Refined Syntax & Semantics Via Taking Contexts Seriously

Philip Saville, University of Oxford

Directions and perspectives in the λ-calculus
Bologna, Jan. '24

A long thread:

new structure ⇝ models express new structures

or programs ⇜

A long thread:

Syntax ⇄ semantics

eg// CCCs, *-autonomous categories, premonoidal categories, ...

eg// linear logic, differential λ-calculus, monadic metalanguage, ...

The trend: refinement on both sides

The trend: refinement on both sides

Syntax ⇄ semantics

graded monads

[Melliès, Katsumata, Fujii, Gaboardi, Orchard, ...]

fuzzy syntax

[de Amorim, Hsu, Katsumata, Gaboardi, Cherigui, ...]

cost analysis

[Niu, Sterling, Grodin, Harper, Gaboardi, ...]

NB: an incomplete list!

The trend: refinement on both sides

Syntax ⇄ semantics

graded monads
[Melliès, Katsumata, Fujii, Gaboardi, Orchard, ...]

fuzzy syntax
[de Amorim, Hsu, Katsumata, Gaboardi, Cherigui, ...]

cost analysis
[Niu, Sterling, Grodin, Harper, Gaboardi, ...]

2-dimensional
[Fiore, Gambino, Hyland, Winskel, Olimpieri, Paquet, Galal, Melliès, ...]

enrichment
[Kavvos, Levy, McDermott-Uustalu, ...]

NB: an incomplete list!

The trend: refinement on both sides

subtle features / relations between programs

rich, expressive models

inc. soundness, completeness, ...

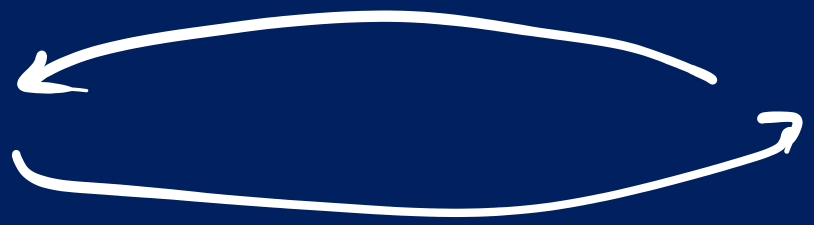
1) Can we canonically extract syntax from semantics?

2) What common ideas can we use for all these cases?

Looking backwards

[Lambek, ...]

(typed) Syntax



categorical semantics

Looking backwards

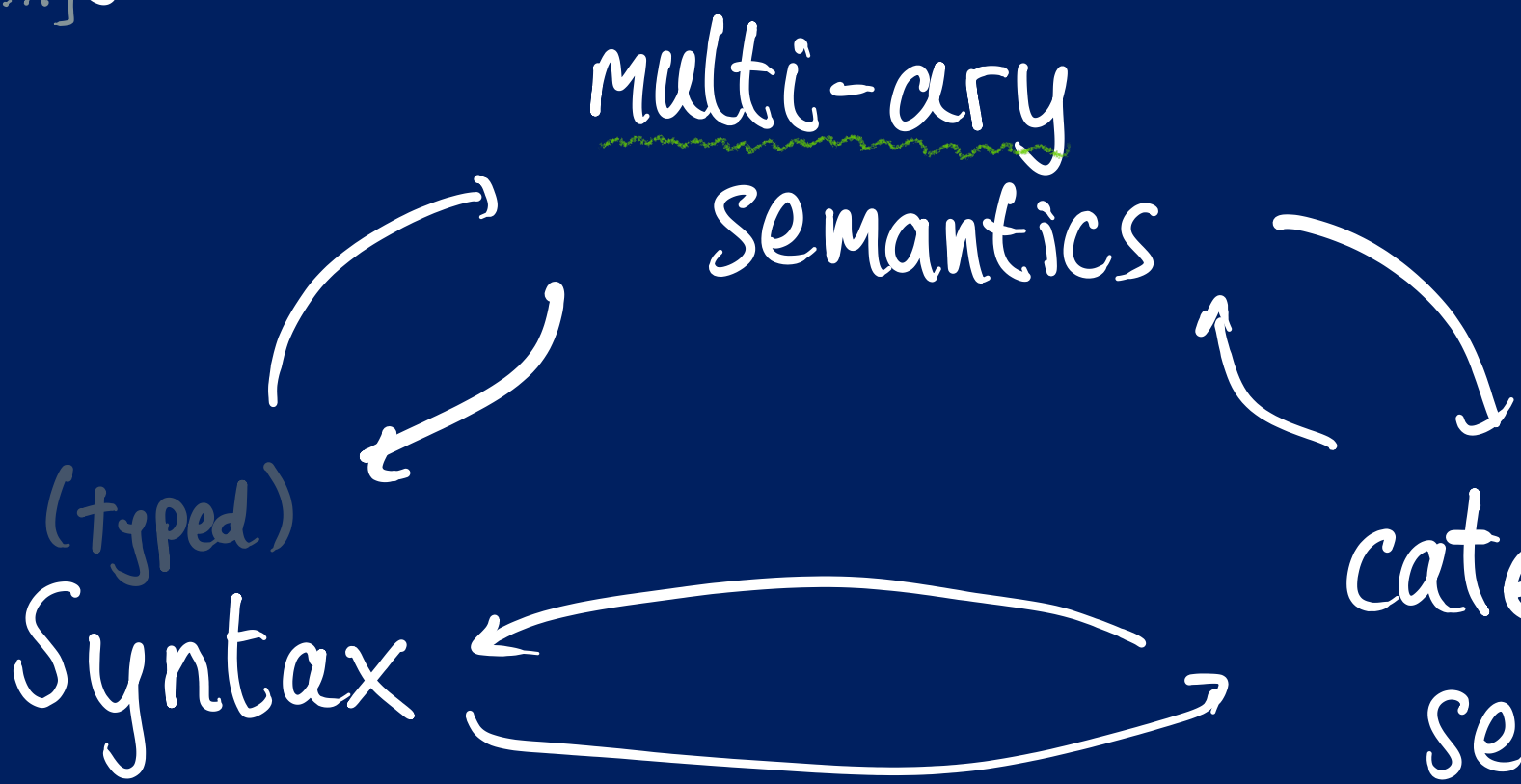
[Lambek, ...]

