Higher-Order DisCoCat

Peirce-Lambek-Montague Semantics

Alexis Toumi, Giovanni de Felice

ACT 2024, Oxford

Outline

Coecke

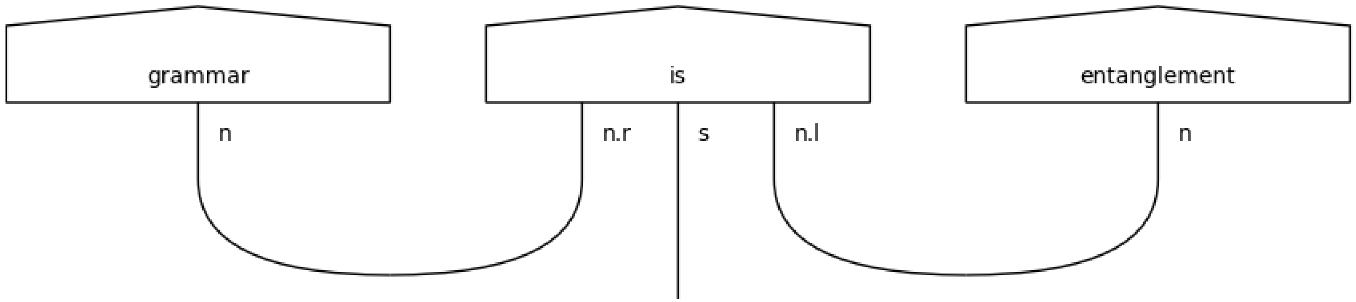
Lambek

Montague

Peirce

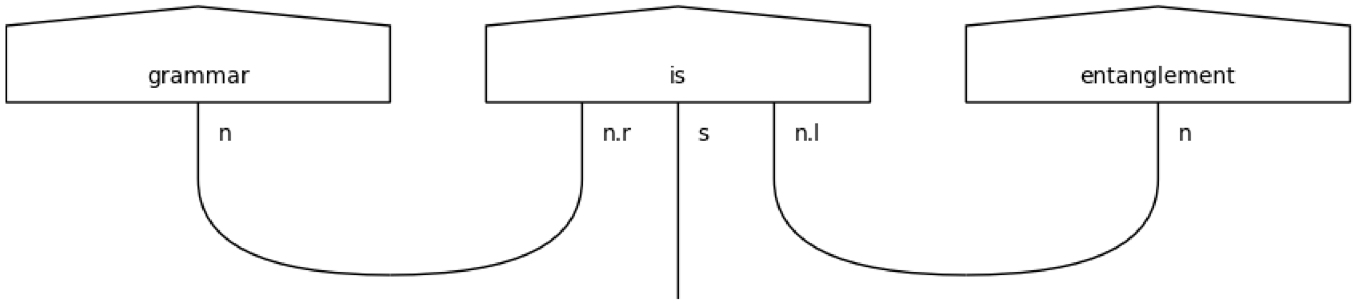
1. DisCoCat $F : \mathbf{G} \to \mathbf{FinVect}$
2. Lambek grammars $\mathbf{G}$ as categories
3. Montague semantics $F : \mathbf{G} \to \Lambda L$
4. Peirce: first-order logic with diagrams
5. Higher-Order DisCoCat $F : \mathbf{G} \to \Lambda D$

DisCoCat $F : \mathbf{G} \to \mathbf{FinVect}$



1. Parse your sentence as a morphism in a closed (rigid) category $\mathbf{G}$
2. Map each type t_i to a finite-dimensional vector space $F(t_i) = \mathbb{R}^d$ and each word w_i of type t_i to a vector $F(w_i \to t_i) \in F(t_i)$
3. Compute the meaning of the sentence as the image of the functor $F : \mathbf{G} \to \mathbf{FinVect}$, i.e. perform tensor network contraction

DisCoCat $F : \mathbf{G} \to \mathbf{FinVect}$



Pros	Cons
both symbolic and statistical	hard to train on big datasets
intuitive graphical language	curse of dimensionality
natural quantum implementation	it's all linear maps!

Categorial Grammar: Context

Die syntaktische Konnexität, Ajdukiewicz (1935)

$$x\,\frac{y}{x} \to y$$

A quasi-arithmetical notation for syntactic description, Bar-Hillel (1953)

$$x(x\backslash y) \to y \leftarrow (y/x)x$$

The Mathematics of Sentence Structure, Lambek (1958)

$$x \otimes (x\backslash y) \to y \leftarrow (y/x) \otimes x$$

Categorial and Categorical Grammars, Lambek (1988)

Categorial Grammar: Definition (1/2)

Definition: Given a set X, we define $T(X) \supseteq X + \{I\}$ where for all $x, y \in T(X)$ we have $(x \backslash y)$, (x/y) and $(x \otimes y) \in T(X)$

Definition: A categorial grammar is a tuple $G = (V, X, D, s)$ where:

- V is a set of words called the **vocabulary**
- X is a set of **basic types** with $s \in X$ the **sentence type**
- $D \subseteq V \times T(X)$ is a set of **dictionary entries**
- In practice, this is extended with a set of **ad-hoc rules** $R \subseteq X \times X$.

Categorial Grammar: Definition (2/2)

Definition: The language of a categorial grammar G is given by:

$$L(G) = \{w_1 \dots w_n \in V^\star \mid \exists\, f : w_1 \otimes \cdots \otimes w_n \to s \in \mathbf{G}\}$$

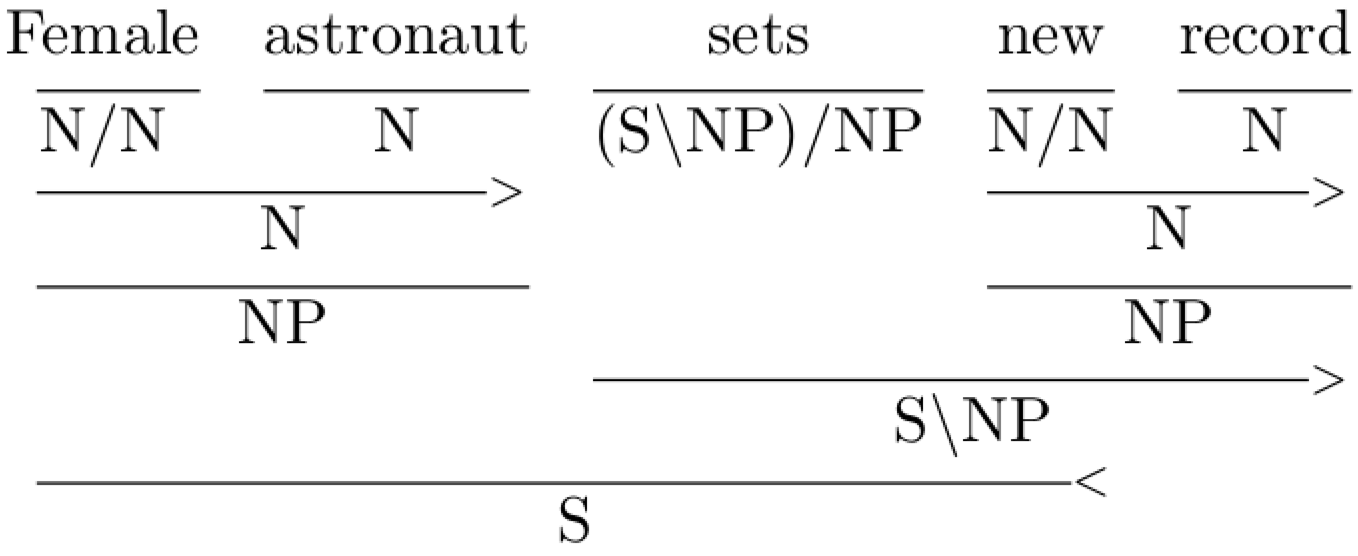
where $\mathbf{G}$ is the free (non-symmetric) closed monoidal category, i.e.

$$\mathbf{G}(y, x \backslash z) \simeq \mathbf{G}(x \otimes y, z) \simeq \mathbf{G}(x, z / y)$$

