Effect-Driven Parsing

Formal studies on a categorical approach to semantic parsing

Matthieu Boyer

7th September 2025

École Normale Supérieure | Yale University

Plan

Plan

Context

Goal. Categorical formalization of a type–effects system for natural-language semantics (following [BC25]).

Method. 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	$\mathtt{e} \to \mathtt{t}$	$\lambda x.\mathbf{planet}\, x$
	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**	

Syntactic Types and Semantic Types

- What is the type of **a cat** or **Jupiter, a planet**?

Syntactic Types and Semantic Types

- What is the type of **a cat** or **Jupiter, a planet**?
- 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

- What is the type of **a cat** or **Jupiter, a planet**?
- 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:

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

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

Plan

Effects as Functors

Monads model side-effects [Mog89].

Effects as Functors

Monads model side-effects [Mog89]. Here, **functors suffice**: we may forbid merging/creating effects; lighter structures are useful.

String Diagrams

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

Categorical foundations: [JS91]; Used as a tool for parsing: [CSC10].

Other Categorical Theories

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

They integrate morphological features inside the syntactic structures without breaking the model chosen for syntax.

Plan

Plan

Notations

- Language: $\mathcal{L}$; words compose denotationally.

Notations

- Language: $\mathcal{L}$; words compose denotationally.
- Base CCC: $\mathcal{C}$; effects: functors $\mathcal{F}(\mathcal{L})$.
- Closure: $\bar{\mathcal{C}} =$ 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:

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

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

Intuitionistic-style Typing Judgements

Typing judgements for natural transformations:

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

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

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

Plan

Information

To present the language we need:

- The syntax of the language.
- An typed lexicon (possibly with effects).

Updated Lexicon

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	$(\mathtt{e} \to \mathtt{t}) \to \mathtt{De}$	$\lambda p.\lambda s.\,\{\langle x, x \mathbin{+\!\!+} s\rangle \mid px\}$
every	$(\mathtt{e} \to \mathtt{t}) \to \mathtt{Ce}$	$\lambda p.\lambda c.\forall x, px \Rightarrow cx$

Introducing Higher-Order Constructs

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

$$\mathbf{future}\left(\mathbf{be}\left(\mathbf{I}, \mathbf{a\ cat}\right)\right) \xrightarrow{\texttt{nat}} \mathbf{be}\left(\mathbf{future}\left(\mathbf{I}\right), \mathbf{a\ cat}\right) \xrightarrow{\texttt{nat}} \mathbf{be}\left(\mathbf{future}\left(\mathbf{I}\right), \mathbf{a\ cat}\right)$$

Introducing Higher-Order Constructs

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

$$\mathbf{future}\left(\mathbf{be}\left(\mathbf{I}, \mathbf{a\ cat}\right)\right) \xrightarrow{\texttt{nat}} \mathbf{be}\left(\mathbf{future}\left(\mathbf{I}\right), \mathbf{a\ cat}\right) \xrightarrow{\texttt{nat}} \mathbf{be}\left(\mathbf{future}\left(\mathbf{I}\right), \mathbf{a\ cat}\right)$$

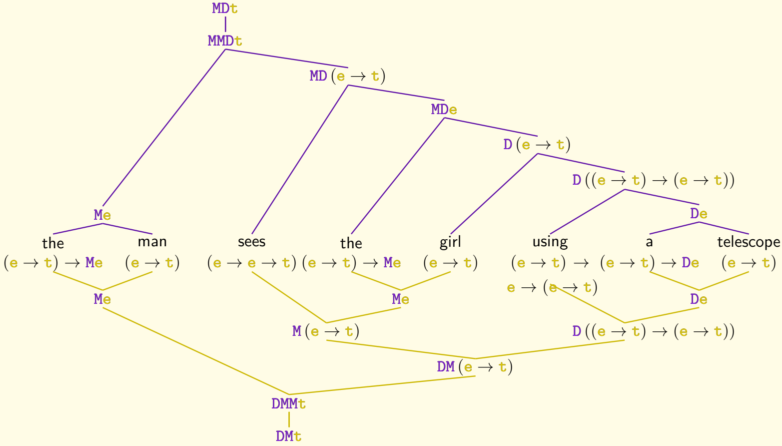
Those constructs are integrated by using natural transformations explaining their propagation through other effects, as those are purely semantic predicates.

Introducing Higher-Order Constructs

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)	$\Gamma \vdash p : \mathtt{e} \to \mathtt{e} \to \mathtt{t}$	$\Pi(p) = \lambda s. \lambda o. (p\,(s)\,(o) \wedge \lvert s \rvert \geq 2)$

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 → (e → t)
(e → t) → De
(e → t)
Me
Me
De
M (e → t)
D ((e → t) → (e → t))
DM (e → t)
DMMt
DMt

Plan

Plan

Handlers

A handler for an effect F is a natural transformation $F \Rightarrow \mathrm{Id}$ ([WSH14]).