Looking backwards

[Lambek, ...]

MANY INPUTS

$f : A_1, \ldots, A_n \longrightarrow B$



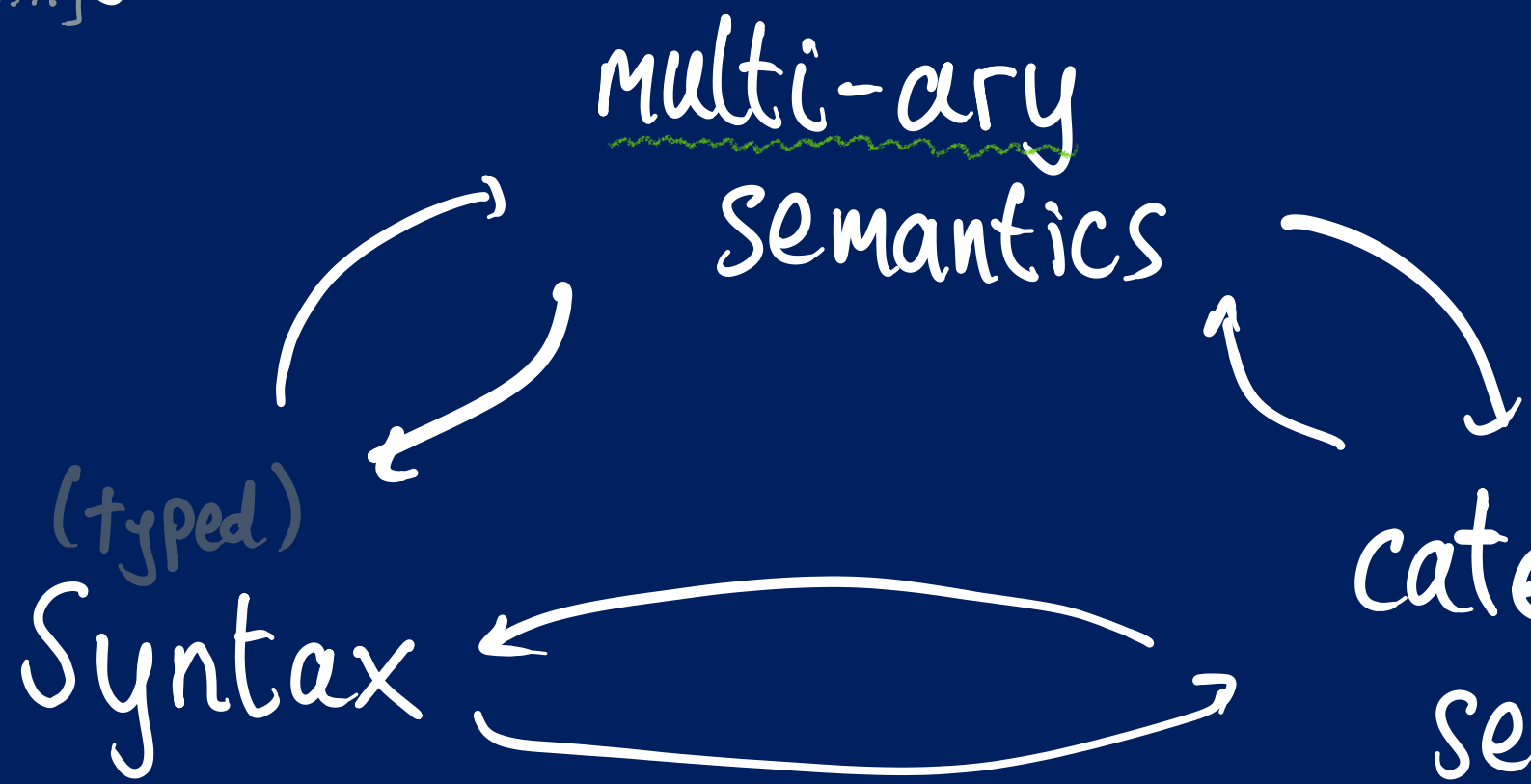
ONE INPUT $f : A \longrightarrow B$

Looking backwards

[Lambek, ...]

MANY INPUTS

$f : A_1, \ldots, A_n \longrightarrow B$



MANY INPUTS

$x_1 : A_1, \ldots, x_n : A_n \vdash t : B$

ONE INPUT $f : A \longrightarrow B$

Bonuses

- resolves the unary / multi-ary mismatch
- distinguishes contexts and product types
- easy to prove soundness + completeness, etc
- a natural way to describe lots of useful language constructs
- naturally generalises

PROPONENTS: Lambek, Hyland, Fiore, Shulman, ...