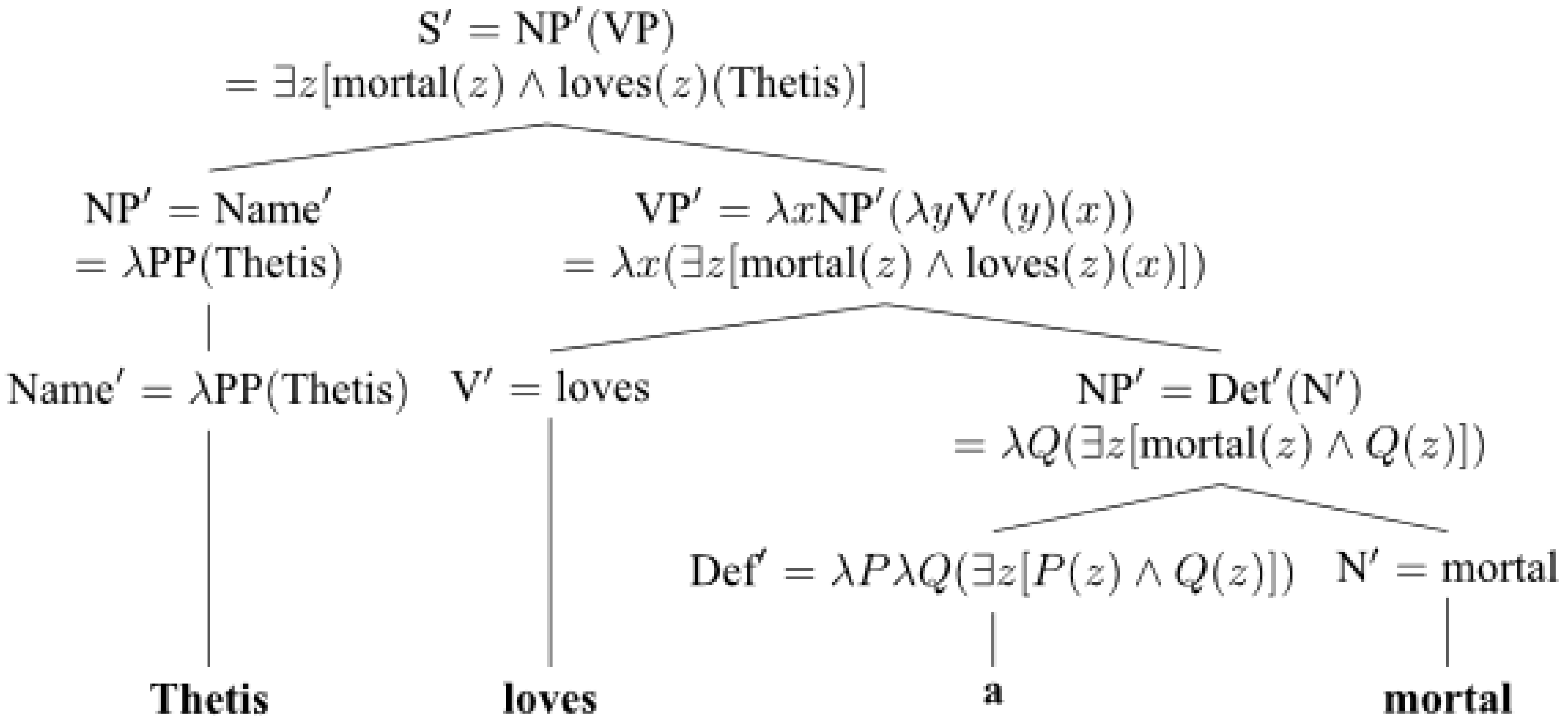
Categorial Grammar: Example

$$V = \{\text{Female, astronaut, sets}, \ldots\} \quad X = \{N, NP, S\}$$

$$D = \{(\text{Female}, N \leftarrow N), (\text{astronaut}, N), \ldots\} \quad R = \{(N, NP)\}$$



Montague Semantics



English as a formal language, Montague (1970)

Formulae as simply-typed lambda terms

Let L be the signature of first-order logic, i.e. $L_0 = \{\tau, \phi\}$ and

L_1	**symbol**	**type**
constants	Alice, Bob, ...	τ
unary predicates	man, sings, ...	$\tau \to \phi$
binary predicates	needs, sees, ...	$\tau \to (\tau \to \phi)$
nullary operators	$\top, \bot$	ϕ
unary operators	$\neg$	$\phi \to \phi$
binary operators	$\wedge, \vee, \to$	$\phi \to (\phi \to \phi)$
quantifiers	$\forall, \exists$	$(\tau \to \phi) \to \phi$

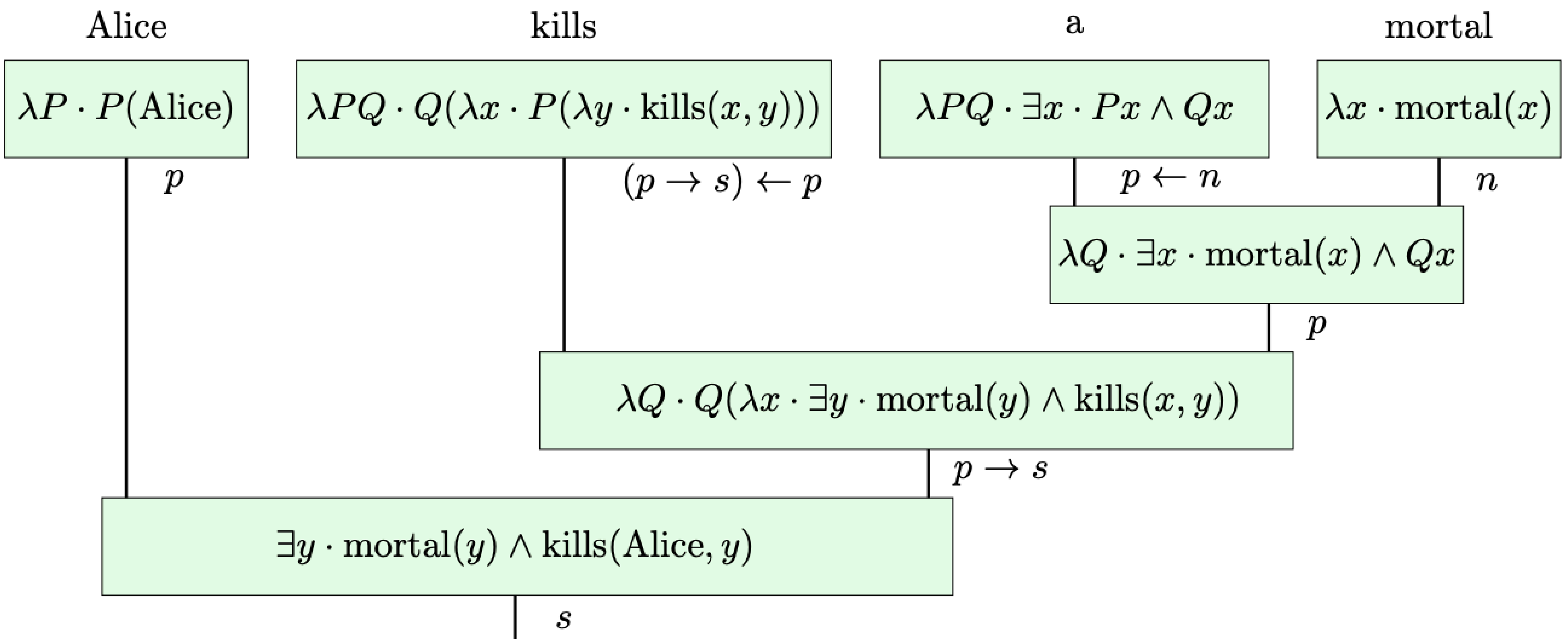
Let ΛL be the free cartesian closed category it generates, i.e. where the lambda terms $f : 1 \to \phi$ reduce to first-order logic formulae.

