Effect-Driven Parsing

Formal studies on a categorical approach to semantic parsing

Matthieu Boyer

7th September 2025

École Normale Supérieure | Yale University

Context

- Categorical formalization of a type–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	λ-Term
planet	$e \to t$	$\lambda x.\mathbf{planet}\, x$
	Generalizes to **common nouns**	
Jupiter	e	$\mathbf{j} \in \mathrm{Var}$
	Generalizes to **proper nouns**	
sleep	$e \to t$	$\lambda x.\mathbf{sleep}\, x$
	Generalizes to **intransitive verbs**	

Syntactic Types and Semantic Types

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

Syntactic Types and Semantic Types

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

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

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

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

Effects as Functors

Monads model side-effects [Mog89].

Effects as Functors

Monads model side-effects [Mog89].

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:

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

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

Intuitionistic-style Typing Judgements

We then have typing judgements for basic combinations:

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

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

Intuitionistic-style Typing Judgements

Typing judgements for natural transformations:

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

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

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

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.

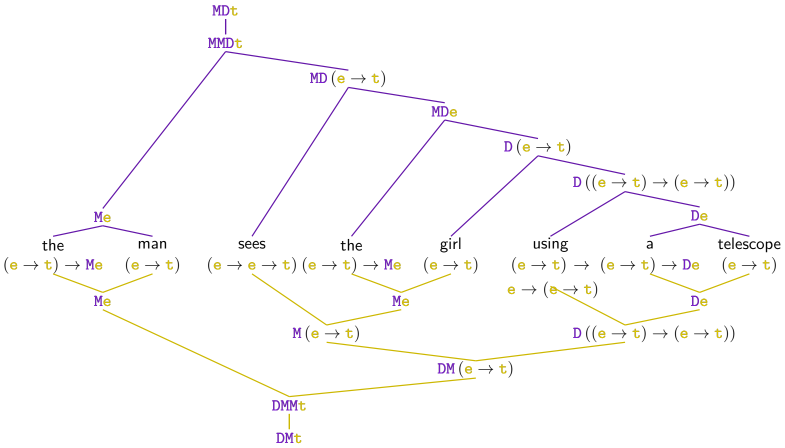
Introducing Higher-Order Constructs

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

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

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

Ambiguity


MDt
MMDt
MD (e → t)
MDe
D (e → t)
D ((e → t) → (e → t))
Me
De
the
man
sees
the
girl
using
a
telescope
(e → t) → Me
(e → t)
(e → e → t)
(e → t) → Me
(e → t)
(e → t) →
(e → t) → De
(e → t)
e → (e → t)
Me
Me
De
M (e → t)
D ((e → t) → (e → t))
DM (e → t)
DMMt
DMt

Handlers

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

Handlers

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

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

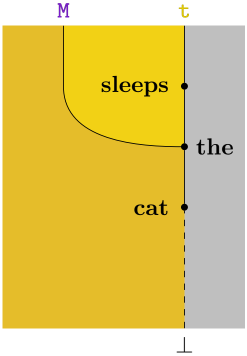
Handlers

Handlers for an effect F are natural transformations $F \Rightarrow \mathrm{Id}$ ([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 diagram 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

$$\texttt{MDt}$$

$$\{\textbf{eats}(\texttt{obj=}m, \texttt{subj=}c) | \textbf{mouse}\,(m)\} \text{ if } \textbf{cat}^{-1}(\top) = \{c\}$$

$$\mathrm{MR}_{\texttt{M}}\mathrm{ML}_{\texttt{D}} >$$

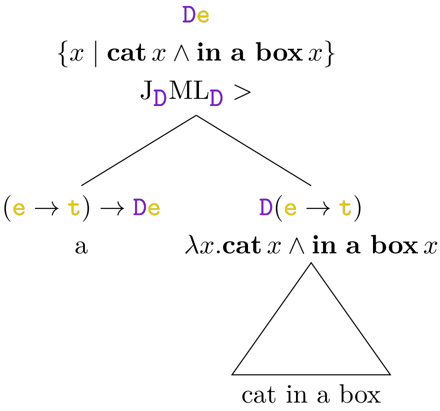
$$\texttt{M}(\texttt{e}) \qquad\qquad \texttt{D}(\texttt{e} \to \texttt{t})$$

$$c \text{ if } \textbf{cat}^{-1}(\top) = \{c\} \quad \{\lambda s.\textbf{eats}(\texttt{obj=}m, \texttt{subj=}s) | \textbf{mouse}\,(m)\}$$

