Synthesising a type theory for cartesian closed bicategories

Marcelo Fiore[†] and Philip Saville[◇]

[†]University of Cambridge
Department of Computer Science and Technology

[◇]University of Edinburgh
School of Informatics

Aim of this work:

A principled construction of a type theory
for cartesian closed bicategories (= CCCs up-to-isomorphism)

Aim of this work:

A principled construction of a type theory
for cartesian closed bicategories (= CCCs up-to-isomorphism)

Originally motivated by **coherence** but interesting in its right!

Aim of this work:

A principled construction of a type theory
for cartesian closed bicategories (= CCCs up-to-isomorphism)

Aim of this work:

A principled construction of an internal language
for cartesian closed bicategories (= CCCs up-to-isomorphism)

Aim of this work:

A principled construction of an internal language for cartesian closed bicategories (= CCCs up-to-isomorphism)

What is an internal language?

Aim of this work:

A principled construction of an internal language for cartesian closed bicategories (= CCCs up-to-isomorphism)

What is an internal language? generally an *informal* term

Aim of this work:

A principled construction of an internal language for cartesian closed bicategories (= CCCs up-to-isomorphism)

What is an internal language? generally an *informal* term

A syntax to describe some semantic structure

$$\text{STLC} \leftrightsquigarrow \text{CCCs}$$

$$\text{string diagrams} \leftrightsquigarrow \text{PROPs}$$

$$\text{MLTT} \leftrightsquigarrow \text{LCCCs}$$

Aim of this work:

A principled construction of an internal language for cartesian closed bicategories (= CCCs up-to-isomorphism)

What is an internal language? generally an *informal* term

A syntax to describe some semantic structure

Typically, a tool for reasoning (*e.g.* less coherence data)

Aim of this work:

A principled construction of an internal language
for cartesian closed bicategories (= CCCs up-to-isomorphism)

What is an internal language? generally an *informal* term

A syntax to describe some semantic structure

Typically, a tool for reasoning (*e.g.* less coherence data)

Language should be sound and complete

Aim of this work:

A principled construction of an internal language
for cartesian closed bicategories (= CCCs up-to-isomorphism)

What is an internal language? generally an *informal* term

A syntax to describe some semantic structure

Typically, a tool for reasoning (*e.g.* less coherence data)

Language should be sound and complete

What is principled?

No arbitrary choices

Based on analysis of algebraic structure

Parallel situation for cartesian closed categories

Aim of this work:

A principled construction of an internal language
for cartesian closed bicategories (= CCCs up-to-isomorphism)

What is an internal language? generally an *informal* term

A syntax to describe some semantic structure

Typically, a tool for reasoning (*e.g.* less coherence data)

Language should be sound and complete

What is principled?

No arbitrary choices

Based on analysis of algebraic structure

Parallel situation for cartesian closed categories

Many benefits! makes life *much* easier

New information about cc-bicats, simpler proofs, new relationships, . . .

Strategy: use tools from universal algebra

Strategy: use tools from universal algebra

1. cc-bicategories = CCCs up-to-isomorphism

Strategy: use tools from universal algebra

1. cc-bicategories = CCCs up-to-isomorphism
2. STLC = internal language of cc-clones (*c.f.* Lambek)

Strategy: use tools from universal algebra

1. cc-bicategories = CCCs up-to-isomorphism
2. STLC = internal language of cc-clones (*c.f.* Lambek)
3. bicategorify: get cc-biclones

Strategy: use tools from universal algebra

1. cc-bicategories = CCCs up-to-isomorphism
2. STLC = internal language of cc-clones (*c.f.* Lambek)
3. bicategorify: get cc-biclones
4. $\Lambda_{\mathrm{ps}}^{\times,\rightarrow}$ = internal language of cc-biclones

Strategy: use tools from universal algebra

1. cc-bicategories = CCCs up-to-isomorphism
2. STLC = internal language of cc-clones (*c.f.* Lambek)
3. bicategorify: get cc-biclones
4. $\Lambda_{\mathrm{ps}}^{\times,\to}$ = internal language of cc-biclones

Today: principles underlying the construction of $\Lambda_{\mathrm{ps}}^{\times,\to}$
$\rightsquigarrow$ focus on STLC

cc-Bicategories

Bicategories

Categories with axioms 'up to isomorphism'.

e.g. profunctors, $\mathrm{Span}(\mathbb{C})$, bicategories of relations, Cat, ...

Composition by universal property ⇒ bicategory

In a category $\mathbb{C}$ with pullbacks:

1. objects: objects of $\mathbb{C}$,
2. 1-cells $A \rightsquigarrow B$: spans $(A \leftarrow X \rightarrow B)$,
3. 2-cells: commutative squares
$$\begin{tikzcd} & X \arrow[dl] \arrow[dr] \arrow[dd, dashed, "h"] & \\ A & & B \\ & Y \arrow[ul] \arrow[ur] & \end{tikzcd}$$

Composition defined by pullback:

$$\begin{tikzcd} & & X \times_B Y \arrow[dl] \arrow[dr] \arrow[dd, phantom, "\lrcorner" rotate=-45, very near start] & & \\ & X \arrow[dl] \arrow[dr] & & Y \arrow[dl] \arrow[dr] & \\ A & & B & & C \end{tikzcd}$$

Composition by universal property $\Rightarrow$ bicategory

In a category $\mathbb{C}$ with pullbacks:

1. objects: objects of $\mathbb{C}$,
2. 1-cells $A \rightsquigarrow B$: spans $(A \leftarrow X \rightarrow B)$,
3. 2-cells: commutative squares
$$\begin{tikzcd} & X \arrow[dl] \arrow[dr] \arrow[d, dashed, "h"] & \\ A & & B \\ & Y \arrow[ul] \arrow[ur] & \end{tikzcd}$$

Composition defined by pullback: $\rightsquigarrow$ associative up to iso

$$\begin{tikzcd} & & X \times_B Y \arrow[dl] \arrow[dr] \arrow[dd, phantom, "\lrcorner" rotate=-45, very near start] & & \\ & X \arrow[dl] \arrow[dr] & & Y \arrow[dl] \arrow[dr] & \\ A & & B & & C \end{tikzcd}$$

Bicategories

- Objects $X \in ob(\mathcal{B})$,

Bicategories

- Objects $X \in ob(\mathcal{B})$,
- *Hom-categories* $\big(\mathcal{B}(X, Y), \bullet, \mathrm{id}\big)$:

Bicategories

- Objects $X \in ob(\mathcal{B})$,
- *Hom-categories* $\big(\mathcal{B}(X, Y), \bullet, \mathrm{id}\big)$:

  1-cells $X \xrightarrow{f} Y$

  2-cells $\begin{tikzcd} X \arrow[r, bend left, "f"{name=U}] \arrow[r, bend right, "f'"'{name=D}] \arrow[phantom, from=U, to=D, "\Downarrow\alpha"] & Y \end{tikzcd}$

- Objects $X \in ob(\mathcal{B})$,
- *Hom-categories* $\big(\mathcal{B}(X,Y), \bullet, \mathrm{id}\big)$:

1-cells $X \xrightarrow{f} Y$

2-cells $$\begin{tikzcd} X \arrow[r, bend left, "f"] \arrow[r, bend right, "f'"'] \arrow[r, phantom, "\Downarrow\alpha"] & Y \end{tikzcd}$$

$$\begin{tikzcd} X \arrow[r, bend left=60, "f"] \arrow[r, "f'" description] \arrow[r, bend right=60, "f''"'] \arrow[r, phantom, bend left=30, "\Downarrow\alpha"] \arrow[r, phantom, bend right=30, "\Downarrow\alpha'"] & Y \end{tikzcd}$$

Bicategories

- Objects $X \in ob(\mathcal{B})$,
- *Hom-categories* $\big(\mathcal{B}(X, Y), \bullet, \mathrm{id}\big)$:

  1-cells $X \xrightarrow{f} Y$

  2-cells
$$\begin{tikzcd} X \arrow[r, bend left, "f"] \arrow[r, bend right, "f'"'] \arrow[r, phantom, "\Downarrow\alpha"] & Y \end{tikzcd}$$
- Identities $\mathrm{Id}_X : X \to X$ and composition

$$\mathcal{B}(Y, Z) \times \mathcal{B}(X, Y) \xrightarrow{\circ_{X,Y,Z}} \mathcal{B}(X, Z)$$

Bicategories

- Objects $X \in ob(\mathcal{B})$,
- *Hom-categories* $\big(\mathcal{B}(X,Y), \bullet, \mathrm{id}\big)$:

  1-cells $X \xrightarrow{f} Y$

  2-cells $\begin{tikzcd} X \arrow[r, bend left, "f"] \arrow[r, bend right, "f'"'] \arrow[r, phantom, "\Downarrow\alpha"] & Y \end{tikzcd}$

- Identities $\mathrm{Id}_X : X \to X$ and composition

$$\mathcal{B}(Y,Z) \times \mathcal{B}(X,Y) \xrightarrow{\circ_{X,Y,Z}} \mathcal{B}(X,Z)$$

$$\begin{tikzcd} X \arrow[r, bend left, "f"] \arrow[r, bend right, "f'"'] \arrow[r, phantom, "\Downarrow\alpha"] & Y \arrow[r, bend left, "g"] \arrow[r, bend right, "g'"'] \arrow[r, phantom, "\Downarrow\beta"] & Z \end{tikzcd}$$

- Objects $X \in ob(\mathcal{B})$,
- *Hom-categories* $\big(\mathcal{B}(X, Y), \bullet, \mathrm{id}\big)$:

  1-cells $X \xrightarrow{f} Y$

  2-cells $\begin{tikzcd} X \arrow[r, bend left, "f"] \arrow[r, bend right, "f'"'] \arrow[r, phantom, "\Downarrow\alpha"] & Y \end{tikzcd}$

- Identities $\mathrm{Id}_X : X \to X$ and composition

$$\mathcal{B}(Y, Z) \times \mathcal{B}(X, Y) \xrightarrow{\circ_{X,Y,Z}} \mathcal{B}(X, Z)$$

- Invertible 2-cells

$$(h \circ g) \circ f \xRightarrow{\mathsf{a}_{h,g,f}} h \circ (g \circ f)$$

$$\mathrm{Id}_X \circ f \xRightarrow{\mathsf{l}_f} f$$

$$g \circ \mathrm{Id}_X \xRightarrow{\mathsf{r}_g} g$$

subject to a triangle law and pentagon law.

Bicategories

- Objects $X \in ob(\mathcal{B})$,
- *Hom-categories* $\big(\mathcal{B}(X, Y), \bullet, \mathrm{id}\big)$:

  1-cells $X \xrightarrow{f} Y$

  2-cells
  $$\begin{tikzcd} X \arrow[r, bend left, "f"] \arrow[r, bend right, "f'"'] \arrow[r, phantom, "\Downarrow\alpha"] & Y \end{tikzcd}$$
- Identities $\mathrm{Id}_X : X \to X$ and composition
$$\mathcal{B}(Y, Z) \times \mathcal{B}(X, Y) \xrightarrow{\circ_{X,Y,Z}} \mathcal{B}(X, Z)$$
- Invertible 2-cells
$$(h \circ g) \circ f \xRightarrow{\mathsf{a}_{h,g,f}} h \circ (g \circ f)$$
$$\mathrm{Id}_X \circ f \xRightarrow{\mathsf{l}_f} f$$
$$g \circ \mathrm{Id}_X \xRightarrow{\mathsf{r}_g} g$$

  subject to a triangle law and pentagon law.

$\rightsquigarrow$ get a 2-category if $\mathsf{a}, \mathsf{l}, r$ all id

Bicategories

Categories with axioms ‘up to isomorphism’.

e.g. profunctors, $\mathrm{Span}(\mathbb{C})$, bicategories of relations, Cat, $\ldots$

Bicategories

Categories with axioms ‘up to isomorphism’.

e.g. profunctors, $\mathrm{Span}(\mathbb{C})$, bicategories of relations, Cat, ...

Cartesian closed bicategories

Cartesian closed categories ‘up to isomorphism’.

Examples:

- Generalised species and cartesian distributors
  bicategorical models of LL, higher category theory
  (Fiore, Gambino, Hyland, Winskel), (Fiore & Joyal)
- Categorical algebra (operads)
  (Gambino & Joyal)
- Game semantics (concurrent games)
  (Yamada & Abramsky, Winskel *et al.*, Paquet)

Cartesian closed bicategories = cc-bicategories

Bicategories $\mathcal{B}$ equipped with families of equivalences

$$\mathcal{B}\left(X, \prod_{i=1}^{n} A_i\right) \quad \simeq \quad \prod_{i=1}^{n} \mathcal{B}(X, A_i)$$

$$\mathcal{B}(X, A \Rightarrow B) \quad \simeq \quad \mathcal{B}(X \times A, B)$$

Not related to Carboni & Walters' "cartesian bicategories"!