Montague Semantics $F : \mathbf{G} \to \Lambda L$

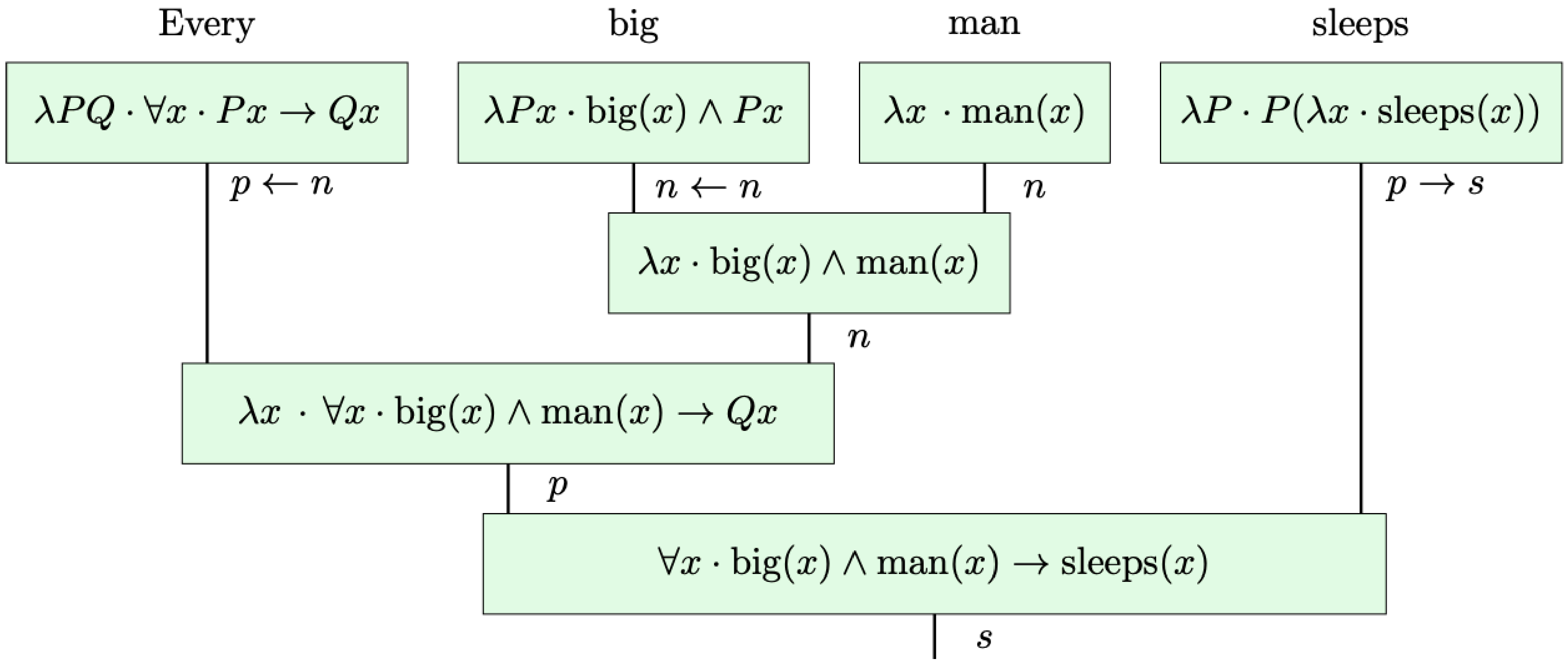
$$F(s) = \phi \qquad F(n) = \tau \to \phi \qquad F(p) = (\tau \to \phi) \to \phi$$

Montague semantics	**word** w	**type** t	**meaning** $F(w \to t)$
proper nouns	Alice	p	$\lambda P.P(Alice)$
common nouns	man	n	$\lambda x.\text{man}(x)$
adjectives	big	$n \leftarrow n$	$\lambda Px.\text{big}(x) \wedge Px$
determiners	every	$p \leftarrow n$	$\lambda PQ.\forall x.Px \to Qx$
intransitive verbs	sleeps	$p \to s$	$\lambda P.P(\lambda x.\text{sleeps}(x))$
transitive verbs	kills	$(p \to s) \leftarrow p$	$\lambda PQ.Q(\lambda x.P(\lambda y.\text{kills}(x, y)))$

Montague Semantics $F : \mathbf{G} \to \Lambda L$



Montague Semantics $F : \mathbf{G} \to \Lambda L$



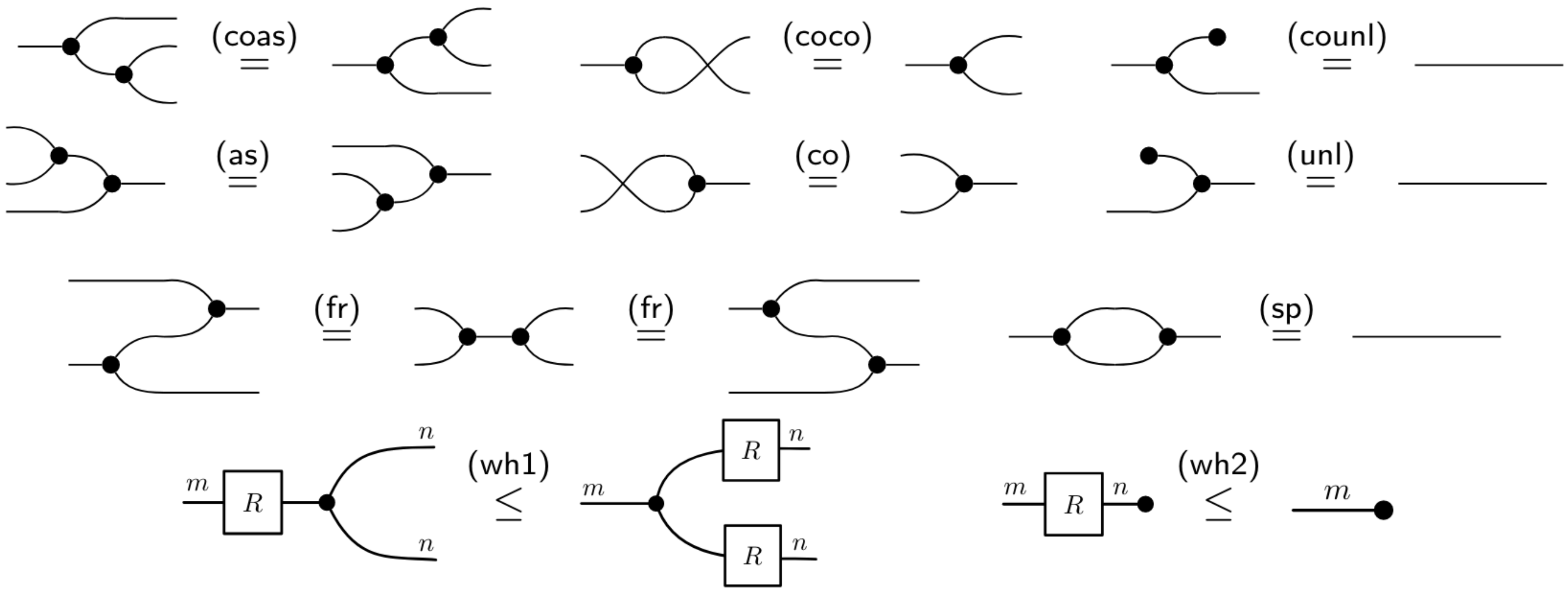
Peirce: first-order logic as string diagrams

PROLEGOMENA TO AN APOLOGY FOR PRAGMATICISM.

COME on, my Reader, and let us construct a diagram to illustrate the general course of thought; I mean a System of diagrammatization by means of which any course of thought can be represented with exactitude.

