

# Effect-Driven Parsing

Formal studies on a categorical approach to semantic parsing

Matthieu Boyer

Yale



PSL



September 9 2025

École Normale Supérieure | Yale University

# Context

---

- Categorical formalization of a type and effects system for natural-language semantics (following [BC25]).
- Develop a graphical, type-driven parsing formalism that derives sentence meaning compositionally from word meanings.

# Typed Semantics for Natural Languages

Expression	Type	$\lambda$ -Term
planet	$e \rightarrow t$	$\lambda x.\text{planet } x$
Generalizes to <b>common nouns</b>		
Jupiter	$e$	$j \in \text{Var}$
Generalizes to <b>proper nouns</b>		
chase	$e \rightarrow e \rightarrow t$	$\lambda o.\lambda s.\text{chase } (o) (s)$
Generalizes to <b>transitive verbs</b>		

# Syntactic Types and Semantic Types

---

- Syntax **a cat** and **the cat** should have the same type: **e**.

# Syntactic Types and Semantic Types

---

- Syntax **a cat** and **the cat** should have the same type: **e**.
- No single canonical **cat** exists: that type cannot be **e**.

# Syntactic Types and Semantic Types

- Syntax **a cat** and **the cat** should have the same type: **e**.
- No single canonical **cat** exists: that type cannot be **e**.

We will use **(side-)effects** to do the difference between them:

$$\mathbf{a\ cat} = \{c \mid \mathbf{cat\ } c\} \quad (\text{Set})$$

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

# Effects as Functors

---

Traditionally ([Mog89]), monads model side-effects.

# Effects as Functors

---

Traditionally ([Mog89]), monads model side-effects.

Here, functors suffice and lighter structures are useful.



# String Diagrams

---

String Diagrams are a formalism ([HM23]) that allows to visually represent the different threads of a computation and the possible side-effects that appear.

# String Diagrams

---

String Diagrams are a formalism ([HM23]) that allows to visually represent the different threads of a computation and the possible side-effects that appear.

Used as a tool for parsing in [CSC10].

# Other Categorical Theories

---

[Mar] and [SM25] use Hopf algebras to give a model for parsing.

[MZ25] use operads to prove results on CFGs.

# Notations

---

- Language  $\mathcal{L}$  of denotationally composed words.
- Base typing CCC  $\mathcal{C}$ ; Effects: functors  $\mathcal{F}(\mathcal{L})$ .

# Notations

---

- Language  $\mathcal{L}$  of denotationally composed words.
- Base typing CCC  $\mathcal{C}$ ; Effects: functors  $\mathcal{F}(\mathcal{L})$ .
- Typing CCC:  $\bar{\mathcal{C}}$  = closure of  $\mathcal{C}$  under  $\mathcal{F}(\mathcal{L})^*$ , products and exponentials.

Intuition: all (effect-sequence, base type) combos, with functions/products.

# Intuitionistic-style Typing Judgements

We then have typing judgements for basic combinations:

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

$$\frac{\Gamma \vdash x : F\tau_1 \quad \Gamma \vdash \varphi : \tau_1 \rightarrow \tau_2}{\Gamma \vdash \varphi x : F\tau_2} \text{fmap}$$

# Intuitionistic-style Typing Judgements

We then have typing judgements for basic combinations:

$$\frac{\Gamma \vdash x : \tau_1 \quad \Gamma \vdash \varphi : \tau_1 \rightarrow \tau_2}{\Gamma \vdash \varphi x : \tau_2} \text{App}$$

$$\frac{\Gamma \vdash x : A\tau_1 \quad \Gamma \vdash \varphi : A(\tau_1 \rightarrow \tau_2)}{\Gamma \vdash \varphi x : A\tau_2} \langle * \rangle$$

# Intuitionistic-style Typing Judgements

Typing judgements for natural transformations:

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

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

$$\forall F \xRightarrow{\theta} G, \quad \frac{\Gamma \vdash x : F\tau \quad \Gamma \vdash G : S' \subseteq \star \quad \tau \in S'}{\Gamma \vdash x : G\tau} \text{nat}$$



# Presentation

---

To present a language in our formalism, we need:

- A syntax;
- A typed dictionary using effects;
- A typed lexicon of non-verbal constructs.