Bicategories $\mathcal{B}$ equipped with families of equivalences

$$\begin{tikzcd} \mathcal{B}\big(X, \prod_{i=1}^{n} A_i\big) \arrow[r, bend left, "(\pi_1 \circ -, \ldots, \pi_n \circ -)"] \arrow[r, phantom, "\perp \simeq"] & \prod_{i=1}^{n} \mathcal{B}(X, A_i) \arrow[l, bend left, "\langle -, \ldots, = \rangle"] \end{tikzcd}$$

(tupling)

$$\begin{tikzcd} \mathcal{B}(X, A \Rightarrow B) \arrow[r, bend left, "\mathrm{eval}_{A,B} \circ (- \times A)"] \arrow[r, phantom, "\perp \simeq"] & \mathcal{B}(X \times A, B) \arrow[l, bend left, "\lambda"] \end{tikzcd}$$

(currying)

Bicategories $\mathcal{B}$ equipped with families of equivalences

$$\begin{tikzcd} \mathcal{B}\left(X, \prod_{i=1}^{n} A_i\right) \arrow[r, bend left, "(\pi_1 \circ -, \ldots, \pi_n \circ -)"] \arrow[r, phantom, "\perp\simeq"] & \prod_{i=1}^{n} \mathcal{B}(X, A_i) \arrow[l, bend left, "\langle -, \ldots, = \rangle"] \end{tikzcd}$$

(tupling)

$$\begin{tikzcd} \mathcal{B}(X, A \Rightarrow B) \arrow[r, bend left, "\mathrm{eval}_{A,B} \circ (- \times A)"] \arrow[r, phantom, "\perp\simeq"] & \mathcal{B}(X \times A, B) \arrow[l, bend left, "\lambda"] \end{tikzcd}$$

(currying)

Triangle laws:

$$\begin{tikzcd} & \pi_i \circ \langle \pi_1 \circ t, \ldots, \pi_n \circ t \rangle \arrow[dr, Rightarrow, "\mathrm{counit}"] & \\ \pi_i \circ t \arrow[ur, Rightarrow, "\pi_i \circ \mathrm{unit}"] \arrow[rr, equal] & & \pi_i \circ t \end{tikzcd}$$

Aim of this work:

A principled construction of an internal language
for cartesian closed bicategories (= CCCs up-to-isomorphism)

Aim of this work:

A principled construction of an internal language
for cartesian closed bicategories (= CCCs up-to-isomorphism)

Our starting point:

A principled construction of an internal language
for cartesian closed categories

Aim of this work:

A principled construction of an internal language
for cartesian closed bicategories (= CCCs up-to-isomorphism)

Our starting point: folklore(?), *c.f.* Lambek, Jacobs,...

A principled construction of an internal language
for cartesian closed categories

An internal language for CCCs

Curry–Howard–Lambek correspondence

$$\begin{tikzcd}
\text{simply-typed lambda calculus} \arrow[r, bend left] & \text{cartesian closed categories} \arrow[l, bend left]
\end{tikzcd}$$

Curry–Howard–Lambek correspondence

$$\begin{tikzcd}\text{simply-typed lambda calculus} \arrow[r, bend left] & \text{cartesian closed categories} \arrow[l, bend left]\end{tikzcd}$$

For every graph [= choice of base types and constants] get $\Lambda^{\times,\rightarrow}(\mathrm{G})$,

Curry–Howard–Lambek correspondence

$$\text{simply-typed lambda calculus} \rightleftarrows \text{cartesian closed categories}$$

For every graph [= choice of base types and constants] get $\Lambda^{\times,\rightarrow}(\mathrm{G})$,

...and a CCC $\mathbb{F}^{\times,\rightarrow}[\mathrm{G}]$:

objects: $\Lambda^{\times,\rightarrow}(\mathrm{G})$-types,

maps $A \rightarrow B$: terms $(x : A \vdash t : B)$ mod $\alpha\beta\eta$

composition: substitution

Curry–Howard–Lambek correspondence

$$\begin{tikzcd} \text{simply-typed lambda calculus} \arrow[r, bend left] & \text{cartesian closed categories} \arrow[l, bend left] \end{tikzcd}$$

For every graph [= choice of base types and constants] get $\Lambda^{\times,\to}(\mathrm{G})$,

…and a CCC $\mathbb{F}^{\times,\to}[\mathrm{G}]$:

objects: $\Lambda^{\times,\to}(\mathrm{G})$-types,
maps $A \to B$: terms $(x : A \vdash t : B)$ mod $\alpha\beta\eta$
composition: substitution

with a free property:

$$\begin{tikzcd} \mathbb{F}^{\times,\to}[\mathrm{G}] \arrow[r, dashed, "{[\![-]\!]}"] & \mathbb{C} \\ \mathrm{G} \arrow[u, hook] \arrow[ur, "h"'] & \end{tikzcd}$$

Curry–Howard–Lambek correspondence

$$\begin{tikzcd} \text{simply-typed lambda calculus} \arrow[r, bend left] & \text{cartesian closed categories} \arrow[l, bend left] \end{tikzcd}$$

For every graph [= choice of base types and constants] get $\Lambda^{\times,\to}(\mathrm{G})$,

... and a CCC $\mathbb{F}^{\times,\to}[\mathrm{G}]$:

objects: $\Lambda^{\times,\to}(\mathrm{G})$-types,
maps $A \to B$: terms $(x : A \vdash t : B)$ mod $\alpha\beta\eta$
composition: substitution

with a free property:

$$\begin{tikzcd} \mathbb{F}^{\times,\to}[\mathrm{G}] \arrow[r, dashed, "{[\![-]\!]}"] & \mathbb{C} \\ \mathrm{G} \arrow[u, hook] \arrow[ur, "h"'] & \end{tikzcd}$$

$\rightsquigarrow$ $t =_{\beta\eta} t' \iff [\![t]\!] = [\![t']\!]$ for every $\mathbb{C}$

Curry–Howard–Lambek correspondence

$$\begin{tikzcd} \mathrm{Graph} \arrow[rr, bend left, "\mathbb{F}^{\times,\to}[-]"] \arrow[rr, phantom, "\perp"] & & \mathsf{CCCat} \arrow[ll, bend left, "\mathsf{forget}"] \end{tikzcd}$$

For every graph [= choice of base types and constants] get $\Lambda^{\times,\to}(\mathrm{G})$,

...and a CCC $\mathbb{F}^{\times,\to}[\mathrm{G}]$:

objects: $\Lambda^{\times,\to}(\mathrm{G})$-types,
maps $A \to B$: terms $(x : A \vdash t : B)$ mod $\alpha\beta\eta$
composition: substitution

with a free property:

$$\begin{tikzcd} \mathbb{F}^{\times,\to}[\mathrm{G}] \arrow[r, dashed, "{[\![-]\!]}"] & \mathbb{C} \\ \mathrm{G} \arrow[u, hook] \arrow[ur, "h"'] & \end{tikzcd}$$

$\rightsquigarrow$ $t =_{\beta\eta} t' \iff [\![t]\!] = [\![t']\!]$ for every $\mathbb{C}$

How good a fit is it, really?

Categories:
 1-input, 1-output

$\overset{?}{\leftrightsquigarrow}$

Typed terms
$(x_1 : A_1, \ldots, x_n : A_n \vdash t : B)$
have n inputs

How good a fit is it, really?

Categories:
1-input, 1-output

$\overset{?}{\leftrightsquigarrow}$

Typed terms
$(x_1 : A_1, \ldots, x_n : A_n \vdash t : B)$
have n inputs

Forced to restrict to unary contexts

How good a fit is it, really?

Categories:
1-input, 1-output $\overset{?}{\leftrightsquigarrow}$ Typed terms $(x_1 : A_1, \ldots, x_n : A_n \vdash t : B)$ have n inputs

Forced to restrict to unary contexts

Relying on the fact

$$[\![x_1 : A_1, \ldots, x_n : A_n]\!] \cong [\![p : \textstyle\prod_n(A_1, \ldots, A_n)]\!]$$

How good a fit is it, really?

Categories:
1-input, 1-output

$\overset{?}{\leftrightsquigarrow}$

Typed terms
$(x_1 : A_1, \ldots, x_n : A_n \vdash t : B)$
have n inputs

Forced to restrict to unary contexts

Relying on the fact

$$\begin{aligned} [\![x_1 : A_1, \ldots, x_n : A_n]\!] &\cong [\![p : \textstyle\prod_n(A_1, \ldots, A_n)]\!] \\ &\cong [\![\textstyle\prod_n(A_1, \ldots, A_n)]\!] \end{aligned}$$

How good a fit is it, really?

Categories:
1-input, 1-output

$\overset{?}{\leftrightsquigarrow}$

Typed terms
$(x_1 : A_1, \ldots, x_n : A_n \vdash t : B)$
have n inputs

Forced to restrict to unary contexts

Relying on the fact

$$\begin{aligned}[\![x_1 : A_1, \ldots, x_n : A_n]\!] &\cong [\![p : \textstyle\prod_n(A_1, \ldots, A_n)]\!] \\ &\cong [\![\textstyle\prod_n(A_1, \ldots, A_n)]\!]\end{aligned}$$

$\rightsquigarrow$ are these the same? Why?

How good a fit is it, really?

Categories:
1-input, 1-output

$\overset{?}{\leftrightsquigarrow}$

Typed terms
$(x_1 : A_1, \ldots, x_n : A_n \vdash t : B)$
have n inputs

Forced to restrict to unary contexts

Relying on the fact

$$\begin{aligned} [\![x_1 : A_1, \ldots, x_n : A_n]\!] &\cong [\![p : \textstyle\prod_n(A_1, \ldots, A_n)]\!] \\ &\cong [\![\textstyle\prod_n(A_1, \ldots, A_n)]\!] \end{aligned}$$

$\rightsquigarrow$ are these the same? Why?

Can have type-theoretic exponentials without products

How good a fit is it, really?

Categories:
1-input, 1-output $\overset{?}{\leftrightsquigarrow}$ Typed terms
$(x_1 : A_1, \ldots, x_n : A_n \vdash t : B)$
have n inputs

Forced to restrict to unary contexts

Relying on the fact

$$\begin{aligned} [\![x_1 : A_1, \ldots, x_n : A_n]\!] &\cong [\![p : \textstyle\prod_n(A_1, \ldots, A_n)]\!] \\ &\cong [\![\textstyle\prod_n(A_1, \ldots, A_n)]\!] \end{aligned}$$

$\rightsquigarrow$ are these the same? Why?

Can have type-theoretic exponentials without products

What if we take many-input behaviour seriously?

What if we take many-input behaviour seriously?