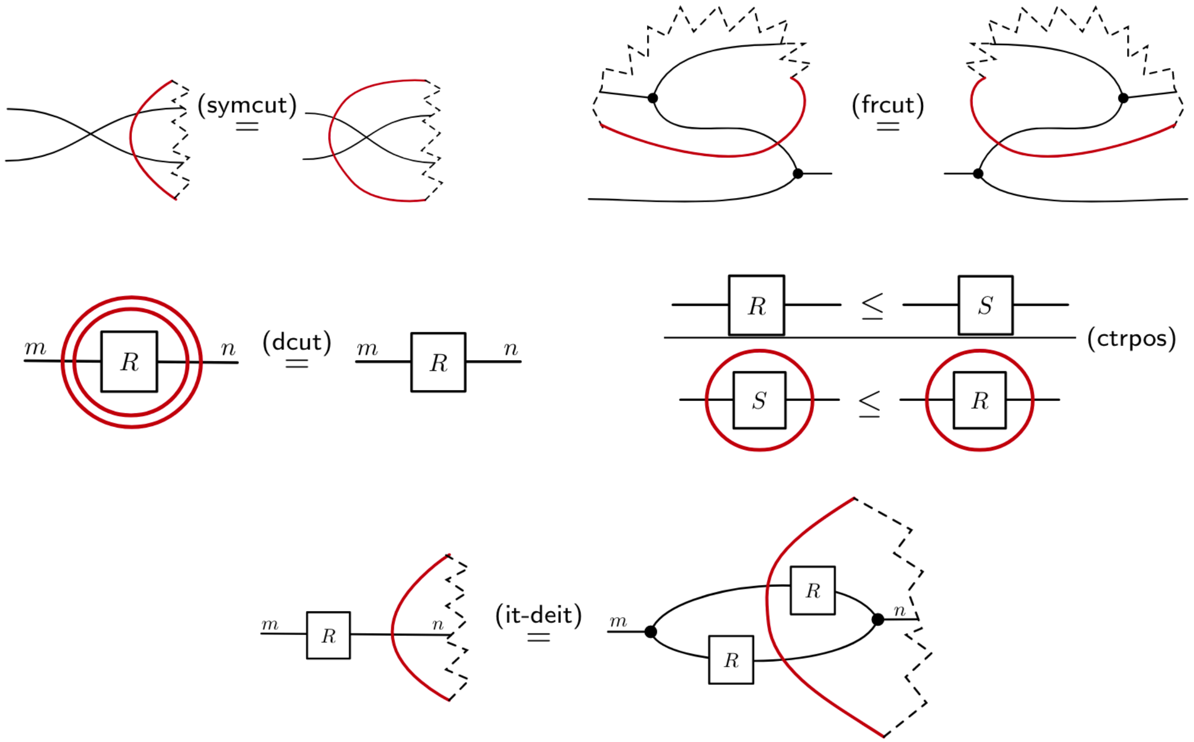
Hand-written as early as 1882, but remained unpublished until 1906.

Peirce: first-order logic as string diagrams



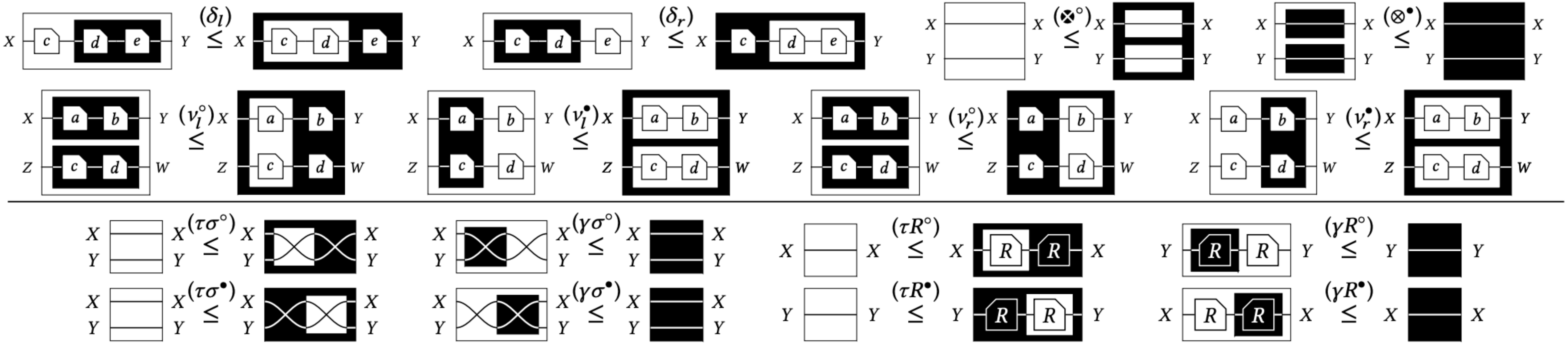
Compositional Diagrammatic First-Order Logic,
Haydon and Sobocinski (2020)

Peirce: first-order logic as string diagrams



Compositional Diagrammatic First-Order Logic,
Haydon and Sobocinski (2020)

Peirce: first-order logic as string diagrams



Diagrammatic Algebra of First Order Logic,
Bonchi, Di Giorgio, Haydon & Sobocinski (2024)

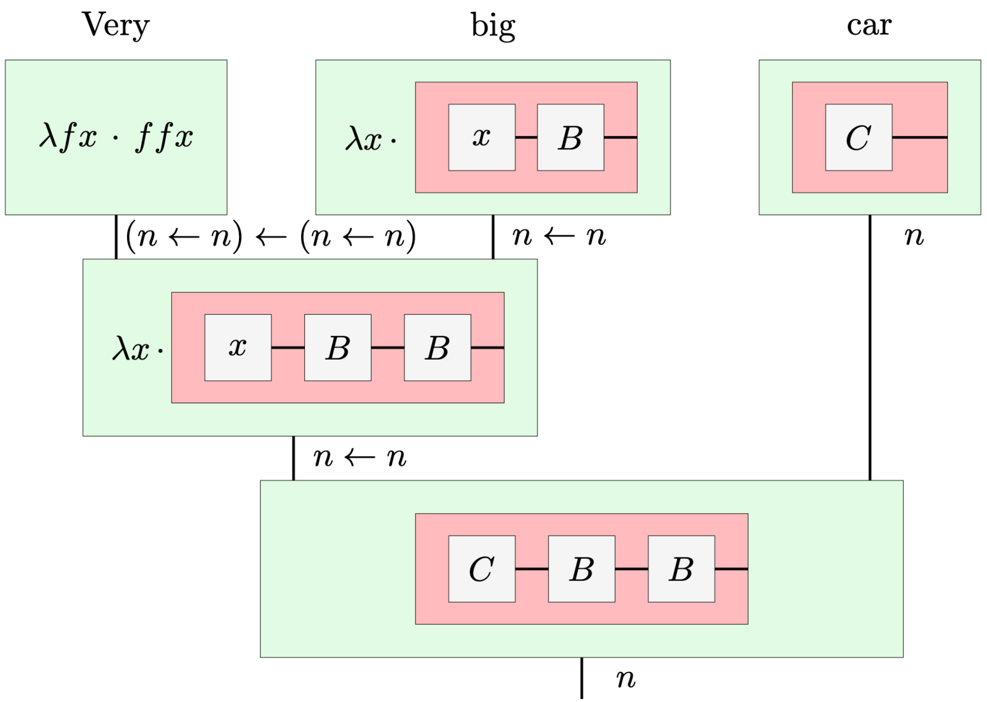
Diagrams as simply-typed lambda terms

Let D be the signature of string diagrams generated by a monoidal signature $\Sigma = (\Sigma_0, \Sigma_1, \texttt{dom}, \texttt{cod})$, i.e. $D_0 = \Sigma_0^\star \times \Sigma_0^\star$ and

D_1	symbol	type
boxes	$f \in \Sigma$	(x, y)
identity	id_x	(x, x)
composition	$\circ_{xyz}$	$(x, y) \to (y, z) \to (x, z)$
tensor	$\otimes_{xyzw}$	$(x, y) \to (z, w) \to (xz, yw)$

