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:

(typed) Syntax $\rightleftarrows$ (categorical) Semantics

new structure $\rightsquigarrow$ models express new

on programs $\leftsquigarrow$ structures

A long thread:

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

Syntax $\rightleftarrows$ Semantics

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

The trend: refinement on both sides

Syntax $\rightleftarrows$ Semantics

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 $\rightleftarrows$ Semantics

graded monads
[Mellies, 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, Mellies, ...]

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

NB: an incomplete list!

The trend: refinement on both sides

Syntax ⇄ Semantics

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

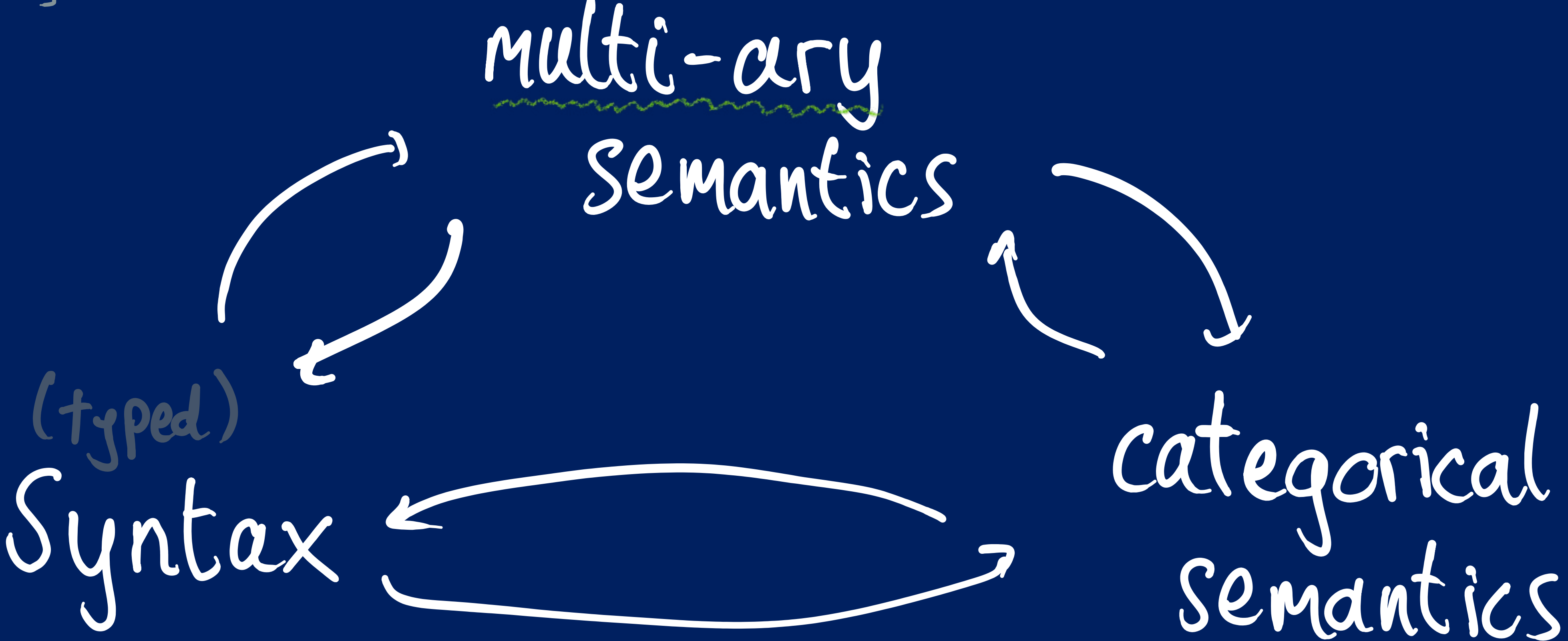
[Lambek, ...]

(typed)
Syntax

categorical
semantics

Looking backwards

[Lambek, ...]



Looking backwards

[Lambek, ...]

MANY INPUTS

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

multi-ary semantics

(typed) Syntax

categorical semantics

ONE INPUT $f : A \longrightarrow B$

Looking backwards

[Lambek, ...]

