Effect-Driven Parsing

Formal studies on a categorical approach to semantic parsing

Matthieu Boyer

30th June 2025

École Normale Supérieure | Yale University

Plan

Plan

Salute

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 that explains how to derive the meaning of a sentence from the meaning of its words.

Lost in translation

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

A, the, your

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

A, the, your

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:

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

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

Plan

DAMN PRINTER WHAT ARE YOU DOING

```
def add(x, y):
  return x + y
```

A pure program.

```
def add(x, y):
  print("I LOVE CHOMSKY")
  return x + y
```

An impure program

DAMN PRINTER WHAT ARE YOU DOING

```python
def add(x, y):
  return x + y
```

A pure program.

```python
def add(x, y):
  print("I LOVE CHOMSKY")
  return x + y
```

An impure program

The addition of the `print` statement modifies the behaviour of the programs: we do not know what actually happens to the memory state of the computer.

This is called a side effect, or simply effect.

Category Theory 101

- A category $\mathcal{C}$ is a structure with things called objects, and ways to go between things called morphisms or arrows.

Category Theory 101

- A category $\mathcal{C}$ is a structure with things called objects, and ways to go between things called morphisms or arrows.
- Objects represent the set of objects of a certain type and arrows represent ways to go from one type to another language. The type of a function is then an object that represents the set of arrows between $A \to B$.

Category Theory 10201

- A functor from a category to another is a morphism between categories. It translates types as well as function between types.

Category Theory 10201

- A functor from a category to another is a morphism between categories. It translates types as well as function between types.

$$\begin{tikzcd} A \arrow[r, "\varphi"] \arrow[d, "F"'] & B \arrow[d, "F"] \\ FA \arrow[r, "F\varphi"'] & FB \end{tikzcd}$$

  Functors represent modifications of a type: they represent effects.

Category Theory 10202

- A natural transformation is a morphism from a functor to another.

Category Theory 10202

- A natural transformation is a morphism from a functor to another.

$$
\begin{tikzcd}
FA \arrow[r, "\theta_A"] \arrow[d, "F\varphi"'] & GA \arrow[d, "G\varphi"] \\
FB \arrow[r, "\theta_B"'] & GB
\end{tikzcd}
$$

Category of Endofunctors

- A monadic effect is a type of effect that can be created from an object, without losing information.
- When an object bears two of the same monadic effect, it can be transformed to only bear one instance of the effect.

Category of Endofunctors

- A monadic effect is a type of effect that can be created from an object, without losing information.
- When an object bears two of the same monadic effect, it can be transformed to only bear one instance of the effect.

Mathematically, we have two natural transformations $\eta : \mathrm{Id} \Rightarrow M$ and $\mu : MM \Rightarrow M$ called unit and multiplication or join.

I'm FREE! Forget it.

An adjunction between two functors $L \dashv R$ is a pair of natural transformations $\eta : \mathrm{Id} \Rightarrow L \circ R$ and $\varepsilon : R \circ L \Rightarrow \mathrm{Id}$.

It mimics the behaviour of a bijection for functor composition.

I'm FREE! Forget it.

An adjunction between two functors $L \dashv R$ is a pair of natural transformations $\eta : \mathrm{Id} \Rightarrow L \circ R$ and $\varepsilon : R \circ L \Rightarrow \mathrm{Id}$.

It mimics the behaviour of a bijection for functor composition.
A classical example is the Read - Write adjunction. It mimics the behaviour of the anaphora: once we have wrote data next to a denotation, reading said data makes us go back to the beginning, or almost.

Plan

Plan

Lexicon, but for the presentation

Let $\mathcal{L}$ be our language (more on that later). We only suppose that our words can be applied to one another in their denotation system.

Lexicon, but for the presentation

Let $\mathcal{L}$ be our language (more on that later). 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.

Lexicon, but for the presentation

Let $\mathcal{L}$ be our language (more on that later). 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 $\mathcal{C}$.
$\bar{\mathcal{C}}$ represents all possible combinations of a sequence of effects and a base type, contains functions and products.

Your honor

We then have typing judgements for basic combinations:

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

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

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

Your honor

We then have typing judgements for basic combinations:

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

Your honor

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:

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

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

Plan

What we need

To present the language, we of course need the syntax of the language, as well as an increased model of our lexicon.