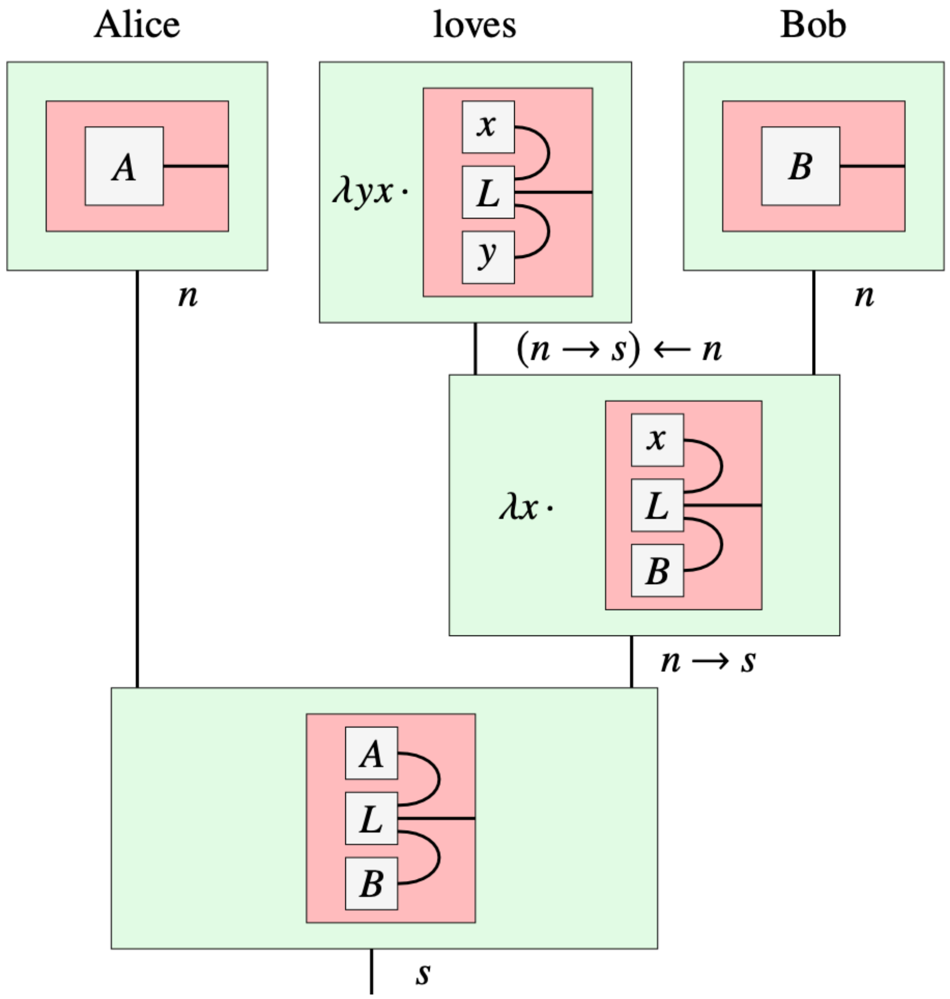
Let ΛD be the free cartesian closed category it generates, i.e. where the lambda terms $f : 1 \to (x, y)$ reduce to string diagrams $x \to y$.

Higher-Order DisCoCat $F : \mathbf{G} \to \Lambda D$



"very big" = "big big"	**word** w	**type** t	**meaning** $F(w \to t)$
common nouns	car	n	car $\in \Sigma$
adjectives	big	$n \leftarrow n$	$\lambda x.\text{big} \circ x$
adverbs	very	$(n \leftarrow n) \leftarrow (n \leftarrow n)$	$\lambda f x. f f x$

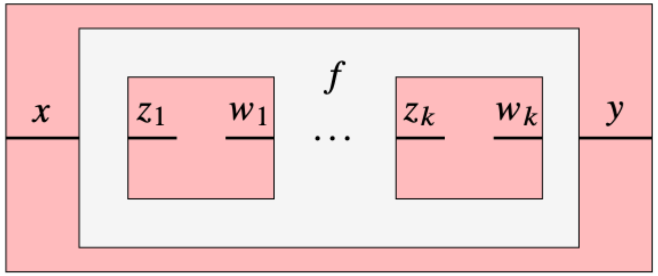
Higher-Order DisCoCat $F : \mathbf{G} \to \Lambda D$



Higher-Order DisCoCat $F : \mathbf{G} \to \Lambda D$

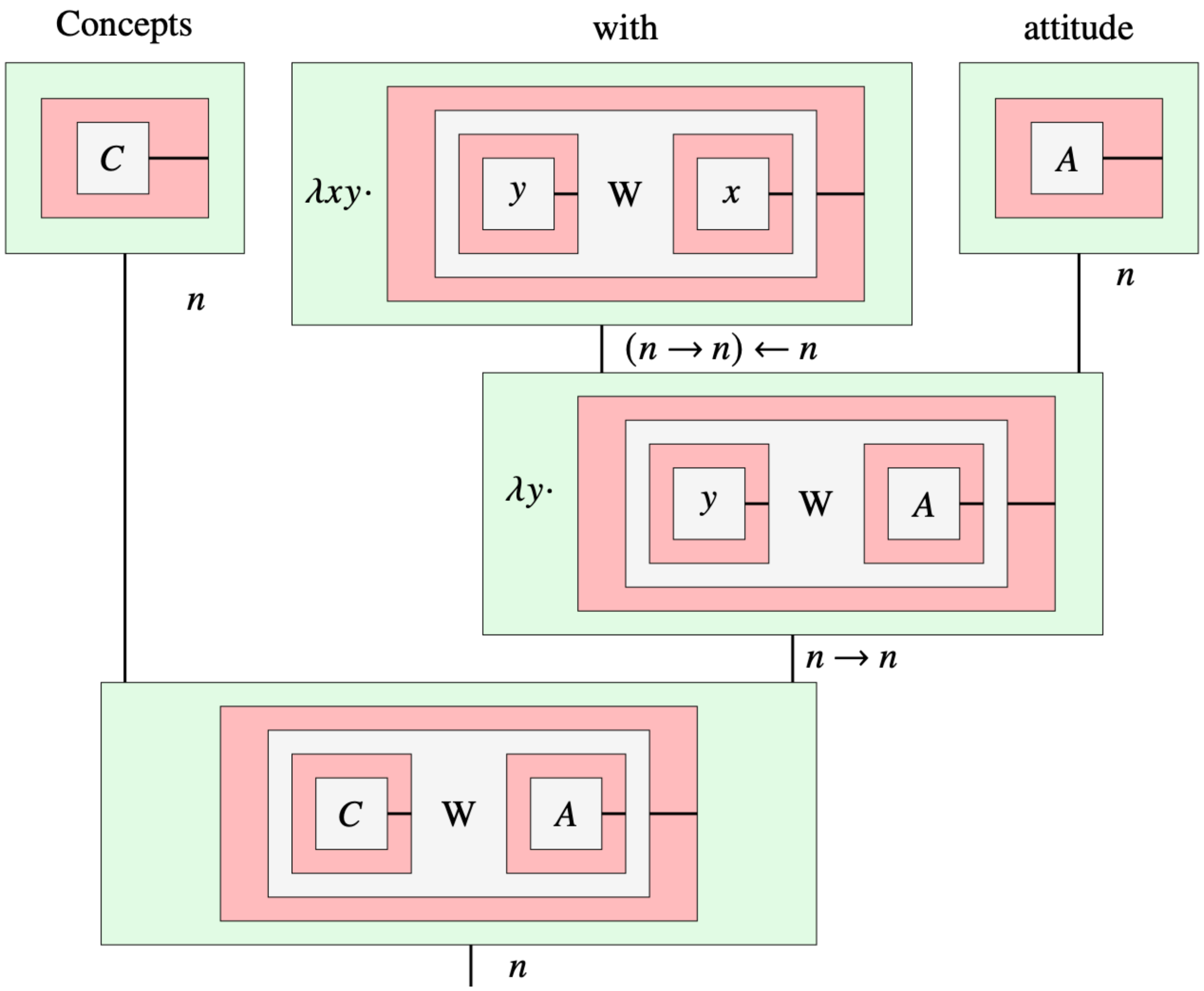
$$\Sigma = (\Sigma_0, \Sigma_1, \texttt{dom}, \texttt{cod}, \text{holes})$$

$$\text{holes}(f) = \big((z_1, w_1), \ldots, (z_k, w_k)\big) \in (\Sigma_0^\star \times \Sigma_0^\star)^\star$$