Bonuses

- resolves the unary / multi-ary mismatch
- distinguishes contexts and product types
- easy to prove soundness + completeness, etc
- a natural way to describe lots of useful language constructs
- naturally generalises

Examples

= has contraction, permutation, weakening

cartesian simple type theories

= has contraction, permutation, weakening

clones

cartesian simple type theories

multicategories, symmetric multicategories

clones

ordered / linear

cartesian simple type theories

multisorted, abstract

def: a clone $\mathbb{C}$ has:

[Hall]

- objects $A, B, C, \ldots$
- multimaps $f, g, \ldots : A_1, \ldots, A_n \longrightarrow B$, $(n \geq 0)$
  including $p_i^{A_1, \ldots, A_n} : A_1, \ldots, A_n \longrightarrow A_i$ for $i = 1, \ldots, n$
- a substitution operation

$$\frac{f : A_1, \ldots, A_n \longrightarrow B \qquad (g_i : \Delta \longrightarrow A_i)_{i=1,\ldots,n}}{f[g_1, \ldots, g_n] : \Delta \longrightarrow B}$$

multisorted, abstract

def: a clone $\mathbb{C}$ has:

[Hall]

- objects $A, B, C, \ldots$
- multimaps $f, g, \ldots : A_1, \ldots, A_n \longrightarrow B$, $(n \geq 0)$
  including $p_i^{A_1, \ldots, A_n} : A_1, \ldots, A_n \longrightarrow A_i$ for $i = 1, \ldots, n$
- a substitution operation

$$\frac{f : A_1, \ldots, A_n \longrightarrow B \qquad (g_i : \Delta \longrightarrow A_i)_{i=1,\ldots,n}}{f[g_1, \ldots, g_n] : \Delta \longrightarrow B}$$

multisorted, abstract

def: a clone $\mathbb{C}$ has:

[Hall]

- objects $A, B, C, \ldots$
- multimaps $f, g, \ldots : A_1, \ldots, A_n \longrightarrow B$, $(n \geq 0)$
  including $p_i^{A_1, \ldots, A_n} : A_1, \ldots, A_n \longrightarrow A_i$ for $i = 1, \ldots, n$
- a substitution operation

$$\frac{f : A_1, \ldots, A_n \longrightarrow B \qquad (g_i : \Delta \longrightarrow A_i)_{i=1,\ldots,n}}{f[g_1, \ldots, g_n] : \Delta \longrightarrow B}$$

multisorted, abstract

def: a clone $\mathbb{C}$ has:

[Hall]

- objects $A, B, C, \ldots$
- multimaps $f, g, \ldots : A_1, \ldots, A_n \to B$, $(n \geq 0)$
  including $p_i^{A_1, \ldots, A_n} : A_1, \ldots, A_n \to A_i$ for $i = 1, \ldots, n$
- a substitution operation

$$\frac{f : A_1, \ldots, A_n \to B \qquad (g_i : \Delta \to A_i)_{i=1,\ldots,n}}{f[g_1, \ldots, g_n] : \Delta \to B}$$

multisorted, abstract

$$p_i[f_1,\dots,f_n] = f_i$$
$$f[p_1,\dots,p_n] = f$$
$$(f[g_1,\dots,g_n])[h_1,\dots,h_m] = f[\dots,g_i[h_\bullet],\dots]$$

def: a clone $\mathbb{C}$ has: [Hall]

- objects $A, B, C, \dots$
- multimaps $f, g, \dots : A_1, \dots, A_n \to B$, $(n \geq 0)$
  including $p_i^{A_1,\dots,A_n} : A_1, \dots, A_n \to A_i$ for $i = 1, \dots, n$
- a substitution operation

$$\frac{f : A_1, \dots, A_n \to B \qquad (g_i : \Delta \to A_i)_{i=1,\dots,n}}{f[g_1, \dots, g_n] : \Delta \to B}$$

simple

And: a type theory has:

- types $A, B, C, \ldots$
- terms $x_1 : A_1, \ldots, x_n : A_n \vdash f : B$,
  including $x_1 : A_1, \ldots, x_n : A_n \vdash x_i : A_i$ for $i = 1, \ldots, n$
- a substitution operation

$$\frac{x_1 : A_1, \ldots, x_n : A_n \vdash f : B \qquad (\Delta \vdash g_i : A_i)_{i=1,\ldots,n}}{\Delta \vdash f[g_1, \ldots, g_n] : B}$$

simple

And: a type theory has:

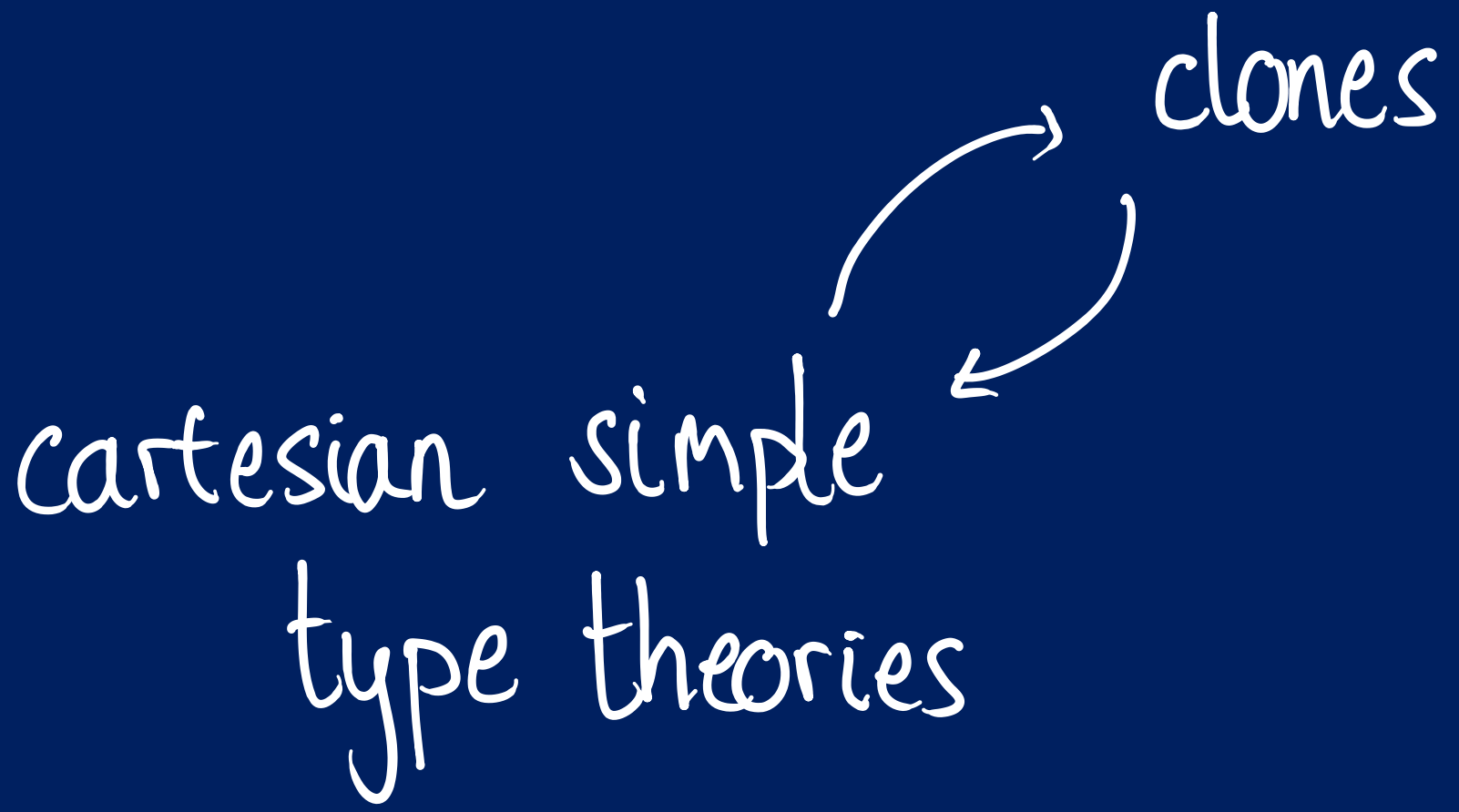
- types $A, B, C, \ldots$
- terms $x_1 : A_1, \ldots, x_n : A_n \vdash f : B$,
  including $x_1 : A_1, \ldots, x_n : A_n \vdash x_i : A_i$ for $i = 1, \ldots, n$
- a substitution operation

$$\frac{x_1 : A_1, \ldots, x_n : A_n \vdash f : B \qquad (\Delta \vdash g_i : A_i)_{i=1,\ldots,n}}{\Delta \vdash f[g_1, \ldots, g_n] : B}$$

$x_i[u_1, \ldots, u_n] = u_i$

$t[x_1, \ldots, x_n] = t$

$t[u_1, \ldots, u_n][v_1, \ldots, v_m] = t[\ldots, u_i[v_\cdot], \ldots]$


clones
cartesian simple
type theories

Syntax from semantics

Syntax from semantics

Cartesian product in category $\mathbb{C}$ $=$ Universal arrow from $\Delta^{(n)} : \mathbb{C} \longrightarrow \mathbb{C}^{\times n}$ to $(A_1, \ldots, A_n) \in \mathbb{C}^{\times n}$

Syntax from semantics

Cartesian product in clone $\mathbb{C}$ $\overset{\text{def}}{=}$ Universal arrow from $\Delta^{(n)} : \mathbb{C} \longrightarrow \mathbb{C}^{\times n}$ to $(A_1, \ldots, A_n) \in \mathbb{C}^{\times n}$

Syntax from semantics

Cartesian product in clone $\mathbb{C}$ $\overset{\text{def}}{=}$ Universal arrow from $\Delta^{(n)} : \mathbb{C}^{\times n} \longrightarrow \mathbb{C}$ to $(A_1, \ldots, A_n) \in \mathbb{C}^{\times n}$

for $n = 2$:

$$t \longmapsto (\pi_1(t), \pi_2(t))$$

$$\mathbb{C}(\Gamma; A \times B) \cong \mathbb{C}(\Gamma; A) \times \mathbb{C}(\Gamma; B)$$

$$\langle t_1, t_2 \rangle \longleftarrow\!\shortmid (t_1, t_2)$$

Syntax from semantics

Cartesian product in clone $\mathbb{C}$ = Universal arrow from $\Delta^{(n)} : \mathbb{C} \longrightarrow \mathbb{C}^{\times n}$ to $(A_1, \ldots, A_n) \in \mathbb{C}^{\times n}$