What if we take many-input behaviour seriously?

$$\begin{tikzcd} \mathrm{Graph} \arrow[rr, bend left, "\mathbb{F}^{\times,\to}[-]"] \arrow[rr, phantom, "\perp"] && \mathsf{CCCat} \arrow[ll, bend left, "U"] \end{tikzcd}$$

What if we take many-input behaviour seriously?

$$
\begin{tikzcd}
& ? \arrow[dl, bend left, "\text{forget}"] \arrow[dr, bend left, "\text{restrict to one input}"] & \\
\mathrm{Graph} \arrow[ur, bend left, "\text{free}"] \arrow[ur, phantom, "\perp"] \arrow[rr, bend left, "\mathbb{F}^{\times,\to}[-]"] \arrow[rr, phantom, "\perp"] & & \mathrm{CCCat} \arrow[ul, bend left] \arrow[ul, phantom, "\perp"] \arrow[ll, bend left, "U"]
\end{tikzcd}
$$

What if we take many-input behaviour seriously?

$$
\begin{tikzcd}
 & \text{cc-clones} \arrow[dl, bend left=15, "\text{forget}"] \arrow[dr, bend left=15, "\text{restrict to one input}"] & \\
\mathrm{Graph} \arrow[ur, bend left=15, "\text{free}"] \arrow[rr, bend left=10, "\mathbb{F}^{\times,\to}[-]"] \arrow[ur, phantom, "\perp"] \arrow[rr, phantom, "\perp"] & & \mathsf{CCCat} \arrow[ul, bend left=15] \arrow[ll, bend left=10, "U"] \arrow[ul, phantom, "\perp"]
\end{tikzcd}
$$

What if we take many-input behaviour seriously?

$$
\begin{tikzcd}
 & \textcolor{blue}{\text{cc-clones}} \arrow[dl, bend left, "\text{forget}"] \arrow[dr, bend left, "\text{restrict to one input}"] & \\
\mathrm{Graph} \arrow[ur, bend left, "\text{free}"] \arrow[ur, phantom, "\perp"] \arrow[rr, bend left, "\mathbb{F}^{\times,\to}[-]"] \arrow[rr, phantom, "\perp"] & & \mathsf{CCCat} \arrow[ul, bend left] \arrow[ul, phantom, "\perp"] \arrow[ll, bend left, "U"]
\end{tikzcd}
$$

free CCC on G

$\wr\Vert$

free cc-clone on G

and restricting to one input

What if we take many-input behaviour seriously?

$$
\begin{tikzcd}
 & \text{cc-clones} \arrow[dl, bend left, "\text{forget}"] \arrow[dr, bend left, "\text{restrict to one input}"] & \\
\mathrm{Graph} \arrow[ur, bend left, "\text{free}"] \arrow[ur, phantom, "\perp"] \arrow[rr, bend left, "\mathbb{F}^{\times,\to}[-]"] \arrow[rr, phantom, "\perp"] & & \mathrm{CCCat} \arrow[ul, bend left] \arrow[ul, phantom, "\perp"] \arrow[ll, bend left, "U"]
\end{tikzcd}
$$

giving a syntax for CCCs

||

giving a syntax for cc-clones

and restricting to unary contexts

Defining an internal language for categories

Defining an internal language for categories

1. Observe clones [Hall, '81] factor the adjunction:

$$
\begin{tikzcd}
 & \text{clones} \arrow[dl, bend left, "\text{forget}"] \arrow[dr, bend left, "\text{restrict to one input}"] & \\
\text{Graph} \arrow[ur, bend left, "\text{free}"] \arrow[ur, phantom, "\perp"] \arrow[rr, bend left, "\text{free}"] & & \text{Cat} \arrow[ul, bend left] \arrow[ul, phantom, "\perp"] \arrow[ll, bend left, "\text{forget}"] \arrow[ll, phantom, "\perp"]
\end{tikzcd}
$$

Defining an internal language for categories

1. Observe clones [Hall, '81] factor the adjunction:

$$\begin{tikzcd}
 & \text{clones} \arrow[dl, bend left, "\text{forget}"] \arrow[dr, bend left, "\text{restrict to one input}"] & \\
\mathrm{Graph} \arrow[ur, bend left, "\text{free}"] \arrow[ur, phantom, "\perp"] \arrow[rr, bend left, "\text{free}"] \arrow[rr, phantom, "\perp"] & & \mathrm{Cat} \arrow[ul, bend left] \arrow[ul, phantom, "\perp"] \arrow[ll, bend left, "\text{forget}"]
\end{tikzcd}$$

2. Define internal language of a clone,

1. Observe clones [Hall, '81] factor the adjunction:

$$
\begin{tikzcd}
 & \text{clones} \arrow[dl, bend left, "\text{forget}"] \arrow[dr, bend left, "\text{restrict to one input}"] & \\
\mathrm{Graph} \arrow[ur, bend left, "\text{free}"] \arrow[ur, phantom, "\perp"] \arrow[rr, bend left, "\text{free}"] \arrow[rr, phantom, "\perp"] & & \mathrm{Cat} \arrow[ll, bend left, "\text{forget}"] \arrow[ul, bend left] \arrow[ul, phantom, "\perp"]
\end{tikzcd}
$$

2. Define internal language of a clone,
3. Internal language of categories

   $\overset{\text{def.}}{=}$ internal language of free clone,
   restricted to unary contexts

Abstract clone $(S, \mathbb{C})$ = abstract theory of substitution:

Abstract clone $(S, \mathbb{C})$ = abstract theory of substitution:

- *Sorts* S,

Abstract clone $(S, \mathbb{C})$ = abstract theory of substitution:

- *Sorts* S,
- Hom-sets $\mathbb{C}(X_1, \ldots, X_n; Y)$ of *operations* $X_1, \ldots, X_n \xrightarrow{t} Y$,

Abstract clone $(S, \mathbb{C})$ = abstract theory of substitution:

- *Sorts* S,
- Hom-sets $\mathbb{C}(X_1, \ldots, X_n; Y)$ of *operations* $X_1, \ldots, X_n \xrightarrow{t} Y$,
- *Projections* $X_1, \ldots, X_n \xrightarrow{\mathsf{p}^{(i)}_{X_1, \ldots, X_n}} X_i$ $(1 \leqslant i \leqslant n)$,

Abstract clone $(S, \mathbb{C})$ = abstract theory of substitution:

- *Sorts* S,
- Hom-sets $\mathbb{C}(X_1, \ldots, X_n; Y)$ of *operations* $X_1, \ldots, X_n \xrightarrow{t} Y$,
- *Projections* $X_1, \ldots, X_n \xrightarrow{\mathsf{p}^{(i)}_{X_1,\ldots,X_n}} X_i \ (1 \leqslant i \leqslant n)$,
- *Substitution* mappings

$$\begin{gathered} \mathbb{C}(X_1, \ldots, X_n; Y) \times \prod_{i=1}^{n} \mathbb{C}(\Gamma; X_i) \to \mathbb{C}(\Gamma; Y) \\ t, (u_1, \ldots, u_n) \mapsto t[u_1, \ldots, u_n] \end{gathered}$$

Abstract clone $(S, \mathbb{C})$ = abstract theory of substitution:

- *Sorts* S,
- Hom-sets $\mathbb{C}(X_1, \ldots, X_n; Y)$ of *operations* $X_1, \ldots, X_n \xrightarrow{t} Y$,
- *Projections* $X_1, \ldots, X_n \xrightarrow{\mathsf{p}^{(i)}_{X_1,\ldots,X_n}} X_i \; (1 \leqslant i \leqslant n)$,
- *Substitution* mappings

$$\begin{gathered} \mathbb{C}(X_1, \ldots, X_n; Y) \times \prod_{i=1}^{n} \mathbb{C}(\Gamma; X_i) \to \mathbb{C}(\Gamma; Y) \\ t, (u_1, \ldots, u_n) \mapsto t[u_1, \ldots, u_n] \end{gathered}$$

such that

$$\begin{aligned} \mathsf{p}^{(k)}[u_1, \ldots, u_n] &= u_k \qquad\qquad (1 \leqslant k \leqslant n) \\ t[\mathsf{p}^{(1)}, \ldots, \mathsf{p}^{(n)}] &= t \\ t\,[u_\bullet]\,[v_\bullet] &= t[u_\bullet[v_\bullet]] \end{aligned}$$

Abstract clone $(S, \mathbb{C})$ = abstract theory of substitution:

- *Sorts* S,
- Hom-sets $\mathbb{C}(X_1, \ldots, X_n; Y)$ of *operations* $X_1, \ldots, X_n \xrightarrow{t} Y$,
- *Projections* $X_1, \ldots, X_n \xrightarrow{\mathsf{p}^{(i)}_{X_1,\ldots,X_n}} X_i$ $(1 \leqslant i \leqslant n)$,
- *Substitution* mappings

$$\begin{gathered}\mathbb{C}(X_1, \ldots, X_n; Y) \times \prod_{i=1}^{n} \mathbb{C}(\Gamma; X_i) \to \mathbb{C}(\Gamma; Y) \\ t, (u_1, \ldots, u_n) \mapsto t[u_1, \ldots, u_n]\end{gathered}$$

such that

$$\begin{aligned} \mathsf{p}^{(k)}[u_1, \ldots, u_n] &= u_k \qquad\qquad (1 \leqslant k \leqslant n) \\ t[\mathsf{p}^{(1)}, \ldots, \mathsf{p}^{(n)}] &= t \\ t\,[u_\bullet]\,[v_\bullet] &= t[u_\bullet[v_\bullet]] \end{aligned}$$

Every clone $(S, \mathbb{C})$ defines a category $\bar{\mathbb{C}}$ and a multicategory $\mathbb{M}\mathbb{C}$

Every clone $(S, \mathbb{C})$ has an internal language [*c.f.* Lambek]:

$$(x_1 : A_1, \ldots, x_n : A_n \vdash t : B) \iff t : A_1, \ldots, A_n \to B$$

Every clone $(S, \mathbb{C})$ has an internal language [*c.f.* Lambek]:

$$(x_1 : A_1, \ldots, x_n : A_n \vdash t : B) \iff t : A_1, \ldots, A_n \to B$$

$$(x_1 : A_1, \ldots, x_n : A_n \vdash x_i : A_i) \leftrightsquigarrow \mathrm{p}^{(i)} : A_1, \ldots, A_n \to A_i$$

$$t[x_i \mapsto u_i] \leftrightsquigarrow t[u_1, \ldots, u_n]$$

Every clone $(S, \mathbb{C})$ has an internal language [*c.f.* Lambek]:

$$(x_1 : A_1, \dots, x_n : A_n \vdash t : B) \iff t : A_1, \dots, A_n \to B$$

$$(x_1 : A_1, \dots, x_n : A_n \vdash x_i : A_i) \leftrightsquigarrow \mathsf{p}^{(i)} : A_1, \dots, A_n \to A_i$$

$$t[x_i \mapsto u_i] \leftrightsquigarrow t[u_1, \dots, u_n]$$

The clone axioms become:

$$\begin{aligned} x_k[x_i \mapsto u_i] &= u_k \qquad\qquad (1 \leqslant k \leqslant n) \\ t[x_i \mapsto x_i] &= t \\ t[x_i \mapsto u_i][y_j \mapsto v_j] &= t[x_i \mapsto u_i[y_j \mapsto v_j]] \end{aligned}$$

Every clone $(S, \mathbb{C})$ has an internal language [*c.f.* Lambek]:

$$(x_1 : A_1, \ldots, x_n : A_n \vdash t : B) \iff t : A_1, \ldots, A_n \to B$$

$$(x_1 : A_1, \ldots, x_n : A_n \vdash x_i : A_i) \leftrightsquigarrow \mathsf{p}^{(i)} : A_1, \ldots, A_n \to A_i$$

$$t[x_i \mapsto u_i] \leftrightsquigarrow t[u_1, \ldots, u_n]$$

Free clone on a graph G:

$$\frac{c \in \mathrm{G}(A, B)}{c : A \to B} \qquad \frac{}{\mathsf{p}^{(i)} : A_1, \ldots, A_n \to A_i}$$

$$\frac{t : A_1, \ldots, A_n \to B \qquad (u_i : \Delta \to A_i)_{i=1,\ldots,n}}{t[u_1, \ldots, u_n] : \Delta \to B}$$

Every clone $(S, \mathbb{C})$ has an internal language [*c.f.* Lambek]:

$$(x_1 : A_1, \ldots, x_n : A_n \vdash t : B) \iff t : A_1, \ldots, A_n \to B$$

$$(x_1 : A_1, \ldots, x_n : A_n \vdash x_i : A_i) \leftrightsquigarrow \mathsf{p}^{(i)} : A_1, \ldots, A_n \to A_i$$

$$t[x_i \mapsto u_i] \leftrightsquigarrow t[u_1, \ldots, u_n]$$

Free clone on a graph $\mathcal{G}$:

$$\frac{c \in \mathcal{G}(A, B)}{x : A \vdash c : B}\ \text{const} \qquad \frac{}{x_1 : A_1, \ldots, x_n : A_n \vdash x_i : A_i}\ \text{var}$$

$$\frac{x_1 : A_1, \ldots, x_n : A_n \vdash t : B \qquad (\Delta \vdash u_i : A_i)_{i=1,\ldots,n}}{\Delta \vdash t[x_i \mapsto u_i] : B}\ \text{subst}$$

free category on G

$\cong$ free clone on G, restricted to unary maps

$$
\begin{tikzcd}
& \text{clones} \arrow[dl, bend left, "\text{forget}"] \arrow[dr, bend left, "\text{restrict to unary maps}"] & \\
\mathrm{Graph} \arrow[ur, bend left, "\text{free}"] \arrow[ur, phantom, "\perp"] \arrow[rr, bend left, "\text{free}"] \arrow[rr, phantom, "\perp"] & & \mathrm{Cat} \arrow[ul, bend left] \arrow[ul, phantom, "\perp"] \arrow[ll, bend left, "\text{forget}"]
\end{tikzcd}
$$

free category on G

$\cong$ free clone on G, restricted to unary maps

$$
\begin{tikzcd}
 & \text{clones} \arrow[dl, bend left, "\text{forget}"] \arrow[dr, bend left, "\text{restrict to unary maps}"] & \\
\mathrm{Graph} \arrow[ur, bend left, "\text{free}"] \arrow[ur, phantom, "\perp"] \arrow[rr, bend left, "\text{free}"] \arrow[rr, phantom, "\perp"] & & \mathrm{Cat} \arrow[ul, bend left] \arrow[ul, phantom, "\perp"] \arrow[ll, bend left, "\text{forget}"]
\end{tikzcd}
$$

giving a syntax for categories

||

giving a syntax for clones

and restricting to unary contexts

giving a syntax for categories

$\parallel$

giving a syntax for clones

and restricting to unary contexts

‘internal language of categories’ [informal]

||

internal language [formal] of the free clone

Defining an internal language for cartesian categories

Defining an internal language for cartesian categories

1. Define fp-clones,

1. Define fp-clones,
2. Observe that:

$$
\begin{tikzcd}
 & \text{fp-clones} \arrow[dl, bend left, "\text{forget}"] \arrow[dr, bend left, "\text{restrict to unary maps}"] & \\
\mathrm{Graph} \arrow[ur, bend left, "\text{free}"] \arrow[rr, bend left, "\text{free}"] \arrow[rr, phantom, "\perp"] & & \mathrm{CartCat} \arrow[ul, bend left] \arrow[ll, bend left, "\text{forget}"]
\end{tikzcd}
$$