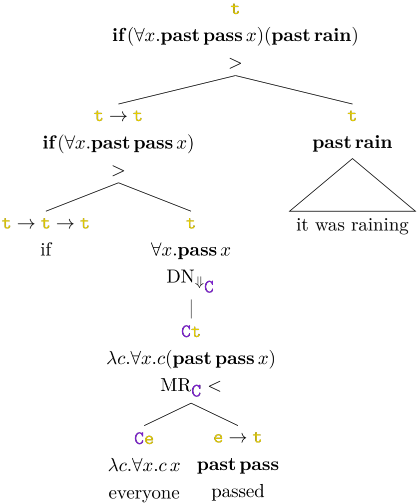
the cat

eats a mouse

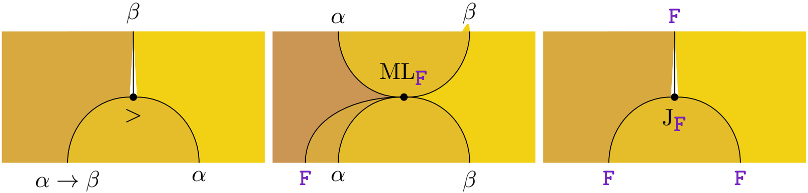
Semantic Parse Trees II



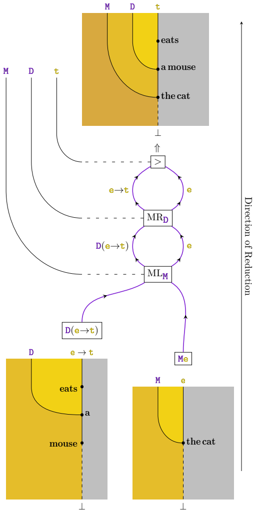
Semantic Parse Trees III



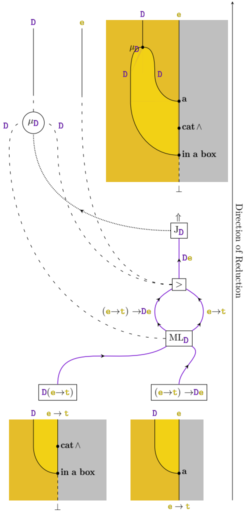
Combinators as String Diagrams



A Parsing Diagram Step

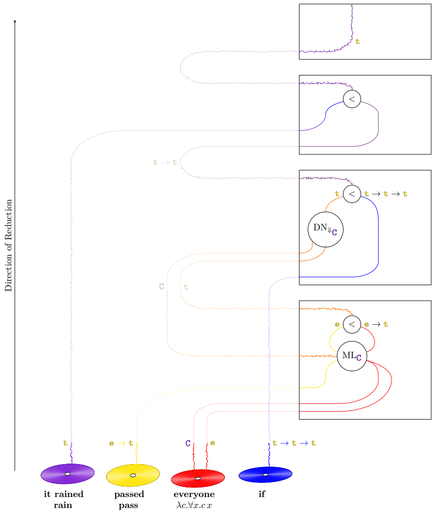


Another Parsing Diagram Step



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	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 **intranitive verbs**	
chase	$e \to e \to t$	$\lambda o.\lambda s.\mathbf{chase}\,(o)\,(s)$
	Generalizes to **transitive verbs**	
be	$(e \to t) \to e \to t$	$\lambda p.\lambda x.px$
she	$r \to e$	$\lambda g.g_0$
it	Ge	$\lambda g.g_0$
which	$(e \to t) \to Se$	$\lambda p.\,\{x \mid px\}$
the	$(e \to t) \to Me$	$\lambda p.x$ if $p^{-1}(\top) = \{x\}$ else $\#$
a	$(e \to t) \to De$	$\lambda p.\lambda s.\,\{\langle x, x + s\rangle \mid px\}$
no	$(e \to t) \to Ce$	$\lambda p.\lambda c.\neg\exists x.px \wedge c\, x$
every	$(e \to t) \to Ce$	$\lambda p.\lambda c.\forall x, px \Rightarrow cx$
·, a ·	$e \to (e \to t) \to We$	$\lambda x.\lambda p.\,\langle x, px\rangle$
as for	$e \to Te$	$\lambda x.\lambda s.\,\langle x, x + s\rangle$
·F	$e \to Fe$	$\lambda v.\,\langle v, \{x \mid x \in D_e\}\rangle$

Functor Denotations

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

$$
\begin{aligned}
>, \beta &::= (\alpha \to \beta), \alpha \\
<, \beta &::= \alpha, (\alpha \to \beta) \\
\mathsf{J}_{\mathtt{F}}\ \mathtt{F}\tau &::= \mathtt{FF}\tau \\
\mathsf{DN}_{\mathtt{C}}\ \tau &::= \mathtt{C}_\tau \tau \\
\mathsf{ML}_{\mathtt{F}}(\alpha, \beta) &::= \mathtt{F}\alpha, \beta \\
\mathsf{MR}_{\mathtt{F}}(\alpha, \beta) &::= \alpha, \mathtt{F}\beta
\end{aligned}
$$

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

Combinator Denotations

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

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

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

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

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

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

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

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

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

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

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

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