free clone with all products = syntax of $\Lambda^{\times}$

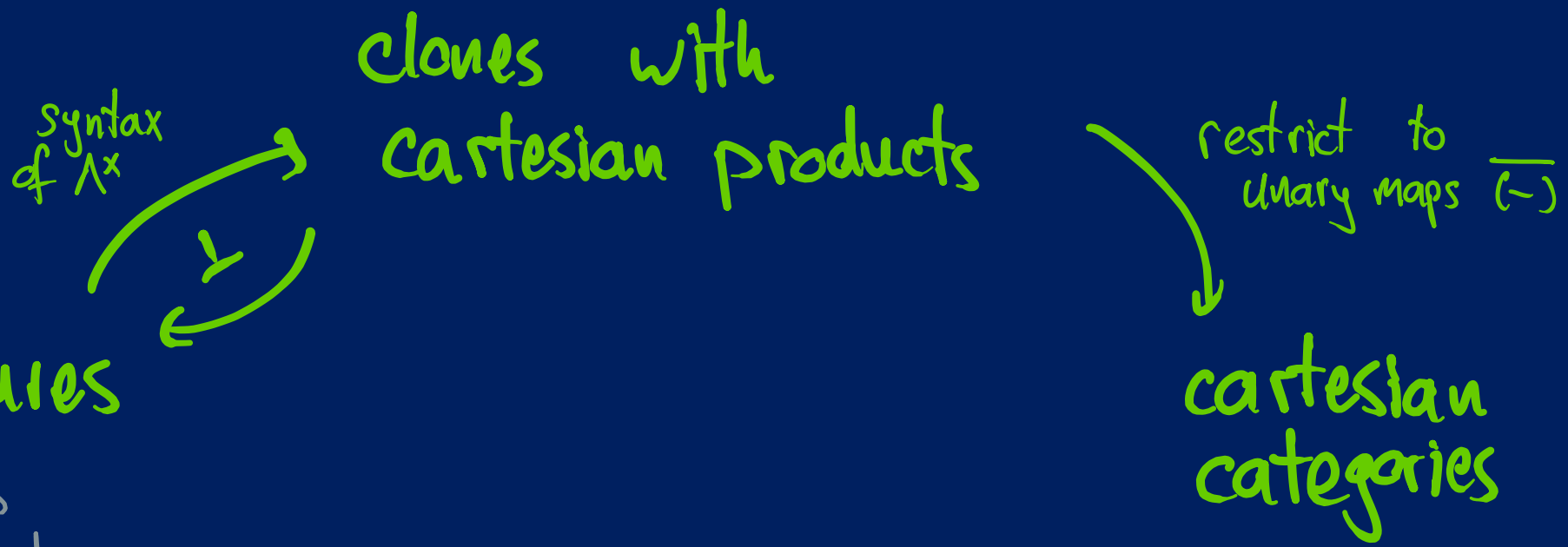
simply-typed λ-calculus with just products

Syntax from semantics

free clone with all products = syntax of $\lambda^\times$ = typed λ-calculus with just products

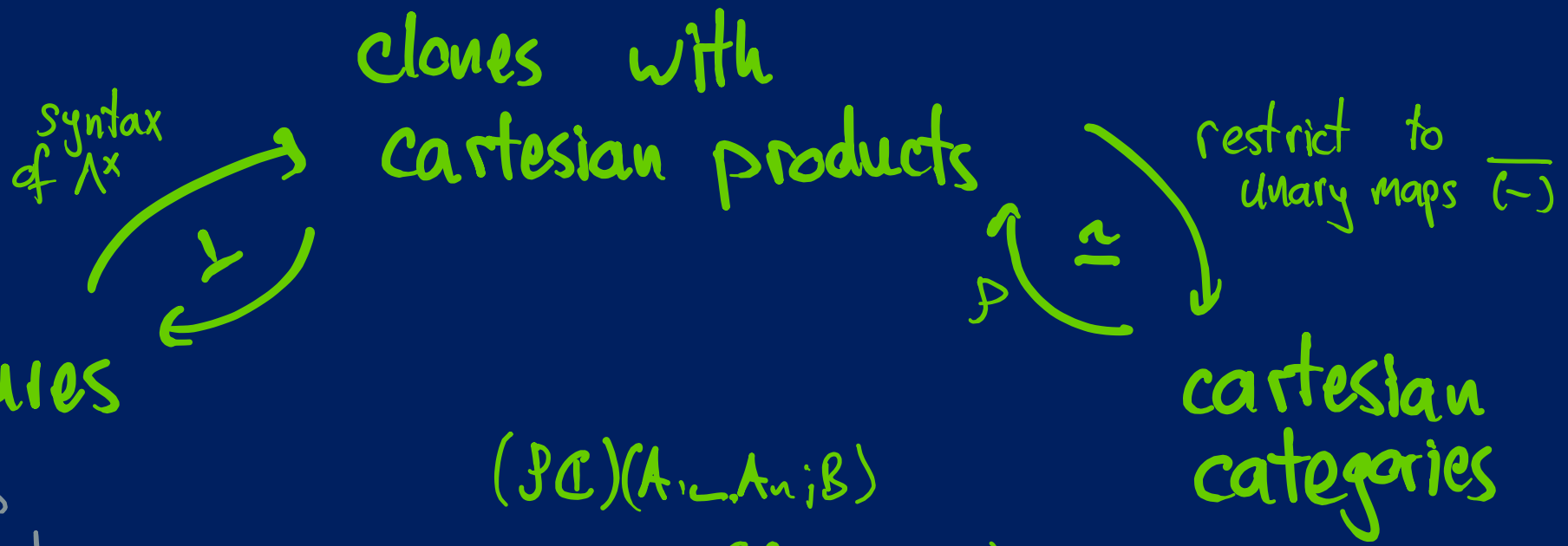
Syntax from semantics

free clone with all products = syntax of $\lambda^\times$ = typed λ-calculus with just products