# Introducing Higher-Order Constructs

---

We implement higher-order semantics (e.g. the future and plural) via functors.

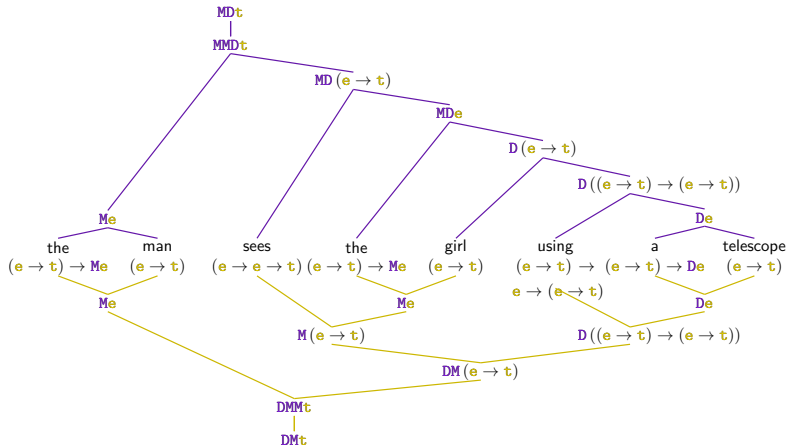
# Introducing Higher-Order Constructs

We implement higher-order semantics (e.g. the future and plural) via functors.

We also enforce the notion of scope islands as in [BC25]:

$$\text{if} : (\mathbf{t} \setminus \mathcal{F}(\mathcal{L})^* \mathbf{Ct}) \rightarrow \mathbf{t} \rightarrow \mathbf{t}$$

# Ambiguity



# Handlers

---

Handlers for an effect  $F$  are natural transformations  $F \Rightarrow \text{Id}$  ([WSH14]) which invert units.

# Handlers

---

Handlers for an effect  $F$  are natural transformations  $F \Rightarrow \text{Id}$  ([WSH14]) which invert units.

- 1 Language-Defined Handlers arise from fundamental properties of the considered effects.

# Handlers

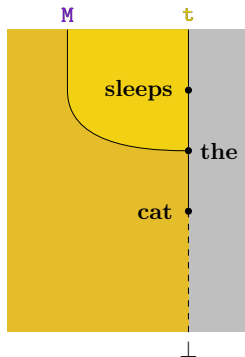
---

Handlers for an effect  $F$  are natural transformations  $F \Rightarrow \text{Id}$  ([WSH14]) which invert units.

- 1 Language-Defined Handlers arise from fundamental properties of the considered effects.
- 2 Speaker-dependant handlers which are dependent on the speaker.

# String Diagrams Representation of Sentences

String diagrams are a representation of the side-effects and types of a sentence across its computation.





# Deformation of String Diagrams

**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.

# Equations on String Diagrams

---

Properties of monads, natural transformations, adjunctions and more can be explained in terms of commutative diagrams, but also as string diagram equations.

Moreover, Theorem 3.1 can be implemented as string diagram equations.

# Confluence of Reductions

**Theorem 3.2 — Confluence** The reduction system defined by the specified equations is confluent and therefore defines normal forms.

**Theorem 3.3 — Normalization Complexity** Normalization is quadratic in the number of natural transformations.

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

# Parsing Algorithm

---

Typing syntax trees is exponentially too long.