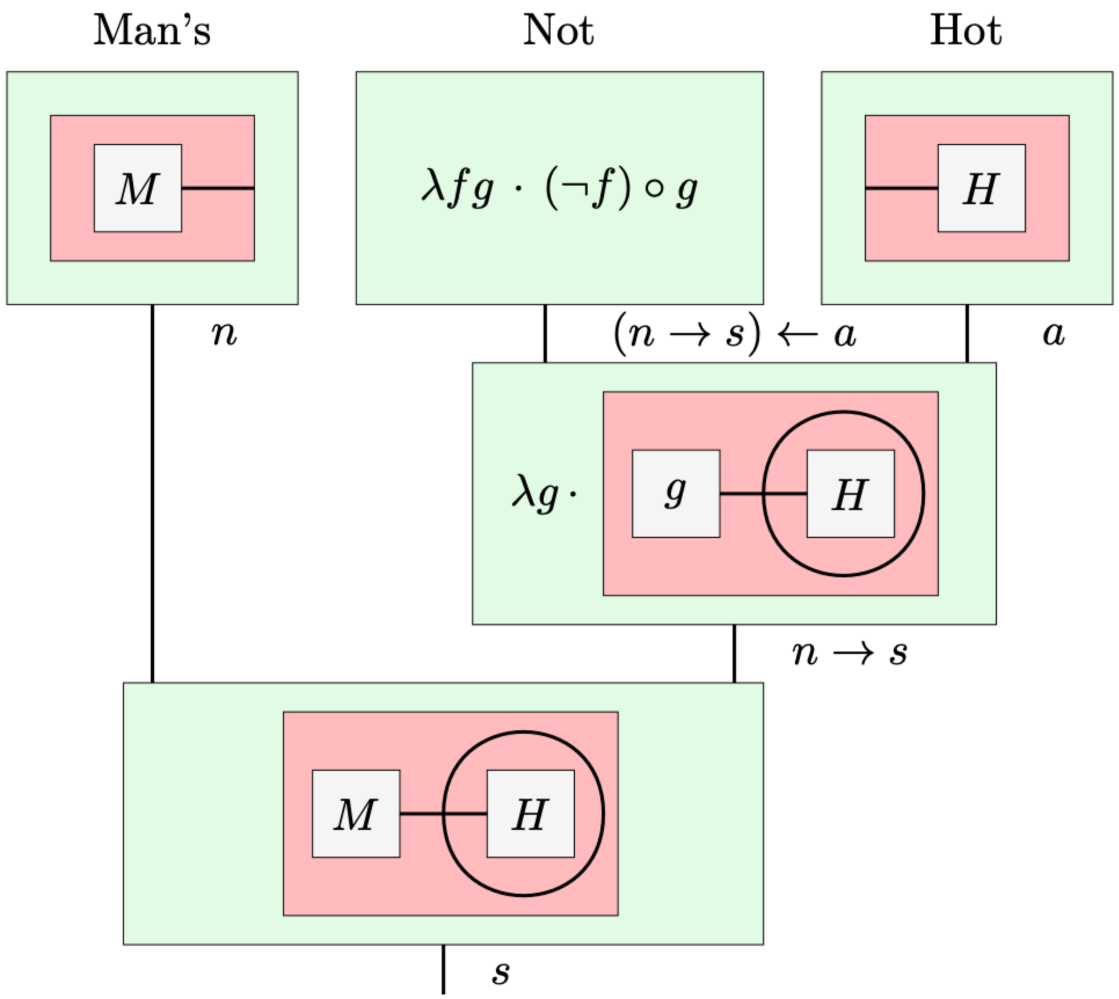


D_1 (continued)	symbol	type
	...	
boxes with holes	$f \in \Sigma$	$(z_1, w_1) \to \cdots \to (z_k, w_k) \to (x, y)$

