Formalizing Typing Rules for Natural Languages using Effects

Matthieu Boyer

16th June 2025

Plan

General Introduction

This work, based on [BC25] aims to provide a categorical formalization of a type and effects system for semantic interpretation of the natural language.

We will develop a graphical formalism for semantic type-driven parsing and prove it is equivalent to a miinmalistic coloured merge interpretation of syntax.

Types in Semantics of Natural Languages

Expression	Type	λ-Term
planet	$\mathtt{e} \to \mathtt{t}$	$\lambda x.\mathbf{planet}\, x$
	Generalizes to **common nouns**	
carnivorous	$(\mathtt{e} \to \mathtt{t})$	$\lambda x.\mathbf{carnivorous}\, x$
	Generalizes to **predicative adjectives**	
skillful	$(\mathtt{e} \to \mathtt{t}) \to (\mathtt{e} \to \mathtt{t})$	$\lambda p.\lambda x.px \wedge \mathbf{skillful}\, x$
	Generalizes to **predicate modifier adjectives**	
Jupiter	$\mathtt{e}$	$\mathbf{j} \in \mathrm{Var}$
	Generalizes to **proper nouns**	
sleep	$\mathtt{e} \to \mathtt{t}$	$\lambda x.\mathbf{sleep}\, x$
	Generalizes to **intransitive verbs**	

Including Non-Determinisms and Anaphoras

What should be the type of expressions such as **a cat** or **Jupiter, a planet**?

Including Non-Determinisms and Anaphoras

What should be the type of expressions such as **a cat** or **Jupiter, a planet**?

Since we should be able to use **a cat** and **the cat** interexchangebly - from a syntax point of view - they should have the same type.

We use effects to do the difference between:

$$\mathbf{a\ cat} = \{c \mid \mathbf{cat}\, c\}$$

$$\mathbf{the\ cat} = x \text{ if } \mathbf{cat}^{-1}(\top) = \{x\} \text{ else } \#$$

Plan

A Type and Effect System

Let $\mathcal{L}$ be our language: a typed lexicon and syntactic rules. We only suppose that our words can be applied to one another in their denotation system.

A Type and Effect System

Let $\mathcal{L}$ be our language: a typed lexicon and syntactic rules. We only suppose that our words can be applied to one another in their denotation system.

Let $\mathcal{C}$ be a cartesian closed category used for typing the lexicon.

Let $\mathcal{F}(\mathcal{L})$ be a set of functors used for representing the words that add an effect to our language.

A Type and Effect System

Let $\mathcal{L}$ be our language: a typed lexicon and syntactic rules. We only suppose that our words can be applied to one another in their denotation system.

Let $\mathcal{C}$ be a cartesian closed category used for typing the lexicon.

Let $\mathcal{F}(\mathcal{L})$ be a set of functors used for representing the words that add an effect to our language.

We consider $\bar{\mathcal{C}}$ the categorical closure of $\mathcal{C}$ under the action of $\mathcal{F}(\mathcal{L})^*$. We close it for the cartesian product and exponential of $\mathcal{C}$.

A Type and Effect System

Let $\mathcal{L}$ be our language: a typed lexicon and syntactic rules. We only suppose that our words can be applied to one another in their denotation system.

Let $\mathcal{C}$ be a cartesian closed category used for typing the lexicon.

Let $\mathcal{F}(\mathcal{L})$ be a set of functors used for representing the words that add an effect to our language.

We consider $\bar{\mathcal{C}}$ the categorical closure of $\mathcal{C}$ under the action of $\mathcal{F}(\mathcal{L})^*$. We close it for the cartesian product and exponential of $\mathcal{C}$.

We also introduce the notions of applicatives and monads, as they grant more flexibility with semantic combinations.