Defining an internal language for cartesian categories

1. Define fp-clones,
2. Observe that:

$$
\begin{tikzcd}
& \text{fp-clones} \arrow[dl, bend left, "\text{forget}"] \arrow[dr, bend left, "\text{restrict to unary maps}"] & \\
\mathrm{Graph} \arrow[ur, bend left, "\text{free}"] \arrow[ur, phantom, "\perp"] \arrow[rr, bend left, "\text{free}"] \arrow[rr, phantom, "\perp"] & & \text{CartCat} \arrow[ul, bend left] \arrow[ul, phantom, "\perp"] \arrow[ll, bend left, "\text{forget}"]
\end{tikzcd}
$$

3. Internal language of cartesian categories
   $\overset{\text{def.}}{=}$ internal language of free fp-clone,
   restricted to unary contexts

An fp-clone [fp = finite-product] is a clone $(S, \mathbb{C})$ with

An fp-clone [fp = finite-product] is a clone $(S, \mathbb{C})$ with

1. for every $A_1, \ldots, A_n$, an object $\prod_n(A_1, \ldots, A_n)$,

An fp-clone [fp = finite-product] is a clone $(S, \mathbb{C})$ with

1. for every $A_1, \ldots, A_n$, an object $\prod_n(A_1, \ldots, A_n)$,
2. maps $\pi_i : \prod_n(A_1, \ldots, A_n) \to A_i$ inducing isomorphisms

$$\begin{tikzcd} \mathbb{C}(\Gamma; \prod_n(A_1, \ldots, A_n)) \arrow[rr, bend left, "{(\pi_1[-], \ldots, \pi_n[-])}"] \arrow[rr, phantom, "\cong"] & & \prod_{i=1}^{n} \mathbb{C}(\Gamma; A_i) \arrow[ll, bend left, "{\langle -, \ldots, = \rangle}"] \end{tikzcd}$$

An fp-clone [fp = finite-product] is a clone $(S, \mathbb{C})$ with

1. for every $A_1, \ldots, A_n$, an object $\prod_n(A_1, \ldots, A_n)$,
2. maps $\pi_i : \prod_n(A_1, \ldots, A_n) \to A_i$ inducing isomorphisms

$$\begin{tikzcd} \mathbb{C}(\Gamma; \prod_n(A_1, \ldots, A_n)) \arrow[r, bend left, "{(\pi_1[-], \ldots, \pi_n[-])}"] \arrow[r, phantom, "\cong"] & \prod_{i=1}^{n} \mathbb{C}(\Gamma; A_i) \arrow[l, bend left, "{\langle -, \ldots, = \rangle}"] \end{tikzcd}$$

Internal language of free fp-clone:

An fp-clone [fp = finite-product] is a clone $(S, \mathbb{C})$ with

1. for every $A_1, \ldots, A_n$, an object $\prod_n(A_1, \ldots, A_n)$,
2. maps $\pi_i : \prod_n(A_1, \ldots, A_n) \to A_i$ inducing isomorphisms

$$\begin{tikzcd} \mathbb{C}(\Gamma; \prod_n(A_1, \ldots, A_n)) \arrow[r, bend left, "(\pi_1[-], \ldots, \pi_n[-])"] \arrow[r, phantom, "\cong"] & \prod_{i=1}^n \mathbb{C}(\Gamma; A_i) \arrow[l, bend left, "\langle -, \ldots, = \rangle"] \end{tikzcd}$$

Internal language of free fp-clone:

$$\frac{A_1, \ldots, A_n \text{ type}}{\prod_n(A_1, \ldots, A_n) \text{ type}}$$

An fp-clone [fp = finite-product] is a clone $(S, \mathbb{C})$ with

1. for every $A_1, \ldots, A_n$, an object $\prod_n(A_1, \ldots, A_n)$,
2. maps $\pi_i : \prod_n(A_1, \ldots, A_n) \to A_i$ inducing isomorphisms

$$\begin{tikzcd} \mathbb{C}(\Gamma; \prod_n(A_1, \ldots, A_n)) \arrow[r, bend left, "{(\pi_1[-], \ldots, \pi_n[-])}"] \arrow[r, phantom, "\cong"] & \prod_{i=1}^n \mathbb{C}(\Gamma; A_i) \arrow[l, bend left, "{\langle -, \ldots, = \rangle}"] \end{tikzcd}$$

Internal language of free fp-clone:

$$\frac{}{p : \prod_n(A_1, \ldots, A_n) \vdash \pi_i(p) : A_i} \; (1 \leqslant i \leqslant n)$$

An fp-clone [fp = finite-product] is a clone $(S, \mathbb{C})$ with

1. for every $A_1, \ldots, A_n$, an object $\prod_n(A_1, \ldots, A_n)$,
2. maps $\pi_i : \prod_n(A_1, \ldots, A_n) \to A_i$ inducing isomorphisms

$$\begin{tikzcd} \mathbb{C}(\Gamma; \prod_n(A_1, \ldots, A_n)) \arrow[r, bend left, "{(\pi_1[-], \ldots, \pi_n[-])}"] \arrow[r, phantom, "\cong"] & \prod_{i=1}^n \mathbb{C}(\Gamma; A_i) \arrow[l, bend left, "{\langle -, \ldots, = \rangle}"] \end{tikzcd}$$

Internal language of free fp-clone:

$$\frac{}{p : \prod_n(A_1, \ldots, A_n) \vdash \pi_i(p) : A_i} \ (1 \leqslant i \leqslant n)$$

$$\frac{(\Gamma \vdash u_i : A_i)_{i=1,\ldots,n}}{\Gamma \vdash \langle u_1, \ldots, u_n \rangle : \prod_n(A_1, \ldots, A_n)}$$

An fp-clone [fp = finite-product] is a clone $(S, \mathbb{C})$ with

1. for every $A_1, \ldots, A_n$, an object $\prod_n(A_1, \ldots, A_n)$,
2. maps $\pi_i : \prod_n(A_1, \ldots, A_n) \to A_i$ inducing isomorphisms

$$\begin{tikzcd} \mathbb{C}(\Gamma; \prod_n(A_1, \ldots, A_n)) \arrow[r, bend left, "(\pi_1[-], \ldots, \pi_n[-])"] \arrow[r, phantom, "\cong"] & \prod_{i=1}^n \mathbb{C}(\Gamma; A_i) \arrow[l, bend left, "\langle -, \ldots, = \rangle"] \end{tikzcd}$$

Internal language of free fp-clone:

$$\frac{}{p : \prod_n(A_1, \ldots, A_n) \vdash \pi_i(p) : A_i} \ (1 \leqslant i \leqslant n)$$

$$\frac{(\Gamma \vdash u_i : A_i)_{i=1,\ldots,n}}{\Gamma \vdash \langle u_1, \ldots, u_n \rangle : \prod_n(A_1, \ldots, A_n)}$$

$$\pi_i(p)[p \mapsto \langle u_1, \ldots, u_n \rangle] = u_i \qquad (\beta)$$

$$\langle \pi_1(p)[p \mapsto u], \ldots, \pi_n(p)[p \mapsto u] \rangle = u \qquad (\eta)$$

An fp-clone [fp = finite-product] is a clone $(S, \mathbb{C})$ with

1. for every $A_1, \ldots, A_n$, an object $\prod_n(A_1, \ldots, A_n)$,
2. maps $\pi_i : \prod_n(A_1, \ldots, A_n) \to A_i$ inducing isomorphisms

$$\begin{tikzcd} \mathbb{C}(\Gamma; \prod_n(A_1, \ldots, A_n)) \arrow[r, bend left, "(\pi_1[-],\ldots,\pi_n[-])"] \arrow[r, phantom, "\cong"] & \prod_{i=1}^n \mathbb{C}(\Gamma; A_i) \arrow[l, bend left, "\langle -,\ldots,= \rangle"] \end{tikzcd}$$

Internal language of free fp-clone:

$$\frac{\Gamma \vdash t : \prod_n(A_1, \ldots, A_n)}{\Gamma \vdash \pi_i(t) : A_i} \ (1 \leqslant i \leqslant n)$$

$$\frac{(\Gamma \vdash u_i : A_i)_{i=1,\ldots,n}}{\Gamma \vdash \langle u_1, \ldots, u_n \rangle : \prod_n(A_1, \ldots, A_n)}$$

$$\pi_i(\langle u_1, \ldots, u_n \rangle) = u_i \qquad (\beta)$$

$$\langle \pi_1(t), \ldots, \pi_n(t) \rangle = t \qquad (\eta)$$

An fp-clone [fp = finite-product] is a clone $(S, \mathbb{C})$ with

1. for every $A_1, \ldots, A_n$, an object $\prod_n(A_1, \ldots, A_n)$,
2. maps $\pi_i : \prod_n(A_1, \ldots, A_n) \to A_i$ inducing isomorphisms

$$\begin{tikzcd} \mathbb{C}(\Gamma; \prod_n(A_1, \ldots, A_n)) \arrow[r, bend left, "{(\pi_1[-], \ldots, \pi_n[-])}"] \arrow[r, phantom, "\cong"] & \prod_{i=1}^{n} \mathbb{C}(\Gamma; A_i) \arrow[l, bend left, "{\langle -, \ldots, = \rangle}"] \end{tikzcd}$$

Internal language of free fp-clone $\rightsquigarrow$ products in STLC

An fp-clone [fp = finite-product] is a clone $(S, \mathbb{C})$ with

1. for every $A_1, \ldots, A_n$, an object $\prod_n(A_1, \ldots, A_n)$,
2. maps $\pi_i : \prod_n(A_1, \ldots, A_n) \to A_i$ inducing isomorphisms

$$\begin{tikzcd} \mathbb{C}(\Gamma; \prod_n(A_1, \ldots, A_n)) \arrow[r, bend left, "{(\pi_1[-], \ldots, \pi_n[-])}"] \arrow[r, phantom, "\cong"] & \prod_{i=1}^{n} \mathbb{C}(\Gamma; A_i) \arrow[l, bend left, "{\langle -, \ldots, = \rangle}"] \end{tikzcd}$$

Internal language of free fp-clone $\rightsquigarrow$ products in STLC

free cartesian category on $\mathbb{G}$

$\cong$ free fp-clone on $\mathbb{G}$, restricted to unary contexts

An fp-clone [fp = finite-product] is a clone $(S, \mathbb{C})$ with

1. for every $A_1, \ldots, A_n$, an object $\prod_n(A_1, \ldots, A_n)$,
2. maps $\pi_i : \prod_n(A_1, \ldots, A_n) \to A_i$ inducing isomorphisms

$$\begin{tikzcd} \mathbb{C}(\Gamma; \prod_n(A_1, \ldots, A_n)) \arrow[r, bend left, "{(\pi_1[-], \ldots, \pi_n[-])}"] \arrow[r, phantom, "\cong"] & \prod_{i=1}^{n} \mathbb{C}(\Gamma; A_i) \arrow[l, bend left, "{\langle -, \ldots, = \rangle}"] \end{tikzcd}$$

Internal language of free fp-clone $\rightsquigarrow$ products in STLC

free cartesian category on $\mathcal{G}$

$\cong$ free fp-clone on $\mathcal{G}$, restricted to unary contexts

---

'internal language of fp-categories' [informal]

||

internal language [formal] of the free fp-clone

An fp-clone [fp = finite-product] is a clone $(S, \mathbb{C})$ with

1. for every $A_1, \ldots, A_n$, an object $\prod_n(A_1, \ldots, A_n)$,
2. maps $\pi_i : \prod_n(A_1, \ldots, A_n) \to A_i$ inducing isomorphisms

$$\begin{tikzcd} \mathbb{C}(\Gamma; \prod_n(A_1, \ldots, A_n)) \arrow[r, bend left, "(\pi_1[-],\ldots,\pi_n[-])"] \arrow[r, phantom, "\cong"] & \prod_{i=1}^n \mathbb{C}(\Gamma; A_i) \arrow[l, bend left, "\langle -,\ldots,= \rangle"] \end{tikzcd}$$

$\rightsquigarrow$ Equivalent to requiring $\mathbb{M}\mathbb{C}$ is representable

An fp-clone [fp = finite-product] is a clone $(S, \mathbb{C})$ with

1. for every $A_1, \ldots, A_n$, an object $\prod_n(A_1, \ldots, A_n)$,
2. maps $\pi_i : \prod_n(A_1, \ldots, A_n) \to A_i$ inducing isomorphisms

$$\begin{tikzcd} \mathbb{C}(\Gamma; \prod_n(A_1, \ldots, A_n)) \arrow[r, bend left, "(\pi_1[-], \ldots, \pi_n[-])"] \arrow[r, phantom, "\cong"] & \prod_{i=1}^{n} \mathbb{C}(\Gamma; A_i) \arrow[l, bend left, "\langle -, \ldots, = \rangle"] \end{tikzcd}$$

In an fp-clone, contexts and products coincide.

