Mathematical and Computational Metatheory of Second-Order Algebraic Theories

Marcelo Fiore
University of Cambridge

HoTTest
17.II.2022

joint work with Dima Szamozvancev

Algebraic Type Theory

$\boxed{?}$ What are type theories? — in logic and computer science

- Foundational — mathematical models
- Pragmatical — formalisation and programming

Working hypothesis: — categorical algebra

Type theories are algebraic systems

Example: Equational Logic of Universal Algebra

[Birkhoff]

	Equational Logic
types	unstructured sorts
terms	(first order) algebraic

Example: Equational Logic of Universal Algebra

Second-Order Equational Logic

	Equational Logic	Second-Order Equational Logic
types	unstructured sorts	(first order) algebraic
terms	(first order) algebraic	second-order

conservative extension

Second-Order Algebraic Theories

[F. & Hur]
[F. & Mahmoud]

Equational/Rewriting deduction system for languages with

type structure: (first-order) algebraic

term structure: (second-order)

- variable-binding operators [Frege]
- parameterised metavariables [Aczel]

Plan

- Second-order algebraic theories by example.
- Mathematical foundations
  vs. computer formalisation
- New mathematical modelling
  & computer implementation:
  - syntax: binding; metavariables
  - substitutions: capture avoiding; meta
  - semantics

Propositional Logic

```
syntax PropLog

type
  * : 0-ary

term
  true  : *
  false : *
  and   : *  *  ->  *
  or    : *  *  ->  *
  not   : *  ->  *

theory

  P : *  |>  and( P , P ) = P

  P : *  |>  not(not( P )) = P

  P,Q : *  |>  not( and(P,Q) ) = or( not(P) , not(Q) )
```

First-Order Logic

```
syntax FOL

type

  * : 0-ary
  N : 0-ary

term

  all :  N.*  ->  *
  ex  :  N.*  ->  *

theory

  P : N.*   |>  not( all(x.P[x]) ) = ex(x. not(P[x]) )

  P,Q : N.*   |>  all(x. and( P[x] , Q[x] ) ) = and( all(x.P[x]) , all(x.Q[x]) )

  P : N.*  Q : *   |>  or( all(x.P[x]) , Q ) = all(x. or( P[x] , Q ))

  P : N.* , n: N   |>  all( x.P[x] ) = and( P[n] , all(x.P[x]) )
```

$$F : N.* \ , \ Q : * \ \vdash \ \exists(x.F[x]) \Longrightarrow Q \ \approx \ \forall(x.\ F[x] \Longrightarrow Q)$$

$$
\begin{aligned}
& \exists(x.\ F[x]) \Longrightarrow Q \\
& \quad \equiv \ \neg(\ \exists(x.F[x])\ ) \lor Q \\
& \quad \approx \ \forall(x.\ \neg\ F[x]\ ) \lor Q \\
& \quad \approx \ \forall(x.\ \neg\ F[x] \lor Q\ ) \qquad (*) \\
& \quad \equiv \ \forall(x.\ F[x] \Longrightarrow Q\ )
\end{aligned}
$$

```
  ∀(x. ¬ F[x] ) ∨ Q  ≋  ∀(x. ¬ F[x] ∨ Q )

is derived from the axiom

  P : N.*  Q : *  ⊢  ∀( x.P[x] ) ∨ Q  ≋  ∀( x. P[x] ∨ Q )

by metasubstituting

  { P :=  y. ¬ F[y] }

intuitively as follows

  ∀( x. P[x] ){ P :=  y. ¬ F[y] }

    ≡  ∀( x. P[x]{ P :=  y. ¬ F[y] } )

    ≡  ∀( x. (¬ F[y]){ x/y } )

    ≡  ∀( x. ¬( F[y]{ x/y } ) )

    ≡  ∀( x. ¬ F[ y{ x/y } ] )

    ≡  ∀( x. ¬ F[x] )
```

Partial Differentiation

```
syntax PDiff

type
  R : 0-ary

term
  zero : R                    |  𝟘
  add  : R  R  ->  R          | _⊕_
  neg  : R  ->  R             |  ⊖_
  one  : R                    |  𝟙
  mult : R  R  ->  R          | _⊗_

  pd   : R.R  R  ->  R        | ∂_|_
```

