$ of finite dimension vector spaces over the reals is monoidal closed with the usual tensor product and the exponent $A \multimap B$ given by the space of $\mathbb{R}$-linear maps from the vector space A to the vector space B.

To interpret the formal language given by a Lambek monoid, Coecke et al. use what they call a quantisation functor, which interprets the terms from the Lambek monoid into a vector space model, preserving the tensor product and the exponents though merging the two exponents into the one of vector spaces, thus somehow *forgetting* the order of the words. The terminology comes from the fact that they used an idea from topological quantum field theory, where there is a similar structure-preserving functor.

Definition 2.20 (Quantisation functor)**.** *Given a Lambek monoid $\mathcal{L}$ and the category of finite dimensional vector spaces $\mathcal{FV}ect$ over $\mathbb{R}$, the quantisation functor $\mathcal{Q} : \mathcal{L} \to \mathcal{FV}ect$ is a strongly monoidal functor, satisfying the following:*

$$\mathcal{Q}(1) \overset{def}{=} \mathbb{R}$$
$$\mathcal{Q}(a \cdot b) \cong \mathcal{Q}(b \cdot a) \overset{def}{=} \mathcal{Q}(a) \otimes \mathcal{Q}(b)$$
$$\mathcal{Q}(a \multimap b) \cong \mathcal{Q}(b \multimapinv a) \overset{def}{=} \mathcal{Q}(a) \multimap \mathcal{Q}(b)$$

Example 2.21. Consider the same words $John, loves, Mary$. Then, $\overrightarrow{John\ loves\ Mary}$ is obtained by the evaluation of $\overrightarrow{John} \otimes \overrightarrow{loves} \otimes \overrightarrow{Mary}$, where the arrow notation $\overrightarrow{John}$ denotes $\mathcal{Q}(John)$.

2.4 Refinement models

This categorical formalism is introduced by Melliès and Zeilberger [40]. It is further developed in [39]. The main idea which justifies the use of refinement systems for our purpose is that they use closed functors as a logical framework, *i.e.* capable of speaking about syntax and semantics in a unified way, which is very much in the same spirit as ACG. We thus give a few definitions that we will use in the result section.

Definition 2.22 (Type refinement system)**.** *A type refinement system is a functor $U : \mathcal{D} \to \mathcal{T}$.*

Definition 2.23 (Refinement)**.** *We say that an object $P \in \mathcal{D}$ refines an object $A \in \mathcal{T}$ (denoted by $P \sqsubset A$) if $U(P) = A$.*

Definition 2.24 (Typing judgement)**.** *A typing judgement is a triple (P, c, Q), where c is a morphism in $\mathcal{T}$ such that $P \sqsubset dom(c)$ and $Q \sqsubset cod(c)$ (denoted by $P \underset{c}{\Longrightarrow} Q$). In the special case where $c = id_A$, (P, c, Q) is also called a subtyping judgement (denoted by $P \Longrightarrow Q$).*

Definition 2.25 (Derivation)**.** *A derivation of a typing judgement (P, c, Q) is a morphism $\alpha : P \to Q$ in $\mathcal{D}$ such that $U(\alpha) = c$ (denoted by $P \underset{c}{\overset{\alpha}{\Longrightarrow}} Q$).*

Example 2.26.

- Any forgetful functor, for instance $U : Grp \to Set$ which sends a group to its underlying set, is a refinement system;
- A motivating example is given by Hoare logic. Let $\mathcal{T}$ be a category with a single object W representing the state space and morphisms $c : W \to W$ to be state transformers.

  Let $\mathcal{D}$ be the category whose objects $P, Q \in \mathcal{D}$ are predicates over the state space W and whose morphisms $(c, \alpha) : P \to Q$ are commands $c : W \to W$ equipped with a verification α that c will take any state satisfying P to a state satisfying Q.

  Finally, let $U : \mathcal{D} \to \mathcal{T}$ be the evident forgetful functor. In this case, a judgement is nothing but a Hoare triple $\{P\}c\{Q\}$, *i.e.* a typing judgement can describe not only entailment but also a side effect.

Definition 2.27 (Monoidal refinement systen)**.** *A refinement system $U : \mathcal{D} \to \mathcal{T}$ between monoidal categories is monoidal when it preserves the monoidal structure on the nose (*i.e. *in the strict sense).*

Example 2.28. Consider the refinement system $U : SubSet \to Set$ from the category of subsets and image inclusion to the category of sets and functions. An object of $SubSet$ is a pair (A, S) of a set A and a subset of that set $S \subseteq A$ while a morphism $(A, S) \to (B, T)$ is a function $f : A \to B$ such that $\forall a. a \in S \Rightarrow f(a) \in T$. Finally, U is taken to be the first projection.

Set is (Cartesian) monoidal with the usual Cartesian product and the terminal object being any singleton $\{*\}$. $SubSet$ is also (Cartesian) monoidal with the terminal object $\{(*, *)\}$ and the product given by:

$$(S \sqsubset A) \cdot (T \sqsubset B) \overset{def}{=} \{(a, b) | a \in S, b \in T\} \sqsubset A \times B$$

In a Cartesian closed category $\mathcal{C}$, one is given for every object $A, B \in \mathcal{C}$ an exponent $A \Rightarrow B$. More generally, a monoidal category may be closed in two ways which collapse into the same if the category is symmetric monoidal. The two exponents $A \multimap B, A \circ\!\!- B$ are respectively called left and right residuals. We can relax even further the definition by calling left (resp. right) residuals any $(A \multimap B)$ (resp. $B \circ\!\!- A$) satisfying the same universal property in a monoidal (non necessary closed) category.

Definition 2.29 (Logical refinement system)**.** *A monoidal refinement is logical when it preserves left and right residuals.*

Example 2.30. The refinement system $U : SubSet \to Set$ is a logical refinement. Set is closed with $A \Rightarrow B \stackrel{def}{=} B^A$ and $SubSet$ with

$$(S \sqsubset A) \Rightarrow (U \sqsubset B) \stackrel{def}{=} \{f | \forall a.a \in S \Rightarrow f(a) \in U\} \sqsubset B^A$$

3 Results

3.1 Abstract Categorial Grammar in Category Theory

Our goal is to tighten links between syntax and semantics. Let us remember the link between λ-calculus and Cartesian Closed Categories (CCC): each simply typed λ-calculus has a model given by a CCC, and conversely every CCC has an internal language giving rise to a simply-typed λ-calculus. Moreover, simply-typed λ-calculus is pure functional programming, and a proper way to add side-effects to functional programming is given by monads (be it monads [41], monad transformers [32], or more recently effect handlers [45]), which come from category theory.

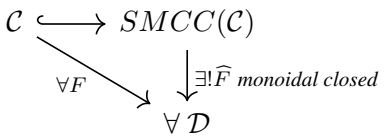
Category theory seems to be a promising way to represent the semantics of a language and add effects to it. As ACG are rooted in simply-typed λ-calculus, it is natural to model them using something similar to CCC. To do so, remember that a CCC models simply-typed λ-calculus whereas a symmetric monoidal closed category models simply-typed *linear* λ-calculus. ACG use the notion of linear simply-typed λ-calculus generated by some set of atomic types. This can be done in category theory as well as follows:

Definition 3.1 (Free symmetric monoidal category (SMCC))**.** *Given an abstract signature $\Sigma = \langle A, C, \tau \rangle$, one can define a free SMCC $\mathcal{C}$ on the discrete category A of atomic types as follows:*

- *objects of $\mathcal{C}$ contain those of A, a unit I, and are closed by tensor and linear exponent,* i.e. $a, b \in \mathcal{C} \Rightarrow a \otimes b, a \multimap b \in \mathcal{C}$
- *morphisms of $\mathcal{C}$ contain those of A, identity morphisms, evaluation maps, are closed by tensor, linear exponent and composition.*

In addition, we add equations to make this construction a SMCC, such as the associativity of composition of morphisms, the fact that identities are left and right neutrals, the associativity of the tensor product, etc. The category is to be thought of as a formal and syntactic construction quotiented by some equations.

We write $SMCC(\mathcal{C})$ for the free SMCC on $\mathcal{C}$. It can be easily shown that it has the universal property of free objects, which in this case is stated as follows, for all symmetric monoidal closed category $\mathcal{D}$:

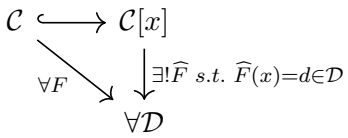


where a functor is monoidal closed when it preserves tensors and exponents on the nose, i.e. *strictly.*

Remark 3.2. This also works for the particular case where the tensor product is the Cartesian product and we obtain a free Cartesian closed category. As all objects defined by a universal property, it is defined up to unique isomorphism, but we take the representing object given above, and we allow ourselves to talk about *the* free SMCC on $\mathcal{C}$. It is also needed as we want the structure to be preserved on the nose, which is required for the unicity of $\widehat{F}$. Finally, we implicitly consider all our categories to be small throughout the whole report.

Then, we need to be able to add constants to our categories. This can be done using the following, which can be found in Lambek and Scott [29].

Definition 3.3 (Polynomial category)**.** *A Cartesian (closed) category $\mathcal{C}$ can be freely adjoined a new morphism, say $x : a \to b$, such that the newly made category is also Cartesian (closed). Such a morphism can be seen as a variable x and enjoys the following universal property:*



where $F, \widehat{F}$ are Cartesian (closed) functors and $\mathcal{D}$ is a Cartesian (closed) category.

We will be interested in the case where x has domain 1, meaning a term that can be typed in the empty context, *i.e.* constants. Any morphism in the new category may be formed using x and it can therefore be seen as a polynomial over the indeterminate x. This explains why in Lambek and Scott [29] such a category is called a polynomial category.

Using that technique, an arbitrary number of constants may be added, and the order is unimportant. This provides us with vocabularies. The next step is to define lexicons. In ACG, these are morphisms of λ-calculus, *i.e.* morphisms which preserve functions. It means they have to preserve exponentials. They also preserve contexts so they should preserve products. Finally, they interpret constants as closed terms so they should preserve the terminal object 1. This leads to the following definition:

Definition 3.4 (Lexicon)**.** *A lexicon $F : \mathcal{C} \to \mathcal{D}$ is a monoidal closed functor F between two monoidal closed categories $\mathcal{C}, \mathcal{D}$.*

Note that the composition of lexicons is still a lexicon. Consider the example where $\mathcal{C}$ is a polynomial category on a free SMCC. Then, given a SMCC $\mathcal{D}$ and using the universal properties given by $\mathcal{C}$ being a free SMCC and a polynomial category, a lexicon $F : \mathcal{C} \to \mathcal{D}$ is entirely determined by its image on basic types and on the added constants. It must respect the unit and the tensor product so it preserves contexts and it sends closed terms to closed terms. Given what we just did, we are led to the following categorical definition of ACG:

Definition 3.5 (ACG)**.** *An ACG is a triple $\langle \mathcal{C}, \mathscr{L}, \mathcal{D} \rangle$ where $\mathcal{C}, \mathcal{D}$ are symmetric monoidal closed categories and where $F : \mathcal{C} \to \mathcal{D}$ is a lexicon.*

Example 3.6 (Montague semantics)**.** Take $\mathcal{C}$ to be the free CCC on $\{n, np, s\}$ with constants such as *John* : $1 \to np$, *loves* : $1 \to (np \multimap np \multimap s)$, *Mary* : $1 \to np$ and $\mathcal{D}$ the free CCC on $\{e, t\}$ with some constants. Let $\mathscr{L} : \mathcal{C} \to \mathcal{D}$ be the lexicon defined by $\mathscr{L}(n) = (e \Rightarrow t), \mathscr{L}(np) = e, \mathscr{L}(s) = t$. Doing this, we recover the classical semantics of Montague.

One can go further, for instance by adding full non-linearity. Consider the case where the categories $\mathcal{C}, \mathcal{D}$ are Cartesian closed categories. These are particular cases of monoidal closed categories, but roughly speaking the extra structure is given by adding morphisms for non-linearity. More precisely, the universal property of the Cartesian product gives for any type A a morphism $\Delta_A : A \to A \times A$ called the diagonal map sending $x \in A$ to the pair $(x, x) \in A \times A$. This provides us with a contraction. As for the projections maps $\pi_i : A_1 \times A_2 \to A_i$, they provide weakening. A nice property is that given a free SMCC, we can freely add the morphisms to make it a free CCC, thus adding non-linearity.

Proposition 3.7. *Let $\mathcal{B}$ be the free symmetric monoidal closed category on a discrete category A of atomic types, and let $\mathcal{B}'$ be the monoidal closed category where projections and diagonal maps for atomic types are freely added, and quotiented by the equations for a Cartesian product. Then, $\mathcal{B}'$ is Cartesian closed.*

Let $F : \mathcal{C} \to \mathcal{D}$ be a lexicon between two free symmetric monoidal closed categories. Then, the extension

$F' : \mathcal{C}' \to \mathcal{D}'$ of F preserving the diagonal and projection maps on the nose is well defined and is still a lexicon.

In addition, given a CCC there is an underlying SMCC that *forgets* the morphisms allowing non-linearity.