Higher-Order DisCoCat $F : \mathbf{G} \to \Lambda D$



Higher-Order DisCoCat $F : \mathbf{G} \to \Lambda D$

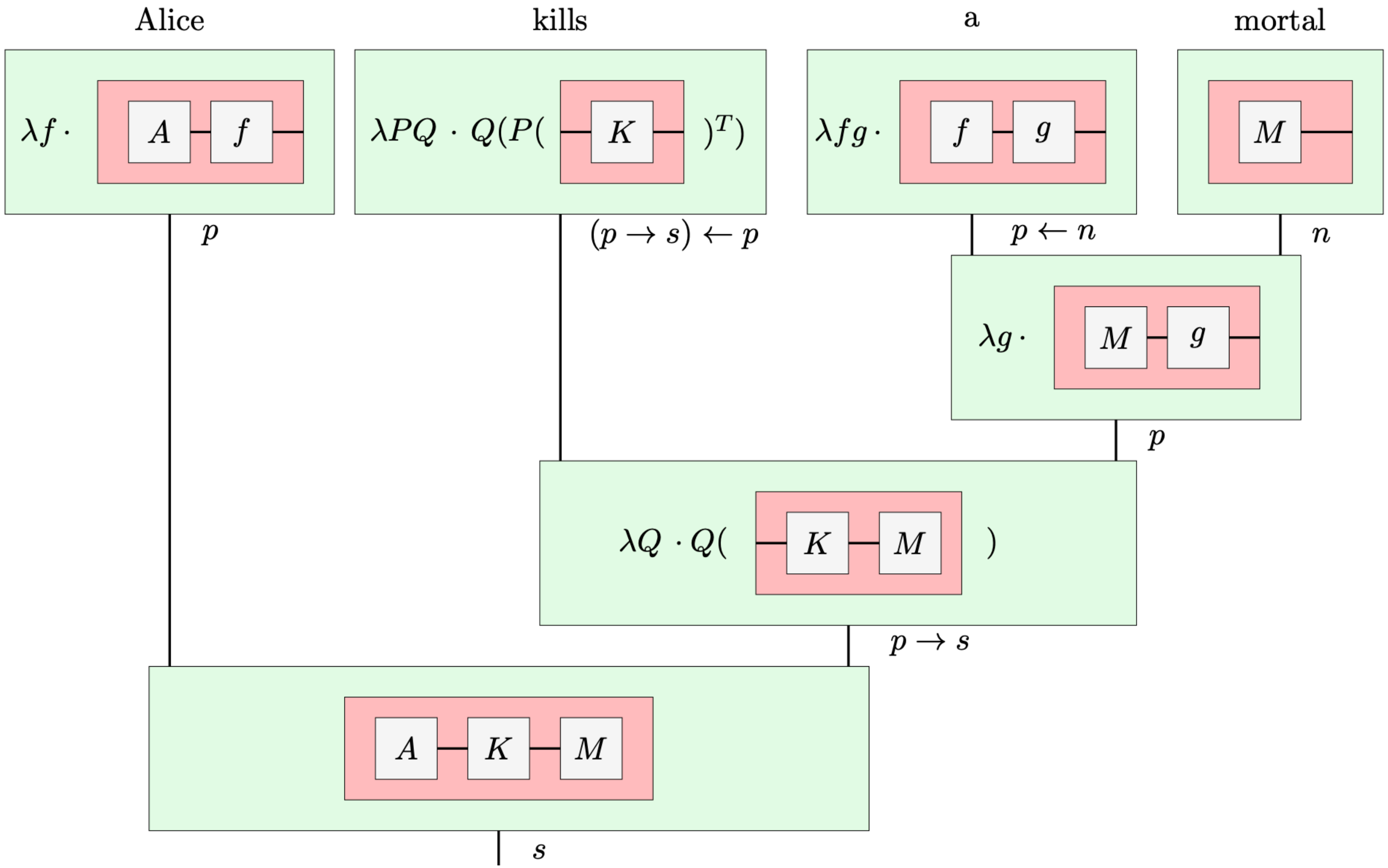


Peirce-Lambek-Montague semantics

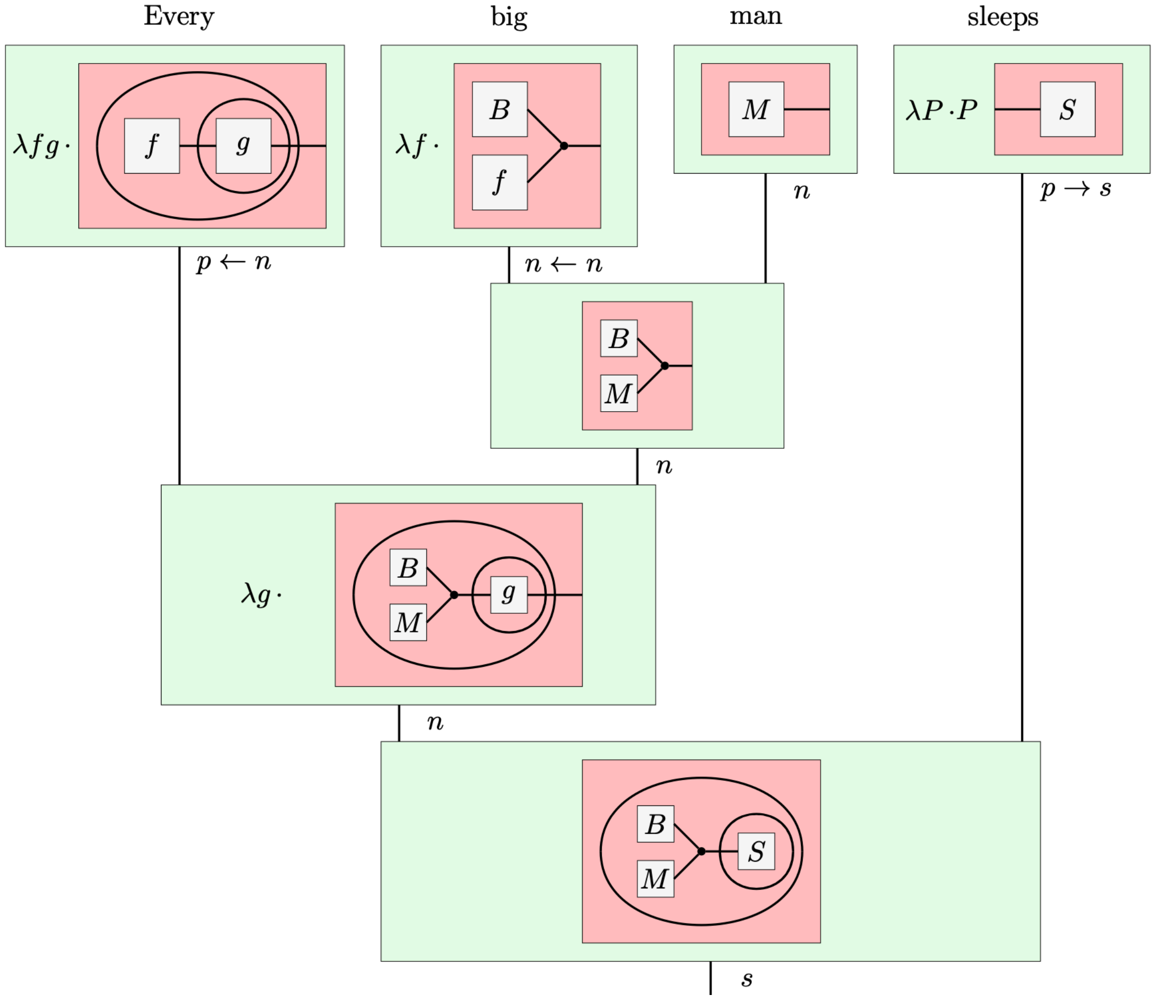
$$F(s) = (1, 1) \qquad F(n) = (1, N) \qquad F(p) = \prod_{x \in \Sigma_0^\star} (N, x) \to (1, x)$$

Peirce-Lambek-Montague	**word** w	**type** t	**meaning** $F(w \to t)$
common nouns	man	n	$(\text{man} : 1 \to N) \in \Sigma$
proper nouns	Alice	p	$\lambda f.\text{Alice} \circ f$
adjectives	big	$n \leftarrow n$	$\lambda f.\text{spider}_{2,1} \circ (\text{big} \otimes f)$
determiners	no	$p \leftarrow n$	$\lambda fg.\text{cut}(g \circ f)$
intransitive verbs	sleeps	$p \to s$	$\lambda P.P(\text{sleeps})$
transitive verbs	kills	$(p \to s) \leftarrow p$	$\lambda PQ.Q(P(\text{kills})^T)$
copula	is	$(p \to s) \leftarrow p$	$\lambda PQ.Q(P(\text{id}_N)^T)$