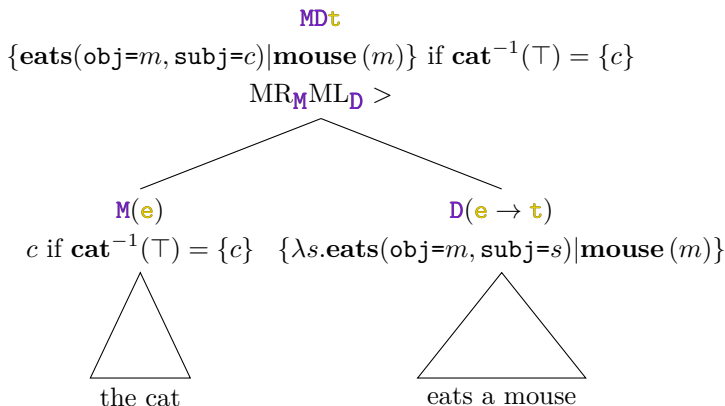
# Parsing Algorithm

---

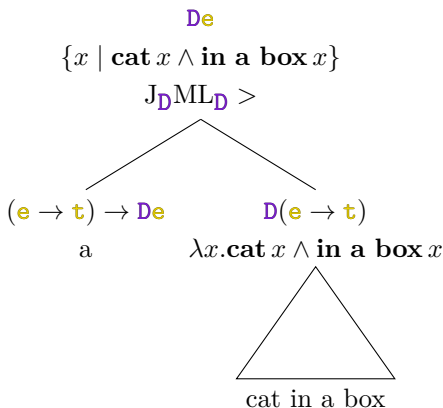
Typing syntax trees is exponentially too long.

We use a Context-Free Grammar in five parts to model our typing system. This gives a complexity in  $\mathcal{O}(|\mathcal{F}(\mathcal{L})| |\mathcal{S}| n^3)$ .

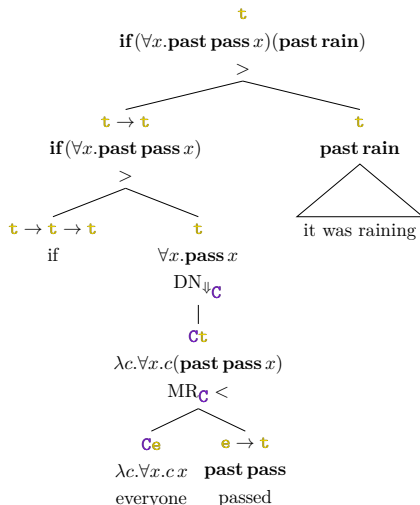
# Semantic Parse Trees I



# Semantic Parse Trees II

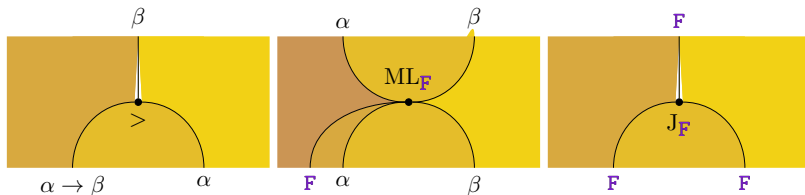


# Semantic Parse Trees III

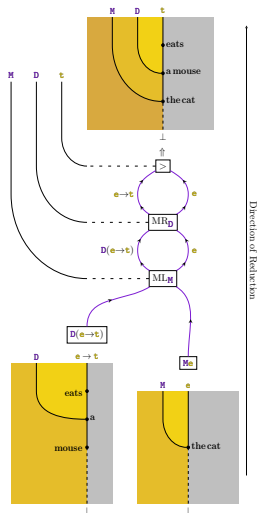




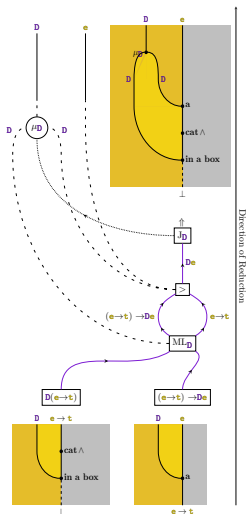
# Combinators as String Diagrams



# A Parsing Diagram Step



# Another Parsing Diagram Step

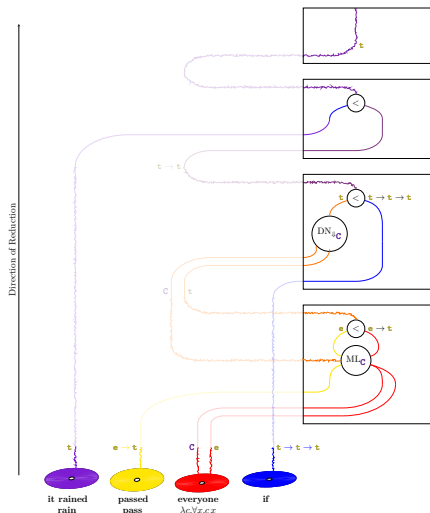


# Building an Intuition

---



# A Full Parsing Diagram



# Reducing the grammar

---

We translate equalities on actual denotations (from combinators or from the denotational system) into the reduction system on string diagrams.