Proposition 3.8. *Let $\mathcal{C}$ be a Cartesian closed category. Then $\mathcal{C}$ is monoidal closed and the extra morphisms given by the universal structure of the Cartesian product are generated by projections and diagonal maps. Let $\mathcal{C}''$ be the subcategory of $\mathcal{C}$ with the same objects as $\mathcal{C}$ and same morphisms except the morphisms generated by projections and diagonal maps. Then $\mathcal{C}''$ is symmetric monoidal closed.*

This might be interesting in the case we have a lexicon behaving linearly on the linear part of the category, which then restricts to a lexicon on the subcategory where no non-linearity is allowed, as the following shows:

Proposition 3.9. *Let $F : \mathcal{C} \to \mathcal{D}$ be a lexicon between two CCC such that $F(f) = \pi \Rightarrow f = \pi$ and $F(f) = \Delta \Rightarrow f = \Delta$ for all $f \in \mathcal{C}$, all projections maps π, and all diagonal maps Δ. Then, the restriction $F'' : \mathcal{C}'' \to \mathcal{D}''$ of F is still a lexicon.*

More generally, one may add or remove non-linearity in a partial way. This might be useful for instance by allowing variables of atomic types to be used several times, and if we impose bound variables to be used at least once, we recover the almost-linear ACG from Kanazawa [27]. If $\mathcal{B}$ is a free SMCC, we denote by $\mathcal{B}'$ the category obtained from $\mathcal{B}$ by freely adding some projections and diagonal maps, and by F' the extension of a lexicon F preserving those projections and diagonal maps strictly. If $\mathcal{B}$ is a CCC, we denote by $\mathcal{B}''$ the category obtained by removing the morphisms generated by some projections and diagonal maps, and by F'' the restriction of a lexicon F. Then, we are led to the following theorem:

Theorem 3.10. *Let $F : \mathcal{C} \to \mathcal{D}$ be a lexicon whose vocabularies are free SMCC. Then, the extension $F' : \mathcal{C}' \to \mathcal{D}'$ is again a lexicon provided that $\mathcal{D}'$ contains the image by F' of the projections and diagonal maps added in $\mathcal{C}'$.*

Let $F : \mathcal{C} \to \mathcal{D}$ be a lexicon whose vocabularies are CCC. Then, the restriction $F'' : \mathcal{C}'' \to \mathcal{D}''$ is again a lexicon provided that no morphisms in $\mathcal{D} - \mathcal{D}''$ are in the image of $\mathcal{C}''$ by F. □

Example 3.11. Consider an ACG $\langle \mathcal{C}, \mathscr{L}, \mathcal{D} \rangle$. If $\mathcal{C}, \mathcal{D}$ are free CCC with constants respecting the almost-linear condition, we obtain almost-linear types by considering the ACG $\langle \mathcal{C}'', \mathscr{L}'', \mathcal{D} \rangle$ where $\mathcal{C}''$ is obtained as the free SMCC on the same base category as $\mathcal{C}$, freely adjoined the same constants as $\mathcal{C}$ and diagonal maps $\Delta_\alpha : \alpha \to \alpha \times \alpha$ for each atomic type α.

If $\mathcal{C}, \mathcal{D}$ are free SMCC with constants, we obtain almost-linear types by considering the ACG $\langle \mathcal{C}', \mathscr{L}', \mathcal{D}' \rangle$ where $\mathcal{C}'$ is the category $\mathcal{C}$ which is freely added diagonal maps $\Delta_\alpha : \alpha \to \alpha \times \alpha$ for each atomic type α, constants may be changed to make use of these new possibilities, $\mathscr{L}'$ is the extension of $\mathscr{L}$ sending diagonal maps to diagonal maps, and $\mathcal{D}'$ is the free extension of $\mathcal{D}$ where the morphisms freely added are at least the image of diagonal maps by $\mathscr{L}'$.

Remark 3.12. The first aim of the internship was to generalise ACG in the setting of Melliès and Zeilberger [40]. We are in the right direction as lexicons are logical refinement systems.

Additional features may be recovered, such as records, variants, and enumerated types. These are extensions of ACG proposed in De Groote and Pogodalla [13]. In the paper it is shown that those are type theoretically given by disjoint union, Cartesian product and unit type. We already have shown we could deal with the Cartesian product and the unit type is given by the terminal object 1. The only thing that remains to be added is the disjoint union, which is categorically given by coproducts, the formal dual of the Cartesian product. It is known that the cocartesian structure from category theory is interpreted in the internal language as pattern-matching.

Particular cases of monoidal closed categories are free bicartesian closed category on a set of atomic types,

i.e. both the Cartesian closed and cocartesian structures are freely added, and this is the right setting to recover what is in [13]. We do not deal with dependent types though, for which a more involved treatment is required. However, adding dependant types to ACG makes one lose decidability and it might be a good reason to consider them as too powerful for our needs.

To further illustrate how flexible the categorical definition is, we may for instance characterise the order 2 of ACG as follows:

Proposition 3.13 (Characterisation of order 2 ACGs)**.** *An ACG on CCC is of order 2 iff any constant is obtained by currying and the universal property of the Cartesian product from morphisms of type* $\alpha_1 \times \cdots \times \alpha_n \multimap \beta$ *where all types are atomic.*

In particular, it means they are obtained by currying and products of the underlying multi-sorted Lawvere theory of the abstract language, where the sorts are given by the atomic types. Note that, for order 3 one could obtain a similar result but we would need an infinity-sorted Lawvere theory, which might help explain the complexity difference between order 2 and 3. This is a small step in a recent paradigm that it would be nice to explain complexity problems using category theory. For instance, see the recent paper of Abramsky, Dawar, and Wang [1] for a natural formulation of existential k-pebble games — a powerful tool used in database theory — using comonads.

Remark 3.14. As lexicons are logical refinement systems, we could easily generalise by defining lexicons to be logical refinement systems, hence the domain and codomain of a lexicon are not necessarily SMCC but monoidal categories with possibly only some exponents. Doing so, we may restrict the order of an ACG by forcing the domain of the lexicon to be monoidal with exponents up to a certain order. For instance, we may restrict exponents to obtain second-order ACG, and it will probably be interesting to study second-order ACG in such a setting.

3.2 Logical relations

First, notice that the technique used by de Groote in [19] for logical relations uses a certain intermediate type transformer T. The idea is that the new feature the extended ACG is dealing with may be provided by a generic transformation on terms and types, then somehow instantiated by $\mathbb{E}$. If we look more carefully, $\mathbb{E}$ is either the identity or a map forgetting the transformation given by T, and $\mathbb{P}$ is actually performing the embedding into the transformed type. Hence we either consider embeddings and projections as the main thing or the type transformer T, but not both. It means we either choose to focus on logical relations between lexicons and T helps in expressing them, or we choose to focus on type transformers T that help construct extended lexicons dealing with a new feature given by T. Both approaches are of interest and we start by the first one.

Suppose given two lexicons $F, G : \mathcal{C} \to \mathcal{D}$ and a pair $(\mathbb{E}, \mathbb{P})$ of embedding and projection families. The pair $(\mathbb{E}, \mathbb{P})$ is parametrised by types $\alpha \in \mathcal{C}$ of the abstract vocabulary, which is reminiscent of natural transformations. This leads us to the following definition:

Definition 3.15 (Logical relation)**.** *A logical relation is a natural transformation* $\mathbb{E} : \mathscr{L}_1 \to \mathscr{L}_2$ *between two lexicons* $\mathscr{L}_1, \mathscr{L}_2 : \mathcal{C} \to \mathcal{D}$.

To express the idea of conservativity, the intuitive idea was that what could be stated using the first lexicon is embedded in what can be stated using the second lexicon, and such that no information is lost, meaning we can go back using projections. This intuition leads to the following:

Definition 3.16 (Conservative extension)**.** *We say that a lexicon* $\mathscr{L}_2 : \mathcal{C} \to \mathcal{D}$ *is a conservative extension of a lexicon* $\mathscr{L}_1 : \mathcal{C} \to \mathcal{D}$ *when there exist two logical relations* $\mathbb{E} : \mathscr{L}_1 \to \mathscr{L}_2, \mathbb{P} : \mathscr{L}_2 \to \mathscr{L}_1$ *such that* $\mathbb{P}\mathbb{E} = Id$.

An interesting point in our the approach is that the existence of the two logical relations only at atomic types and for the types of the constants is sufficient to extend it.

Theorem 3.17. *Suppose $\mathcal{C}$ is a free CCC. Then $\mathscr{L}_2 : \mathcal{C} \to \mathcal{D}$ is a conservative extension of $\mathscr{L}_1 : \mathcal{C} \to \mathcal{D}$ if there exist two natural families $\mathbb{E} : \mathscr{L}_1 \to \mathscr{L}_2, \mathbb{P} : \mathscr{L}_2 \to \mathscr{L}_1$ indexed over basic types and types of constants of $\mathcal{C}$ only such that $\mathbb{P}\mathbb{E} = Id$.* □

We are now able to express the examples given by de Groote [19] in our setting:

Example 3.18 (Intentionalisation)**.** For every type α, we define

$$\begin{array}{ll} \mathbb{E}_\alpha : e \to (s \multimap e) & \mathbb{P}_\alpha : (s \multimap e) \to e \\ \mathbb{E}_\alpha(t) = \lambda i.t[\mathrm{I} := i] & \mathbb{P}_\alpha(t) = t\ \mathrm{I} \end{array}$$

Then, $\mathbb{P}_\alpha\mathbb{E}_\alpha(t) = t$ for every term $t : \alpha$.

Example 3.19 (Dynamisation)**.** We define

$$\begin{array}{llll} \mathbb{E}_e : e \to e & \mathbb{E}_t : t \to (c \multimap (c \multimap t) \multimap t) & \mathbb{P}_e : e \to e & \mathbb{P}_t : (c \multimap (c \multimap t) \multimap t) \to t \\ \mathbb{E}_e(t) = t & \mathbb{E}_t t = \lambda ck.t \wedge (kc) & \mathbb{P}_e t = t & \mathbb{P}_t t = t\ \mathrm{nil}\ (\lambda c.\mathrm{true}) \end{array}$$

We have the following, considering equalities modulo α, β, η-equivalence:

$$\begin{aligned} \mathbb{P}_e\mathbb{E}_e t &= (\lambda ck.t \wedge (kc))\ \mathrm{nil}\ (\lambda c.\mathrm{true}) \\ &= t \wedge ((\lambda c.\mathrm{true})\ \mathrm{nil}) \\ &= t \wedge \mathrm{true} \\ &= t \end{aligned}$$

This still does not work automatically for logical constants though, as they are given special translations. Also note that the logical connectives obey equations such as $a \wedge \top = a$ for all $a : t$.

Example 3.20 (Type raising)**.** We define, assuming the existence of the same constants L as in the paper of de Groote, the following:

$$\begin{array}{ll} \mathbb{E}_n : (e \multimap t) \to (e \multimap t) : t \mapsto t & \mathbb{P}_n : (e \multimap t) \to (e \multimap t) : t \mapsto t \\ \mathbb{E}_{np} : e \to ((e \multimap t) \multimap t) : t \mapsto \lambda k.kt & \mathbb{P}_{np} : ((e \multimap t) \multimap t) \to e : t \mapsto \mathbb{L}t \\ \mathbb{E}_s : t \to t : t \mapsto t & \mathbb{P}_s : t \to t : t \mapsto t \end{array}$$

L may actually be obtained by requiring the unit of the continuation monad to be a split-monomorphism, *i.e.* it has a left inverse.

In addition, we are able to express more interesting examples from Shan [52]:

Example 3.21 (Interrogatives)**.** The powerset monad $T\alpha = \alpha \to t$ is a crude model of non-determinism. For instance, the set of individuals m defined by $m \stackrel{def}{=} \{\text{John, Mary}\} : Te$ can be thought of as a non-deterministic individual : there is an ambiguity between *John* and *Mary*. Now suppose you want to add the possibility for non determinism in your language, such that it extends the deterministic semantics in a conservative way. You may do so as follows. Suppose you have a lexicon $\mathscr{L}_1 : \mathcal{C} \to \mathcal{D}$, where $\mathcal{D}$ has the two element set t of truth values. Then, a non deterministic version $\mathscr{L}_2 : \mathcal{C} \to \mathcal{D}$ is given by $\mathscr{L}_2(\alpha) \stackrel{def}{=} T\mathscr{L}_1(\alpha)$ on atomic types and by direct image on terms. It is however not clear how to define conservativity in that case. It is much easier in the next example.

Example 3.22 (Focus)**.** It is a variation of the powerset monad, where there is a designated element at each step, which captures the intuition that this element is focused, and that we keep track of a set of non-deterministic alternatives and a particular alternative in the set. It is defined by $T\alpha = \{(x_0, x_1) | x_0 \in x_1 \subset \alpha\}$. Given a lexicon $\mathscr{L}_1$, we define in a similar way as before $\mathscr{L}_2(\alpha) \stackrel{def}{=} T\mathscr{L}_1(\alpha)$ on atomic types. Then, we show $\mathscr{L}_2$ is a conservative extension of $\mathscr{L}_1$ using the following:

$$\mathbb{E}_\alpha : \mathscr{L}_1(\alpha) \to T\mathscr{L}_1 \qquad \mathbb{P}_\alpha : T\mathscr{L}_1 \to \mathscr{L}_1$$
$$\mathbb{E}_\alpha(t) = (t, \{t\}) \qquad \mathbb{P}_\alpha((t, L)) = t$$

Example 3.23 (Quantification)**.** We can express words requiring quantification such as *everyone* using the continuation monad $T\alpha = (\alpha \to t) \to t$. It avoids the need for quantifier raising and allows to deal with words manipulating the continuation in a non trivial way such as *everyone* which be assigned the λ-term $\lambda c. \forall x. c(x)$ which is not of the form $\lambda c. c(\dots)$. We may establish conservativity along the same way as in type-raising, but it also needs to suppose given constants L.