Syntax from semantics

free clone with all products = syntax of $\lambda^\times$ = typed λ-calculus with just products

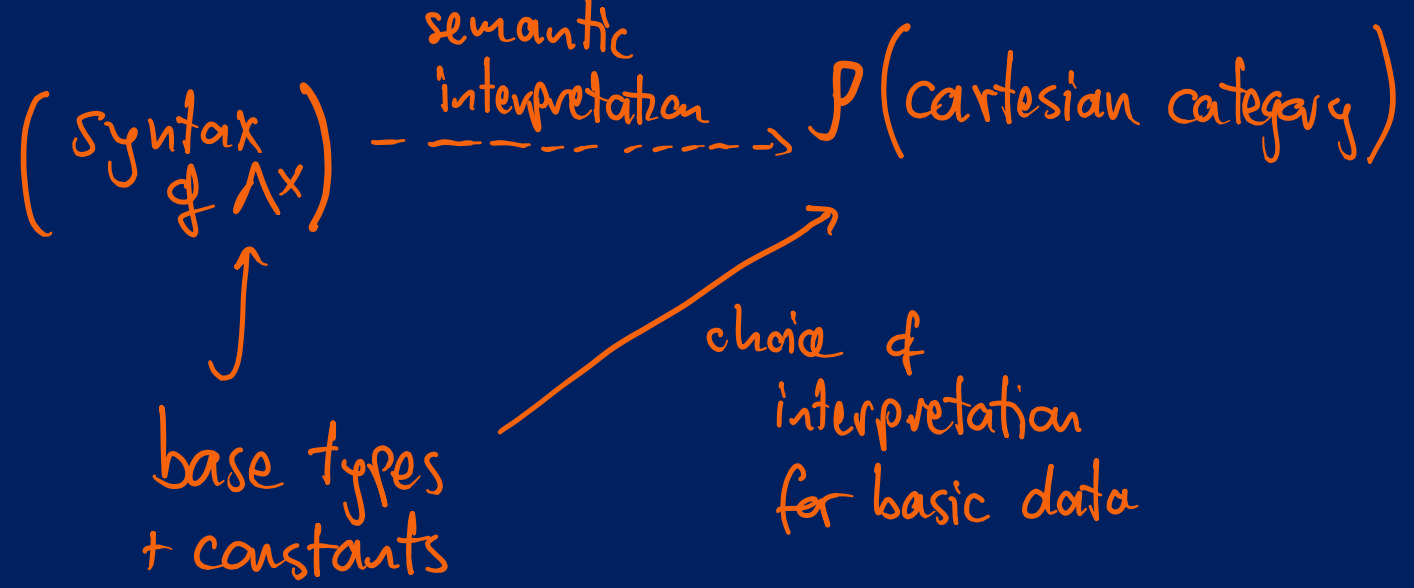


Syntax from semantics

$\Rightarrow$ free cartesian category $=$ $\lambda^\times$-terms $x : A \vdash t : B$