An fp-clone [fp = finite-product] is a clone $(S, \mathbb{C})$ with

1. for every $A_1, \ldots, A_n$, an object $\prod_n(A_1, \ldots, A_n)$,
2. maps $\pi_i : \prod_n(A_1, \ldots, A_n) \to A_i$ inducing isomorphisms

$$\begin{tikzcd} \mathbb{C}(\Gamma; \prod_n(A_1, \ldots, A_n)) \arrow[r, bend left, "(\pi_1[-], \ldots, \pi_n[-])"] \arrow[r, phantom, "\cong"] & \prod_{i=1}^n \mathbb{C}(\Gamma; A_i) \arrow[l, bend left, "\langle -, \ldots, = \rangle"] \end{tikzcd}$$

In an fp-clone, contexts and products coincide.

There exist maps

$$\left(u_i : \prod_n(A_1, \ldots, A_n) \to A_i\right)_{i=1,\ldots,n}$$

$$t : A_1, \ldots, A_n \to \prod_n(A_1, \ldots, A_n)$$

such that

$$t[u_1, \ldots, u_n] = \mathsf{p}^{(1)}_{\prod_n(A_1,\ldots,A_n)}$$

$$u_i[t] = \mathsf{p}^{(i)}_{A_1,\ldots,A_n} \quad (i=1,\ldots,n)$$

Defining an internal language for cc-structure

Defining an internal language for cc-structure

$$
\begin{tikzcd}
& \text{cc-clones} \arrow[dl, bend left, "\text{forget}"] \arrow[dr, bend left, "\text{restrict to unary maps}"] & \\
\mathrm{Graph} \arrow[ur, bend left, "\text{free}"] \arrow[ur, phantom, "\perp"] \arrow[rr, bend left, "\text{free}"] \arrow[rr, phantom, "\perp"] & & \mathrm{CCCat} \arrow[ll, bend left, "\text{forget}"] \arrow[ul, bend left] \arrow[ul, phantom, "\perp"]
\end{tikzcd}
$$

free cartesian closed category on G

$\cong$ free cc-clone on G, restricted to unary maps

Defining an internal language for cc-structure

$$
\begin{tikzcd}
& \color{blue}{\text{cc-clones}} \arrow[dl, bend left, "\text{forget}"] \arrow[dr, bend left, "\text{restrict to unary maps}"] & \\
\mathrm{Graph} \arrow[ur, bend left, "\text{free}"] \arrow[ur, phantom, "\perp"] \arrow[rr, bend left, "\text{free}"] \arrow[rr, phantom, "\perp"] & & \mathrm{CCCat} \arrow[ul, bend left] \arrow[ul, phantom, "\perp"] \arrow[ll, bend left, "\text{forget}"]
\end{tikzcd}
$$

---

free cartesian closed category on $\mathbb{G}$

$\cong$ free cc-clone on $\mathbb{G}$, restricted to unary maps

1. Define cc-clones,
2. Internal language of CCCs

   $\stackrel{\text{def.}}{=}$ internal language of free cc-clone,
   restricted to unary contexts

A clone $(S, \mathbb{C})$ has exponentials if $\mathrm{M}\mathbb{C}$ has exponentials.

A clone $(S, \mathbb{C})$ has exponentials if $\mathrm{M}\mathbb{C}$ has exponentials.

A clone $(S, \mathbb{C})$ with exponentials has

1. for every A, B, an object $A \Rightarrow B$,

A clone $(S, \mathbb{C})$ has exponentials if $\mathrm{M}\mathbb{C}$ has exponentials.

A clone $(S, \mathbb{C})$ with exponentials has

1. for every A, B, an object $A \Rightarrow B$,
2. maps $\mathrm{eval}_{A,B} : A \Rightarrow B, A \to B$ inducing isomorphisms

$$\begin{tikzcd} \mathbb{C}(X_1, \dots, X_n; A \Rightarrow B) \arrow[r, bend left, "\mathrm{eval}\left[(-)[\mathsf{p}^{(1)},\dots,\mathsf{p}^{(n)}],\mathsf{p}^{(n+1)}\right]"] \arrow[r, phantom, "\cong"] & \mathbb{C}(X_1, \dots, X_n, A; B) \arrow[l, bend left, "\lambda"] \end{tikzcd}$$

A clone $(S, \mathbb{C})$ has exponentials if $\mathrm{M}\mathbb{C}$ has exponentials.

A clone $(S, \mathbb{C})$ with exponentials has

1. for every A, B, an object $A \Rightarrow B$,
2. maps $\mathrm{eval}_{A,B} : A \Rightarrow B, A \to B$ inducing isomorphisms

$$
\begin{tikzcd}
\mathbb{C}(X_1, \dots, X_n; A \Rightarrow B) \arrow[rr, bend left, "\mathrm{eval}\left[(-)[\mathsf{p}^{(1)}, \dots, \mathsf{p}^{(n)}], \mathsf{p}^{(n+1)}\right]"] \arrow[rr, phantom, "\cong"] & & \mathbb{C}(X_1, \dots, X_n, A; B) \arrow[ll, bend left, "\lambda"]
\end{tikzcd}
$$

for $n = 1$:

$$
\text{if } t : X, A \to B \text{ then } \mathrm{eval}\left[t[\mathsf{p}^{(1)}], \mathsf{p}^{(2)}\right] : X \to (A \Rightarrow B)
$$

A clone $(S, \mathbb{C})$ has exponentials if $\mathrm{M}\mathbb{C}$ has exponentials.

A clone $(S, \mathbb{C})$ with exponentials has

1. for every A, B, an object $A \Rightarrow B$,
2. maps $\mathrm{eval}_{A,B} : A \Rightarrow B, A \to B$ inducing isomorphisms

$$\begin{tikzcd} \mathbb{C}(X_1, \ldots, X_n; A \Rightarrow B) \arrow[r, bend left, "\mathrm{eval}\left[(-)[\mathsf{p}^{(1)},\ldots,\mathsf{p}^{(n)}],\mathsf{p}^{(n+1)}\right]"] \arrow[r, phantom, "\cong"] & \mathbb{C}(X_1, \ldots, X_n, A; B) \arrow[l, bend left, "\lambda"] \end{tikzcd}$$

for $n = 1$:

$$\text{if } t : X, A \to B \text{ then } \mathrm{eval}\left[t[\mathsf{p}^{(1)}], \mathsf{p}^{(2)}\right] : X \to (A \Rightarrow B)$$

$\rightsquigarrow$ *c.f.* $\mathrm{eval} \circ \langle f \circ \pi_1, \pi_2 \rangle = \mathrm{eval} \circ (f \times A)$

An fp-clone is a clone $(S, \mathbb{C})$ with

1. for every $A_1, \ldots, A_n$, an object $\prod_n(A_1, \ldots, A_n)$,
2. maps $\pi_i : \prod_n(A_1, \ldots, A_n) \to A_i$ inducing isomorphisms

$$\begin{tikzcd} \mathbb{C}(\Gamma; \prod_n(A_1, \ldots, A_n)) \arrow[r, bend left, "{(\pi_1[-], \ldots, \pi_n[-])}"] \arrow[r, phantom, "\cong"] & \prod_{i=1}^{n} \mathbb{C}(\Gamma; A_i) \arrow[l, bend left, "{\langle -, \ldots, = \rangle}"] \end{tikzcd}$$

A clone $(S, \mathbb{C})$ with exponentials has

1. for every A, B, an object $A \Rightarrow B$,
2. maps $\mathrm{eval}_{A,B} : A \Rightarrow B, A \to B$ inducing isomorphisms

$$\begin{tikzcd} \mathbb{C}(X_1, \ldots, X_n; A \Rightarrow B) \arrow[r, bend left, "{\mathrm{eval}[(-)[\mathsf{p}^{(1)}, \ldots, \mathsf{p}^{(n)}], \mathsf{p}^{(n+1)}]}"] \arrow[r, phantom, "\cong"] & \mathbb{C}(X_1, \ldots, X_n, A; B) \arrow[l, bend left, "\lambda"] \end{tikzcd}$$

An fp-clone is a clone $(S, \mathbb{C})$ with

1. for every $A_1, \ldots, A_n$, an object $\prod_n(A_1, \ldots, A_n)$,
2. maps $\pi_i : \prod_n(A_1, \ldots, A_n) \to A_i$ inducing isomorphisms

$$\begin{tikzcd} \mathbb{C}(\Gamma; \prod_n(A_1, \ldots, A_n)) \arrow[r, bend left, "{(\pi_1[-], \ldots, \pi_n[-])}"] \arrow[r, phantom, "\cong"] & \prod_{i=1}^{n} \mathbb{C}(\Gamma; A_i) \arrow[l, bend left, "{\langle -, \ldots, = \rangle}"] \end{tikzcd}$$

A clone $(S, \mathbb{C})$ with exponentials has

1. for every A, B, an object $A \Rightarrow B$,
2. maps $\mathrm{eval}_{A,B} : A \Rightarrow B, A \to B$ inducing isomorphisms

$$\begin{tikzcd} \mathbb{C}(X_1, \ldots, X_n; A \Rightarrow B) \arrow[r, bend left, "{\mathrm{eval}[(-)[\mathsf{p}^{(1)}, \ldots, \mathsf{p}^{(n)}], \mathsf{p}^{(n+1)}]}"] \arrow[r, phantom, "\cong"] & \mathbb{C}(X_1, \ldots, X_n, A; B) \arrow[l, bend left, "\lambda"] \end{tikzcd}$$

A clone is cartesian closed [cc-] if it is an fp-clone with exponentials.

internal language of free cc-clone

||

simply-typed lambda calculus

internal language of free cc-clone

||

simply-typed lambda calculus

'internal language of cartesian closed categories' [informal]

||

internal language [formal] of the free cc-clone

||

simply-typed lambda calculus

$\Lambda_{\mathrm{ps}}^{\times,\to}$: an internal language for cc-bicategories

A recipe for $\Lambda_{\mathrm{ps}}^{\times,\to}$, modulo technicalities

1. Define cc-biclones,
2. Define internal language of a biclone,

$$\begin{tikzcd}
& \text{cc-biclones} \arrow[dl, bend left, "\text{forget}"] \arrow[dr, bend left, "\text{restrict to unary maps}"] & \\
\mathrm{Graph} \arrow[ur, bend left, "\text{free}"] \arrow[rr, bend left, "\text{free}"] \arrow[ur, phantom, "\perp"] & & \text{CCBicat} \arrow[ll, bend left, "\text{forget}"] \arrow[ul, bend left] \arrow[ul, phantom, "\perp"] \arrow[ll, phantom, "\perp"]
\end{tikzcd}$$

3. Internal language of cc-bicategories
   $\overset{\text{def.}}{=}$ internal language of free cc-biclone,
   restricted to unary contexts