[Plotkin]

```
theory

  f : (R,R).R , g,h : R.R |> z : R

    |-    pd( x.f[g[x],h[x]] , z )
       =
          add(
            mult( pd( x.f[x,h[z]] , g[z] ) , pd( x.g[x] , z ) )
            ,
            mult( pd( y.f[g[z],y] , h[z] ) , pd( x.h[x] , z ) )
            )


[ R · R ⊩ R ] [ R ⊩ R ] [ R ⊩ R ] ▹ ⌊ R ⌋

  ⊢     ∂₀ 𝔞⟨ 𝔟⟨x₀⟩ ◂ 𝔠⟨x₀⟩ ⟩
    ≈ₐ
      ( ∂ 𝔞⟨ x₀ ◂ 𝔠⟨x₁⟩ ⟩ | 𝔟⟨x₀⟩ ) ⊗ (∂₀ 𝔟⟨x₀⟩)
     ⊕
      ( ∂ 𝔞⟨ 𝔟⟨x₁⟩ ◂ x₀ ⟩ | 𝔠⟨x₀⟩ ) ⊗ (∂₀ 𝔠⟨x₀⟩)
```

```
-- Unary chain rule

∂₀ 𝔞⟨ 𝔟⟨ x₀ ⟩ ⟩

  ≋⟨ ax ∂Ch₂ with 《 𝔞⟨ x₀ ⟩ ◃ 𝔟⟨ x₀ ⟩ ◃ 𝟘 》 ⟩

      ( ∂ 𝔞⟨ x₀ ⟩ ∣ 𝔟⟨ x₀ ⟩ ) ⊗ (∂₀ 𝔟⟨ x₀ ⟩ )
    ⊕
      ( ∂ 𝔞⟨ 𝔟⟨ x₁ ⟩ ⟩ ∣ 𝟘 ) ⊗ (∂₀ 𝟘)

  ≋⟨ cong[ thm ∂𝟘 ] inside
       (∂ 𝔞⟨ x₀ ⟩ ∣ 𝔟⟨ x₀ ⟩) ⊗ (∂₀ 𝔟⟨ x₀ ⟩) ⊕ ((∂ 𝔞⟨ 𝔟⟨ x₁ ⟩ ⟩ ∣ 𝟘) ⊗ ◌ᶜ) ⟩

      ( ∂ 𝔞⟨ x₀ ⟩ ∣ 𝔟⟨ x₀ ⟩ ) ⊗ (∂₀ 𝔟⟨ x₀ ⟩)
    ⊕
      ( ∂ 𝔞⟨ 𝔟⟨ x₁ ⟩ ⟩ ∣ 𝟘 ) ⊗ 𝟘

  ≋⟨ cong[ thm 𝟘X⊗ᴿ with 《 (∂ 𝔞⟨ (𝔟⟨ x₁ ⟩) ⟩ ∣ 𝟘) 》 ] inside
       (∂ 𝔞⟨ x₀ ⟩ ∣ 𝔟⟨ x₀ ⟩) ⊗ (∂₀ 𝔟⟨ x₀ ⟩) ⊕ ◌ᶜ ⟩

      ( ∂ 𝔞⟨ x₀ ⟩ ∣ 𝔟⟨ x₀ ⟩ ) ⊗ (∂₀ 𝔟⟨ x₀ ⟩)
    ⊕
      𝟘

  ≋⟨ thm 𝟘U⊕ᴿ with 《 (∂ 𝔞⟨ x₀ ⟩ ∣ 𝔟⟨ x₀ ⟩) ⊗ (∂₀ 𝔟⟨ x₀ ⟩) 》 ⟩

    (∂ 𝔞⟨ x₀ ⟩ ∣ 𝔟⟨ x₀ ⟩) ⊗ ∂₀ 𝔟⟨ x₀ ⟩
```