Typing Rules

We then have typing judgements for basic combinations:

$$\frac{\Gamma \vdash x : \tau \qquad \Gamma \vdash F \in \mathcal{F}(\mathcal{L})}{\Gamma \vdash Fx : F\tau}\,\textsf{Cons}$$

$$\frac{\Gamma \vdash x : F\tau_1 \qquad \Gamma \vdash \varphi : \tau_1 \to \tau_2}{\Gamma \vdash \varphi x : F\tau_2}\,\texttt{fmap}$$

$$\frac{\Gamma \vdash x : \tau_1 \qquad \Gamma \vdash \varphi : \tau_1 \to \tau_2}{\Gamma \vdash \varphi x : \tau_2}\,\textsf{App}$$

Typing Rules

We then have typing judgements for basic combinations:

$$\frac{\Gamma \vdash x : A\tau_1 \qquad \Gamma \vdash \varphi : A\,(\tau_1 \to \tau_2)}{\Gamma \vdash \varphi x : A\tau_2}\text{<*>}$$

Typing Rules

Typing judgements for natural transformations:

$$\frac{\Gamma \vdash x : \tau}{\Gamma \vdash x : A\tau}\texttt{pure/return}$$

$$\frac{\Gamma \vdash x : MM\tau}{\Gamma \vdash x : M\tau}\texttt{>>=}$$

More generally:

$$\forall F \overset{\theta}{\Longrightarrow} G, \qquad \frac{\Gamma \vdash x : F\tau \qquad \Gamma \vdash G : S' \subseteq \star \qquad \tau \in S'}{\Gamma \vdash x : G\tau}\texttt{nat}$$

To ensure termination and decidability, we prevent the use of the unit rule out of the blue, more on that later.

Plan

Handlers

A handler for an effect F is a natural transformation $F \Rightarrow \mathrm{Id}$.

Handlers are not generally language-defined and are speaker-dependent. Adjunctions and comonads provide handlers intrinsic to their effects. Handlers should also be exact inverses to monadic and applicative units: this justifies semantically why we can remove the usage of the unit rule out of certain situations.

A large question we have to solve before parsing is whether two denotations will always yield the same result, considering effect handling.

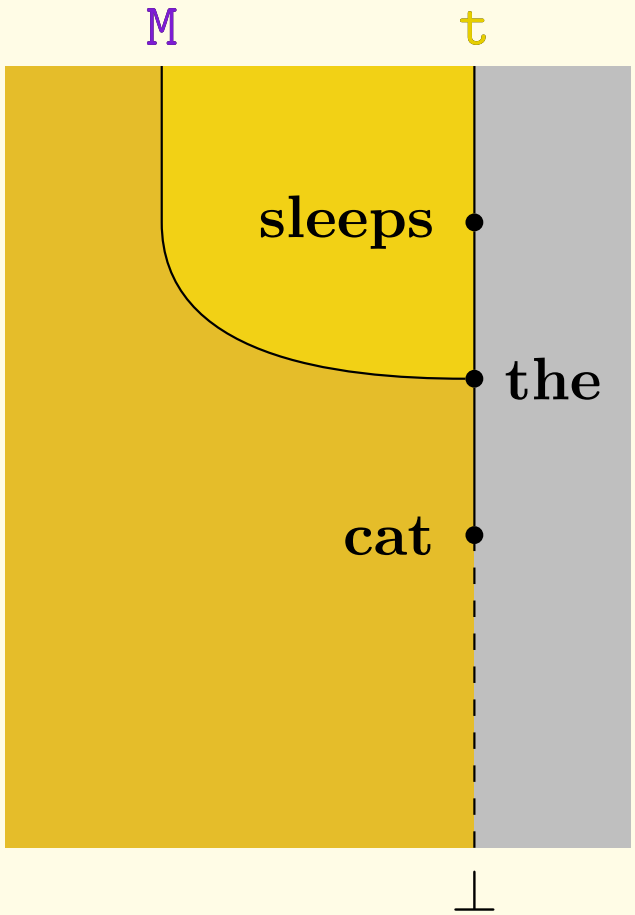
String Diagrams in Denotations - 1

We will represent our denotations as string diagrams to make computations easier and to better understand what happens to each effect when handling.

This allows us to look at equality of string diagrams instead of complex commutative diagrams.

What happens when combining diagrams and why they have such a shape will be detailed in the following section.