However, what has just been given is somehow an incomplete or unsatisfactory explanation. In addition, in the above examples we actually knew the extension was going to be conservative as the now hidden type transformer T is given by a monad whose unit is a monomorphism. Indeed, intentionalisation is given by the reader monad $T\alpha = c \to \alpha$ for a certain type c, type raising and quantification by the continuation monad $T\alpha = (\alpha \to \omega) \to \omega$ for a certain type ω, in these cases t, and finally focus by the pointed powerset monad $T\alpha = \{(x_0, x_1) | x_0 \in x_1 \subset \alpha\}$.

It is therefore of interest to look in detail what can be said using the second point of view. We have the following proposition, giving a partial explanation to what happened in the previous examples:

Theorem 3.24. *If $\mathscr{L}_1 : \mathcal{C} \to \mathcal{D}$ is a lexicon between free CCC with constants and $T : \mathcal{D} \to \mathcal{D}$ a monad on T whose unit is a split-monomorphism, then there exists a conservative extension $\mathscr{L}_2 : \mathcal{C} \to \mathcal{D}$ of $\mathscr{L}_1$ given on atomic types α by $\mathscr{L}_2(\alpha) \stackrel{def}{=} T\mathscr{L}_1(\alpha)$.* □

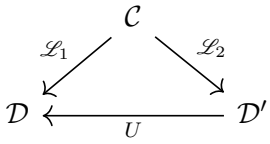
Finally, we may see conservativity and logical relations in the setting of Melliès and Zeilberger [40]. Remember that lexicons are logical refinement systems. Then, logical relations are sorts of morphisms of lexicons. This leads to the following from Melliès and Zeilberger [39]:

Definition 3.25 (Morphism of refinement systems)**.** *Given a pair of refinement systems $t : \mathcal{D} \to \mathcal{T}$ and $b : \mathcal{E} \to \mathcal{B}$, a morphism of refinement systems is a pair $F = (F_\mathcal{D}, F\mathcal{T})$ of functors $F_\mathcal{D} : \mathcal{D} \to \mathcal{E}$ and $F_\mathcal{T} : \mathcal{T} \to \mathcal{B}$ such that the following square commutes strictly:*

$$\begin{array}{ccc} \mathcal{D} & \xrightarrow{F_\mathcal{D}} & \mathcal{E} \\ {\scriptstyle t}\downarrow & & \downarrow{\scriptstyle b} \\ \mathcal{T} & \xrightarrow[F_\mathcal{T}]{} & \mathcal{B} \end{array}$$

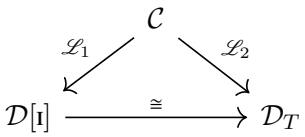
This leads to expressing conservativity results in a more uniform way, as all our work is expressed within the framework of refinements systems. However, we have to be more specific about what is the good way to represent conservativity. We have seen that a functor is a refinement system, hence the paradigm that the domain has more information that the codomain and the functor *forgets* information. We are interested in the particular case where the top functor of a morphism of refinement system is the identity, and $\mathscr{L}_2$ is a conservative extension of $\mathscr{L}_1$ if we can forget about some of the structure given by $\mathscr{L}_2$ to get $\mathscr{L}_1$, *i.e.* if there is a functor U such that $U \circ \mathscr{L}_2 = \mathscr{L}_1$, thus the following definition:

Definition 3.26 (Conservative extension)**.** *Given a pair of lexicons $\mathscr{L}_1 : \mathcal{C} \to \mathcal{D}$ and $\mathscr{L}_2 : \mathcal{C} \to \mathcal{D}'$, we say that $\mathscr{L}_2$ is a conservative extension of $\mathscr{L}_1$ when there exists a morphism of refinement systems from $\mathscr{L}_2$ to $\mathscr{L}_1$ of the form $(Id_\mathcal{C}, U)$,* i.e. *a functor $U : \mathcal{D}' \to \mathcal{D}$ such that the following triangle commutes strictly:*



Note that even if the condition on U seems to be weak, the fact that it has to make a triangle commute strictly, where the two other functors are very constrained — in that case monoidal closed — makes it very constrained too. This definition is really convenient as it indeed captures conservativity for all the examples, and in a more uniform way.

Example 3.27 (Intentionalisation)**.** The picture is the following:



In some sense, the constant I seemed roguish and the extended semantics given by $\mathscr{L}_2$ just seemed to do the same as $\mathscr{L}_1$ but in a proper way. What the above diagram shows is that those techniques are actually equivalent and a particular case of a theorem which can be found in the book of Lambek and Scott [29].

In the case where $\mathrm{I} : 1 \to C$, the theorem states that $\mathcal{D}[\mathrm{I}]$ is isomorphic to the Kleisli category of $\mathcal{D}$ for the reader monad T given by $TA = S \Rightarrow A$. In particular, the isomorphism sends any term t to $\lambda i.t[\mathrm{I} := i]$ which is precisely what we expect. Then, the reverse of this functor is a functor U taken as refinement system.

For the remaining examples, they may all be dealt with using the following theorem:

Theorem 3.28. *If $\mathscr{L}_1 : \mathcal{C} \to \mathcal{D}$ is a lexicon on free CCC with constants and $T : \mathcal{D} \to \mathcal{D}$ a monad such that the assignment $\alpha \mapsto T\alpha$ is injective, then there exists a conservative extension $\mathscr{L}_2$ of $\mathscr{L}_1$ given on atomic types α by $\mathscr{L}_2(\alpha) \stackrel{def}{=} T\mathscr{L}_1(\alpha)$.* □

Example 3.29 (Dynamisation)**.** There is a morphism of refinement systems $U : \mathcal{D} \to \mathcal{D}$ between the old lexicon and the dynamised version of it. It verifies $U(e) = e$ and $U(c \to (c \to t) \to t) = t$ as expected. This time, there is no problem with logical connectives as dynamic logical connectives are sent by U to their non-dynamic counterparts.

Example 3.30 (Type raising)**.** Similarly, there is a morphism of refinement systems $U : \mathcal{D} \to \mathcal{D}$ such that $U(t) = t$, $U(e \to t) = e \to t$ and $U((e \to t) \to t) = e$. Considering the conservative extension in this way avoids the need for the existence of $\mathbb{L}$.

Example 3.31 (Interrogatives)**.** Finally, there is a morphism of refinement system U sending $(\alpha \to t)$ to α. It matches for instance the intuition that non-determinism is added on top of determinism, allowing more flexibility.

And this also works in a very similar way for *focus* and *quantification*. We could go further using this notion of morphism of refinement systems but this is outside the scope of this report.

3.3 Distributions as enrichment

What we have done so far lies within abstract formal frameworks, looking for modularity and strong foundations. However, what is most commonly used in practice are learning methods. The core idea is that they rely on probabilities and are efficient at disambiguation, but do not capture subtle linguistics phenomena and hence are limited to simple sentences. One of the objectives of our work is to achieve a small step forward in

unifying formal linguistics and learning methods in a simple, nice and formal model. To achieve that, we import one of such methods which has been successfully used in linguistics and imported in a formal framework, developed in Coecke, Grefenstette, and Sadrzadeh [10] and improved in Preller [48].

Their framework relies on the fact that a Lambek monoid is in category theory a monoidal biclosed category. Hence it is equipped with a tensor and exponents. We obviously have it too with ACG and particular cases of ACG are those whose abstract language is generated by n, np, s which may benefit of the power of disambiguation of distributional models. We thus consider the following reformulation of the quantisation functor:

Definition 3.32 (Quantisation functor)**.** *A quantisation functor $\mathcal{Q} : \mathcal{C} \to \mathcal{FV}ect$ is a monoidal closed functor from a monoidal closed category $\mathcal{C}$ to the category $\mathcal{FV}ect$ of finite dimensional vector spaces over $\mathbb{R}$.*

The problem is though that we would like to use this functor whenever needed for disambiguation, but we want to keep the main categories we are working with in the central place. Indeed, we would like to keep the formal system very apparent, and category theory is to help us achieve that. We would like to avoid functors going everywhere whenever an information is needed. Thus, it would be nice if the vector space is at our disposal in the model, but somehow hidden, and transported when we are considering different ACG, composing them, or extending them.

One way to see this is to consider enriched categories. A locally small category $\mathcal{C}$ is a category such that for every $A, B \in \mathcal{C}$, the collection of morphisms $[A, B]$ from A to B is a set. It is a category enriched over Set. More generally, a category $\mathcal{C}$ enriched over $\mathcal{V}$ is such that for every $A, B \in \mathcal{C}$, $[A, B]$ is an object of $\mathcal{D}$. It is a natural way to add structure to the collection of morphisms of a category. We are here obviously interested in the case where $\mathcal{C}$ is a vocabulary and $\mathcal{D}$ is $\mathcal{FV}ect$.

Here is how we proceed. First, every monoidal closed category is naturally enriched over itself, taking $[A, B]$ to be $A \multimap B$. Next, suppose given a category $\mathcal{C}$ enriched over a monoidal closed category $\mathcal{V}$ and a monoidal closed functor $F : \mathcal{V} \to \mathcal{W}$. Then $\mathcal{C}$ is naturally $\mathcal{W}$-enriched. In our case, using the quantisation functor we are provided with monoidal closed categories naturally enriched over $\mathcal{FV}ect$. The nice thing is then that all our framework lifts to the enriched setting with no difficulties. Hence the compositional distributional framework is recovered in our setting as a $\mathcal{FV}ect$-enrichment of the vocabularies.

Theorem 3.33. *All the constructions of the previous sections lift to the setting of $\mathcal{FV}ect$-categories.* □

It is actually also convenient because it paves the way for further enrichments, allowing for more disambiguations and adding more effects in the framework. For instance, it might be the proper way to add probabilities and some kinds of contexts.

Example 3.34 (Montague semantics)**.** We may lift the lexicon from example 3.6 to distributional models. For instance, it may be seen as adding disambiguation on words such as *cat, dog* : n in a very similar way to what is happening with Lambek monoids.

Logical connectives may be conveniently recovered in the vector space model. Following Preller [48], the distributional model is called *logical functional* if the quantisation functor sends the type s of sentences to a two-dimensional space S with canonical basis vectors $\top, \bot$, nouns to sums of basic vectors, determiners and attributive adjectives to projectors, verbs and predicative adjectives to predicates, and logical words to logical connectives.

In this setting, given an orthonormal basis A and the free vector space on it V, a predicate is a linear map $P : V \to S$ such that any basis vector $a \in A$ verifies $P(a) \in \{0, \top, \bot\}$. It is a predicate on A if $P(a) \neq 0$ for all $a \in A$. Let X be a vector in V and P a predicate on A. The logic is four-valued with the following : $P(X)$ is *true* if $P(X)$ is not 0 and co-linear to $\top$, *false* is $P(X)$ is not 0 and co-linear to $\bot$, *mute* if $P(X) = 0$, and *mixed* if $P(X) = \alpha\top + \beta\bot$ for some $\alpha, \beta \neq 0$.

For instance there are linear maps $true : V \to S$ and $false : V \to S$ sending every $a \in A$ respectively to $\top$ and $\bot$. Logical connectives are given by the linear maps $not : S \to S, and : S \otimes S \to S, or : S \otimes S \to S$ and $ifthen : S \otimes S \to S$ determined by their values on the basis vectors $z \in \{\top \otimes \top, \top \otimes \bot, \bot \otimes \top, \bot \otimes \bot\}$ thus

$$and(z) = \begin{cases} \top & \text{if } z = \top \otimes \top \\ \bot & \text{else} \end{cases} \qquad or(z) = \begin{cases} \bot & \text{if } z = \bot \otimes \bot \\ \top & \text{else} \end{cases}$$

$$ifthen(z) = \begin{cases} \bot & \text{if } z = \top \otimes \bot \\ \top & \text{else} \end{cases} \qquad not(\top) = \bot \quad not(\bot) = \top$$

If P and Q are predicates on A, then so are $not \circ P, and \circ \langle P, Q\rangle, or \circ \langle P, Q\rangle, ifthen \circ \langle P, Q\rangle$.

Example 3.35. Again considering Montague semantics, we may now see the lexicon as interpreting nouns of type n such as *cat* into predicates over the individuals of type $e \to t$, but in the vector-space setting, permitting to use the machinery of disambiguation as desired. For instance, the individuals that are *cats* are sent to a vector co-linear to $\top$ whereas individuals such as *cats and dogs* have the truth value *mixed* as some of them are indeed *cats* while the dogs are not.