Formal Metatheory of Second-Order Abstract Syntax

Marcelo Fiore and Dmitrij Szamozvancev. 2022. Formal Metatheory of Second-Order Abstract Syntax. *Proc. ACM Program. Lang.* 6, POPL, Article 53 (January 2022), 29 pages. https://doi.org/10.1145/3498715

We present a mathematically-inspired language-formalisation framework implemented in Agda. The system translates the description of a syntax signature with variable-binding operators into an intrinsically-encoded, inductive data type equipped with syntactic operations such as weakening and substitution, along with their correctness properties. The generated metatheory further incorporates metavariables and their associated operation of metasubstitution, which enables second-order equational/rewriting reasoning. The underlying mathematical foundation of the framework – initial algebra semantics – derives compositional interpretations of languages into their models satisfying the semantic substitution lemma by construction.

Applications:

- Formalisation and reasoning
  - Logic | calculi
  - Programming | calculi
- Symbolic computation
  - Deduction systems | rapid prototyping
  - Programing calculi | rapid prototyping
  - Abstract machines | formal translations

Mathematical Theory [F. & Plotkin & Turi]

Presheaf Model of Abstract Syntax with Variable Binding

- Types : T
- Category of contexts and renamings : $\mathbb{F}[T]$
- Universe of discourse : $\mathcal{T} \in \left(\underline{\underline{\mathrm{Set}}}^{\mathbb{F}[T]}\right)^T$

$\mathcal{T}_\alpha(\Gamma) \sim$ terms of type α in context Γ

Example: Presheaf of variables

$$V_\alpha(\Gamma) = \mathbb{F}[T](\langle\alpha\rangle, \Gamma)$$

- Combinatorial constructions

sums $\qquad$ products $\qquad$ powers to representables

$$\coprod_i \mathcal{T}_i \quad , \quad \prod_i \mathcal{T}_i \quad , \quad \mathcal{T}_\beta^{\,y\alpha}$$

Terms of Type β in a context extended by α

$$\mathcal{T}_\beta^{\,y\alpha}(\Gamma) = \mathcal{T}_\beta(\alpha \cdot \Gamma)$$

- Combinatorial constructions

sums, products, powers to representables

$$\coprod_i \mathcal{T}_i \quad , \quad \prod_i \mathcal{T}_i \quad , \quad \mathcal{T}_\beta^{\,\mathcal{V}_\alpha}$$

Terms of Type β in a context extended by α

$$\mathcal{T}_\beta^{\,\mathcal{V}_\alpha}(\Gamma) = \mathcal{T}_\beta(\alpha \cdot \Gamma)$$

Example:

$$t ::= x \mid t_1 @ t_2 \mid \lambda x . t$$

$$\mathcal{T} \cong \mathcal{V} + \mathcal{T} \times \mathcal{T} + \mathcal{T}^{\mathcal{V}}$$

Initial-Algebra Semantics

[ADJ]

Syntax signatures $\rightsquigarrow$ Polynomial functors

Abstract syntax $\rightsquigarrow$ Initial algebras

- universal representation
- compositional semantics
- induction principles

Initial-Algebra Semantics

[ADJ]

Syntax signatures ⇝ Polynomial functors

Abstract syntax ⇝ Initial algebras
- universal representation
- compositional semantics
- induction principles

Thesis

* Type-theoretic rules are syntactic descriptions of polynomial diagrams.
* The abstract syntax is the initial algebra for the associated polynomial functor.

Framework

syntax

second-order signature $\Sigma \rightsquigarrow P_\Sigma \circlearrowright \left(\underline{\underline{\mathrm{Set}}}^{\mathbb{F}[T]}\right)^T$

$$\begin{tikzcd} P_\Sigma\text{-}\underline{\underline{\mathrm{Alg}}} \arrow[d, bend right, ""'] \arrow[d, phantom, "\dashv"] \\ \left(\underline{\underline{\mathrm{Set}}}^{\mathbb{F}[T]}\right)^T \arrow[u, ""'] \end{tikzcd}$$

semantics

$T_\Sigma \circlearrowright \left(\underline{\underline{\mathrm{Set}}}^{\mathbb{F}[T]}\right)^T$

term monad.