String Diagrams in Denotations - 2

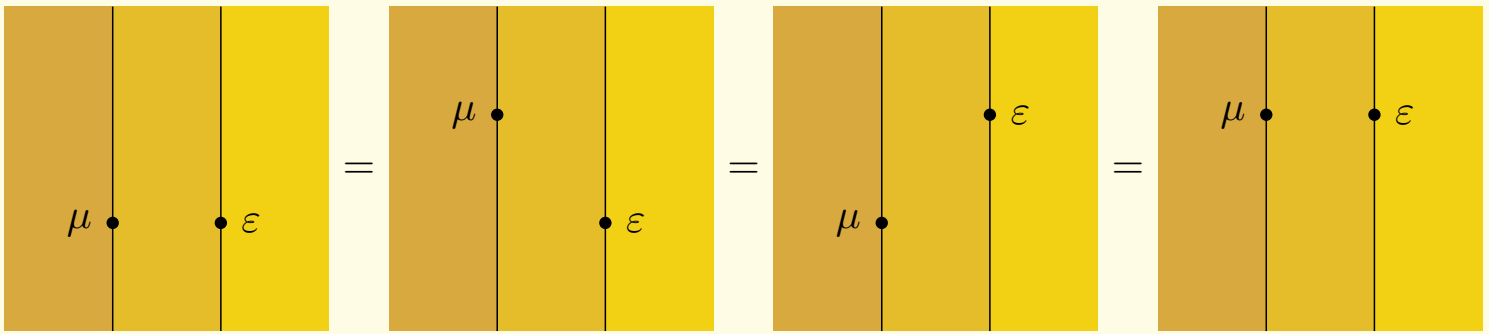


String Diagram Isotopy

Theorem 3.1 — **Theorem 3.1 [Sel10], Theorem 1.2 [JS91]** A well-formed equation between morphism terms in the language of monoidal categories follows from the axioms of monoidal categories if and only if it holds, up to planar isotopy, in the graphical language.

Commutation in String Diagrams

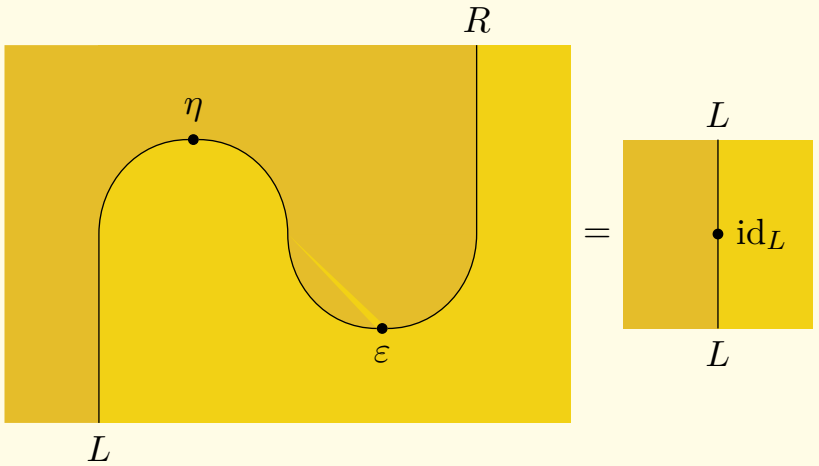
First, the elevator equations are a consequence of 3.1:



(🛗)

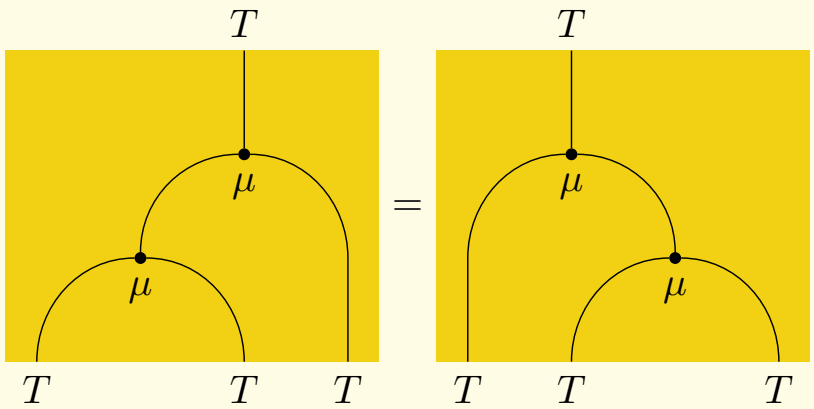
Commutation in String Diagrams

The Snake equations are a rewriting of the properties of an adjunction:



Commutation in String Diagrams

The Monadic equations are a rewriting of the properties of a monad:



(μ)

Reduction Scheme

[DV22] proposed a combinatorial description to check in linear time for equality under Theorem 3.1.

Reduction Scheme

[DV22] proposed a combinatorial description to check in linear time for equality under Theorem 3.1.