3.4 Adding sub-typing

One aspect that has not been explored yet in ACG is the possibility to add sub-typing and deal with coercions phenomena. Again, we want to add it in a modular way in order to have a great level of generality. We also want it to be compatible with the previous sections.

It is of interest for disambiguation to be able to add information on top of typing. For instance, consider the sentence *He met the criteria*. The verb *meet* has two possible interpretations: *encounter* and *satisfy*. *encounter* expects individuals whereas *satisfy* does not. Hence the coercion from the type of individuals i to the type of the argument of *encounter* is allowed whereas the one from i to the type of the argument of *satisfy* is rejected.

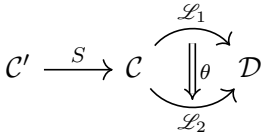
This is a particular instance of a sub-typing problem and it arises very quickly in linguistics. Suppose you are given an ACG $\langle \mathcal{C}, F, \mathcal{D}\rangle$ where $\mathcal{C}$ is freely generated by n, np, s and some constants. We would like to be able to enrich the typing of the abstract vocabulary to be able to deal with coercion issues. A way to do so is given by the following:

Definition 3.36 (ACG with sub-typing)**.** *An ACG $\langle \mathcal{C}, F, \mathcal{D}\rangle$ may be equipped with a logical refinement system $S : \mathcal{C}' \to \mathcal{C}$ for sub-typing from a monoidal closed category $\mathcal{C}'$ to the category $\mathcal{C}$ of the abstract vocabulary of the ACG.*

You may notice that U is just another lexicon, but in this report we do not consider the ACG $\langle \mathcal{C}', F \circ S, \mathcal{D}\rangle$. The reason lies in the role that is given to each functor. Roughly speaking, F is making a link between an abstract and an object vocabularies while U is enriching the type of $\mathcal{C}$ for disambiguation purposes.

This functor is to be seen on top of the ACG and as such should not be affected by logical relations. Fortunately, the different ways to express logical relations from the previous section lift to ACG with sub-typing using the following notion of right-action of a functor on a natural transformation:

Definition 3.37 (Right-action)**.** *Given the following situation:*



The functor S acts on the natural transformation $\theta : \mathscr{L}_1 \to \mathscr{L}_2 : \mathcal{C} \to \mathcal{D}$ and transports it into the natural transformation $\theta \circ_R S : \mathscr{L}_1 \circ S \to \mathscr{L}_s \circ S : \mathcal{C}' \to \mathcal{D}$ whose instance at α is defined as the morphism $\mathscr{L}_1 \circ S(\alpha) \overset{\theta_{S(\alpha)}}{\longrightarrow} \mathscr{L}_2 \circ S(\alpha)$

Proposition 3.38. *ACG with sub-typing may be considered in the setting of $\mathcal{FV}ect$-categories. In addition, the previous notions of logical relations lift to this setting.*

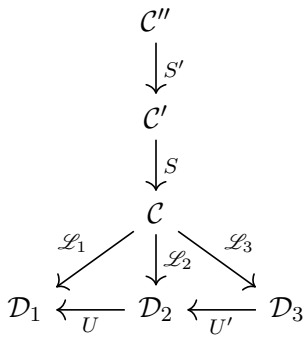
Sub-typing may however be considered in the setting of enriched categories and it allows more disambiguation combined with the distributional semantics we already added. We just added one more layer in our model which allows for more control but without adding more expressivity. It is just a sound and convenient way to represent and deal with more phenomena. This convenience is showed in the following examples:

Example 3.39. Consider an ACG $\langle \mathcal{C}, \mathscr{L}, \mathcal{D} \rangle$ for which $\mathcal{C}$ is given by a free SMCC on basic types $\{n, np, s\}$. Then, we can construct the free SMCC $\mathcal{C}'$ whose basic types are pairs such as $(n, \textit{dog}), (n, \textit{man}), (n, \textit{cat})$ and the sub-typing functor is given by the evident forgetful functor.

The same process can be used to add possible cases, *i.e.* pattern-matching. For instance you may consider a sub-typing functor S sending *(human,female), (human,male), (human,female+male)* to *human*.

Finally, we may deal with coercion phenomena. For instance, suppose given a constant *read* : $A \to B$ for some types A, B. Now understand A as the physical-informational $p \cdot i$ in dot-types, and you may solve the problem of coercion in *He read the rumour* by considering a sub-typing functor S sending $p, i, p \cdot i$ to A.

To sum up, our framework is of the following general scheme:



where we can compose sub-typing, ACG, logical relations, adding in a very controlled way all sorts of features such as pattern-matching, non-linearity, sub-typing, or distributional semantics.

4 Conclusion and further work

Due to space constraints, we will not repeat the conclusion of our report here, but refer the reader to page 2 for a summary of contributions and a discussion of future work. We also encourage the interested reader to go to appendix C for several detailed ideas for future work.

5 Acknowledgement

I am very grateful to my advisor, Philippe de Groote. I need to thank Aleksander Maskharashvili for his advice and time explaining various notions and ideas. I am also thankful to Sylvain Pogodalla, Paul-André Melliès and Noam Zeilberger for insightful discussions. Many thanks to the reviewers. Finally, I need to say that the Sémagramme team was really great to be part of.

References

[1] Samson Abramsky, Anuj Dawar, and Pengming Wang. "The pebbling comonad in finite model theory". In: *arXiv preprint arXiv:1704.05124* (2017).

[2] Jirí Adámek, Horst Herrlich, and George E. Strecker. *Abstract and Concrete Categories. The Joy of Cats*. Wiley-Interscience, 1990.

[3] Thorsten Altenkirch, James Chapman, and Tarmo Uustalu. "Monads need not be endofunctors". In: *International Conference on Foundations of Software Science and Computational Structures*. Springer. 2010, pp. 297–311.

[4] Andrea Asperti and Giuseppe Longo. *Categories, types, and structures: an introduction to category theory for the working computer scientist*. MIT press, 1991.

[5] S. Awodey. *Category Theory*. Oxford Logic Guides, 2006.

[6] Hendrik Pieter Barendregt et al. *The lambda calculus*. Vol. 2. North-Holland Amsterdam, 1984.

[7] Michael Barr and Charles Wells. *Toposes, triples and theories*. Vol. 278. Springer-Verlag New York, 1985.

[8] Gilad Ben-Avi and Yoad Winter. "The Semantics of Intensionalization". In: *ESSLLI 2007* (2007), p. 98.

[9] Francis Borceux and Isar Stubbe. "Short introduction to enriched categories". In: *Current Research in Operational Quantum Logic*. Springer, 2000, pp. 167–194.

[10] Bob Coecke, Edward Grefenstette, and Mehrnoosh Sadrzadeh. "Lambek vs. Lambek: Functorial vector space semantics and string diagrams for Lambek calculus". In: *Annals of pure and applied logic* 164.11 (2013), pp. 1079–1100.

[11] Robin Cooper et al. "Probabilistic type theory and natural language semantics". In: *LiLT (Linguistic Issues in Language Technology)* 10 (2015).

[12] Philippe De Groote and Makoto Kanazawa. "A note on intensionalization". In: *Journal of Logic, Language and Information* 22.2 (2013), pp. 173–194.

[13] Philippe De Groote and Sylvain Pogodalla. "On the expressive power of abstract categorial grammars: Representing context-free formalisms". In: *Journal of Logic, Language and Information* 13.4 (2004), pp. 421–438.

[14] Stanislas Dehaene et al. "The neural representation of sequences: from transition probabilities to algebraic patterns and linguistic trees". In: *Neuron* 88.1 (2015), pp. 2–19.

[15] Jean Gillibert and Christian Retoré. *Category theory, logic and formal linguistics: some connections, old and new*. 2014.

[16] Michele Giry. "A categorical approach to probability theory". In: *Categorical aspects of topology and analysis*. Springer, 1982, pp. 68–85.

[17] Eric Goubault and Sergio Rajsbaum. "A simplicial complex model of dynamic epistemic logic for fault-tolerant distributed computing". In: *arXiv preprint arXiv:1703.11005* (2017).

[18] Philippe de Groote. "Modularity and compositionality: The case of temporal modifiers". In: *Semantics and Linguistic Theory*. Vol. 25. 2017, pp. 656–675.

[19] Philippe de Groote. "On logical relations and conservativity". In: *NLCS'15. Third Workshop on Natural Language and Computer Science*. Vol. 32. 2015, pp. 1–11.

[20] Philippe de Groote. "Towards abstract categorial grammars". In: *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*. Association for Computational Linguistics. 2001, pp. 252–259.

[21] Philippe de Groote. “Tree-adjoining grammars as abstract categorial grammars”. In: *TAG+ 6, Proceedings of the sixth International Workshop on Tree Adjoining Grammars and Related Frameworks*. 2002, pp. 145–150.

[22] Philippe de Groote and Sarah Maarek. “Type-theoretic extensions of abstract categorial grammars”. In: (2007).

[23] Mark P Jones and Luc Duponcheel. *Composing monads*. Tech. rep. Technical Report YALEU/DCS/RR-1004, Department of Computer Science. Yale University, 1993.

[24] Aravind K Joshi and Yves Schabes. “Tree-adjoining grammars”. In: *Handbook of formal languages*. Springer, 1997, pp. 69–123.

[25] Laura Kallmeyer and Rainer Osswald. “Syntax-driven semantic frame composition in Lexicalized Tree Adjoining Grammars”. In: *Journal of Language Modelling* 1.2 (2014), pp. 267–330.

[26] Ohad Kammar and Matija Pretnar. “No value restriction is needed for algebraic effects and handlers”. In: *Journal of Functional Programming* 27 (2017).

[27] Makoto Kanazawa. “Parsing and generation as datalog queries”. In: *ACL*. Vol. 7. 2007, pp. 176–183.

[28] Oleg Kiselyov. “Applicative Abstract Categorial Grammar”. In: *NLCS@ ICALP/LICS*. 2015, pp. 29–38.

[29] Joachim Lambek and Philip J Scott. *Introduction to higher-order categorical logic*. Vol. 7. Cambridge University Press, 1988.

[30] Alex Lascarides and Nicholas Asher. “Segmented discourse representation theory: Dynamic semantics with discourse structure”. In: *Computing meaning*. Springer, 2008, pp. 87–124.

[31] Ekaterina Lebedeva. “Expression de la dynamique du discours á l’aide de continuations”. PhD thesis. PhD thesis, Université de Lorraine, 2012. URL http://hal. inria. fr/docs/00/78/32/45/PDF/Thesis-submitted_version. pdf, 2012.

[32] Sheng Liang, Paul Hudak, and Mark Jones. “Monad transformers and modular interpreters”. In: *Proceedings of the 22nd ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. ACM. 1995, pp. 333–343.

[33] Saunders Mac Lane. *Categories for the working mathematician*. Vol. 5. Springer Science & Business Media, 2013.

[34] Saunders MacLane and Ieke Moerdijk. *Sheaves in geometry and logic: A first introduction to topos theory*. Springer Science & Business Media, 2012.

[35] Jirka Maršík. “Effects and Handlers in Natural Language”. PhD thesis. Université de Lorraine, 2016.

[36] Jirka Maršík and Maxime Amblard. “Introducing a Calculus of Effects and Handlers for Natural Language Semantics”. In: *International Conference on Formal Grammar*. Springer. 2016, pp. 257–272.

[37] Scott Martin and Carl Pollard. “Hyperintensional Dynamic Semantics”. In: *Formal grammar*. Springer. 2012, pp. 114–129.

[38] Conor McBride and Ross Paterson. “Applicative programming with effects”. In: *Journal of functional programming* 18.01 (2008), pp. 1–13.

[39] Paul-André Melliès and Noam Zeilberger. “An Isbell duality theorem for type refinement systems”. In: *Mathematical Structures in Computer Science* (2017), pp. 1–39.

[40] Paul-André Melliès and Noam Zeilberger. “Functors are type refinement systems”. In: *ACM SIGPLAN Notices*. Vol. 50. 1. ACM. 2015, pp. 3–16.

[41] Eugenio Moggi. “Notions of computation and monads”. In: *Information and computation* 93.1 (1991), pp. 55–92.

[42] Uwe Mönnich. "Adjunction as substitution: An algebraic formulation of regular, context-free and tree adjoining languages". In: *arXiv preprint cmp-lg/9707012* (1997).

[43] Richard Montague. "English as a formal language". In: (1970).

[44] Richard Montague. "The proper treatment of quantification in ordinary English". In: *Philosophy, language, and artificial intelligence*. Springer, 1973, pp. 141–162.

[45] Gordon Plotkin and Matija Pretnar. "Handlers of algebraic effects". In: *European Symposium on Programming*. Springer. 2009, pp. 80–94.

[46] Carl Pollard. "Are (Linguists') Propositions (Topos) Propositions?" In: *Logical Aspects of Computational Linguistics* (2011), pp. 205–218.

[47] Carl Pollard. "Covert movement in logical grammar". In: *Logic and grammar*. Springer, 2011, pp. 17–40.

[48] Anne Preller. "From logical to distributional models". In: *arXiv preprint arXiv:1412.8527* (2014).

[49] James Pustejovsky. "The generative lexicon". In: *Computational linguistics* 17.4 (1991), pp. 409–441.