Handlers should invert monadic and applicative units: motivating the omission of the unit rule out of certain situations.

Defining Handlers

Two types of handlers:

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

Defining Handlers

Two types of handlers:

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

Scope Islands

The notion of handlers allows us to enforce the notion of scope islands as introduced in [BC25].

Scope Islands

The notion of handlers allows us to enforce the notion of scope islands as introduced in [BC25].

We would for example have:

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

to enforce in the type system the resolution of continuations.

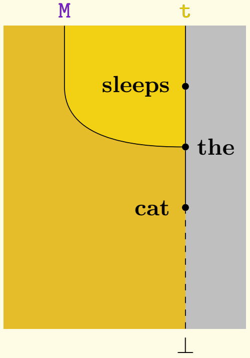
Plan

String Diagrams Representation of Sentences

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

String Diagrams Representation of Sentences

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



Plan

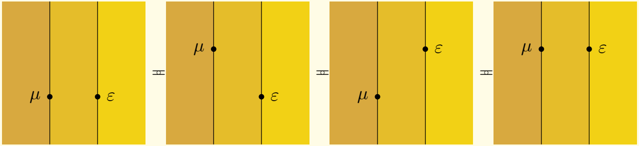
Deformation of String Diagrams

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 I

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

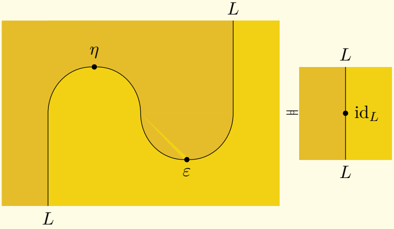
The Elevator Equations represent Theorem 3.1:



(🛗)

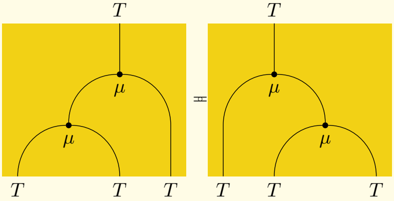
Equations on String Diagrams II

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



Equations on String Diagrams III

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



(μ)

Confluence of Reductions 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.

Confluence of Reductions 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

CFG Model of Parsing

We use a Context-Free Grammar in five parts to model our typing system:

1. We reintroduce the grammar defining the type and effect system.

CFG Model of Parsing

We use a Context-Free Grammar in five parts to model our typing system:

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].

CFG Model of Parsing

We use a Context-Free Grammar in five parts to model our typing system:

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.

CFG Model of Parsing

We use a Context-Free Grammar in five parts to model our typing system:

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.

CFG Model of Parsing

We use a Context-Free Grammar in five parts to model our typing system:

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.

CFG Model of Parsing

We use a Context-Free Grammar in five parts to model our typing system:

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.

CFG Model of Parsing

We use a Context-Free Grammar in five parts to model our typing system:

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.

Complexity in $\mathcal{O}(|\mathcal{F}(\mathcal{L})|\,|\mathcal{S}|\,n^3)$.

Semantic Parse Trees I

$$\mathtt{De}$$

$$\{x \mid \mathbf{cat}\, x \wedge \mathbf{in\ a\ box}\, x\}$$

$$\mathrm{J}_{\mathtt{D}}\mathrm{ML}_{\mathtt{D}} >$$

$$(\mathtt{e} \to \mathtt{t}) \to \mathtt{De}$$

a

$$\mathtt{D}(\mathtt{e} \to \mathtt{t})$$

$$\lambda x.\mathbf{cat}\, x \wedge \mathbf{in\ a\ box}\, x$$

cat in a box

Semantic Parse Trees II

$\mathtt{MDt}$

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

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

$\mathtt{M}(\mathtt{e})$

c if $\mathbf{cat}^{-1}(\top) = \{c\}$

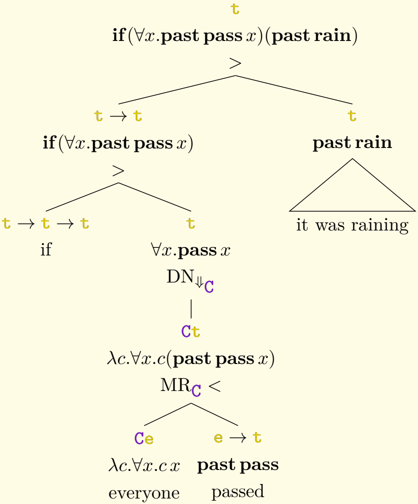
the cat

$\mathtt{D}(\mathtt{e} \to \mathtt{t})$

$\{\lambda s.\mathbf{eats}(\mathtt{obj}{=}m, \mathtt{subj}{=}s) | \mathbf{mouse}\,(m)\}$