Computer Implementation Approaches

- In programming languages
  - Ad hoc
  - No correctness guarantees.
- In proof assistants
  - Strong invariants for correctness
  - computation

Mathematics vs. Proof Assistant

- formalise the mathematical model in the proof assistant
  - computationally problematic
- adapt the mathematics to the proof assistant

TENSION

presheaves $\longleftrightarrow$ families

$\left(\underline{\underline{\mathrm{Set}}}^{\mathbb{F}[T]}\right)^T$ $\left(\underline{\underline{\mathrm{Set}}}^{T^*}\right)^T$

Adjoint Modalities

monad $\Diamond \dashv \Box$ comonad

$$
\begin{tikzcd}
\mathbb{C}_0 \arrow[r, hook] \arrow[d] & \underline{\underline{\mathrm{Set}}}^{\mathbb{C}_0} \arrow[loop left] \arrow[loop right] \arrow[d, bend right=50, "\blacklozenge"'] \arrow[d, bend left=50, "\blacksquare"] \\
\mathbb{C} \arrow[r, hook] & \underline{\underline{\mathrm{Set}}}^{\mathbb{C}^{op}} \arrow[u]
\end{tikzcd}
$$

families

presheaves

$\underline{\underline{\mathrm{Set}}}^{\mathbb{C}^{op}} \cong \Diamond\text{-}\underline{\mathrm{Alg}}$ (actions)

$\underline{\underline{\mathrm{Set}}}^{\mathbb{C}^{op}} \cong \Box\text{-}\underline{\mathrm{coAlg}}$ (coactions)

$\blacklozenge F(\Gamma) = \sum_{\Delta} F(\Delta) \times \mathbb{C}(\Gamma, \Delta)$

$\blacksquare F(\Gamma) = \prod_{\Delta} \mathbb{C}(\Delta, \Gamma) \Rightarrow F\Delta$

Presheaves as Indexed Families with Algebraic Model Structure in Proof Assistants?

Internal languages for presheaves compiled to families via adjoint modalities?

Families $\left(\underline{Set}^{T^*}\right)^T$ as universe of discourse

- Names: $N_\alpha(\Gamma) = (\langle\alpha\rangle \equiv \Gamma)$
- Combinatorial constructions

$$\coprod_i F_i \quad , \quad \prod_i F_i$$

Families $(\underline{Set}^{T^*})^T$ as universe of discourse

- Names: $N_\alpha(\Gamma) = (\langle\alpha\rangle \equiv \Gamma)$
- Combinatorial constructions

$$\coprod_i F_i \quad , \quad \prod_i F_i$$

Day convolution

$$(F \otimes G)(\Gamma) = \sum_{\Gamma = \Gamma_1 + \Gamma_2} F(\Gamma_1) \times G(\Gamma_2)$$

$$(F \multimap G)(\Gamma) = \prod_{\Delta} F(\Delta) \Rightarrow G(\Gamma + \Delta)$$

Calculus

- $V = \blacklozenge N$ variables are freely generated from names

  $I = |V|$ indices

- $N_\alpha \multimap \mathcal{F}_\beta = \mathcal{F}_\beta(\alpha \cdot -)$ context extension

- $| P^{\blacklozenge X} | = X \multimap |P|$

- $\blacklozenge(\mathcal{F}) \times \blacklozenge(\mathcal{G}) = \blacklozenge(\mathcal{F} \otimes \mathcal{G})$

- $(\blacksquare \mathcal{F})^{\tau} = \blacksquare(\mathcal{F}^{|\tau|})$

Initial Algebra Semantics

Example: λ-calculus $\quad t ::= x \mid t_1 @ t_2 \mid \lambda x.t$

in presheaves

$$\mathcal{T} \cong V + \mathcal{T} \times \mathcal{T} + \mathcal{T}^V$$

$$\big\downarrow$$

$$|\mathcal{T}| \cong |V| + |\mathcal{T}| \times |\mathcal{T}| + |\mathcal{T}^{\blacklozenge N}|$$

$$\cong I + |\mathcal{T}| \times |\mathcal{T}| + N \multimap |\mathcal{T}|$$

$$\big\downarrow$$

in families

$$\mathcal{F} \cong I + \mathcal{F} \times \mathcal{F} + N \multimap \mathcal{F}$$