[50] Exequiel Rivas and Mauro Jaskelioff. "Notions of computation as monoids". In: *arXiv preprint arXiv:1406.4823* (2014).

[51] Chung-chieh Shan. "Linguistic side effects". PhD thesis. Harvard University Cambridge, Massachusetts, 2005.

[52] Chung-chieh Shan. "Monads for natural language semantics". In: *arXiv preprint cs/0205026* (2002).

[53] Tao Xue and Zhaohui Luo. "Dot-types and their implementation". In: *Logical aspects of computational linguistics* (2012), pp. 234–249.

[54] Ryo Yoshinaka. "Linearization of affine abstract categorial grammars". In: *Proceedings of the 11th conference on Formal Grammar*. 2006, pp. 185–199.

A Categorical foundations

For a survey on category theory in general, we recommend Mac Lane [33], Awodey [5], and Adámek, Herrlich, and Strecker [2]. For a survey more focused on computer science, see Lambek and Scott [29] and Asperti and Longo [4].

A.1 Monads

Definition A.1 (Monad)**.** *Let X be a category. A* monad (T, η, μ) *in X consists of an endofunctor $T : X \to X$ and natural transformations $\eta : 1 \dashrightarrow T$ and $\mu : T^2 \dashrightarrow T$ such that the following diagrams commute:*

$$\begin{array}{ccc} T^3 & \xrightarrow{T\mu} & T^2 \\ {\scriptstyle \mu T}\downarrow & & \downarrow{\scriptstyle \mu} \\ T^2 & \xrightarrow{\mu} & T \end{array} \quad \text{and} \quad \begin{array}{ccccc} 1 \circ T & \xrightarrow{\eta T} & T^2 & \xleftarrow[T\eta]{} & T \circ 1 \\ \downarrow{\scriptstyle =} & & \downarrow{\scriptstyle \mu} & & \downarrow{\scriptstyle =} \\ T & \xrightarrow{=} & T & \xleftarrow[=]{} & T \end{array}$$

The first diagram expresses the associativity of the multiplication and the second one expresses that the compatibility conditions between the unit and the multiplication.

An adjunction $F \dashv G$ for $F : X \to A$ always gives rise to a monad as follows. Let $(F, G, \eta, \epsilon) : X \rightharpoonup A$ be the adjunction, so the unit is $\eta : I \dashrightarrow GF$ and the counit $\epsilon : FG \dashrightarrow I$. Set

- $T = GF : X \to X$, so T is an endofunctor,

- $\eta : T \dot{\rightarrow} 1$ as the unit,
- $\mu = G\epsilon F : GFGF \dot{\rightarrow} GF$ as the multiplication, so $\mu : T^2 \dot{\rightarrow} T$

Then (T, η, μ) is a monad in X.

Definition A.2 (Strong monad)**.** *A monad is strong when its underlying functor is strong, which means it has a tensorial strength satisfying the following axioms :*

$$\begin{array}{ccc} I \otimes TA & \xrightarrow{st} & T(I \otimes A) \\ & \searrow^{\lambda} & \downarrow{\scriptstyle T(\lambda)} \\ & & TA \end{array} \qquad \begin{array}{ccccc} (A \otimes B) \otimes TC & & \xrightarrow{\qquad st \qquad} & & T((A \otimes B) \otimes C) \\ \downarrow{\scriptstyle \alpha} & & & & \downarrow{\scriptstyle T(\alpha)} \\ A \otimes (B \otimes TC) & \xrightarrow[1 \otimes st]{} & A \otimes T(B \otimes C) & \xrightarrow[st]{} & T(A \otimes (B \otimes C)) \end{array}$$

The strength has to be compatible with the unit and multiplication of the monad, which means it also has to satisfy the following coherence axioms :

$$\begin{array}{ccc} A \otimes B & \xrightarrow{1 \otimes \eta} & A \otimes TB \\ & \searrow^{\eta} & \downarrow{\scriptstyle st} \\ & & T(A \otimes B) \end{array} \qquad \begin{array}{ccccc} A \otimes T^2B & \xrightarrow{st} & T(A \otimes TB) & \xrightarrow{T(st)} & T^2(A \otimes B) \\ \downarrow{\scriptstyle 1 \otimes \mu} & & & & \downarrow{\scriptstyle \mu} \\ A \otimes TB & & \xrightarrow[st]{\qquad\qquad} & & T(A \otimes B) \end{array}$$

A.2 Monoidal categories

Definition A.3 (Strict monoidal category)**.** *A* strict monoidal category $(B, \cdot, e)$ *is a category* B *with a bifunctor* $\cdot : B \times B \to B$*, which is associative and has* e *as a two-sided unit.*

A strict monoidal category is just a monoid in Cat.

Here a *bifunctor* just refers to a functor out of a product category.

Definition A.4 (Relaxed monoidal category)**.** *A* relaxed monoidal category *or* lax monoidal category *is* $(B, \cdot, e, \alpha, \lambda, \rho)$ *in which the associativity and unit laws don't hold on the nose, but only up to natural isomorphisms* α*,* λ*,* ρ *witnessing the associativity, left unit law and right unit law respectively.*

A (relaxed or strict) monoidal category is called *Cartesian* if the monoid structure $(C, I, \otimes)$ is given by $(C, T, \times)$ for T the terminal object and $\times$ the usual category-theoretic product.

Definition A.5 (monoidal functor)**.** *A functor between monoidal categories that preserves the (closed) monoidal structure – so there are arrows making the associativity and unit diagrams commute. If there are just arrows (not isomorphisms) for these diagrams, the functor is called a* lax *or a* relaxed *monoidal functor. A* strong monoidal functor *is a monoidal functor such that these arrows are isomorphisms.*

For example, a functor that preserves finite products preserves the tensor product structure on a Cartesian monoidal category (since this structure is just the product), and so is a strong monoidal functor.

Definition A.6 (Closed monoidal category)**.** *A monoidal category is called* closed *if every functor* $(-) \otimes B$ *given by tensoring with (any object)* B *has a right adjoint* $(-)^B$*. The right adjoint is commonly denoted by* $B \multimap (-)$*,* $[B, -]$*,* $A \mapsto (B \Rightarrow A)$ *or* $(-)^B$*; the last one is often used when the tensor product on* C *is the Cartesian product.*

The adjoint rule forces a form of evaluation onto the functor $(-)^B$, just as in the case of exponentials. Thus, specifying a monoidal category is closed is equivalent to specifying that for each $A, B \in C$ we have an object $A \multimap B$ and a morphism $ev : (A \multimap B) \otimes A \to B$ satisfying the usual universal property for evaluation. This

defines a functor $C^{op} \times C \to C$, called the *internal hom functor*. Then $A \Rightarrow B$ is called the *internal hom* of A and B. More formally:

Definition A.7 (Internal hom functor)**.** *For C a monoidal category, an* internal hom *is a functor $(-) \multimap (-) : C^{op} \times C \to C$ such that for each $X \in C$ we have an adjoint pair $(-) \otimes X \dashv X \multimap (-)$.*

Thus, a monoidal category with an internal hom is the same thing as a closed monoidal category. To construct this, use currying. Define a covariant family of functors $F_X : X \mapsto (-) \otimes X$. This is equivalent to defining a contravariant family $F'_X : X \mapsto X \multimap (-)$. Then set $G(X, Y) := F'_X(Y) = Y \otimes X$. This defines a functor $C^{op} \times C \to C$, as required.

Intuitively, the internal hom assigns an object $X, Y \mapsto X \multimap Y$ that acts like the hom object of arrows from X to Y.

A.3 Enriched categories

Definition A.8. *$\mathcal{V}$-category is a generalisation of category theory in which we allow the hom functors to take values in any closed monoidal category, not just Set. (Usually it is required to have hom-*sets*. In this construction, we generalise to hom-*objects *taken from a closed monoidal category).*

Any locally small category is enriched over Set. For any category C we have hom-sets $C(a, b)$. The composition and identity rules specify mappings saying that our hom-sets satisfy the associativity and unit laws:

$$C(a, b) \times C(b, c) \to C(a, c)$$
$$1 \to X(a, a)$$

where $\times$ is the usual Cartesian product in Set. So the hom-objects are just sets (standard sets of arrows), and the enriched operation is usual composition.

Example A.9.

Let C be a poset category. Let the hom functors take values in the Cartesian closed category $\mathbf{2}$ by setting $C(a, b)$ as $true$ if $a \geq b$ and $false$ else.

These objects satisfy the required associativity and unit laws in $\mathbf{2}$, by taking the arrows to be entailment and the unit to be $true$:

$$C(a, b) \wedge C(b, c) \vdash C(a, c)$$
$$true \vdash X(a, a)$$

because if $a \geq b$ and $b \geq c$ then $a \geq c$ by the transitivity law, while always $a \geq a$ by the reflexivity law.

For a K-additive category (cf. homological algebra) C, the hom-sets are K-module morphisms and the tensor product gives K-linear maps

$$C(a, b) \otimes C(b, c) \to C(a, c)$$
$$K \to X(a, a)$$

We enrich a category C by using the monoidal structure to define composition satisfying the category axioms within C. As in the above cases, we replace our hom sets with hom objects from some underlying (closed monoidal) category.

Definition A.10 (Enriched category)**.** *Let $(\mathcal{V}, \otimes, I)$ be a (lax or strict) monoidal category. We take a set R of 'objects' $r, s, t \in \mathcal{V}$ and to each pair assign an 'arrow' object $R(r, s)$. We call this the* hom object *of arrows from r to s. Arrow composition is then given by the tensor operation as $R(s, t) \otimes R(r, s) \to R(r, t)$. For units, we assign to each object r an arrow $e \to R(r, r)$ in $\mathcal{V}$. Then we make the whole thing satisfy the usual category axioms to get a $\mathcal{V}$-category.*

More formally, we can think of this as making the following assignments: a $\mathcal{V}$-category is a pair $(X, d : X \times X \to \mathcal{V})$ given by

- a set $X \in \mathcal{V}$, the 'objects' of our enriched category; and
- a function $X \times X \to \mathcal{V}$ assigning hom objects to each pair

such that composition is given by the tensor product, and the associativity and unit laws hold.

Note that to turn this into an actual set-theoretic category (at the moment we only have hom objects and not hom *sets*) we need to apply an underlying functor.

Another way of viewing an (enriched) $\mathcal{V}$-category is as a set X with a (lax) monoidal functor.

Definition A.11 (Enriched category)**.** *Let $\mathcal{V}$ be a monoidal category. Then a* category enriched over $\mathcal{V}$ *or* $\mathcal{V}$-category *is a set X with a lax monoidal functor $\Phi = \Phi_d$ given by*

$$\mathcal{V}^{op} \xrightarrow{y} Set^X \xrightarrow{d} Set^{X \times X}$$

where we identify $Set^{X \times X}$ with $Hom(X, X)$ in the bicategory of spans.

A.3.1 Examples of enriched categories

A closed monoidal category is enriched: we take the hom objects to be given by the internal hom $A \multimap B$.

Proposition A.12 (Closed monoidal categories are enriched over themselves)**.** *Let $(C, \otimes, I)$ be a closed monoidal category. Then C is a C-category.*

Proof. We take:

- objects: objects $A, B \in C$,
- hom objects: exponentials $A \multimap B$ given by the closed property

We need to define a composition operation $(B \multimap C) \otimes (A \multimap B) \to (A \multimap B)$ and a choice of unit $1 \to (A \multimap A)$.

For the composition, we use evaluation and the category's associator:

$$(B \multimap C \otimes A \multimap B) \otimes A \xrightarrow{assoc} B \multimap C \otimes (A \multimap B \otimes A) \xrightarrow{id \otimes eval} B \multimap C \otimes B \xrightarrow{eval} C$$

This is a map $(B \multimap C \otimes A \multimap B) \otimes A \to C$ and hence equivalent, by the adjunction, to a map $B \multimap C \otimes [A \multimap B \to A \multimap C$, as required. For the identity we take the left unit from the category $1 \otimes A \to A$ and argue likewise. □

Example A.13.

Let Ab be the category of Abelian groups with their homomorphisms. It is a monoidal category with tensor product as the tensor. Then an Ab-category with one object is a monoid (category with one object) that gains the extra structure of another operation, with a unit. Since we're enriching over Ab, moreover, this extra structure is Abelian – so we get a ring structure.

A poset is an enriched category over $\mathbf{2}$.

A.3.2 Enriched functors

We have a corresponding idea of enriched functors as functors that preserve the enriched structure. Roughly speaking, this is just a bunch of arrows from our closed monoidal category, taking one collection of objects over $\mathcal{V}$ to the other collection of objects over $\mathcal{V}$ and preserving the hom objects.

So for $\mathcal{V}$-enriched categories $\mathcal{C}$ and $\mathcal{D}$, this needs to map

- maps of objects: (objects of $\mathcal{V}$ acting as objects of $\mathcal{C}$) to (objects of $\mathcal{V}$ acting as objects of $\mathcal{D}$), and
- maps of arrows: (pairs of objects of $\mathcal{V}$ acting as arrows in $\mathcal{C}$) to (pairs of objects of $\mathcal{V}$ acting as arrows of $\mathcal{D}$)

such that the identity and composition laws for a functor hold for the 'composition' operations defined in $\mathcal{C}$ and $\mathcal{D}$.