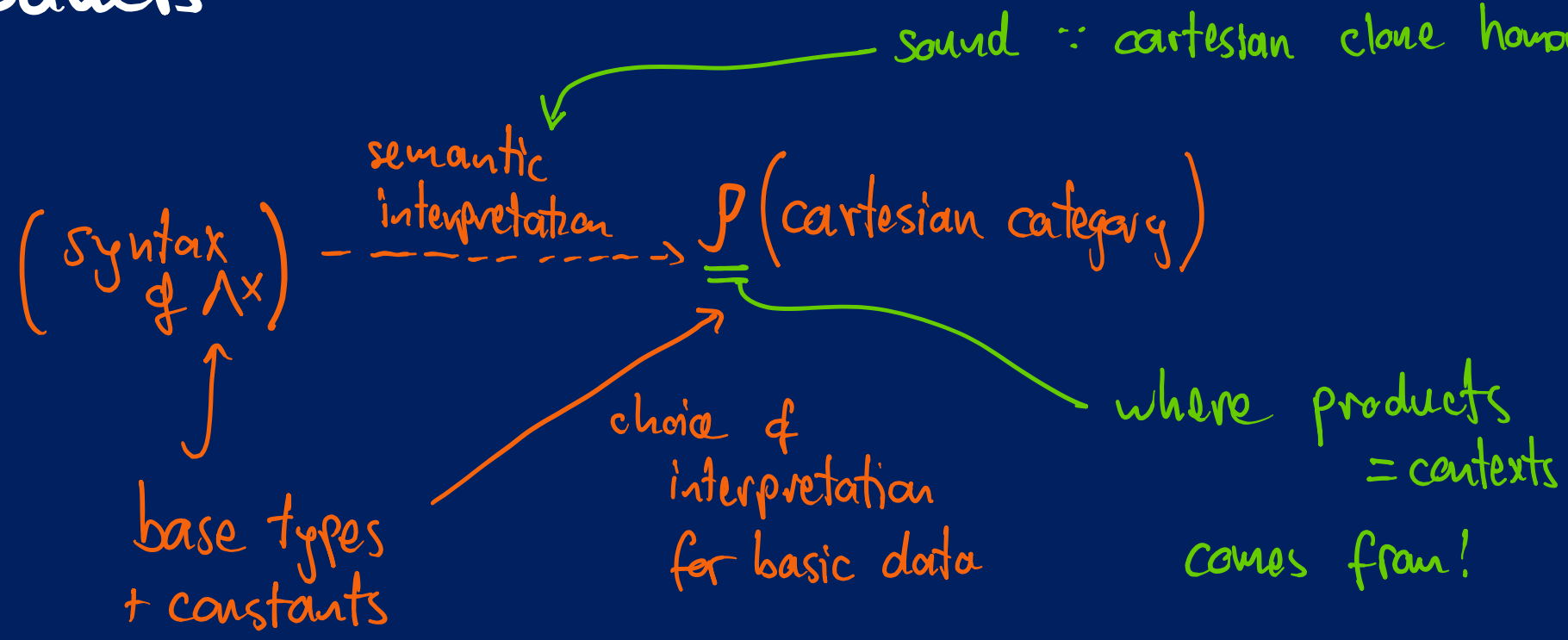
Syntax from semantics

free clone with all products = syntax of $\lambda^\times$ = typed λ-calculus with just products



Syntax from semantics

free clone with all products = syntax of $\lambda^\times$ = typed λ-calculus with just products



Syntax from semantics

objects $A, B, \ldots$
1-cells $f, g : A \to B$
2-cells $\tau : f \Rightarrow g$

a monad in a 2-category $\mathfrak{C}$ consists of:

- an object C
- a 1-cell $T : C \longrightarrow C$
- 2-cells $\eta : id_C \Longrightarrow T$
  $\mu : T \circ T \Longrightarrow T$

+ axioms

monad in 2-category of categories, functors, nat. trans. = usual def" of monad!

Syntax from semantics

[jww. Nayan Rajesh]

instantiating in the 2-category of clones:

monad on $\mathbb{C}$ = a type TA for each type A,
a unit return : $A \longrightarrow TA$,
a bind (>>=) operation

Syntax from semantics

[jww. Nayan Rajesh]

$$\text{monad on } \mathbb{C} \quad = \quad \frac{\Gamma \vdash t : A}{\Gamma \vdash \mathrm{return}(t) : TA}, \quad \frac{\Gamma, x : A \vdash t : TB \qquad \Gamma \vdash u : TA}{\Gamma \vdash \mathrm{let}\ x = u\ \mathrm{in}\ t : TB}$$

...

Syntax from semantics

[jww. Nayan Rajesh]

$$\text{monad on } \mathbb{C} \;=\; \frac{\Gamma \vdash t : A}{\Gamma \vdash \mathrm{return}(t) : TA}\,,\quad \frac{\Gamma, x : A \vdash t : TB \qquad \Gamma \vdash u : TA}{\Gamma \vdash \mathrm{let}\ x = u\ \mathrm{in}\ t : TB}$$

...

free clone equipped with a monad = syntax of monadic Moggi's metalanguage

Syntax from semantics

[jww. Nayan Rajesh]

free clone $\mathbb{C}$ equipped with a monad T = syntax of Moggi's monadic metalanguage

↳ what about strengths?

Syntax from semantics

[jww. Nayan Rajesh]
[see also: Kock, Slattery]

free clone $\mathbb{C}$ equipped with a monad T = syntax of Moggi's monadic metalanguage

if $\mathbb{C}$ has products, T becomes a strong monad on the cartesian category $\bar{\mathbb{C}}$ = restrict $\mathbb{C}$ to unary maps

(linear version: monoidal)

Syntax from semantics

many similar examples: [see especially Shulman et al...]

- (linear) exponential types
- $\otimes$ and $\&$ types in linear λ-calculus [Hyland + de Paiva, ...]
- ⋮

Syntax from semantics

many similar examples:

- (linear) exponential types
- $\otimes$ and & types in linear λ-calculus

conjecture: also can get

- recursive types
- logical relations

Syntax from semantics

many similar examples:

- (linear) exponential types
- $\otimes$ and & types in linear λ-calculus

conjecture: also can get

- recursive types
- **logical** relations ⟿ so a toolkit for reasoning about programs!

Syntax from semantics: a heuristic

1) instantiate the structure in the (2-)category of clones

2) free such clone = required syntax

↝ automatically sound + complete WRT these models

3) (clones) $\rightleftarrows$ (categories) gives categorical semantics

Refined syntax from semantics

1) instantiate the structure in the (2-)category of generalised clones

2) free such clone = required syntax

3) (generalised clones) $\rightleftarrows$ ("categories") gives expected semantics

An example: cartesian closed bicategories

(w/ Fiore)

cartesian closed category

$$\mathbb{C}(X, A \times B) \cong \mathbb{C}(X, A) \times \mathbb{C}(X, B)$$

$$\mathbb{C}(X \times A, B) \cong \mathbb{C}(X, A \multimap B)$$

(β) $\quad f = \mathrm{eval} \circ (\Lambda f \times A)$

(η) $\quad g = \Lambda(\mathrm{eval} \circ (g \times A))$

An example : cartesian closed bicategories

(w/ Fiore)

cartesian closed bicategory

$$\mathbb{C}(X, A \times B) \simeq \mathbb{C}(X, A) \times \mathbb{C}(X, B)$$

hom-categories

$$\mathbb{C}(X \times A, B) \simeq \mathbb{C}(X, A \multimap B)$$

(β) $\quad f \cong \mathrm{eval} \circ (\Lambda f \times A)$

(η) $\quad g \cong \Lambda(\mathrm{eval} \circ (g \times A))$

eg// generalised species

An example : cartesian closed bicategories

(w/ Fiore)

