# On a Categorical Type and Effect Inference Structure for Semantic Denotation Combinations in Natural Languages: Constructing a Purely Functional Semantic Parser of English

Matthieu Boyer

May 7, 2025

## Contents

**A Bibliography** **3**

**B Other Considered Things** **4**

B.1 Typing with a Product Category (and a bit of polymorphism) . . . . . . . . . . 4

B.2 Enhanced Grammar . . . . . . . . . . 4

**Abstract**

The main idea is that you can consider some words to act as functors in a typing category, for example determiners: they don't change the way a word acts in a sentence, but more the setting in which the word works by adding an effect to the work. The linguistics is based mostly on work by Bumford and Charlow (2025).

$$\frac{\dfrac{}{\Gamma \vdash l : \mathtt{a} \to \mathtt{b}}\,\top}{\Gamma \vdash r : \mathtt{a} \to \mathtt{b}} \tag{1}$$

## Introduction

Moggi (1989) provided a way to view monads as effects in functional programming. This allows for new modes of combination in a compositional semantic formalism, and provides a way to model words which usually raise problems with the traditional lambda-calculus representation of the words. In particular we consider words such as *the* or *a* whose application to common nouns results in types that should be used in similar situations but with largely different semantics, and model those as functions whose application yields an effect. This allows us to develop typing judgements and an extended typing system for compositional semantics of natural languages. Type-driven compositional semantics acts under the premise that given a set of words and their denotations, a set of grammar rules for composition and their associated typing judgements, we are able to form an enhanced parser for natural language which provides a mathematical representation of the meaning of sentences, as proposed by Heim and Kratzer (1998).

This is not the first time a categorical representation of a compositional semantics of natural language is proposed, Coecke et al. (2010) already suggested an approach based on monoidal categories using an outside model of meaning. However, what we propose here is a representation of the different capabilities of words as categorical constructs: we allow for a wider set of representations inside our model of meaning, trading non-determinism for additional structures. In this regard, we basically combine the grammatical type and the meaning of a word by having our denotations be associated with a type: there is no need from an additional category outside of our typing category and we limit ourselves to reducing non-determinism by limiting our possibilities for combinations with a provided CFG[1] of the language.

The focus of our system is to allow more flexibility in denotations, leading to more possibilities of combinations.

## Acknowledgements

Thanks to Antoine Groudiev for his precious insights on the direction the snakes for (**??**) and (**??**) should face.

[1] Or another model that could generate our language.

# A Bibliography


Andrej Bauer and Matija Pretnar. 2014. An Effect System for Algebraic Effects and Handlers. *Logical Methods in Computer Science*, Volume 10, Issue 4.

Dylan Bumford and Simon Charlow. 2025. Effect-driven interpretation: Functors for natural language composition.

Bob Coecke, Mehrnoosh Sadrzadeh, and Stephen Clark. 2010. Mathematical Foundations for a Compositional Distributional Model of Meaning.

Antonin Delpeuch and Jamie Vicary. 2022. Normalization for planar string diagrams and a quadratic equivalence algorithm.

Michael Goodale. 2022. Manifolds as conceptual representations in formal semantics.

Julian Grove and Jean-Philippe Bernardy. 2023. Probabilistic compositional semantics, purely.

Irene Heim and Angelika Kratzer. 1998. *Semantics in Generative Grammar*. Number 13 in Blackwell Textbooks in Linguistics. Blackwell, Malden, MA.

James Higginbotham. 1984. English Is Not a Context-Free Language. *Linguistic Inquiry*, 15(2):225–234.

André Joyal and Ross Street. 1991. The geometry of tensor calculus, I. *Advances in Mathematics*, 88(1):55–112.

Jiří Maršík and Maxime Amblard. Algebraic Effects and Handlers in Natural Language Interpretation.

Paul-André Melliès and Noam Zeilberger. 2015. Functors are Type Refinement Systems. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 3–16, Mumbai India. ACM.

E. Moggi. 1989. Computational lambda-calculus and monads. In *[1989] Proceedings. Fourth Annual Symposium on Logic in Computer Science*, pages 14–23, Pacific Grove, CA, USA. IEEE Comput. Soc. Press.

B Partee. Lecture 2. Lambda abstraction, NP semantics, and a Fragment of English. *Formal Semantics*.

Gordon D. Plotkin and Matija Pretnar. 2013. Handling Algebraic Effects. *Logical Methods in Computer Science*, Volume 9, Issue 4.

Mehrnoosh Sadrzadeh and Reinhard Muskens. 2018. Static and Dynamic Vector Semantics for Lambda Calculus Models of Natural Language.

Peter Selinger. 2010. A survey of graphical languages for monoidal categories. volume 813, pages 289–355.

Birthe van den Berg and Tom Schrijvers. 2024. A framework for higher-order effects & handlers. *Science of Computer Programming*, 234:103086.

Philip Wadler. 1989. Theorems for free! In *Proceedings of the Fourth International Conference on Functional Programming Languages and Computer Architecture*, FPCA '89, pages 347–359, New York, NY, USA. Association for Computing Machinery.

V. Wang-Maścianica. 2023. *String Diagrams for Text*. http://purl.org/dc/dcmitype/Text, University of Oxford.

Nicolas Wu, Tom Schrijvers, and Ralf Hinze. 2014. Effect handlers in scope. In *Proceedings of the 2014 ACM SIGPLAN Symposium on Haskell*, Haskell '14, pages 1–12, New York, NY, USA. Association for Computing Machinery.

# B Other Considered Things

## B.1 Typing with a Product Category (and a bit of polymorphism)

Another way to start would be to consider product categories: one for the main type system and one for the effects. Let $\mathcal{C}_0$ be a closed cartesian category representing our main type system. Here we again consider constants and full computations as functions $\perp \to \tau$ or $\tau \in \mathrm{Obj}\left(\mathcal{C}_0\right)$. Now, to type functions and functors, we need to consider a second category: We consider $\mathcal{C}_1$ the category representing the free monoid on $\mathcal{F}\left(\mathcal{L}\right)$. Monads and Applicatives will generate relations in that monoid. To ease notation we will denote *functor types* in $\mathcal{C}_1$ as lists written with head on the left.

Finally, let $\mathcal{C} = \mathcal{C}_0 \times \mathcal{C}_1$ be the product category. This will be our typing category. This means that the real type of objects will be $(\perp \to \tau, [])$, which we will still denote by $\tau$. We will denote by $F_n \cdots F_0 \tau$ the type of an object, as if it were a composition of functions[2].

In that paradigm, functors simply append to the head of the *functor type* (with the same possible restrictions as before, though I do not see what they would be needed for) while functions will take a polymorphic form: $x : L\tau_1 \mapsto \varphi x : L\tau_2$ and $\varphi$'s type can be written as $\star\tau_1 \to \star\tau_2$.

## B.2 Enhanced Grammar

[2] It is!