Thus the definition is :

Definition A.14 (Enriched functor)**.** *Let $\mathcal{C}$ and $\mathcal{D}$ be $\mathcal{V}$-categories. Then a* $\mathcal{V}$-functor $F : \mathcal{C} \to \mathcal{D}$ *is a mapping of the objects of $\mathcal{C}$ into those of $\mathcal{D}$, together with the assignment of a $\mathcal{V}$-morphism $\mathcal{C}(a,b) \xrightarrow{F_{a,b}} \mathcal{D}(Fa, Fb)$ for each pair (a,b) of $\mathcal{C}$-objects, we have a functorial assignment:*

$$\begin{array}{ccc} k & \xrightarrow{\eta^{\mathcal{C}}_a} & \mathcal{C}(a,a) \\ & {\scriptstyle \eta^{\mathcal{D}}_F a} \searrow \quad \swarrow & \\ & \mathcal{D}(Fa, Fa) & \end{array} \qquad \begin{array}{ccc} \mathcal{C}(a,b) \otimes \mathcal{C}(b,c) & \xrightarrow{\mu^{\mathcal{C}}_{a,b,c}} & \mathcal{C}(a,c) \\ {\scriptstyle F_{a,b} \otimes F_{b,c}} \downarrow & & \downarrow {\scriptstyle F_{a,c}} \\ \mathcal{D}(Fa,Fb) \otimes \mathcal{D}(Fb,Fc) & \xrightarrow[\mu^{\mathcal{D}}_{Fa,Fb,Fc}]{} & \mathcal{D}(Fa,Fc) \end{array}$$

where μ^X and η^X represent the multiplication and the unit in the $\mathcal{V}$-category X.

The second diagram says that (multiplying according to the rule for $\mathcal{C}$, then applying the morphism assigned by the enriched functor) is the same as (applying the morphisms from the enriched functor, then multiplying according to the rule for $\mathcal{D}$).

More equationally, the definition is:

Definition A.15 (Enriched functor)**.** *Let $\mathcal{V}$ be a monoidal category and suppose that $\mathcal{C}$ and $\mathcal{D}$ are $\mathcal{V}$-categories. An* enriched functor *or* $\mathcal{V}$-functor $F : \mathcal{C} \to \mathcal{D}$ *consists of*

- *a function $F_0 : \mathcal{C}_0 \to \mathcal{D}_0$ between the respective underlying sets,*
- *a $(\mathcal{C}_0 \times \mathcal{C}_0)$-indexed collection of morphisms in $\mathcal{V}$, $F_{x,y} : \mathcal{C}(x,y) \to \mathcal{D}(F_0 x, F_0 y)$, compatible with the enriched identities and compositions on $\mathcal{C}$ and $\mathcal{D}$, and*
- *such that the maps $F_{x,y}$ respect the functorial axioms.*

Here we see the functor is a collection of morphisms $\mathcal{C}(x,y) \to \mathcal{D}(F_0 x, F_0 y)$ – and the close connection between strengths on functors and natural transformations of this form.

A.3.3 Enriched natural transformations

In the non-enriched setting, a natural transformation $\alpha : F \dashrightarrow G$ is a family of maps $\alpha_x : Fx \to Gx$ such that we have the following naturality square:

$$\begin{array}{ccc} Fx & \xrightarrow{Ff} & Fy \\ {\scriptstyle \alpha_x}\downarrow & & \downarrow{\scriptstyle \alpha_y} \\ Gx & \xrightarrow[Gf]{} & Gy \end{array}$$

In the enriched setting, we need to replace the composition by the tensor product $\otimes$. Suppose we have functors $F, G : \mathcal{C} \to \mathcal{D}$ for $\mathcal{C}$ and $\mathcal{D}$ both $\mathcal{V}$-categories. As in the non-enriched case, we want a family of maps $Fa \to Ga$; this corresponds to a family of objects $\mathcal{D}(Fa, Ga)$ indexed by $a \in \mathcal{C}$. We can collect these together as the product $\prod_{a \in \mathcal{C}} \mathcal{D}(Fa, Ga)$, recalling that a product $\prod_{(x:I)} B_x$ corresponds to I-indexed sequences in B.

We want our family of maps to be such that the two ways round the naturality diagram commute, where composition is given by the tensor product. In other words, we want to ensure that, for each object $\mathcal{C}(a, b)$ – each arrow $a \to b$ – then the two routes round the naturality square from Fa to Gb are equal.

Expressing this as a process: we want to be able to take any $a \to b$, apply F and G to obtain $Fa \to Fa$ and $Ga \to Gb$ such that the composite $Fa \to Fb \to Gb$ is equal to the composite $Fa \to Ga \to Gb$:

$$\begin{array}{ccc} Ga & \longrightarrow & Gb \\ \downarrow & & \downarrow \\ Fa & \longrightarrow & Fb \end{array}$$

In other words, we want the two ways round below to be equal:

$$\begin{array}{ccc} Ga & \longrightarrow & Gb \\ \uparrow & \nearrow_{\otimes} & \\ Fa & & \end{array} \quad \text{and} \quad \begin{array}{ccc} & & Gb \\ & \nearrow_{\otimes} & \uparrow \\ Fa & \longrightarrow & Fb \end{array}$$

But the top diagram amounts to an object $\mathcal{D}(Ga, Gb) \otimes \mathcal{D}(Fa, Ga)$, and the bottom diagram amounts to an object $\mathcal{D}(Fb, Gb) \otimes \mathcal{D}(Fa, Fb)$. Commutativity of the above square is then the claim that using the multiplication to evaluate each of these tensor products gives the same thing.

So we can express our naturality condition as saying that we want a mapping such that for any object $\mathcal{C}(a, b)$, then we have that applying multiplication μ to get

$$\mathcal{D}(Ga, Gb) \otimes \mathcal{D}(Fa, Ga) \xrightarrow{\mu} \mathcal{D}(Fa, Gb)$$

is equal to applying multiplication to get

$$\mathcal{D}(Fb, Gb) \otimes \mathcal{D}(Fa, Fb) \xrightarrow{\mu} \mathcal{D}(Fa, Gb)$$

This can all be summarised by requiring a map α such that the diagram below commutes:

$$\begin{array}{ccccc} & & I \otimes \mathcal{C}(a,b) & \xrightarrow{G \otimes \alpha} & \mathcal{D}(Ga, Gb) \otimes \mathcal{D}(Fa, Fa) \\ & \overset{\lambda}{\nearrow} & & & \searrow^{\mu} \\ \mathcal{C}(a,b) & & & & \mathcal{D}(Fa, Gb) \\ & \underset{\rho}{\searrow} & & & \nearrow_{\mu} \\ & & \mathcal{C}(a,b) \otimes I & \xrightarrow[\alpha \otimes F]{} & \mathcal{D}(Fb, Gb) \otimes \mathcal{D}(Fa, Fb) \end{array}$$

This gives us:

Definition A.16 (Enriched natural transformation)**.** *An* enriched natural transformation *of $\mathcal{V}$-functors $F, G : \mathcal{C} \nrightarrow \mathcal{D}$ is a family of arrows in $\mathcal{V}$*

$$\alpha = \{\alpha_a : I \to \mathcal{D}(Fa, Ga)\}$$

such that the above diagram commutes.

Alternatively, we can see this as defining a form of equalizer. Recall that in Set, an equalizer of functions f and g is the collection of elements in their common domain where they agree: $Eq(f, q) = \{x \in X : fx) = g(x)\}$. Similarly, we can use the equalizer to define an enriched natural transformation as a sub-collection of the collection of all arrows $(Fa \to Ga)_{(a \in \mathcal{C})}$ such that the two routes round the naturality diagram are equal. We choose maps f and g out of $(Fa \to Ga)_{(a \in \mathcal{C})}$ to represent each of the two routes, and then take their equalizer.

Recall that a product $\prod_{(i:I)} A_i$ can be seen as I-indexed sequences taking elements from $A = (A_i)_{(i \in I)}$. So a non-enriched natural transformation $\alpha : F \nrightarrow G : \mathcal{C} \to \mathcal{D}$ is as a $\mathcal{C}$-indexed family $(\alpha_c : Fc \to Gc)_{(c \in \mathcal{C})}$, and hence an element of $\prod_{(c \in C)} \mathcal{D}(Fc, Gc)$ satisfying an extra (naturality) condition.

Now let $\mathcal{C}$ and $\mathcal{D}$ be $\mathcal{V}$-categories. To capture the naturality square, we look at all maps $\mathcal{C}(a, b) \to \mathcal{D}(Fa, Gb)$ as before – that is, at $\prod_{(a,b \in \mathcal{C})}[\mathcal{C}(a, b), \mathcal{D}(Fa, Gb)]$, where $[-, -]$ represents the internal hom. We choose f and g to be the two routes round the diagram from the previous definition of an enriched natural transformation, so we have

$$f, g : \prod_{(a \in \mathcal{C})} \mathcal{C}(Fa, Ga) \rightrightarrows \prod_{(a,b \in \mathcal{C})} [\mathcal{C}(a, b), \mathcal{D}(Fa, Gb)]$$

Then we define the $\mathcal{V}$-object of natural transformations $F \nrightarrow G$ to be the equalizer of f and g:

Definition A.17 (Enriched natural transformation)**.** *Let $\mathcal{C}$ and $\mathcal{D}$ be $\mathcal{V}$-categories, and $F, G : \mathcal{C} \nrightarrow \mathcal{D}$ enriched functors. Let f and g be the maps representing the two routes round the diagram from the previous definition. Then the* $\mathcal{V}$-object of natural transformations *$Y^a(F, G)$ is the equalizer in $\mathcal{V}$ of the diagram*

$$f, g : \prod_{(a \in C)} \mathcal{C}(Fa, Ga) \rightrightarrows \prod_{(a,b \in \mathcal{C})} [\mathcal{C}(a, b), \mathcal{D}(Fa, Gb)]$$

A.3.4 The effect of functors on the underlying monoidal category

It turns out that a closed monoidal functor is sufficient to push an enriched structure between categories :

Proposition A.18 (Closed monoidal functors carry enrichments)**.** *Let $\Phi : \mathcal{V} \to \mathcal{W}$ be a closed monoidal functor, and $\mathcal{C}$ a $\mathcal{V}$-category. Then we can define a $\mathcal{W}$-category $\Phi\mathcal{C}$ with the same set of objects as $\mathcal{C}$.*

Proof. Set the hom objects to be those given by applying the functor to hom objects in $\mathcal{C}$: $(\Phi\mathcal{C}) = \Phi(\mathcal{C}(a, b))$. Since our functor is closed monoidal, the diagrams expressing $\mathcal{C}$ as an enriched category carry over, and $\Phi\mathcal{C}$ is an enriched category. □

Similarly, applying a functor of the underlying closed monoidal categories also allows us to take $\mathcal{V}$-functors to $\mathcal{W}$-functors. Given a $\mathcal{V}$-functor $F : \mathcal{C} \to \mathcal{D}$ (so an enriched one, not a functor of the underlying categories), then setting $(\Phi f)_{a,b} := \Phi(f_{a,b})$ gives a $\mathcal{W}$-functor $\Phi\mathcal{C} \to \Phi\mathcal{D}$.

B Proofs

We give most of the proofs of propositions claimed in the result section. The equalities are considered modulo α, β, η-equivalence. 1 may denote the terminal object, the unit of a monoidal category or the identity morphism. For convenience in the reading of the proof, we sometimes skip indices.

Proposition 3.7. *Let $\mathcal{B}$ be the free symmetric monoidal closed category on a discrete category A of atomic types, and let $\mathcal{B}'$ be the monoidal closed category where projections and diagonal maps for atomic types are freely added, and quotiented by the equations for a Cartesian product. Then, $\mathcal{B}'$ is Cartesian closed.*

Let $F : \mathcal{C} \to \mathcal{D}$ be a lexicon between two free symmetric monoidal closed categories. Then, the extension $F' : \mathcal{C}' \to \mathcal{D}'$ of F preserving the diagonal and projection maps on the nose is well defined and is still a lexicon.

Proof. 1. $\mathcal{B}'$ is a CCC: we are working by freely adding morphisms and equations. Hence $\mathcal{B}'$ is still monoidal closed and it is Cartesian because we just added what was required. Thus $\mathcal{B}'$ is Cartesian monoidal closed;

2. Just define $F'(\pi_A) = \pi_{FA}, F'(\Delta_A) = \Delta_{FA}$, F' equals F on the rest of $\mathcal{C}$, and extend F to be a functor by induction, *i.e.* defining $F'(f \circ g) \stackrel{def}{=} F'(f) \circ F'(g)$.