Theorem 3.3 — **Confluence** Our reduction system is confluent and therefore defines normal forms:

1. Right reductions are confluent and therefore define *right* normal forms for diagrams under the equivalence relation induced by exchange.
2. Equational reductions are confluent and therefore define *equational* normal forms for diagrams under the equivalence relation induced by exchange.

Polynomial Time Reductions

Theorem 3.4 — **Normalization Complexity** Reducing a diagram to its normal form is done in quadratic time in the number of natural transformations in it.

This is accomplished using a formalism based on [DV22].

Plan

Combining Phrases

A natural way to do the parsing would be to compute the valid syntactic parsing trees and then to map our typing rules on top of those.

However, english is not a context-free language [Hig84] and is moreover inherently ambiguous as the sentence *The man sees the girl with a telescope* proves it.

CFGs

We use a Context-Free Grammar to model our typing system and take its product with the syntax defining grammar.

$$
\begin{array}{lcl}
\tau & ::= & \tau \Rightarrow \tau \\
 & | & \langle \tau, \tau \rangle \\
 & | & \bar{\tau} \\
 & | & \mathrm{F}\tau \\
 & & \\
\bar{\tau} & ::= & \texttt{t} \mid \texttt{e} \mid \cdots \\
 & & \\
\mathrm{F} & ::= & \mathcal{F}\,(\mathcal{L})^* \\
 & & \\
\mathsf{Tree} & ::= & \tau, \tau
\end{array}
$$

$$
\begin{array}{lcll}
\mathsf{SM} & ::= & \mathsf{ML} \mid \mathsf{MR} & \textit{(fmap)} \\
 & | & \mathsf{UL} \mid \mathsf{UR} & \texttt{(pure)} \\
 & | & \mathsf{A} & \texttt{(Struct)} \\
 & | & \mathsf{J} & \texttt{(join)} \\
 & | & \mathsf{C} & \texttt{(counit)} \\
 & | & \mathsf{ER} \mid \mathsf{EL} & \texttt{(eject)} \\
 & | & \mathsf{DN} & \texttt{(closure)} \\
 & & & \\
\mathsf{M} & ::= & \mathsf{SM}, \mathsf{M} & \\
 & | & \mathsf{BComb} & \\
 & & & \\
\mathsf{BComb} & ::= & > \mid < \mid \wedge \mid \vee &
\end{array}
$$

CFGs - 2

$$
\begin{array}{lcl}
>, \beta & ::= & (\alpha \to \beta), \alpha \\
<, \beta & ::= & \alpha, (\alpha \to \beta) \\
\wedge, \alpha \to \mathtt{t} & ::= & (\alpha \to \mathtt{t}), (\alpha \to \mathtt{t}) \\
\vee, \alpha \to \mathtt{t} & ::= & (\alpha \to \mathtt{t}), (\alpha \to \mathtt{t})
\end{array}
$$

$$
\begin{array}{lcl}
\mathsf{J}_{\mathtt{F}}\ \mathtt{F}\tau & ::= & \mathtt{F}\mathtt{F}\tau \\
\mathsf{DN}_{\mathtt{C}}\ \tau & ::= & \mathtt{C}_{\tau}\tau
\end{array}
$$

$$
\begin{array}{lcl}
\mathsf{ML}_{\mathtt{F}}(\alpha, \beta) & ::= & \mathtt{F}\alpha, \beta \\
\mathsf{MR}_{\mathtt{F}}(\alpha, \beta) & ::= & \alpha, \mathtt{F}\beta \\
\mathsf{A}_{\mathtt{F}}(\alpha, \beta) & ::= & \mathtt{F}\alpha, \mathtt{F}\beta \\
\mathsf{UR}_{\mathtt{F}}(\alpha \to \alpha', \beta) & ::= & \mathtt{F}\alpha \to \alpha', \beta \\
\mathsf{UL}_{\mathtt{F}}(\alpha, \beta \to \beta') & ::= & \alpha, \mathtt{F}\beta \to \beta' \\
\mathsf{C}_{\mathtt{LR}}(\mathtt{L}\alpha, \mathtt{R}\beta) & ::= & (\alpha, \beta) \\
\mathsf{ER}_{\mathtt{R}}(\mathtt{R}(\alpha \to \alpha'), \beta) & ::= & \alpha \to \mathtt{R}\alpha', \beta \\
\mathsf{EL}_{\mathtt{R}}(\alpha, \mathtt{R}(\beta \to \beta')) & ::= & \alpha, \beta \to \mathtt{R}\beta'
\end{array}
$$

CFGs - 3

This grammar works in five major sections:

1. We reintroduce the grammar defining the type and effect system.
2. We introduce a structure for the semantic parse trees and their labels, the combination modes from [BC25].
3. We introduce rules for basic type combinations.
4. We introduce rules for higher-order unary type combinators.
5. We introduce rules for higher-order binary type combinators.