What's that mean ?

Expression	Type	λ-Term
it	$\mathtt{Ge}$	$\lambda g.g_0$
$\cdot$, **a** $\cdot$	$\mathtt{e} \to (\mathtt{e} \to \mathtt{t}) \to \mathtt{We}$	$\lambda x.\lambda p.\, \langle x, px \rangle$
which	$(\mathtt{e} \to \mathtt{t}) \to \mathtt{Se}$	$\lambda p.\, \{x \mid px\}$
the	$(\mathtt{e} \to \mathtt{t}) \to \mathtt{Me}$	$\lambda p.x$ if $p^{-1}(\top) = \{x\}$ else $\#$
a	$(\mathtt{e} \to \mathtt{t}) \to \mathtt{De}$	$\lambda p.\lambda s.\, \{\langle x, x + s \rangle \mid px\}$
every	$(\mathtt{e} \to \mathtt{t}) \to \mathtt{Ce}$	$\lambda p.\lambda c.\forall x, px \Rightarrow cx$

Independence

Using the notion of functors, we can also implement higher-order semantic constructions in our lexicon, such as the future, without caring about morphological markers:

Independence

Using the notion of functors, we can also implement higher-order semantic constructions in our lexicon, such as the future, without caring about morphological markers:

$$\mathbf{future}\,(\mathbf{be}\,(\mathbf{I}, \mathbf{a\ cat})) \xrightarrow{\beta} \mathbf{be}\,(\mathbf{future}\,(\mathbf{I})\,, \mathbf{a\ cat}) \xrightarrow{\beta} \mathbf{be}\,(\mathbf{future}\,(\mathbf{I})\,, \mathbf{a\ cat})$$

Those constructs are integrated by using natural transformations explaining their propagation through other effects, as those are purely semantic predicates.

Independence

For the plural, this gives:

CN(P)	$\Gamma \vdash p : (\mathtt{e} \to \mathtt{t})$	$\Pi(p) = \lambda x.\,(px \wedge \lvert x\rvert \geq 2)$
ADJ(P)	$\Gamma \vdash p : (\mathtt{e} \to \mathtt{t})$	$\Pi(p) = \lambda x.\,(px \wedge \lvert x\rvert \geq 2)$
	$\Gamma \vdash p : (\mathtt{e} \to \mathtt{t}) \to (\mathtt{e} \to \mathtt{t})$	$\Pi(p) = \lambda \nu.\lambda x.\,(p\,(\nu)\,(x) \wedge \lvert x\rvert \geq 2)$
NP	$\Gamma \vdash p : \mathtt{e}$	$\Pi(p) = p$
	$\Gamma \vdash p : (\mathtt{e} \to \mathtt{t}) \to \mathtt{t}$	$\Pi(p) = \lambda \nu.p\,(\Pi \nu)$
IV(P)/VP	$\Gamma \vdash p : \mathtt{e} \to \mathtt{t}$	$\Pi(p) = \lambda o.\,(po \wedge \lvert x\rvert \geq 2)$
TV(P)	$\Gamma \vdash p : \mathtt{e} \to \mathtt{e} \to \mathtt{t}$	$\Pi(p) = \lambda s.\lambda o.\,(p\,(s)\,(o) \wedge \lvert s\rvert \geq 2)$

Independence

Plan

Plan

Q

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

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.

Alphabetical philosophy

There are two main types of handlers that are of interest to us:

Alphabetical philosophy

There are two main types of handlers that are of interest to us:

1. Language-Defined Handlers, which are defined with adjunctions and comonads, for example. Those arise from fundamental properties of the considered effects.

Alphabetical philosophy

There are two main types of handlers that are of interest to us:

1. Language-Defined Handlers, which are defined with adjunctions and comonads, for example. Those arise from fundamental properties of the considered effects.
2. Speaker-dependant handlers, which are considered when retrieving the denotation from a sentence from under the effects that arose in the computation of its meaning. Those need to be considered dependent on the speaker because for example of the multiple ways to solve non-determinism.

Friday

The notion of handlers allows us to enforce the notion of scope islands. To do so, it would suffice to ask that the words enclosing the island, are not defined on not certain effectful types and make handlers a part of the combination modes.