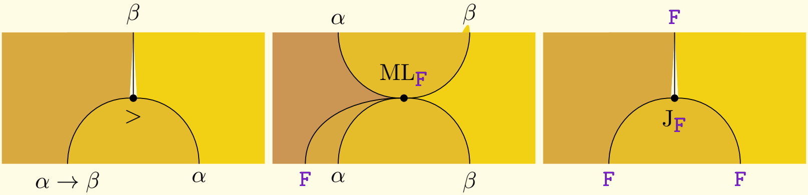
eats a mouse

Semantic Parse Trees III

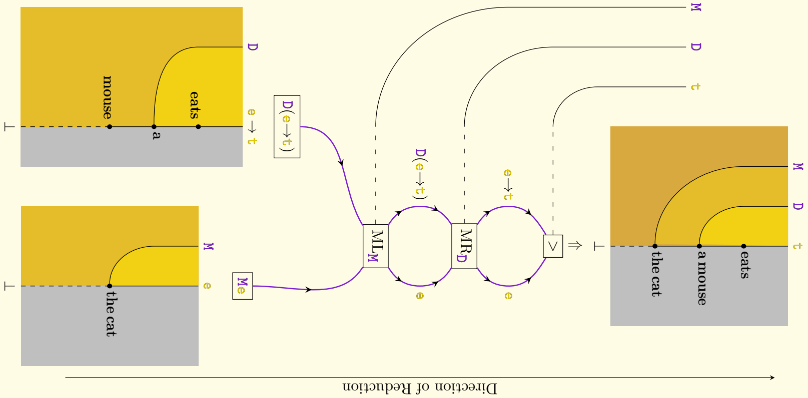


Plan

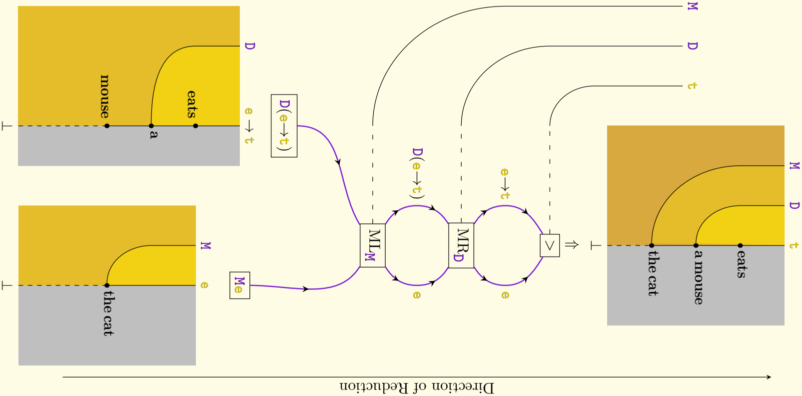
Combinators as String Diagrams



A Parsing Diagram Step

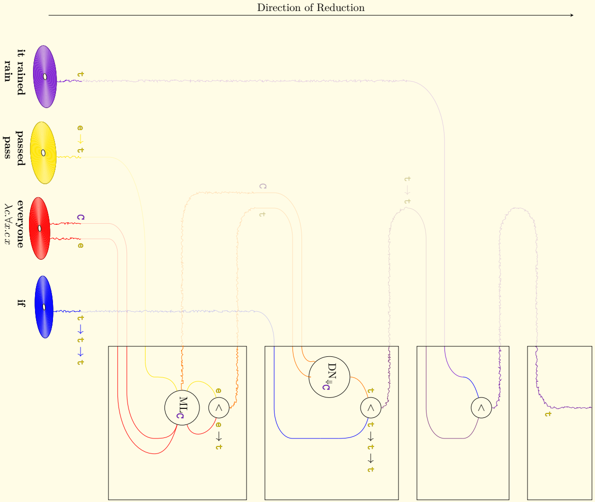


A Parsing Diagram Step



Building an Intuition

A Full Parsing Diagram



Plan

Reducing the grammar

We introduce denotations for our combinators, to allow us to define reductions that relieve a bit the ambiguity of the parsing grammar:

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

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

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

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

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

Reductions

Reductions include the following:

- When two effects commute, we choose an order to apply them.
- We use UR instead of using MR or DNMR when possible.
- We always use modes J, C and DN as early as possible. The fact this works is a consequence of Theorem 3.1 on the denotation diagrams.
- All the rules provided in the previous section can be rewritten here too.

Conclusion

We have formalised an enhancement of usual type systems for natural language semantics.

Thanks to the introduction of the string diagrams, this did not come at the cost of comprehension of the system nor efficiency.

Conclusion

We have formalised an enhancement of usual type systems for natural language semantics.

Thanks to the introduction of the string diagrams, this did not come at the cost of comprehension of the system nor efficiency.

I would like to thank Simon Charlow for his advice, 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?

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

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

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

Combinator Denotations

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

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

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

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

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

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

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

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

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

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

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

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