Combinators

The combinators introduced in the previous table all have actual effects on the denotations:

$$>= \lambda\varphi.\lambda x.\varphi x$$

$$<= \lambda x.\lambda\varphi.\varphi x$$

$$\mathsf{ML}_{\mathtt{F}} = \lambda M.\lambda x.\lambda y.(\mathtt{fmap}_{\mathtt{F}}\lambda a.M(a,y))x$$

$$\mathsf{MR}_{\mathtt{F}} = \lambda M.\lambda x.\lambda y.(\mathtt{fmap}_{\mathtt{F}}\lambda b.M(x,b))y$$

$$\mathsf{A}_{\mathtt{F}} = \lambda M.\lambda x.\lambda y.(\mathtt{fmap}_{\mathtt{F}}\lambda a.\lambda b.M(a,b))(x)\texttt{<*>}y$$

$$\mathsf{UL}_{\mathtt{F}} = \lambda M.\lambda x.\lambda\varphi.M(x,\lambda b.\varphi(\eta_{\mathtt{F}}b))$$

$$\mathsf{UR}_{\mathtt{F}} = \lambda M.\lambda\varphi.\lambda y.M(\lambda a.\varphi(\eta_{\mathtt{F}}a),y)$$

$$\mathsf{J}_{\mathtt{F}} = \lambda M.\lambda x.\lambda y.\mu_{\mathtt{F}}M(x,y)$$

$$\mathsf{C}_{\mathtt{LR}} = \lambda M.\lambda x.\lambda y.\varepsilon_{\mathtt{LR}}(\mathtt{fmap}_{\mathtt{L}}(\lambda l.\mathtt{fmap}_{\mathtt{R}}(\lambda r.M(l,r))(y))(x$$

$$\mathsf{EL}_{\mathtt{R}} = \lambda M.\lambda\varphi.\lambda y.M(\Upsilon_{\mathtt{R}}\varphi,y)$$

$$\mathsf{ER}_{\mathtt{R}} = \lambda M.\lambda x.\lambda\varphi.M(x,\Upsilon_{\mathtt{R}}\varphi)$$