□

Proposition 3.8. *Let $\mathcal{C}$ be a Cartesian closed category. Then $\mathcal{C}$ is monoidal closed and the extra morphisms given by the universal structure of the Cartesian product are generated by projections and diagonal maps. Let $\mathcal{C}''$ be the subcategory of $\mathcal{C}$ with the same objects as $\mathcal{C}$ and same morphisms except the morphisms generated by projections and diagonal maps. Then $\mathcal{C}''$ is symmetric monoidal closed.*

Proof. 1. $\mathcal{C}$ is monoidal closed as it is Cartesian monoidal closed;

2. Let's denote by Δ the diagonal map, *i.e.* $\Delta \stackrel{def}{=} \langle id, id \rangle$. If $h = \langle f, g \rangle$ is a map given by the universal property of the Cartesian product, then it is easy to show that $(f \times g) \circ \Delta$ satisfies the same universal property, thus $h = (f \times g) \circ \Delta$ is obtained using Δ. If we further remove projections, there are only maps of the kind $f_1 \times \cdots \times f_n$;

3. $\mathcal{C}''$ is monoidal by the same Cartesian monoidal structure. The symmetry $\sigma_{A,B} : A \times B \to B \times A$ is given by the universal property of the product $\langle \pi_2, \pi_1 \rangle$. The closed structure is given by the restriction of the closed structure on $\mathcal{C}$.

□

Proposition 3.9. *Let $F : \mathcal{C} \to \mathcal{D}$ be a lexicon between two CCC such that $F(f) = \pi \Rightarrow f = \pi$ and $F(f) = \Delta \Rightarrow f = \Delta$ for all $f \in \mathcal{C}$, all projections maps π, and all diagonal maps Δ. Then, the restriction $F'' : \mathcal{C}'' \to \mathcal{D}''$ of F is still a lexicon.*

Proof. As lexicons are monoidal closed functor and $\mathcal{C}'', \mathcal{D}''$ are monoidal closed subcategories of monoidal closed categories, the only thing to show is that F'' is well defined. As F was, this is equivalent to saying that no arrow in $\mathcal{D} - \mathcal{D}''$ is in the image of $\mathcal{C}''$ by F. This is ensured by the condition in the proposition. □

Theorem 3.10. *Let $F : \mathcal{C} \to \mathcal{D}$ be a lexicon whose vocabularies are free SMCC. Then, the extension $F' : \mathcal{C}' \to \mathcal{D}'$ is again a lexicon provided that $\mathcal{D}'$ contains the image by F' of the projections and diagonal maps added in $\mathcal{C}'$.*

Let $F : \mathcal{C} \to \mathcal{D}$ be a lexicon whose vocabularies are CCC. Then, the restriction $F'' : \mathcal{C}'' \to \mathcal{D}''$ is again a lexicon provided that no morphisms in $\mathcal{D} - \mathcal{D}''$ are in the image of $\mathcal{C}''$ by F. □

Proof. It is simple generalisation of the previous propositions. The only requirement is for the image of the lexicon to fit, which is ensured by the hypothesis, and the rest is automatically transported to the restriction or extension of the lexicon. □

Proposition 3.13 (Characterisation of order 2 ACGs)**.** *An ACG on CCC is of order 2 iff any constant is obtained by currying and the universal property of the Cartesian product from morphisms of type* $\alpha_1 \times \cdots \times \alpha_n \multimap \beta$ *where all types are atomic.*

Proof. Suppose given a constant c. If the codomain of c is a product type $B_1 \times \cdots \times B_k$, it is equivalent to k constants by the universal property of the Cartesian product. Hence we suppose that all the constants have a non product type.

Then, c is of the form $1 \to (A_1 \to A_2 \to \cdots \to B$, where A_i may be products, but can't contain abstractions as the constants are of order at most 2. By currying, we obtain $(1 \times A_1 \times ...) \to B$. Therefore it is isomorphic to $(\alpha_1 \times \cdots \times \alpha_n) \to B$ for some α_i of atomic types.

Now, B may be a product type, and we can redo the previous steps, and the process eventually terminates. This finally gives the desired morphisms of the form $(\alpha_1 \times \cdots \times \alpha_m) \to \beta$ for all α_i and β of atomic types.

For the other direction in the proof, just notice that morphisms of this kind are of order at most 2, and that this order does not grow using the universal property of the Cartesian product and currying, where the order of a term whose domain is a product $(a \times b) \to c$ is the order of its curried version $a \to (b \to c)$, defined by induction. □

Theorem 3.17. *Suppose* $\mathcal{C}$ *is a free CCC. Then* $\mathscr{L}_2 : \mathcal{C} \to \mathcal{D}$ *is a conservative extension of* $\mathscr{L}_1 : \mathcal{C} \to \mathcal{D}$ *if there exist two natural families* $\mathbb{E} : \mathscr{L}_1 \to \mathscr{L}_2, \mathbb{P} : \mathscr{L}_2 \to \mathscr{L}_1$ *indexed over basic types and types of constants of* $\mathcal{C}$ *only such that* $\mathbb{P}\mathbb{E} = Id$*.* □

Proof. Suppose given $(-)^*, (\bar{-}) : \mathcal{C} \to \mathcal{D}$ two lexicons.

We show the proof in the case where the categories are CCC. The proof is even simpler in SMCC. We define by induction $\mathbb{E}_{\alpha\times\beta} \stackrel{def}{=} \mathbb{E}_\alpha \times \mathbb{E}_\beta$ and $\mathbb{E}_{\alpha\multimap\beta} \stackrel{def}{=} \Lambda(\mathbb{E}_\beta \circ ev(1 \times \mathbb{P}_\alpha)) = \mathbb{P}_\alpha \multimap \mathbb{E}_\beta$. Thus, we define $\mathbb{P}$ in a similar way, *i.e.* by induction $\mathbb{P}_{\alpha\times\beta} \stackrel{def}{=} \mathbb{P}_\alpha \times \mathbb{P}_\beta$ and $\mathbb{P}_{\alpha\multimap\beta} \stackrel{def}{=} \mathbb{E}_\alpha \multimap \mathbb{P}_\beta$.

It easily gives $\mathbb{P}_\alpha\mathbb{E}_\alpha = 1_\alpha$ for all types α by induction as it is true on atomic types. Indeed, we have the following equalities:

$$\begin{aligned}\mathbb{P}_{\alpha\times\beta}\mathbb{E}_{\alpha\times\beta} &= (\mathbb{P}_\alpha \times \mathbb{P}_\beta) \circ (\mathbb{E}_\alpha \times \mathbb{E}_\beta)\\ &= (\mathbb{P}_\alpha \circ \mathbb{E}_\alpha) \times (\mathbb{P}_\beta \circ \mathbb{E}_\beta)\\ &= 1_{\mathscr{L}_1\alpha} \times 1_{\mathscr{L}_1\beta}\\ &= 1_{\mathscr{L}_1(\alpha\times\beta)}\end{aligned}$$

$$\begin{aligned}\mathbb{P}_{\alpha\multimap\beta}\mathbb{E}_{\alpha\multimap\beta} &= (\mathbb{E}_\alpha \multimap \mathbb{P}_\beta) \circ (\mathbb{P}_\alpha \multimap \mathbb{E}_\beta)\\ &= (\mathbb{P}_\alpha \circ \mathbb{E}_\alpha) \multimap (\mathbb{P}_\beta \circ \mathbb{E}_\beta)\\ &= 1_{\mathscr{L}_1\alpha} \multimap 1_{\mathscr{L}_1\beta}\\ &= 1_{\mathscr{L}_1(\alpha\multimap\beta)}\end{aligned}$$

For convenience in the following we rename the lexicons, so $\mathbb{E} : (-)^* \dot{\to} (\bar{-})$ is a natural transformation is what we show next. To do so, we need to prove commuting squares for every morphism f in $\mathcal{C}$. There are several kinds of such morphisms, and we have the commutative squares for each of the associated squares. It is shown by induction.

- identity are both left and right neutrals and are preserved by functors so they are ok.

- projection maps:

$$\begin{array}{ccc} \alpha^* \times \beta^* & \xrightarrow{\mathbb{E}_{\alpha\times\beta}} & \bar{\alpha} \times \bar{\beta} \\ {\scriptstyle \pi_1}\downarrow & & \downarrow{\scriptstyle \pi_1} \\ \alpha^* & \xrightarrow[\mathbb{E}_\alpha]{} & \bar{\alpha} \end{array}$$

  The square commutes by the universal property of the Cartesian product and the definition of $\mathbb{E}_{\alpha\times\beta}$. Similarly for the second projection and $\mathbb{P}$.

- products:

$$\begin{array}{ccc} \alpha^* \times \beta^* & \xrightarrow{\mathbb{E}_{\alpha\times\beta}} & \bar{\alpha} \times \bar{\beta} \\ {\scriptstyle f^*\times g^*}\downarrow & & \downarrow{\scriptstyle \bar{f}\times\bar{g}} \\ \gamma^* \times \delta^* & \xrightarrow[\mathbb{E}_{\gamma\times\delta}]{} & \bar{\gamma} \times \bar{\delta} \end{array}$$

  The square commutes by induction, the definition of $\mathbb{E}$ on a product and the fact that $\times$ is a bifunctor. Similarly for $\mathbb{P}$.

- universal property of product:

$$\begin{array}{ccc} \alpha^* \times \beta^* & \xrightarrow{\mathbb{E}_{\alpha\times\beta}} & \bar{\alpha} \times \bar{\beta} \\ {\scriptstyle \langle f^*,\, g^*\rangle}\uparrow & & \uparrow{\scriptstyle \langle \bar{f},\, \bar{g}\rangle} \\ \gamma^* & \xrightarrow[\mathbb{E}_\gamma]{} & \bar{\gamma} \end{array}$$

  By property of the Cartesian product and by induction, we have

$$\langle \bar{f},\bar{g}\rangle \circ \mathbb{E}_\gamma = \langle \bar{f} \circ \mathbb{E}_\gamma,\, \bar{g} \circ \mathbb{E}_\gamma\rangle = \langle \mathbb{E}_\alpha \circ f^*,\, \mathbb{E}_\beta \circ g^*\rangle$$

  We conclude by the universal property of the Cartesian product and the definition of $\mathbb{E}_{\alpha\times\beta}$.

- evaluation map:

$$\begin{array}{ccc} (\alpha^* \multimap \beta^*) \times \alpha^* & \xrightarrow{\mathbb{E}_{(\alpha\multimap\beta)\times\alpha}} & (\bar{\alpha} \multimap \bar{\beta}) \times \bar{\alpha} \\ {\scriptstyle ev}\downarrow & & \downarrow{\scriptstyle ev} \\ \beta^* & \xrightarrow[\mathbb{E}_\beta]{} & \bar{\beta} \end{array}$$

  This case is the only tricky one and we have to use all the hypotheses. We also need to use the universal property of Λ in the definition of $\mathbb{E}_{\alpha\multimap\beta}$. The above diagram expands to the following:

$$\begin{array}{ccccc} (\alpha^* \multimap \beta^*) \times \alpha^* & \xrightarrow{1\times\mathbb{E}_\alpha} & (\alpha^* \multimap \beta^*) \times \bar{\alpha} & \xrightarrow{\Lambda(\mathbb{E}_\beta ev(1\times\mathbb{P}_\alpha))\times 1} & (\bar{\alpha} \multimap \bar{\beta}) \times \bar{\alpha} \\ {\scriptstyle 1_{(\alpha^*\multimap\beta^*)\times\alpha^*}}\downarrow & \swarrow{\scriptstyle 1\times\mathbb{P}_\alpha} & & \searrow{\scriptstyle \mathbb{E}_\beta ev(1\times\mathbb{P}_\alpha)} & \downarrow{\scriptstyle ev} \\ (\alpha^* \multimap \beta^*) \times \alpha^* & \xrightarrow[ev]{} & \beta^* & \xrightarrow[\mathbb{E}_\beta]{} & \bar{\beta} \end{array}$$

  - Top-left triangle commutes as $\mathbb{P}_\alpha\mathbb{E}_\alpha = 1$;
  - Middle triangle commutes by definition;
  - Top-right triangle commutes by universal property of Λ.

Finally, the first line is just $\mathbb{E}_{(\alpha\multimap\beta)\times\alpha}$, and as 1 is a neutral, the above diagram is exactly the square from above.

- exponentials: for all $f : \gamma \to \alpha,\ g : \beta \to \gamma$ we have the following square:

$$\begin{array}{ccc} \alpha^* \multimap \beta^* & \xrightarrow{\mathbb{E}_{\alpha\multimap\beta}} & \bar{\alpha} \multimap \bar{\beta} \\ {\scriptstyle f^*\multimap g^*}\downarrow & & \downarrow{\scriptstyle \bar{f}\multimap\bar{g}} \\ \gamma^* \multimap \delta^* & \xrightarrow[\mathbb{E}_{\gamma\multimap\delta}]{} & \bar{\gamma} \multimap \bar{\delta} \end{array}$$

Then, by induction and the definition of $\mathbb{E}_{\alpha\multimap\beta}$, it is similar to the product except that $\multimap$ is covariant on the right and contravariant on the left.