Friday

The notion of handlers allows us to enforce the notion of scope islands. To do so, it would suffice to ask that the words enclosing the island, are not defined on not certain effectful types and make handlers a part of the combination modes.

We would for example have:

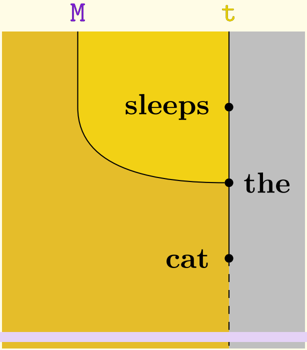
$$\textbf{if} \;:\; (\mathtt{t} \setminus \mathcal{F}\mathcal{L}^{*}\mathtt{C}\mathtt{t}) \to \mathtt{t} \to \mathtt{t}$$

Plan

Pantone I

A string diagram is a representation of the side-effects and types of a sentence across its computation.

The lines are functors (effects or base types), the nodes are natural transformations.



This diagram for example represents the sentence *The cat sleeps*. The order of the words and position of the strings will be explained in detail in the

Plan

Tying the knots

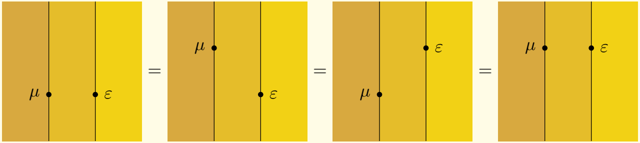
String diagrams will be the formalism we use to compute equality between denotations, and especially handling the denotations.

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.

Version 7.0. I

Every property of the functors, monads, natural transformations, adjunctions and more can be explained in terms of commutative diagrams, but also as string diagrams.

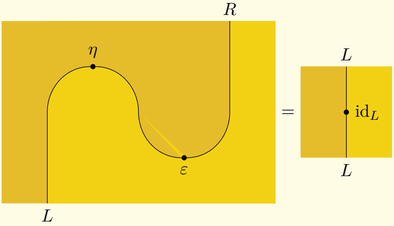
First, the elevator equations are a consequence of 3.1:



(🛗)

Version 7.0. II

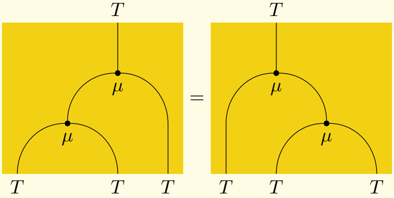
The Snake equations are a rewriting of the properties of an adjunction:



(🐍)

Version 7.0. III

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



(μ)

Bubba Gump Shrimps I

Theorem 3.2 — 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.

Bubba Gump Shrimps II

Theorem 3.3 — 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

Plan

Chomsky did nothing wrong. I

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

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

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

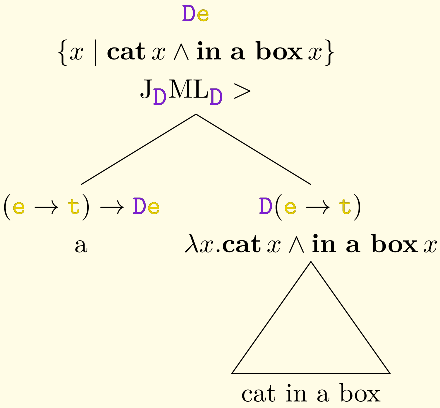
$$
\begin{aligned}
\mathsf{ML}_{\mathtt{F}}(\alpha, \beta) &::= \mathtt{F}\alpha, \beta \\
\mathsf{MR}_{\mathtt{F}}(\alpha, \beta) &::= \alpha, \mathtt{F}\beta \\
\mathsf{A}_{\mathtt{F}}(\alpha, \beta) &::= \mathtt{F}\alpha, \mathtt{F}\beta \\
\mathsf{UR}_{\mathtt{F}}(\alpha \to \alpha', \beta) &::=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

Chomsky did nothing wrong. II

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.