$$\mathsf{DN}_{\Downarrow} = \lambda M.\lambda x.\lambda y.\Downarrow M(x,y)$$

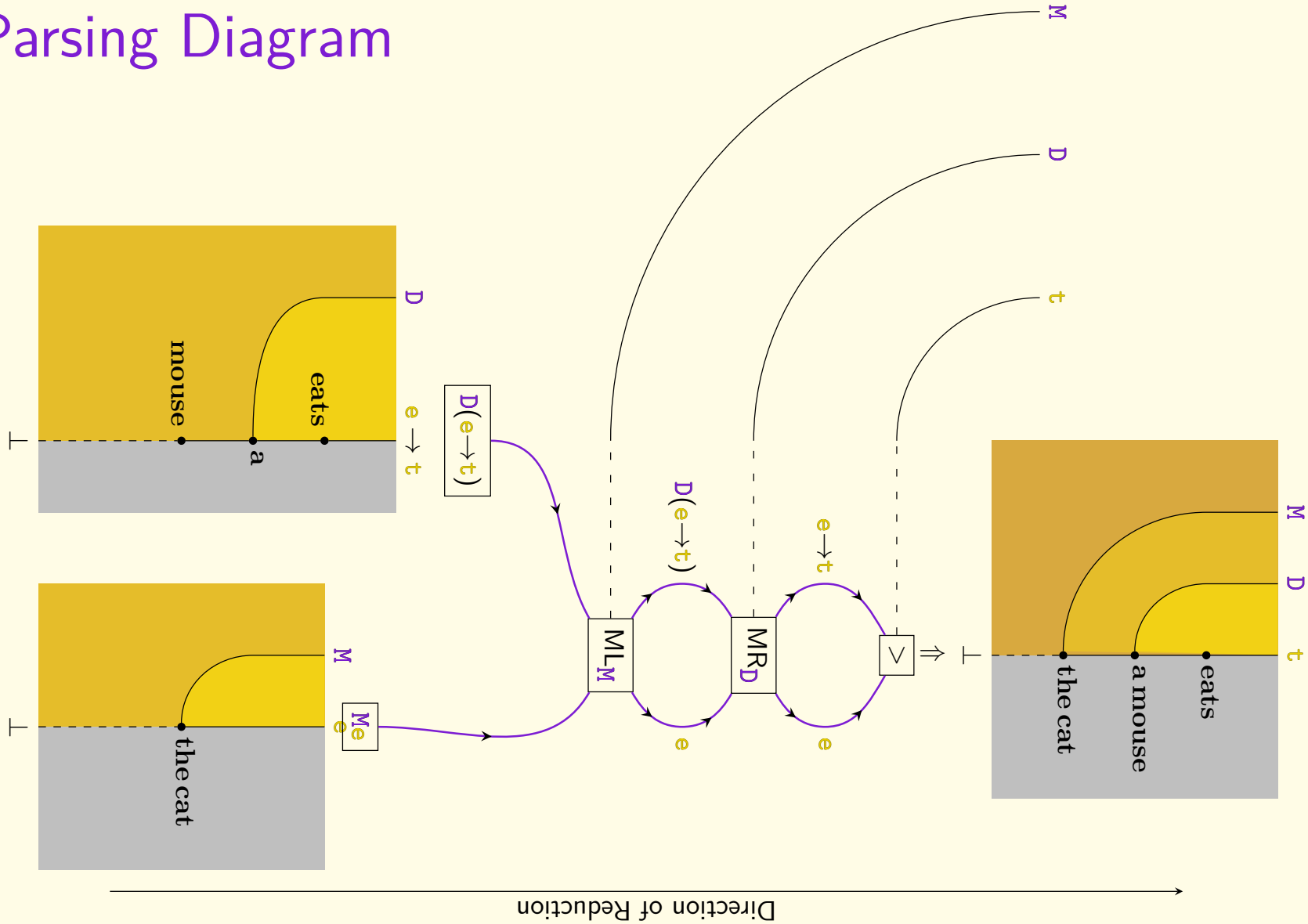
This is how we actually compute a denotation for a sentence knowing its components.