Commutation of effects, Theorem 3.1 and more, allow a reduction of the constant in the algorithmic complexity.

# Conclusion

---

- Theoretical enhancement of a type system for natural language semantics;

# Conclusion

---

- Theoretical enhancement of a type system for natural language semantics;
- No load added neither on the user (comprehension) nor the parser (efficiency).



# Conclusion

---

- Theoretical enhancement of a type system for natural language semantics;
- No load added neither on the user (comprehension) nor the parser (efficiency).

I would like to thank Simon Charlow for his advice and guidance, my mother for the knitting, Antoine Groudiev for the rotation of the snakes in equation labels, Bella Senturia, Bob Frank and Paul-André Melliès for their suggestions of papers to read about categories and linguistics.

Thank you for your attention.

Do you have any questions?

# Lexicon

Expression	Type	$\lambda$ -Term
planet	$\mathbf{e} \rightarrow \mathbf{t}$ Generalizes to <b>common nouns</b>	$\lambda x.\text{planet } x$
carnivorous	$(\mathbf{e} \rightarrow \mathbf{t})$ Generalizes to <b>predicative adjectives</b>	$\lambda x.\text{carnivorous } x$
skillful	$(\mathbf{e} \rightarrow \mathbf{t}) \rightarrow (\mathbf{e} \rightarrow \mathbf{t})$ Generalizes to <b>predicate modifier adjectives</b>	$\lambda p.\lambda x.px \wedge \text{skillful } x$
Jupiter	$\mathbf{e}$ Generalizes to <b>proper nouns</b>	$\mathbf{j} \in \text{Var}$
sleep	$\mathbf{e} \rightarrow \mathbf{t}$ Generalizes to <b>intransitive verbs</b>	$\lambda x.\text{sleep } x$
chase	$\mathbf{e} \rightarrow \mathbf{e} \rightarrow \mathbf{t}$ Generalizes to <b>transitive verbs</b>	$\lambda o.\lambda s.\text{chase } (o) (s)$
be	$(\mathbf{e} \rightarrow \mathbf{t}) \rightarrow \mathbf{e} \rightarrow \mathbf{t}$	$\lambda p.\lambda x.px$
she	$\mathbf{x} \rightarrow \mathbf{e}$	$\lambda g.g_0$
it	$\mathbf{G}\mathbf{e}$	$\lambda g.g_0$
which	$(\mathbf{e} \rightarrow \mathbf{t}) \rightarrow \mathbf{S}\mathbf{e}$	$\lambda p.\{x \mid px\}$
the	$(\mathbf{e} \rightarrow \mathbf{t}) \rightarrow \mathbf{M}\mathbf{e}$	$\lambda p.x \text{ if } p^{-1}(\top) = \{x\} \text{ else } \#$
a	$(\mathbf{e} \rightarrow \mathbf{t}) \rightarrow \mathbf{D}\mathbf{e}$	$\lambda p.\lambda s.\{ \langle x, x \# s \rangle \mid px \}$
no	$(\mathbf{e} \rightarrow \mathbf{t}) \rightarrow \mathbf{C}\mathbf{e}$	$\lambda p.\lambda c.\neg \exists x.px \wedge c x$
every	$(\mathbf{e} \rightarrow \mathbf{t}) \rightarrow \mathbf{C}\mathbf{e}$	$\lambda p.\lambda c.\forall x.px \Rightarrow c x$
$\cdot, a \cdot$	$\mathbf{e} \rightarrow (\mathbf{e} \rightarrow \mathbf{t}) \rightarrow \mathbf{W}\mathbf{e}$	$\lambda x.\lambda p.\langle x, px \rangle$
as for	$\mathbf{e} \rightarrow \mathbf{T}\mathbf{e}$	$\lambda x.\lambda s.\langle x, x \# s \rangle$
$\cdot \mathbf{F}$	$\mathbf{e} \rightarrow \mathbf{F}\mathbf{e}$	$\lambda v.\langle v, \{x \mid x \in D_e\} \rangle$