A recipe for $\Lambda_{\mathrm{ps}}^{\times,\to}$, modulo technicalities

1. Define cc-biclones,
2. Define internal language of a biclone,

$$
\begin{tikzcd}
 & \begin{array}{c}\text{cc-biclones}\\ +\ \text{strict maps}\end{array} \arrow[dl, bend left, "\text{forget}"] \arrow[dr, bend left, "\text{restrict to unary maps}"] & \\
\mathrm{Graph} \arrow[ur, bend left, "\text{free}"] \arrow[ur, phantom, "\perp"] \arrow[rr, bend left, "\text{free}"] \arrow[rr, phantom, "\perp"] & & \begin{array}{l}\text{cc-bicategories}\\ +\ \text{strict maps}\end{array} \arrow[ll, bend left, "\text{forget}"] \arrow[ul, bend left] \arrow[ul, phantom, "\perp"]
\end{tikzcd}
$$

3. Internal language of cc-bicategories
   $\overset{\text{def.}}{=}$ internal language of free cc-biclone,
   restricted to unary contexts

A recipe for $\Lambda_{\mathrm{ps}}^{\times,\to}$, modulo technicalities

1. Define cc-biclones,
2. Define internal language of a biclone,
3. Internal language of cc-bicategories
   $\stackrel{\text{def.}}{=}$ internal language of free cc-biclone,
   restricted to unary contexts

A recipe for an internal language for bicategories

1. Define biclones,
2. Define notion of internal language,
3. Internal language of bicategories
   $\overset{\text{def.}}{=}$ internal language of free biclone,
   restricted to unary contexts

Biclone $(S, \mathcal{C})$ = abstract theory of bicategorical substitution:

Biclone $(S, \mathcal{C})$ = abstract theory of bicategorical substitution:

- *Sorts* S,

Biclone $(S, \mathcal{C})$ = abstract theory of bicategorical substitution:

- *Sorts* S,
- Hom-categories $\big(\mathcal{C}(X_1, \ldots, X_n; Y), \bullet, \mathrm{id}\big)$,

Biclone $(S, \mathcal{C})$ = abstract theory of bicategorical substitution:

- *Sorts* S,
- Hom-categories $\big(\mathcal{C}(X_1, \ldots, X_n; Y), \bullet, \mathrm{id}\big)$,

$$\begin{tikzcd} X_1, \ldots, X_n \arrow[r, bend left, "t"] \arrow[r, bend right, "t'"'] \arrow[r, phantom, "\Downarrow\tau"] & Y \end{tikzcd}$$

Biclone $(S, \mathcal{C})$ = abstract theory of bicategorical substitution:

- *Sorts* S,
- Hom-categories $\big(\mathcal{C}(X_1, \dots, X_n; Y), \bullet, \mathrm{id}\big)$,
- *Projection* 1-cells $\mathsf{p}_{X_\bullet}^{(i)} : X_1, \dots, X_n \to X_i$ $(1 \leqslant i \leqslant n)$,

Biclone $(S, \mathcal{C})$ = abstract theory of bicategorical substitution:

- *Sorts* S,
- Hom-categories $\big(\mathcal{C}(X_1, \ldots, X_n; Y), \bullet, \mathrm{id}\big)$,
- *Projection* 1-cells $\mathsf{p}_{X_\bullet}^{(i)} : X_1, \ldots, X_n \to X_i$ $(1 \leqslant i \leqslant n)$,
- *Substitution* functors

$$\begin{gathered}\mathcal{C}(X_1, \ldots, X_n; Y) \times \textstyle\prod_{i=1}^{n} \mathcal{C}(\Gamma; X_i) \to \mathcal{C}(\Gamma; Y) \\ t, (u_1, \ldots, u_n) \mapsto t[u_1, \ldots, u_n] \\ \tau, (\sigma_1, \ldots, \sigma_n) \mapsto \tau[\sigma_1, \ldots, \sigma_n]\end{gathered}$$

Biclone $(S, \mathcal{C})$ = abstract theory of bicategorical substitution:

- *Sorts* S,
- Hom-categories $\big(\mathcal{C}(X_1, \ldots, X_n; Y), \bullet, \mathrm{id}\big)$,
- *Projection* 1-cells $\mathrm{p}^{(i)}_{X_\bullet} : X_1, \ldots, X_n \to X_i$ $(1 \leqslant i \leqslant n)$,
- *Substitution* functors

$$\begin{gathered}\mathcal{C}(X_1, \ldots, X_n; Y) \times \textstyle\prod_{i=1}^{n} \mathcal{C}(\Gamma; X_i) \to \mathcal{C}(\Gamma; Y) \\ t, (u_1, \ldots, u_n) \mapsto t[u_1, \ldots, u_n] \\ \tau, (\sigma_1, \ldots, \sigma_n) \mapsto \tau[\sigma_1, \ldots, \sigma_n]\end{gathered}$$

- Structural isomorphisms

$$\mathrm{p}^{(k)}[u_1, \ldots, u_n] \xRightarrow{\varrho^{(k)}_{u_\bullet}} u_k \qquad (1 \leqslant k \leqslant n)$$

$$t[\mathrm{p}^{(1)}, \ldots, \mathrm{p}^{(n)}] \xRightarrow{\iota_t} t$$

$$t\,[u_\bullet]\,[v_\bullet] \xRightarrow{\mathrm{assoc}_{t; u_\bullet; v_\bullet}} t[u_\bullet[v_\bullet]]$$

Biclone $(S, \mathcal{C})$ = abstract theory of bicategorical substitution:

- *Sorts* S,
- Hom-categories $\big(\mathcal{C}(X_1, \ldots, X_n; Y), \bullet, \mathrm{id}\big)$,
- *Projection* 1-cells $\mathsf{p}^{(i)}_{X_\bullet} : X_1, \ldots, X_n \to X_i$ $(1 \leqslant i \leqslant n)$,
- *Substitution* functors

$$\begin{gathered}\mathcal{C}(X_1, \ldots, X_n; Y) \times \textstyle\prod_{i=1}^{n} \mathcal{C}(\Gamma; X_i) \to \mathcal{C}(\Gamma; Y) \\ t, (u_1, \ldots, u_n) \mapsto t[u_1, \ldots, u_n] \\ \tau, (\sigma_1, \ldots, \sigma_n) \mapsto \tau[\sigma_1, \ldots, \sigma_n]\end{gathered}$$

- Structural isomorphisms

$$\begin{aligned}\mathsf{p}^{(k)}[u_1, \ldots, u_n] &\xRightarrow{\varrho^{(k)}_{u_\bullet}} u_k \qquad\qquad (1 \leqslant k \leqslant n) \\ t[\mathsf{p}^{(1)}, \ldots, \mathsf{p}^{(n)}] &\xRightarrow{\iota_t} t \\ t\,[u_\bullet]\,[v_\bullet] &\xRightarrow{\mathrm{assoc}_{t; u_\bullet; v_\bullet}} t[u_\bullet[v_\bullet]]\end{aligned}$$

subject to a triangle law and pentagon law.

Biclone $(S, \mathcal{C})$ = abstract theory of bicategorical substitution:

- *Sorts* S,

Every biclone defines a bicategory and a bi-multicategory

- Hom-categories $\big(\mathcal{C}(X_1, \dots, X_n; Y), \bullet, \mathrm{id}\big)$,
- *Projection* 1-cells $\mathsf{p}^{(i)}_{X_\bullet} : X_1, \dots, X_n \to X_i$ $(1 \leqslant i \leqslant n)$,
- *Substitution* functors

$$\begin{gathered}\mathcal{C}(X_1, \dots, X_n; Y) \times \textstyle\prod_{i=1}^{n} \mathcal{C}(\Gamma; X_i) \to \mathcal{C}(\Gamma; Y) \\ t, (u_1, \dots, u_n) \mapsto t[u_1, \dots, u_n] \\ \tau, (\sigma_1, \dots, \sigma_n) \mapsto \tau[\sigma_1, \dots, \sigma_n]\end{gathered}$$

- Structural isomorphisms

$$\begin{aligned}\mathsf{p}^{(k)}[u_1, \dots, u_n] &\xRightarrow{\varrho^{(k)}_{u_\bullet}} u_k \qquad (1 \leqslant k \leqslant n) \\ t[\mathsf{p}^{(1)}, \dots, \mathsf{p}^{(n)}] &\xRightarrow{\iota_t} t \\ t\,[u_\bullet]\,[v_\bullet] &\xRightarrow{\mathsf{assoc}_{t;u_\bullet;v_\bullet}} t[u_\bullet[v_\bullet]]\end{aligned}$$

subject to a triangle law and pentagon law.

Internal language for biclones [*c.f.* Hilken, Seely, Hirschowitz]

Internal language for biclones [*c.f.* Hilken, Seely, Hirschowitz]

terms:

$$
\begin{array}{c}
(x_1 : A_1, \ldots, x_n : A_n \vdash t : B) \\
\updownarrow \\
t : A_1, \ldots, A_n \to B
\end{array}
$$

Internal language for biclones [*c.f.* Hilken, Seely, Hirschowitz]

terms:

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

$$\updownarrow$$

$$t : A_1, \ldots, A_n \to B$$

rewrites:

$$(x_1 : A_1, \ldots, x_n : A_n \vdash \tau : t \Rightarrow t' : B)$$

$$\updownarrow$$

$$\tau : t \Rightarrow t' : A_1, \ldots, A_n \to B$$

Internal language for biclones [c.f. Hilken, Seely, Hirschowitz]

terms:

$$
\begin{array}{c}
(x_1 : A_1, \ldots, x_n : A_n \vdash t : B) \\
\Updownarrow \\
t : A_1, \ldots, A_n \to B
\end{array}
$$

rewrites:

$$
\begin{array}{c}
(x_1 : A_1, \ldots, x_n : A_n \vdash \tau : t \Rightarrow t' : B) \\
\Updownarrow \\
\tau : t \Rightarrow t' : A_1, \ldots, A_n \to B
\end{array}
$$

what is the internal language of the free biclone on G?

A type theory for biclones = internal language of free biclone

Hom-categories $\big(\mathcal{C}(X_1, \ldots, X_n; Y), \bullet, \mathrm{id}\big)$

A type theory for biclones = internal language of free biclone

Hom-categories $\big(\mathcal{C}(X_1, \ldots, X_n; Y), \bullet, \mathrm{id}\big)$

Judgements:

Hom-categories $\big(\mathcal{C}(X_1, \ldots, X_n; Y), \bullet, \mathrm{id}\big)$

Judgements:

- Relating *terms*: $\Gamma \vdash t : B$

A type theory for biclones = internal language of free biclone

Hom-categories $\big(\mathcal{C}(X_1, \dots, X_n; Y), \bullet, \mathrm{id}\big)$

Judgements:

- Relating *terms*: $\Gamma \vdash t : B$
- Relating *rewrites*: $\Gamma \vdash \tau : t \Rightarrow t' : B$

A type theory for biclones = internal language of free biclone

Hom-categories $\left(\mathcal{C}(X_1, \ldots, X_n; Y), \bullet, \mathrm{id}\right)$

Judgements:

- Relating *terms*: $\Gamma \vdash t : B$
- Relating *rewrites*: $\Gamma \vdash \tau : t \Rightarrow t' : B$
- Equational theory $\Gamma \vdash \tau \equiv \tau' : t \Rightarrow t' : B$

A type theory for biclones = internal language of free biclone

Hom-categories $\left(\mathcal{C}(X_1, \ldots, X_n; Y), \bullet, \mathrm{id}\right)$

Judgements:

- Relating *terms*: $\Gamma \vdash t : B$
- Relating *rewrites*: $\Gamma \vdash \tau : t \Rightarrow t' : B$
- Equational theory $\Gamma \vdash \tau \equiv \tau' : t \Rightarrow t' : B$

Vertical composition:
$$\frac{\Gamma \vdash \tau' : t' \Rightarrow t'' : B \qquad \Gamma \vdash \tau : t \Rightarrow t' : B}{\Gamma \vdash \tau' \bullet \tau : t \Rightarrow t'' : B}$$

Identities:
$$\frac{\Gamma \vdash t : B}{\Gamma \vdash \mathrm{id}_t : t \Rightarrow t : B}$$

A *substitution* functor

$$\mathcal{C}(X_1, \ldots, X_n; Y) \times \prod_{i=1}^{n} \mathcal{C}(\Gamma; X_i) \to \mathcal{C}(\Gamma; Y)$$
$$t, (u_1, \ldots, u_n) \mapsto t[u_1, \ldots, u_n]$$
$$\tau, (\sigma_1, \ldots, \sigma_n) \mapsto \tau[\sigma_1, \ldots, \sigma_n]$$