String Diagram Combination

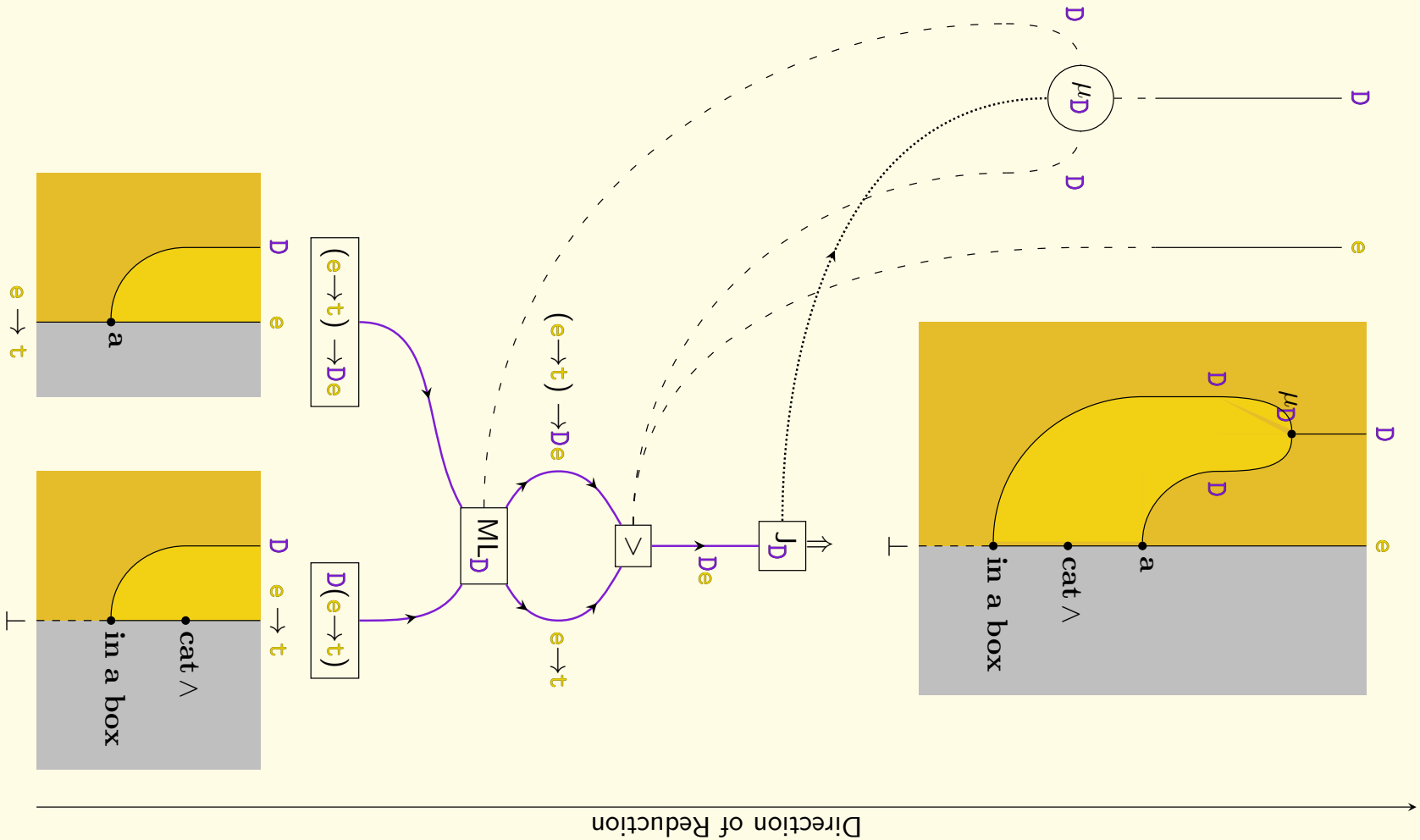
Given our grammar, we could build parsing trees, but that would blur the actual usefulness of our grammar and our string diagrammatic representation of sentences.

We thus consider diagrams whose 1-cells are objects in $\bar{\mathcal{C}}$, i.e. types and effects and whose natural transformations are the combinators of our grammar.

A Parsing Diagram



Another Parsing Diagram



Coproduct approach to Syntax

In [Mar], the theory that all syntactic operations can be expressed using the product and coproduct of a Hopf algebra is developed.

In an upcoming paper, Isabella Sentura and Matilde Marcolli prove using that approach that morphosyntactic trees and morphological trees added to syntactic trees yield the same results.
In a similar fashion, as their is a trivial way to map our string diagrams to labelled trees as proposed in [BC25], and since labelled trees can be derived from coproducts and products, we show that our way of defining parsing is actually equivalent in its integrations to the minimalistic theory.

A more direct proof is in the works.

Bibliography I

Dylan Bumford and Simon Charlow. *Effect-Driven Interpretation: Functors for Natural Language Composition*. Mar. 2025. (Visited on 24/04/2025).

Antonin Delpeuch and Jamie Vicary. *Normalization for Planar String Diagrams and a Quadratic Equivalence Algorithm*. Jan. 2022. DOI: 10.48550/arXiv.1804.07832. arXiv: 1804.07832. (Visited on 31/03/2025).

Bibliography II

James Higginbotham. “English Is Not a Context-Free Language”. In: *Linguistic Inquiry* 15.2 (1984), pp. 225–234. ISSN: 0024-3892. JSTOR: 4178381. (Visited on 06/05/2025).

André Joyal and Ross Street. “The Geometry of Tensor Calculus, I”. In: *Advances in Mathematics* 88.1 (July 1991), pp. 55–112. ISSN: 00018708. DOI: 10.1016/0001-8708(91)90003-P. (Visited on 28/03/2025).

Bibliography III

Marcolli, Matilde et Chomsky, Noam et Berwick, Robert C. *Mathematical Structure of Syntactic Merge*. (Visited on 29/05/2025).

Peter Selinger. "A Survey of Graphical Languages for Monoidal Categories". In: vol. 813. 2010, pp. 289–355. DOI: 10.1007/978-3-642-12821-9_4. arXiv: 0908.3347 [math]. (Visited on 03/04/2025).