# Functor Denotations

Constructor	fmap	Typeclass
$\mathbf{G}(\tau) = \mathbf{r} \rightarrow \tau$	$\mathbf{G}\varphi(x) = \lambda r. \varphi(xr)$	Monad
$\mathbf{W}(\tau) = \tau \times \mathbf{t}$	$\mathbf{W}\varphi(\langle a, p \rangle) = \langle \varphi a, p \rangle$	Monad
$\mathbf{S}(\tau) = \{\tau\}$	$\mathbf{S}\varphi(\{x\}) = \{\varphi(x)\}$	Monad
$\mathbf{C}(\tau) = (\tau \rightarrow \mathbf{t}) \rightarrow \mathbf{t}$	$\mathbf{C}\varphi(x) = \lambda c. x(\lambda a. c(\varphi a))$	Monad
$\mathbf{T}(\tau) = \mathbf{s} \rightarrow (\tau \times \mathbf{s})$	$\mathbf{D}\varphi(\lambda s. \{\langle x, x \# s \rangle \mid px\}) = \lambda s. \langle \varphi x, \varphi x \# s \rangle$	Monad
$\mathbf{F}(\tau) = \tau \times \mathbf{S}\tau$	$\mathbf{F}\varphi(\langle v, \{x \mid x \in D_e\} \rangle) = \langle \varphi(v), \{x \mid x \in D_e\} \rangle$	Monad
$\mathbf{D}(\tau) = \mathbf{s} \rightarrow \mathbf{S}(\tau \times \mathbf{s})$	$\mathbf{D}\varphi(\lambda s. \{\langle x, x \# s \rangle \mid px\}) = \lambda s. \{\langle \varphi x, \varphi x \# s \rangle \mid px\}$	Monad
$\mathbf{M}(\tau) = \tau + \perp$	$\mathbf{M}\varphi(x) = \begin{cases} \varphi(x) & \text{if } \Gamma \vdash x : \tau \\ \# & \text{if } \Gamma \vdash x : \# \end{cases}$	Monad

# CFG of English

---

CP ::= DP, VP  
| Cmp, CP  
| CP, CBar

CBar ::= Cor, CP

Dbar ::= Cor, DP

DP ::= DP, Dbar  
| Dmp, DP  
| Det, NP  
| Gen, TN

Gen ::= DP, GenD

NP ::= AdjP, NP  
| NP, AdjP

AdjP ::= TAdj, DP  
| Deg, AdjP

VP ::= TV, DP  
| AV, CP  
| VP, AdvP

TV ::= DV, DP

AdvP ::= TAdv, DP

# CFG for Parsing

$$>, \beta \quad ::= \quad (\alpha \rightarrow \beta), \alpha$$

$$<, \beta \quad ::= \quad \alpha, (\alpha \rightarrow \beta) \quad A_F(\alpha, \beta) \quad ::= \quad F\alpha, F\beta$$

$$UR_F(\alpha \rightarrow \alpha', \beta) \quad ::= \quad F\alpha \rightarrow \alpha', \beta$$

$$J_F F\tau \quad ::= \quad FF\tau$$

$$UL_F(\alpha, \beta \rightarrow \beta') \quad ::= \quad \alpha, F\beta \rightarrow \beta'$$

$$DN_C \tau \quad ::= \quad C_\tau \tau$$

$$C_{LR}(L\alpha, R\beta) \quad ::= \quad (\alpha, \beta)$$

$$ML_F(\alpha, \beta) \quad ::= \quad F\alpha, \beta$$

$$ER_R(R(\alpha \rightarrow \alpha'), \beta) \quad ::= \quad \alpha \rightarrow R\alpha', \beta$$

$$MR_F(\alpha, \beta) \quad ::= \quad \alpha, F\beta$$

$$EL_R(\alpha, R(\beta \rightarrow \beta')) \quad ::= \quad \alpha, \beta \rightarrow R\beta'$$

# Combinator Denotations

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

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

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

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

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

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

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

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

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

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

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

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