Where cats are stuck. I



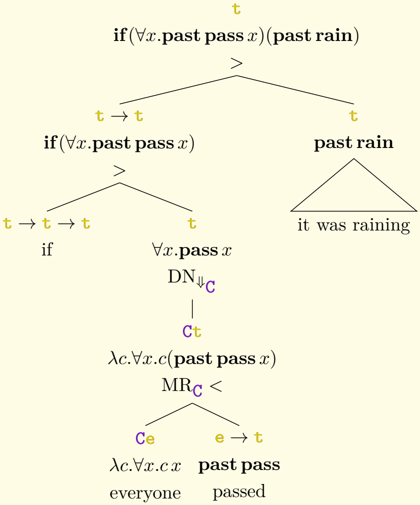
Where cats are stuck. II

$$\texttt{MDt}$$
$$\{\mathbf{eats}(\texttt{obj=}m, \texttt{subj=}c) | \mathbf{mouse}\,(m)\} \text{ if } \mathbf{cat}^{-1}(\top) = \{c\}$$
$$\mathrm{MR}_{\texttt{M}}\mathrm{ML}_{\texttt{D}} >$$

$$\texttt{M}(\texttt{e})$$
$$c \text{ if } \mathbf{cat}^{-1}(\top) = \{c\}$$
the cat

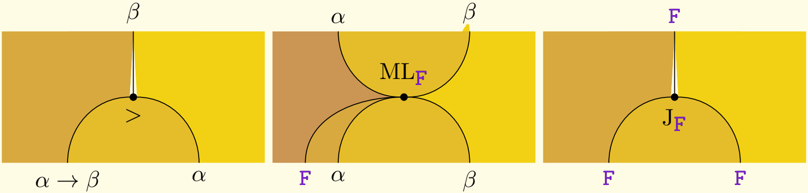
$$\texttt{D}(\texttt{e} \to \texttt{t})$$
$$\{\lambda s.\mathbf{eats}(\texttt{obj=}m, \texttt{subj=}s) | \mathbf{mouse}\,(m)\}$$
eats a mouse

Where cats are stuck. III



Plan

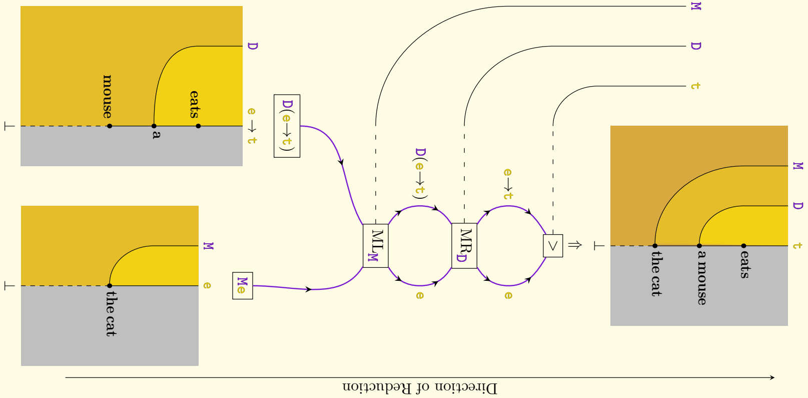
Legos, with strings



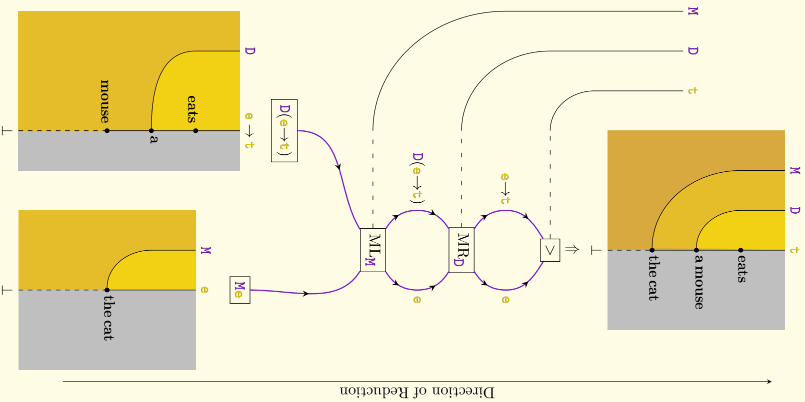
Crayons

Based on the mathematical definitions of the combinators, we can also define reduction rules as a part of the grammar (and later, on our string diagrams).
This allows us to reduce the grammar while still using the properties of string diagrams to our advantage in the proofs.

Under the old willow



Under the older willow



Knitting with words

Willows, but better!