A *substitution* functor

$$\mathcal{C}(X_1, \ldots, X_n; Y) \times \prod_{i=1}^{n} \mathcal{C}(\Gamma; X_i) \to \mathcal{C}(\Gamma; Y)$$
$$t, (u_1, \ldots, u_n) \mapsto t[u_1, \ldots, u_n]$$
$$\tau, (\sigma_1, \ldots, \sigma_n) \mapsto \tau[\sigma_1, \ldots, \sigma_n]$$

Explicit substitution:

$$\frac{x_1 : A_1, \ldots, x_n : A_n \vdash t : B \qquad (\Delta \vdash u_i : A_i)_{i=1.,n}}{\Delta \vdash t\,\{x_i \mapsto u_i\} : B}$$

$$\frac{x_1 : A_1, \ldots, x_n : A_n \vdash \tau : t \Rightarrow t' : B \qquad (\Delta \vdash \sigma_i : u_i \Rightarrow u'_i : A_i)_{i=1,\ldots,n}}{\Delta \vdash \tau\,\{x_i \mapsto \sigma_i\} : t\,\{x_i \mapsto u_i\} \Rightarrow t'\,\{x_i \mapsto u'_i\} : B}$$

$\rightsquigarrow$ binds the variables $x_1, \ldots, x_n$

A type theory for biclones

Structural isomorphisms $\varrho^{(k)}, \iota, \mathsf{assoc}$

Structural isomorphisms $\varrho^{(k)}, \iota, \mathsf{assoc}$

Distinguished invertible rewrites *e.g.*:

$$\frac{(\Delta \vdash u_i : A_i)_{i=1,\ldots,n}}{x_1 : A_1, \ldots, x_n : A_n \vdash \varrho^{(k)}_{u_\bullet} : x_k\,\{x_i \mapsto u_i\} \stackrel{\cong}{\Longrightarrow} u_k : A_k}\ (1 \leqslant k \leqslant n)$$

the free bicategory on G

$\simeq$ free biclone on G, restricted to unary maps

the free bicategory on G

$\simeq$ free biclone on G, restricted to unary maps

syntax for bicategories

$$\parallel$$

syntax for free biclone, restricted to unary contexts

the free bicategory on G

$\simeq$ free biclone on G, restricted to unary maps

syntax for bicategories

$\parallel$

syntax for free biclone, restricted to unary contexts

'internal language of bicategories' [informal]

$\parallel$

internal language [formal] of the free biclone

1. Define fp-biclones,
2. Extract internal language $\Lambda^{\times}_{\mathrm{ps}}$ of free fp-biclone,
3. Restrict to unary contexts

   $\rightsquigarrow$ get internal language of fp-bicategories.

1. Define fp-biclones,
2. Extract internal language $\Lambda^{\times}_{\mathrm{ps}}$ of free fp-biclone,
3. Restrict to unary contexts
   $\rightsquigarrow$ get internal language of fp-bicategories.

An fp-clone is a clone $(S, \mathbb{C})$ with

1. for every $A_1, \ldots, A_n$, an object $\prod_n(A_1, \ldots, A_n)$,
2. maps $\pi_i : \prod_n(A_1, \ldots, A_n) \to A_i$
   inducing isomorphisms

$$\begin{tikzcd} \mathbb{C}(\Gamma; \prod_n(A_1, \ldots, A_n)) \arrow[r, bend left, "{(\pi_1[-], \ldots, \pi_n[-])}"] \arrow[r, phantom, "\cong"] & \prod_{i=1}^{n} \mathbb{C}(\Gamma; A_i) \arrow[l, bend left, "{\langle -, \ldots, = \rangle}"] \end{tikzcd}$$

$\rightsquigarrow$ equivalent to asking that $\mathrm{M}\mathbb{C}$ is representable.

1. Define fp-biclones,
2. Extract internal language $\Lambda^{\times}_{\mathrm{ps}}$ of free fp-biclone,
3. Restrict to unary contexts

   $\rightsquigarrow$ get internal language of fp-bicategories.

An fp-biclone is a biclone $(S, \mathcal{C})$ with

1. for every $A_1, \ldots, A_n$, an object $\prod_n(A_1, \ldots, A_n)$,
2. maps $\pi_i : \prod_n(A_1, \ldots, A_n) \to A_i$

   inducing adjoint equivalences

$$\begin{tikzcd} \mathcal{C}(\Gamma; \prod_n(A_1, \ldots, A_n)) \arrow[r, bend left, "(\pi_1[-], \ldots, \pi_n[-])"] \arrow[r, phantom, "\perp\simeq"] & \prod_{i=1}^{n} \mathcal{C}(\Gamma; A_i) \arrow[l, bend left, "\langle -, \ldots, = \rangle"] \end{tikzcd}$$

$\rightsquigarrow$ equivalent to asking that $\mathrm{M}\mathcal{C}$ is representable.

A type theory for fp-bicategories = internal language of free fp-biclone

1-cells $\pi_i : \prod_n(A_1, \ldots, A_n) \to A_i \qquad (1 \leqslant i \leqslant n)$

A type theory for fp-bicategories = internal language of free fp-biclone

1-cells $\pi_i : \prod_n(A_1, \ldots, A_n) \to A_i \qquad (1 \leqslant i \leqslant n)$

Projections $$\frac{}{p : \prod_n(A_1, \ldots, A_n) \vdash \pi_i(p) : A_i} \; (1 \leqslant i \leqslant n)$$

Equivalences
$$\begin{tikzcd} \mathcal{B}(\Gamma, \prod_n(A_1, \ldots, A_n)) \arrow[rr, bend left, "(\pi_1[-],\ldots,\pi_n[-])"] \arrow[rr, phantom, "\perp\simeq"] & & \prod_{i=1}^n \mathcal{B}(\Gamma, A_i) \arrow[ll, bend left, "\langle -,\ldots,= \rangle"] \end{tikzcd}$$

A type theory for fp-bicategories = internal language of free fp-biclone

$$\begin{tikzcd} \text{Equivalences} \quad \mathcal{B}(\Gamma, \prod_n(A_1, \ldots, A_n)) \arrow[rr, bend left, "(\pi_1[-],\ldots,\pi_n[-])"] \arrow[rr, phantom, "\perp\simeq"] & & \prod_{i=1}^n \mathcal{B}(\Gamma, A_i) \arrow[ll, bend left, "\langle -,\ldots,= \rangle"] \end{tikzcd}$$

- Mapping on objects $\langle -, \ldots, = \rangle$.

A type theory for fp-bicategories = internal language of free fp-biclone

$$\text{Equivalences} \quad \begin{tikzcd} \mathcal{B}(\Gamma, \prod_n(A_1, \ldots, A_n)) \arrow[r, bend left, "{(\pi_1[-],\ldots,\pi_n[-])}"] \arrow[r, phantom, "\perp\simeq"] & \prod_{i=1}^{n} \mathcal{B}(\Gamma, A_i) \arrow[l, bend left, "{\langle -,\ldots,= \rangle}"] \end{tikzcd}$$

- Mapping on objects $\langle -, \ldots, = \rangle$.
- Counit $\varpi^{(i)}_{t_\bullet} : \pi_i[\langle t_1, \ldots, t_n \rangle] \overset{\cong}{\Longrightarrow} t_i$

$$\text{Equivalences}\quad \begin{tikzcd} \mathcal{B}(\Gamma, \prod_n(A_1,\dots,A_n)) \arrow[r, bend left, "(\pi_1[-],\dots,\pi_n[-])"] \arrow[r, phantom, "\perp\simeq"] & \prod_{i=1}^n \mathcal{B}(\Gamma, A_i) \arrow[l, bend left, "\langle -,\dots,=\rangle"] \end{tikzcd}$$

- Mapping on objects $\langle -, \dots, = \rangle$.
- Counit $\varpi^{(i)}_{t_\bullet} : \pi_i[\langle t_1, \dots, t_n\rangle] \stackrel{\cong}{\Longrightarrow} t_i$
- For every $(\alpha_i : \pi_i[u] \Rightarrow t_i)_{i=1,\dots,n}$,

$$\begin{tikzcd} & \pi_i[\langle t_1,\dots,t_n\rangle] \arrow[dr, "\varpi^{(i)}_{t_\bullet}"] & \\ \pi_i[u] \arrow[ur, dashed, "{\pi_i[\mathsf{p}^\dagger(\alpha_1,\dots,\alpha_n)]}", "\exists!"'] \arrow[rr, "\alpha_i"'] & & t_i \end{tikzcd}$$

Equivalences

$$\begin{tikzcd} \mathcal{B}(\Gamma, \prod_n(A_1, \ldots, A_n)) \arrow[rr, bend left, "(\pi_1\{-\},\ldots,\pi_n\{-\})"] \arrow[rr, phantom, "\perp\simeq"] & & \prod_{i=1}^n \mathcal{B}(\Gamma, A_i) \arrow[ll, bend left, "\langle -,\ldots,= \rangle"] \end{tikzcd}$$

Tupling map

$$\frac{(\Gamma \vdash t_i : A_i)_{i=1,\ldots,n}}{\Gamma \vdash \mathsf{tup}(t_1 \ldots, t_n) : \prod_n(A_1, \ldots, A_n)}$$

A type theory for fp-bicategories = internal language of free fp-biclone

Equivalences
$$
\begin{tikzcd}
\mathcal{B}(\Gamma, \prod_n(A_1, \dots, A_n)) \arrow[rr, bend left, "(\pi_1\{-\}, \dots, \pi_n\{-\})"] \arrow[rr, phantom, "\perp\simeq"] & & \prod_{i=1}^{n} \mathcal{B}(\Gamma, A_i) \arrow[ll, bend left, "\langle -, \dots, = \rangle"]
\end{tikzcd}
$$

Tupling map
$$\frac{(\Gamma \vdash t_i : A_i)_{i=1,\dots,n}}{\Gamma \vdash \mathrm{tup}(t_1 \dots, t_n) : \prod_n(A_1, \dots, A_n)}$$

Counit (β-law)
$$\frac{(\Gamma \vdash t_i : A_i)_{i=1,\dots,n}}{\Gamma \vdash \varpi^{(k)}_{t_\bullet} : \pi_k\{\mathrm{tup}(t_1 \dots, t_n)\} \stackrel{\cong}{\Longrightarrow} t_k : A_k} \; (1 \leqslant k \leqslant n)$$

A type theory for fp-bicategories = internal language of free fp-biclone

Equivalences
$$\begin{tikzcd} \mathcal{B}(\Gamma, \prod_n(A_1, \dots, A_n)) \arrow[r, bend left, "(\pi_1\{-\},\dots,\pi_n\{-\})"] \arrow[r, phantom, "\perp\simeq"] & \prod_{i=1}^n \mathcal{B}(\Gamma, A_i) \arrow[l, bend left, "\langle -,\dots,=\rangle"] \end{tikzcd}$$

Tupling map
$$\frac{(\Gamma \vdash t_i : A_i)_{i=1,\dots,n}}{\Gamma \vdash \mathrm{tup}(t_1 \dots, t_n) : \prod_n(A_1, \dots, A_n)}$$

Counit (β-law)
$$\frac{(\Gamma \vdash t_i : A_i)_{i=1,\dots,n}}{\Gamma \vdash \varpi^{(k)}_{t_\bullet} : \pi_k\{\mathrm{tup}(t_1 \dots, t_n)\} \overset{\cong}{\Longrightarrow} t_k : A_k} \ (1 \leqslant k \leqslant n)$$

Mediating 2-cell
$$\frac{(\Gamma \vdash \alpha_i : \pi_i\{u\} \Rightarrow t_i : A_i)_{i=1,\dots,n}}{\Gamma \vdash \mathrm{p}^\dagger(\alpha_1, \dots, \alpha_n) : u \Rightarrow \mathrm{tup}(t_1, \dots, t_n) : \prod_n(A_1, \dots, A_n)}$$

+ three equational rules. $\rightsquigarrow$ η-law is derivable

the free fp-bicategory on G
$\simeq$ free fp-biclone on G, restricted to unary maps

the free fp-bicategory on G
$\simeq$ free fp-biclone on G, restricted to unary maps

syntax for fp-bicategories

$\shortparallel$

syntax for free fp-biclone, restricted to unary contexts

the free fp-bicategory on $\mathcal{G}$
$\simeq$ free fp-biclone on $\mathcal{G}$, restricted to unary maps

syntax for fp-bicategories

$$\parallel$$

syntax for free fp-biclone, restricted to unary contexts

'internal language of fp-bicategories' [informal]

$$\parallel$$

internal language [formal] of the free fp-biclone

The recipe for $\Lambda_{\mathrm{ps}}^{\times,\to}$

1. Define cc-biclones,
2. Extract internal language $\Lambda_{\mathrm{ps}}^{\times,\to}$ of free cc-biclone,
3. Restrict to unary contexts

   $\rightsquigarrow$ get internal language of cc-bicategories.