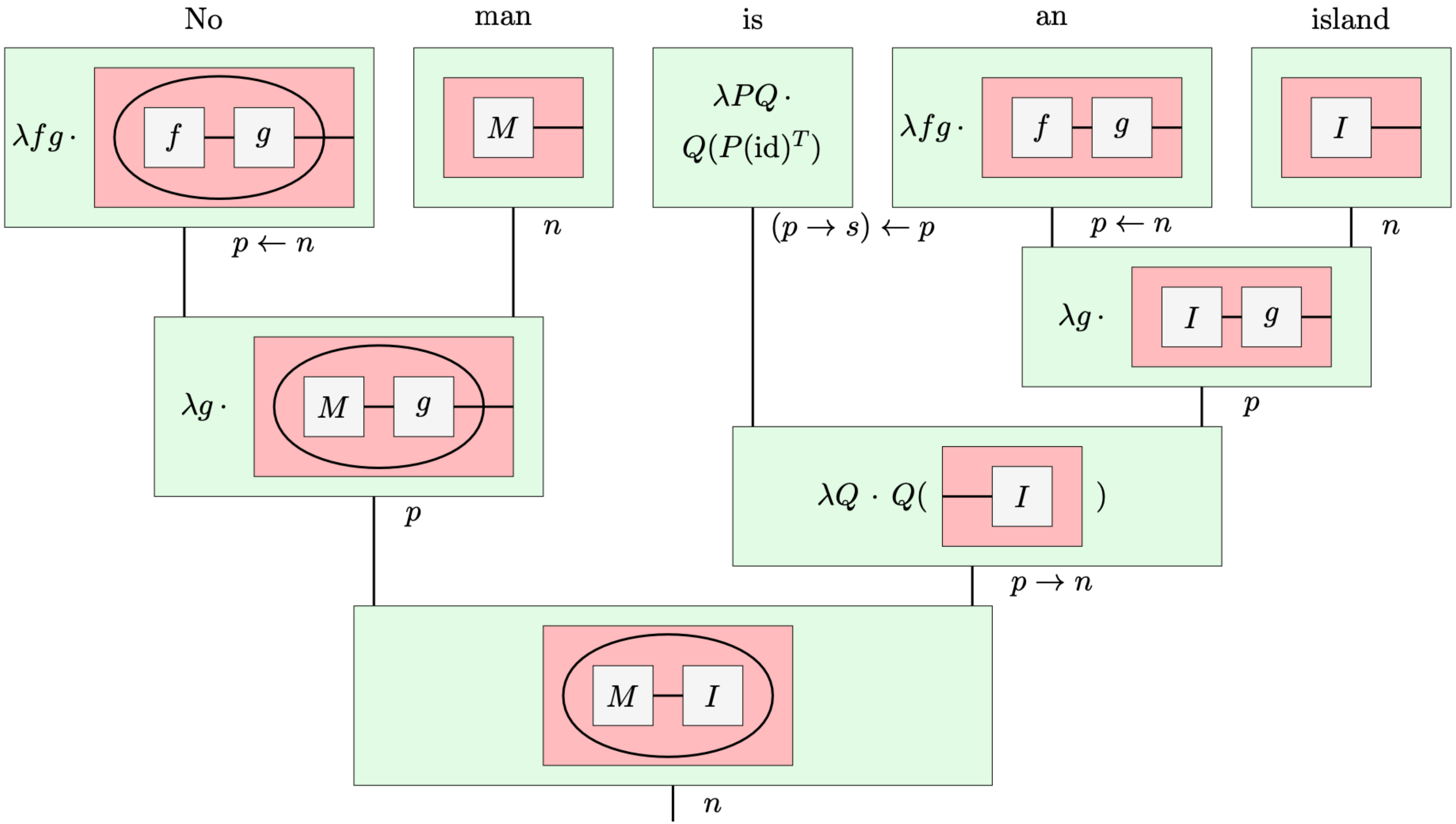
Peirce-Lambek-Montague semantics



Peirce-Lambek-Montague semantics



Peirce-Lambek-Montague semantics



Implementation (1/4): Formula

```python
from discopy import frobenius, utils
from discopy.tensor import Dim, Tensor

@utils.factory  # Ensure the subclass is closed under composition
class Formula(frobenius.Diagram):
    ty_factory = frobenius.PRO  # i.e. natural numbers as objects

    def eval(self, size: int) -> Tensor[bool]:
        return frobenius.Functor(
            ob=lambda _: Dim(size),
            ar=lambda box: box.data,
            cod=Category(Dim, Tensor[bool]))(self)

class Cut(frobenius.Bubble, Formula): ...
class Ligature(frobenius.Spider, Formula): ...
class Predicate(frobenius.Box, Formula): ...

Id, Formula.bubble_factory = Formula.id, Cut
Tensor[bool].bubble = lambda self, **_: self.map(lambda x: not x)
```

Implementation (2/4): Grammar

```
from discopy.grammar.categorial import Ty, Word, Eval

n, p, s = Ty('n'), Ty('p'), Ty('s')  # noun, phrase and sentence

man, island = (Word(noun, n) for noun in ("man", "island"))
_is = (Word(verb, (p >> s) << p) for verb in ("is"))
no, an = (Word(det, p << n) for det in ("no", "an"))

no_man_is_an_island = (no @ man @ _is @ an @ island
    >> Eval(p << n) @ ((p >> s) << p) @ Eval(p << n)
    >> p @ Eval((p >> s) << p) >> Eval(p >> s))
```

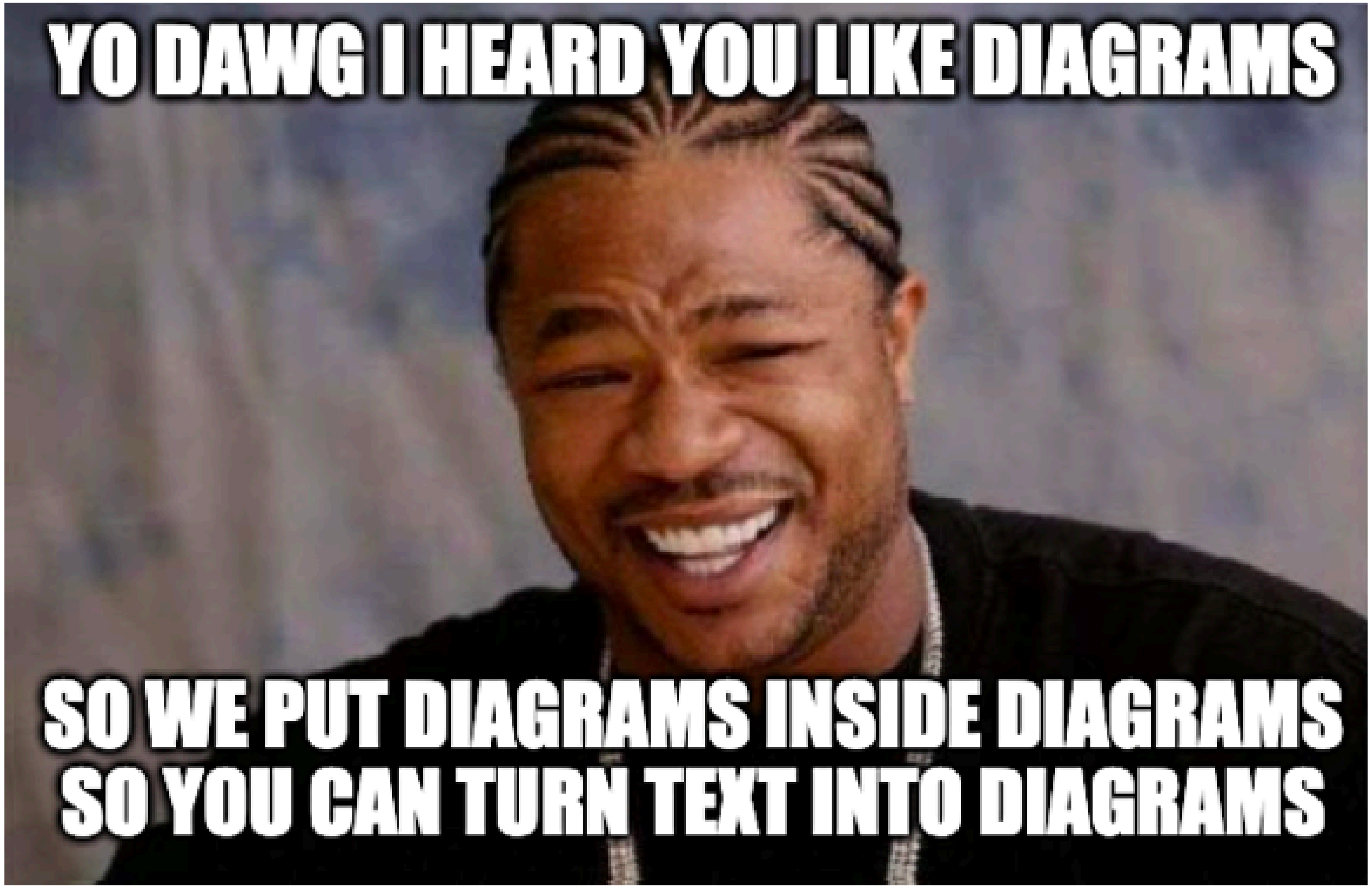
Implementation (3/4): Functor

```
from discopy import closed, python

M, I = (Predicate("M", 0, 1, data) for P, data in zip("MI", unary_predicates))

F = closed.Functor(
    cod=closed.Category(tuple[type, ...], python.Function),
    ob={s: Formula, n: Formula, p: Callable[[Formula], Formula]},
    ar={Alice: lambda: lambda f: A >> f,
        sleeps: lambda: lambda P: P(S.dagger()),
        man: lambda: M, island: lambda: I,
        big: lambda: lambda f: f @ B >> Ligature(2, 1, frobenius.PRO(1)),
        _is: lambda: lambda P: lambda Q: Q(P(Id(1)).dagger()),
        kills: lambda: lambda P: lambda Q: Q(P(K).dagger()),
        no: lambda: lambda f: lambda g: (f >> g).bubble(),
        some: lambda: lambda f: lambda g: f >> g,
        every: lambda: lambda f: lambda g: (f >> g.bubble()).bubble()})
```

Implementation (4/4): Evaluation

```python
from random import choice

size = 42  # Generating a random interpretation to test our model
random_bits = lambda n=size: [choice([True, False]) for _ in range(n)]

unary_predicates = is_man, is_island = [random_bits() for _ in range(5)]

evaluate = lambda sentence: bool(F(sentence)().eval(size))

assert evaluate(no_man_is_an_island) == all(
    not is_man[x] or not is_island[x] for x in range(size))
```

Thank you!


YO DAWG I HEARD YOU LIKE DIAGRAMS
SO WE PUT DIAGRAMS INSIDE DIAGRAMS
SO YOU CAN TURN TEXT INTO DIAGRAMS