cartesian closed biclone

$$\mathbb{C}(\Gamma; A \times B) \simeq \mathbb{C}(\Gamma; A) \times \mathbb{C}(\Gamma; B)$$

$$\mathbb{C}(\Gamma, A; B) \simeq \mathbb{C}(\Gamma; A \to B)$$

$$\Gamma \vdash \beta : (\lambda x.t)\, u \Rightarrow t\{x \mapsto u\} : B$$

$$\Gamma \vdash \eta : t \Rightarrow \lambda x.(t^{x}\ x) : A \to B$$

An example: cartesian closed bicategories

(w/ Fiore)

cartesian closed biclone

↳ easy to prove soundness + completeness

"correct by construction"

↳ strong justification for design choices

(canonical!)

What's next?

The trend: refinement on both sides

Syntax ⇄ categorical semantics

graded monads

[Melliès, Katsumata, Fujii, Gaboardi, Orchard, ...]

fuzzy syntax

[de Amorim, Hsu, Katsumata, Gaboardi, Cherigui, ...]

cost analysis

[Niu, Sterling, Grodin, Harper, Gaboardi, ...]

2-dimensional

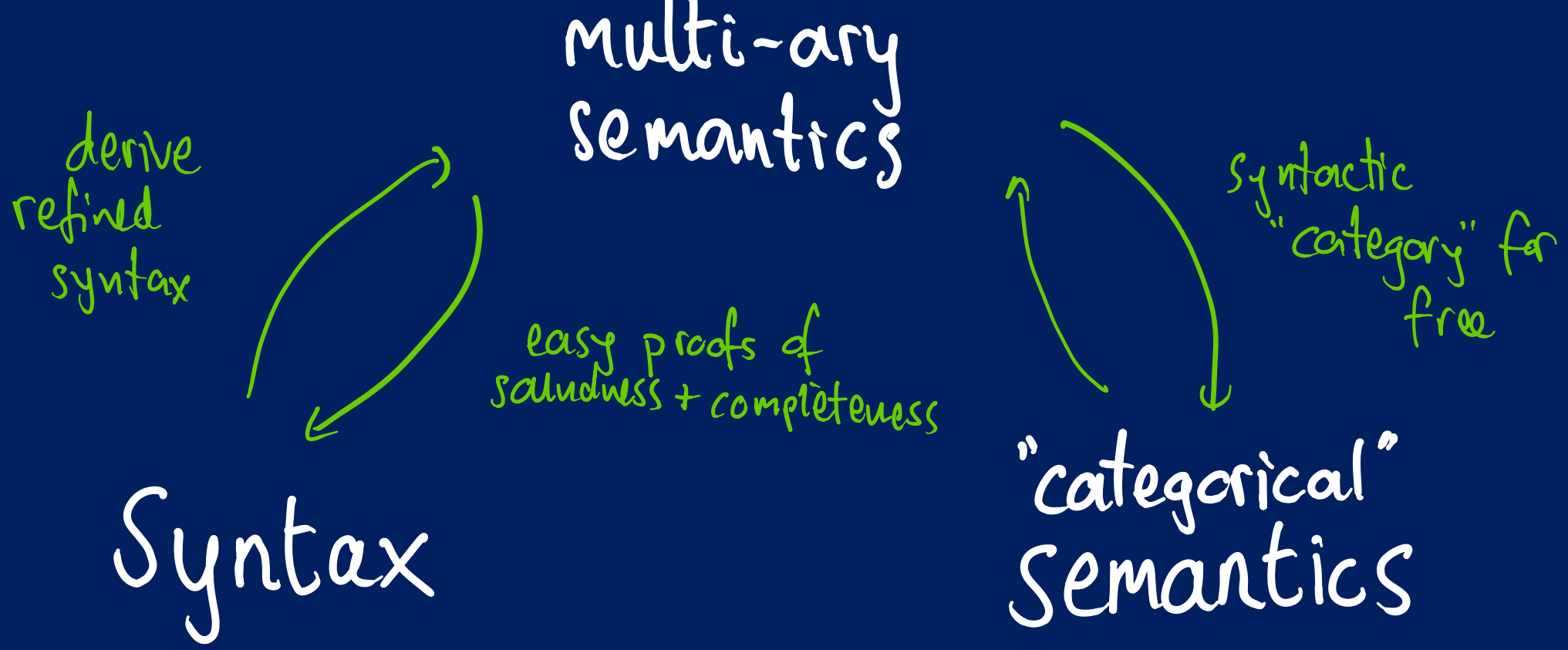
[Fiore, Gambino, Hyland, Winskel, Olimpieri, Paquet, Galal, Melliès, ...]

enrichment

[Kavvos, Levy, McDermott-Uustalu, ...]

NB: an incomplete list!

The future? Refined syntax via multi-ary



WIP: "duoidal enrichment" covers effects, CBPV, graded, ...

[w/ Rajesh]

WIP: "duoidal enrichment" covers effects, CBPV, graded, ...

[w/ Rajesh]

Future work:

- other bases of enrichment (eg. for cost analysis, metaprogramming, ...)
- other syntax from constructions in Clone
  - build a framework for many languages
- enriched clones and enriched Universal algebra

The future? Refined syntax via multi-ary