Note that we can compose such conservative extensions, thus having a nice way to add several extensions in a conservative manner. □

Theorem 3.33. *All the constructions of the previous sections lift to the setting of $\mathcal{FV}ect$-categories.* □

Proof. We suppose we are a given quantisation functor $\mathcal{Q}$. We already sketched how to naturally enrich a category using a quantisation functor. Lexicons are monoidal closed functors, so lexicons preserve exponentials and hence preserve the natural enriched structure of monoidal closed categories. Let $\mathscr{L} : \mathcal{C} \to \mathcal{D}$ be a lexicon. Then, $\mathcal{C}$ is naturally a $\mathcal{C}$-category and $\mathcal{D}$ a $\mathcal{D}$-category. Using $\mathscr{L}$, $\mathcal{C}$ becomes a $\mathcal{D}$ category, and using the quantisation functor, they both become $\mathcal{FV}ect$ categories. Next, $\mathscr{L}$ is naturally $\mathcal{D}$-enriched. We may now use again what is used the appendix on the effect of functors on the underlying monoidal category to transport $\mathcal{V}$-lexicons into $\mathcal{W}$-lexicons and $\mathscr{L}$ is then $\mathcal{FV}ect$-enriched using the quantisation functor.

All the propositions about linearity and non-linearity don't change.

Logical relations become enriched natural transformations, and nothing else changes in the proofs. Everything in the refinement systems work in the enriched setting by basically adding the word "enriched" when needed. □

Proposition 3.38. *ACG with sub-typing may be considered in the setting of $\mathcal{FV}ect$-categories. In addition, the previous notions of logical relations lift to this setting.*

Proof. We already shown that we can work in the setting of enriched categories for ACG and as sub-typing functors are just lexicons, we may obviously work in this setting.

For logical relations, the version using functors is again evident given the theorem on lifting things to the enriched setting.

For the version using natural transformations, we just use the notion of right action to define new natural transformations out of the old ones. More precisely, consider a sub-typing functor $S : \mathcal{C}' \to \mathcal{C}$ for two ACG $\langle \mathcal{C}, \mathscr{L}_1, \mathcal{D}\rangle$, $\langle \mathcal{C}, \mathscr{L}_2, \mathcal{D}\rangle$ such that the second ACG is a conservative extension of the first one. Then, there exist two natural transformations $\mathbb{E}, \mathbb{P}$ such that $\mathbb{PE} = 1$. We define two new natural transformations $\mathbb{E}' \stackrel{def}{=} \mathbb{E} \circ_R S$ and $\mathbb{P}' \stackrel{def}{=} \mathbb{P} \circ_R S$. Those new natural transformations verify $\mathbb{P}'\mathbb{E}' = 1$. Indeed, for all type α we have $\mathbb{P}'_\alpha\mathbb{E}'_\alpha = \mathbb{P}_{S\alpha}\mathbb{E}_{S\alpha} = 1$. □

C Ideas for future work

C.1 Monad transformers and composition of monads

There are close links between monad transformers, monads and handlers. See Shan [52, 51] and Rivas and Jaskelioff [50] for monads and monad transformers used in linguistics. See for instance Kammar and Pretnar [26] for a nice use of handlers. It could be useful to use ideas from the thesis Maršík [35]. Composition of monads, which is not possible in general, has been developed, for instance in Jones and Duponcheel [23] and it has been found some sufficient conditions to compose monads, which might be satisfied in our case.

In particular, we could maybe see $\mathbb{B}_{\alpha\to\beta}$ as a monad morphism from the free monad or the extension of a monad using an adjunction. We could also gain in generality by considering relative monads from Altenkirch, Chapman, and Uustalu [3], which are no longer endofunctors and might well suit our needs. We could also wonder about the usefulness of the list monad in our case : non determinism, parallel computation, different points of view might be added on top of what we already have.

Lastly, we might have operators to compose monads or find a way to obtain a monad out of two using a sort of coproduct or pushout.

C.2 Tree Adjoining Grammars (TAG)

We could explore how TAG [24] behave from a categorical point of view as their image under the embedding $TAG \to ACG$ from [21]. In addition, it might be possible to see synchronous TAG as a functor $TAG \to TAG$, and we might consider similar ideas for extensions of TAG such as multiTAG.

Another point of view could be to explore further the weak equivalence between TAG and linear monadic context free tree grammars given in Mönnich [42]. Then, this could also be equivalent to some structure in a presheaf category. Lastly, the link between this last category and the encoding into the category of ACG might lead to new results, either categorically or more oriented toward linguistics.

C.3 Special case of pragmatics as a monad

Let $\mathcal{C}$ be a SMCC with constants, seen as abstractly modelling the meaning of a sentence. We consider a category whose objects are such categories and morphisms are lexicons. Then, we would like to see the extension of the semantics, in a way a special case of pragmatics, as follows. A semantics is a special case of pragmatics, the trivial pragmatics where there is no information brought by pragmatics on top of the information given by semantics. Pragmatics brings information to semantics, thus extends it. An iterated pragmatics is just pragmatics, we just have to sum the information obtained. Thus we can think of the pragmatics as a 2-monad T whose unit $\eta_{\mathcal{C}}$ sends the semantics $\mathcal{C}$ to the trivial pragmatics $T\mathcal{C}$. The multiplication states that a pragmatics of pragmatics can be flatten. Now we can see a morphism of the Kleisli category $F : \mathcal{C} \to T\mathcal{D}$ as the extension of the semantics $\mathcal{D}$, *i.e.* a pragmatics, taking information from the semantics of $\mathcal{C}$. It is a straightforward generalisation of the idea of continuation at a higher level of generality, highlighting the linguistics aspect of the situation. In addition, it might be a proper way to deal with local states — for instance people at work today versus people at work tomorrow — or to talk about different point of views — such as what I know about people at work versus what you know —.

What we have just shown is somehow just another view on semantics as pragmatics being in a sense the part of semantics we have not yet captured.

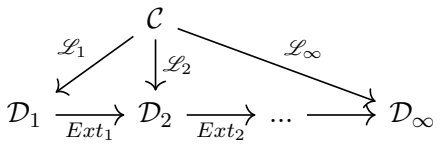
C.4 ∞-semantics

An idea could be to extend semantics again and again to deal with more and more phenomena. This is somehow intuitively what happens when you are to answer a question. If you don't take time to answer, basically you give a rough *yes/no* answer. But if you are given more time, you may think about exceptions, details, and the many subtleties that might influence your answer.

The *real* semantics would then be obtained as the colimit of these semantics using only partial information and a finite number of ideas/computations. An interesting fact with this point of view is the idea that you could compute every step to obtain a more and more accurate semantics.

Of course we can only compute a finite number of these semantics and get somehow a finite approximation of the real semantics. But it can be arbitrarily precise and this is actually close to the intuition we have just exemplified of the way the brain seems to work. If we are to give a short answer then our brain computes roughly the answer. But on the other hand if we are to give a more thoughtful answer, the brain takes much more time and it takes many things into consideration.

Here is a small diagram of how things could be seen abstractly:



where Ext means an extension of the semantics.

Maybe a good framework to see this is using the notion of nerve. Another route would be to go in presheaf categories, maybe those in the refinement systems of Melliès and Zeilberger [39].

Imagine the semantics of an answer is given in a CCC, or more precisely a topos [7, 34], which is, roughly speaking, the right place to do higher-order logic. Then, computing an answer for a longer period of time means you take more things into consideration, and it may be seen as a bigger topos equipped with a morphism from the old topos to the new one. And so on, this defines an ordered family of topoi. The question is whether the limit exists, *i.e.* whether there is a Platonic truth for which we may only have finite approximation. A very good point in favour of this view is the following theorem from MacLane and Moerdijk [34]:

Theorem C.1. *The category of topoi and logical morphisms has filtered colimits.* □

C.5 On composing monads

There is no way in general to compose two monads. However, we provide here a way to systematically compose two special kinds of monads, leading to an interesting way to compose extensions of semantics.

Remember that $\mathcal{C}[x : 1 \to A, y : 1 \to B]$ is the same as $\mathcal{C}[z : 1 \to A \times B]$ and that $\mathcal{C}[x]$ is isomorphic to the Kleisli category $\mathcal{C}_T$ for the monad T given by $T\alpha = A \Rightarrow \alpha$. Hence we can for instance deal with intentionalisation using x. Later, if we also want to deal with some temporality and add a similar procedure to the one of intentionalisation, there is a canonical way to do so using the following :

$$\mathcal{C}_T \xrightarrow{\cong} \mathcal{C}[x] \to \mathcal{C}[x, y] \xrightarrow{\cong} \mathcal{C}_M$$

for the monad M given by $M\alpha = A \times B \Rightarrow \alpha$.

C.6 Adding probabilities for disambiguation

An interesting way to explore would be dealing syntactically with ambiguities with non-determinism. Instead of computing a λ-term, it is possible for instance using the powerset monad, to compute a set of possible terms. It could also be refined using sub-typing.

On top of that, we could use probabilities on each term of the list of a list monad. One trivial thing to do could be to multiply probabilities when performing the application of a functor. Then, terms below a certain threshold would be removed. Maybe less trivial, when performing such an application, *i.e.* when a concatenation occurs, we could compare how likely the two words fit together and give some probability to it. We would remove things under a certain threshold and things could be compositional, but in a more elaborated way.

It is very close to the ideas used in parsing but could be raised at the semantics level. Consider *He met the criteria*. *meet* could mean *satisfy* or *encounter* but it is clear because of *the criteria* that in the above sentence it means *satisfy*. Thus, if we have some (semantic) knowledge that *meet* is used as *encounter*, the sentence would be rejected.

This technique could be developed further as follows. Imagine you get information on John, saying he is pacific and harmless. If we have to resolve an anaphora such as *He kills dogs*, most likely *he* will not be John. This would combine learning techniques to remove ambiguities at a higher level of generality and within a nice and powerful abstract framework.

C.7 Higher order predicates

C.7.1 Effectus theory

Recently people have developed a theory to deal with quantum computation : not only one should deal with $0, 1$ but one should consider a whole distribution. This can be done using a monad and working in the associated Kleisli category. Predicates over X are then maps $X \to 2 = 1 + 1$ in the Kleisli category, *i.e.* a map $X \to T2$ in the original category. This fruitful idea could be used in our case. It requires to have a distributive category, which in particular has finite coproducts, to deal with non-determinism. It offers a simple and nice framework to deal with higher order predicates.

However this raises several questions : would it be better to work in the current category or to work in the arrow category, where objects are morphisms of the original category. The former might me simpler but might not work or be powerful enough. The latter is closer to the intuitive idea that predicates take terms — *i.e.* morphisms in the category — as arguments.

C.7.2 Free cocompletion – topos

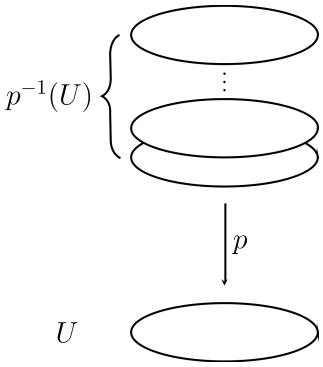
Now that we have put things into category theory, there are powerful techniques to freely add things to categories. One of them is the free cocompletion of a category. Given any category $\mathcal{C}$, the free cocompletion of $\mathcal{C}$ is a cocomplete category containing $\mathcal{C}$ and which satisfy a universal property. It can be shown that the category $\widehat{\mathcal{C}}$ of (contravariant) presheaves over $\mathcal{C}$ verifies such a universal property. In addition this category has a lot of nice properties (see for example my M1 report) and is often a topos or close to a topos. Getting such a topos would really help include higher order predicates in a very natural way into the theory. In addition, it can all be done using $\mathcal{V}$-enriched categories.

C.7.3 Hyperdoctrines

Another route could be to look in more details at the theory of hyperdoctrines (Lawvere). It allows to treat quantifiers at a very high level of generality and could be a natural way to generalise our theory to obtain more powerful (categorical) tools.

C.7.4 Fibrations – covering space

Maybe one could use $\mathbb{E}, \mathbb{P}$ to see semantics enrichment in a way similar to what people do in Homotopy Type Theory, using fibrations and similar ideas.



For instance, one could see possible worlds as covering spaces. In addition, we could possibly do the same to deal with time, conditional scope, or maybe even points of view from different people.

Another interesting point of view would be the following. Imagine semantics as a topological space. A meaning is given by an open set and it represents the idea that it can be extended, refined, compared with another. In a word, we may handle them in a way arguably close to the use we intuitively have of it, for instance when describing something like a feeling. Then, ambiguity can be seen as an open set made of several disjoint open sets. Two things or more don't agree when the intersection of their meaning is the empty set. Then, syntax is to semantics what invariant groups are to topological spaces. In particular, two non-homotopic paths are incompatible as for their meaning. In addition, choosing one path over the other might make it pass through one out of two components of an open set, thus a disambiguation. The idea is that syntax is rough and several meanings can be adjoined to syntax. On the other hand, a meaning can be described using several words, thus working up to homotopy or something close to it.

Lastly, this topological space could be given by a metric space or a manifold. This would have more structure than just the structure of topological space. In addition it could enable to learn the topology using learning methods or even manifold reconstruction. We could also imagine a directed version of all of it using directed algebraic topology [17] which might help dealing with non-invertible parameters such as time.

C.7.5 Several enrichments

We have seen that it might be fruitful to consider enriched categories in our framework to encompass distributional models. Another enrichment could be to enrich over Booleans or Abelian groups to gain extra structure. In addition, we might enrich over a product of categories equipped with point-wise tensor product to combine the effects of several enrichments. Other possibilities could be to enrich over topological spaces, metric spaces or even probability spaces, for instance using ideas from Cooper et al. [11] — which all have good reasons to be considered given the previous sections —.