MANY INPUTS

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

multi-ary semantics

(typed) Syntax

categorical semantics

MANY INPUTS

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

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

muticategories, 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 (multisorted, abstract) 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

<u>def</u>: 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

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

$$p_i[f_1, \ldots, f_n] = f_i$$

$$f[p_1, \ldots, p_n] = f$$

$$(f[g_1, \ldots, g_n])[h_1, \ldots, h_m] = f[\ldots, g_i[h_\bullet], \ldots]$$

And: a type theory has: (simple type theory)

- 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

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

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

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

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

And: a type theory (simple) 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

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

clones

cartesian simple
type theories

Syntax from semantics

Syntax from semantics

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

Syntax from semantics

cartesian product in $\stackrel{\text{def}}{=}$ universal arrow from

clone $\mathbb{C}$ $\quad$ $\Delta^{(n)} : \mathbb{C}^{\times n} \longrightarrow \mathbb{C}$

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 $n = 2$:

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

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

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

Syntax from semantics

cartesian product in **clone** $\mathbb{C}$ = universal arrow from $\Delta^{(n)} : \mathbb{C}^{\times n} \longrightarrow \mathbb{C}$ to $(A_1, \dots, A_n) \in \mathbb{C}^{\times n}$

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

$$
\begin{tikzcd}
\text{signatures} \arrow[r, bend left, "\text{syntax of } \Lambda^{\times}"] \arrow[r, phantom, "\perp"] & \text{clones with cartesian products} \arrow[l, bend left]
\end{tikzcd}
$$

signatures = base types + constants

Syntax from semantics

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

$$
\begin{tikzcd}
& \text{clones with cartesian products} \arrow[dl, bend left] \arrow[dr, "\text{restrict to unary maps } \overline{(-)}"] & \\
\text{signatures} \arrow[ur, bend left, "\text{syntax of } \Lambda^{\times}"] \arrow[ur, phantom, "\perp"] & & \text{cartesian categories}
\end{tikzcd}
$$

signatures $=$ base types $+$ constants

Syntax from semantics

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

$$\begin{tikzcd}[column sep=large]
\text{signatures} \arrow[r, bend left, "\text{syntax of } \Lambda^{\times}"] \arrow[r, phantom, "\perp"] & \text{clones with cartesian products} \arrow[l, bend left] \arrow[r, bend left, "\text{restrict to unary maps } \overline{(-)}"] \arrow[r, phantom, "\simeq"] & \text{cartesian categories} \arrow[l, bend left, "\mathcal{P}"]
\end{tikzcd}$$

signatures $=$ base types $+$ constants

$$(\mathcal{P}\mathbb{C})(A_1, \dots, A_n; B) := \mathbb{C}\left(\textstyle\prod_{i=1}^{n} A_i ; B\right)$$

Syntax from semantics

$$
\begin{tikzcd}
 & \text{clones with cartesian products} \arrow[dl, bend left, "\perp"] \arrow[dr, bend left, "\text{restrict to unary maps}"] & \\
\text{signatures} \arrow[ur, bend left, "\text{syntax } \& \ \Lambda^{\times}"] & & \text{cartesian categories} \arrow[ul, bend left, "\overline{(-)}", "\simeq"']
\end{tikzcd}
$$

= base types + constants

$\Rightarrow$ 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

$$
\begin{tikzcd}
\left(\text{syntax of } \Lambda^{\times}\right) \arrow[r, dashed, "\text{semantic interpretation}"] & \mathcal{P}(\text{cartesian category}) \\
\text{base types + constants} \arrow[u] \arrow[ur, "\text{choice of interpretation for basic data}"'] &
\end{tikzcd}
$$

Syntax from semantics

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

sound :: cartesian clone homomorphism

semantic interpretation

$$\left(\text{syntax of } \Lambda^{\times}\right) \dashrightarrow \mathcal{P}\left(\text{cartesian category}\right)$$

base types + constants

choice of interpretation for basic data

where products = contexts comes from!

Syntax from semantics

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

a <u>monad</u> in a 2-category $\mathcal{C}$
consists of:

- an object c
- a 1-cell $T : c \longrightarrow c$
- 2-cells $\eta : \mathrm{id}_c \Longrightarrow T$

  $\mu : T \circ T \Longrightarrow T$

+ axioms

monad in 2-category of categories, functors, nat. trans. = usual def^n 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 $\mathrm{return} : A \to TA$,
a bind operation
(>>=)

Syntax from semantics

[jww. Nayan Rajesh]

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

...

Syntax from semantics

[jww. Nayan Rajesh]

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

...

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

[see especially Shulman et al...]

many similar examples:

- (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 $\rightsquigarrow$ 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) $(\text{clones}) \rightleftarrows (\text{categories})$ 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) $\left(\begin{array}{c}\text{generalised}\\\text{clones}\end{array}\right) \rightleftarrows$ ("categories") gives expected semantics

An example: cartesian closed bicategories

(w/ Fiore)

cartesian closed category

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

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

(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

hom-categories

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

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

eg// generalised species

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

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

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

$$\text{Syntax} \rightleftarrows \text{categorical semantics}$$

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

multi-ary semantics

derive refined syntax

easy proofs of soundness + completeness

syntactic "category" for free

Syntax

"categorical" Semantics

<u>WIP</u>: "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

multi-ary semantics

derive refined syntax

easy proofs of soundness + completeness

syntactic "category" for free

Syntax

"categorical" semantics