The recipe for $\Lambda_{\mathrm{ps}}^{\times,\rightarrow}$

1. Define cc-biclones,
2. Extract internal language $\Lambda_{\mathrm{ps}}^{\times,\rightarrow}$ of free cc-biclone,
3. Restrict to unary contexts
   $\rightsquigarrow$ get internal language of cc-bicategories.

'internal language for cc-bicategories' [informal]

$\parallel$

internal language [formal] of free cc-biclone

$\Lambda_{\mathrm{ps}}^{\times,\to}$ as STLC up-to-isomorphism

$\Lambda_{\mathrm{ps}}^{\times,\rightarrow}$ as STLC up-to-isomorphism

$\rightsquigarrow$ Rewrites in $\Lambda_{\mathrm{ps}}^{\times,\rightarrow}$ witness $\beta\eta$-equalities

$\Lambda_{\mathrm{ps}}^{\times,\to}$ as STLC up-to-isomorphism

$\rightsquigarrow$ Rewrites in $\Lambda_{\mathrm{ps}}^{\times,\to}$ witness $\beta\eta$-equalities

STLC-terms embed into $\Lambda_{\mathrm{ps}}^{\times,\to}$

$\Lambda_{\mathrm{ps}}^{\times,\to}$-terms evaluate to STLC-terms $\overline{t\{x\mapsto u\}} := \overline{t}[x\mapsto\overline{u}]$

$\Lambda_{\mathrm{ps}}^{\times,\to}$ as STLC up-to-isomorphism

$\rightsquigarrow$ Rewrites in $\Lambda_{\mathrm{ps}}^{\times,\to}$ witness $\beta\eta$-equalities

STLC-terms embed into $\Lambda_{\mathrm{ps}}^{\times,\to}$

$\Lambda_{\mathrm{ps}}^{\times,\to}$-terms evaluate to STLC-terms $\quad \overline{t\{x \mapsto u\}} := \overline{t}[x \mapsto \overline{u}]$

$$\left(\text{STLC-terms } \Gamma \vdash t : B\right) \big/_{\beta\eta} \cong \left(\Lambda_{\mathrm{ps}}^{\times,\to}\text{-terms } \Gamma \vdash t : B\right) \big/_{\cong_B^\Gamma}$$

$$t \cong_B^\Gamma t' \overset{\text{def.}}{\iff} \Gamma \vdash \tau : t \overset{\cong}{\Longrightarrow} t' : B \text{ for some invertible } \tau$$

$\Lambda_{\mathrm{ps}}^{\times,\to}$ as STLC up-to-isomorphism

$\rightsquigarrow$ Rewrites in $\Lambda_{\mathrm{ps}}^{\times,\to}$ witness $\beta\eta$-equalities

STLC-terms embed into $\Lambda_{\mathrm{ps}}^{\times,\to}$

$\Lambda_{\mathrm{ps}}^{\times,\to}$-terms evaluate to STLC-terms $\quad \overline{t\{x\mapsto u\}} := \overline{t}[x\mapsto\overline{u}]$

$$\left(\text{STLC-terms } \Gamma \vdash t : B\right)\big/_{\beta\eta} \cong \left(\Lambda_{\mathrm{ps}}^{\times,\to}\text{-terms } \Gamma \vdash t : B\right)\big/_{\cong_B^\Gamma}$$

$$(\Gamma \vdash t =_{\beta\eta} t' : B) \iff (\Gamma \vdash \tau : (\!|\, t \,|\!) \overset{\cong}{\Rightarrow} (\!|\, t' \,|\!) : B)$$

$$t \cong_B^\Gamma t' \overset{\text{def.}}{\iff} \Gamma \vdash \tau : t \overset{\cong}{\Rightarrow} t' : B \text{ for some invertible } \tau$$

$\Lambda_{\mathrm{ps}}^{\times,\to}$ as a logic of program transformations

$$\left(\text{STLC-terms } \Gamma \vdash t : B\right)\Big/_{\beta\eta} \cong \left(\Lambda_{\mathrm{ps}}^{\times,\to}\text{-terms } \Gamma \vdash t : B\right)\Big/_{\cong^{\Gamma}_{B}}$$

$$\left(\Gamma \vdash t =_{\beta\eta} t' : B\right) \iff \left(\Gamma \vdash \tau : (\!|\, t \,|\!) \xRightarrow{\cong} (\!|\, t' \,|\!) : B\right)$$

$\Lambda_{\mathrm{ps}}^{\times,\to}$ as a logic of program transformations

$$\left(\text{STLC-terms } \Gamma \vdash t : B\right)\Big/_{\beta\eta} \cong \left(\Lambda_{\mathrm{ps}}^{\times,\to}\text{-terms } \Gamma \vdash t : B\right)\Big/_{\cong_B^\Gamma}$$

$$(\Gamma \vdash t =_{\beta\eta} t' : B) \iff (\Gamma \vdash \tau : (\!|\, t \,|\!) \overset{\cong}{\Longrightarrow} (\!|\, t' \,|\!) : B)$$

$\Lambda_{\mathrm{ps}}^{\times,\to}$ as a logic of program transformations

$$\left(\text{STLC-terms } \Gamma \vdash t : B\right)\Big/_{\beta\eta} \;\cong\; \left(\Lambda_{\mathrm{ps}}^{\times,\to}\text{-terms } \Gamma \vdash t : B\right)\Big/_{\cong^{\Gamma}_{B}}$$

$$(\Gamma \vdash t =_{\beta\eta} t' : B) \iff (\Gamma \vdash \tau : (\!| t |\!) \overset{\cong}{\Longrightarrow} (\!| t' |\!) : B)$$

$\rightsquigarrow$ Equational theory = quotient out **loops** in rewriting:

$$\begin{tikzcd}
& \pi_i\{\langle \pi_1 \circ t, \ldots, \pi_n \circ t \rangle\} \arrow[dr, Rightarrow, "\beta"] & \\
\pi_i\{t\} \arrow[ur, Rightarrow, "\pi_i(p)\{p \mapsto \eta\}"] \arrow[rr, equal] & & \pi_i\{t\}
\end{tikzcd}$$

$\Lambda_{\mathrm{ps}}^{\times,\to}$ as a logic of program transformations

$$\left(\text{STLC-terms } \Gamma \vdash t : B\right)\Big/_{\beta\eta} \cong \left(\Lambda_{\mathrm{ps}}^{\times,\to}\text{-terms } \Gamma \vdash t : B\right)\Big/_{\cong_B^\Gamma}$$

$$(\Gamma \vdash t =_{\beta\eta} t' : B) \iff (\Gamma \vdash \tau : (\!| t |\!) \overset{\cong}{\Rightarrow} (\!| t' |\!) : B)$$

$\rightsquigarrow$ Equational theory = quotient out **loops** in rewriting:

$$\begin{tikzcd}
 & \pi_i\{\langle \pi_1 \circ t, \dots, \pi_n \circ t\rangle\} \arrow[dr, Rightarrow, "\beta"] & \\
\pi_i\{t\} \arrow[ur, Rightarrow, "\pi_i(p)\{p \mapsto \eta\}"] \arrow[rr, equal] & & \pi_i\{t\}
\end{tikzcd}$$

$\rightsquigarrow$ With **coherence**: rewrites unique mod $\equiv$

Using $\Lambda_{\mathrm{ps}}^{\times,\to}$ to prove coherence [LICS'20, thesis]

Coherence à la Mac-Lane: 'all diagrams commute'

Using $\Lambda_{\mathrm{ps}}^{\times,\rightarrow}$ to prove coherence [LICS'20, thesis]

Coherence à la Mac-Lane: $\boxed{\text{if } \sigma, \tau : t \Rightarrow t' \text{ then } \sigma \equiv \tau}$

Using $\Lambda_{\mathrm{ps}}^{\times,\rightarrow}$ to prove coherence [LICS'20, thesis]

Coherence à la Mac-Lane: $\boxed{\text{if } \sigma, \tau : t \Rightarrow t' \text{ then } \sigma \equiv \tau}$

Two proofs for cc-bicategories:

1. via the Yoneda embedding,
2. prove coherence of $\Lambda_{\mathrm{ps}}^{\times,\rightarrow}$

Using $\Lambda_{\mathrm{ps}}^{\times,\rightarrow}$ to prove coherence [LICS'20, thesis]

Coherence à la Mac-Lane: $\boxed{\text{if } \sigma, \tau : t \Rightarrow t' \text{ then } \sigma \equiv \tau}$

Two proofs for cc-bicategories:

1. via the Yoneda embedding,
2. prove coherence of $\Lambda_{\mathrm{ps}}^{\times,\rightarrow}$

Strategy:

bicategorify Fiore's semantic normalisation-by-evaluation for STLC

Summary and future questions

- STLC is the internal language of the free cc-clone.

Summary and future questions

- STLC is the internal language of the free cc-clone.
- To construct an internal language for cc-bicategories:
    1. Define cc-biclones,
    2. The internal language of cc-bicategories $\Lambda_{\mathrm{ps}}^{\times,\to}$ is the internal language of the free cc-biclone.

Summary and future questions

- STLC is the internal language of the free cc-clone.
- To construct an internal language for cc-bicategories:
  1. Define cc-biclones,
  2. The internal language of cc-bicategories $\Lambda_{\mathrm{ps}}^{\times,\rightarrow}$ is the internal language of the free cc-biclone.
- $\Lambda_{\mathrm{ps}}^{\times,\rightarrow}$ is a calculus for $\beta\eta$-rewriting modulo a specific equational theory.

Summary and future questions

- STLC is the internal language of the free cc-clone.
- To construct an internal language for cc-bicategories:
    1. Define cc-biclones,
    2. The internal language of cc-bicategories $\Lambda_{\mathrm{ps}}^{\times,\rightarrow}$ is the internal language of the free cc-biclone.
- $\Lambda_{\mathrm{ps}}^{\times,\rightarrow}$ is a calculus for $\beta\eta$-rewriting modulo a specific equational theory.

- Refine NbE argument to extract canonical normal forms.
- Does a similar argument work for the linear case?
- What does this say about bicategorical models of LL?
- What denotational meaning can we give to rewrites?
- Proof-theoretic implications?

$\boxed{\text{1-cells}}$

$$\mathrm{eval}_{A,B} : (A \Rightarrow B) \times A \to B$$

$\boxed{\text{Adjoint equivalences}}$

$$\begin{tikzcd} \mathcal{B}(X, A \Rightarrow B) \arrow[r, bend left, "\mathrm{eval}_{A,B} \circ (- \times A)"] \arrow[r, phantom, "\perp\simeq"] & \mathcal{B}(X \times A, B) \arrow[l, bend left, "\lambda"] \end{tikzcd}$$

Rules for exponentials

explicit weakening by x

free variable in context

$$\epsilon \bullet \mathsf{eval}\left\{(-)\left\{\mathrm{inc}_x\right\}, x\right\} \quad \frac{\overset{(x:A)}{\mathsf{eval}\left\{u\left\{\mathrm{inc}_x\right\}, x\right\} \Rightarrow t : B}}{u \Rightarrow \lambda x.t : A \Rrightarrow B} \quad e^{\dagger}(x\,.\,-)$$

$$\frac{}{f : A \Rrightarrow B, x : A \vdash \mathsf{eval}(f, x) : B}\ \mathsf{eval} \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \Rrightarrow B}\ \mathsf{lam}$$

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma, x : A \vdash \epsilon_t : \mathsf{eval}\left\{(\lambda x.t)\left\{\mathrm{inc}_x\right\}, x\right\} \Rightarrow t : B}\ \epsilon\text{-intro } (\beta\text{-rule})$$

$$\frac{\Gamma, x : A \vdash t : B \qquad \Gamma \vdash u : A \Rrightarrow B \qquad \Gamma, x : A \vdash \alpha : \mathsf{eval}\left\{u\left\{\mathrm{inc}_x\right\}, x\right\} \Rightarrow t : B}{\Gamma \vdash e^{\dagger}(x\,.\,\alpha) : u \Rightarrow \lambda x.t : A \Rrightarrow B}\ e^{\dagger}(x\,.\,\alpha)\text{-intro}$$

+ three equational rules $\rightsquigarrow$ η-rule derivable