$$
\begin{tikzcd}
\left(\underline{\underline{Set}}^{\mathbb{F}[T]}\right)^{T} \arrow[r, "P_\Sigma"] \arrow[d] & \left(\underline{\underline{Set}}^{\mathbb{F}[T]}\right)^{T} \arrow[d] \\
\left(\underline{\underline{Set}}^{T^*}\right)^{T} \arrow[r, "P_\Sigma^{\downarrow}"'] & \left(\underline{\underline{Set}}^{T^*}\right)^{T}
\end{tikzcd}
$$

Initial $P_\Sigma^{\downarrow}$-algebras lift to initial P_Σ-algebras

Initial-Algebra Lifting Theorem

$$\begin{tikzcd} S\text{-coAlg} \arrow[r, "F_\lambda"] \arrow[d] & S\text{-coAlg} \arrow[d] \\ \mathcal{E} \arrow[r, "F"'] & \mathcal{E} \end{tikzcd}$$

Initial F-algebras lift to initial F_λ-algebras

$$\begin{tikzcd} FA \arrow[rr, "\alpha \text{ initial}"] \arrow[d] & & A \arrow[d, "\exists!"'] \\ FSA \arrow[r, "\lambda"'] & SFA \arrow[r, "S\alpha"'] & SA \end{tikzcd}$$

coalgebra structure

```
syntax STLC | Λ

type
  N   : 0-ary
  _↣_ : 2-ary

term

  app :  α ↣ β  α  ->  β  | _$_

  lam :  α.β  ->  α ↣ β   | ƛ_
```

```
data Λ : Familyₛ where

  var  : ℐ ⇾ Λ

  _$_  : Λ (α ↣ β) Γ → Λ α Γ → Λ β Γ

  ƛ_   : Λ β (α • Γ) → Λ (α ↣ β) Γ
```

automatically generated intrinsically-typed encoding

[Altenkirch & Reus, Benton et al, Allais et al]

Substitution

Idea: metaoperation

$$\mathcal{T}_\alpha(\Gamma) \times \prod_{\gamma \in \Gamma} \mathcal{T}_\gamma(\Delta) \longrightarrow \mathcal{T}_\alpha(\Delta)$$

subject to axioms.

Substitution

Idea: metaoperation

$$\mathcal{T}_\alpha(\Gamma) \times \prod_{\gamma \in \Gamma} \mathcal{T}_\gamma(\Delta) \longrightarrow \mathcal{T}_\alpha(\Delta)$$

subject to axioms.

Presheaf model: monoid structure

$$\mathcal{T} \bullet \mathcal{T} \longrightarrow \mathcal{T}$$

for a substitution tensor product

$$(\mathcal{T} \bullet \mathcal{S})_\alpha(\Delta) = \int^{\Gamma} \mathcal{T}_\alpha(\Gamma) \times \prod_{\gamma \in \Gamma} \mathcal{S}_\alpha(\Delta)$$

Syntax with Substitution

$$\begin{tikzcd}
 & P_2(\mathcal{T}) \arrow[d, "\mathrm{con}"'] & \\
V \arrow[r, "\mathrm{var}"] & \mathcal{T} & \mathcal{T} \bullet \mathcal{T} \arrow[l, "\mathrm{sub}"']
\end{tikzcd}$$

compatibility condition

monoid structure

Syntax with Substitution

$$
\begin{tikzcd}
 & P_\Sigma(\mathcal{T}) \arrow[d, "con"'] & \\
V \arrow[r, "var"] & \mathcal{T} & \mathcal{T} \bullet \mathcal{T} \arrow[l, "sub"']
\end{tikzcd}
$$

compatibility condition

pointed strength

$$
\begin{tikzcd}
P_\Sigma(\mathcal{T}) \bullet \mathcal{T} \arrow[r] \arrow[d, "con \bullet id"'] & P_\Sigma(\mathcal{T} \bullet \mathcal{T}) \arrow[r, "P_\Sigma(sub)"] & P_\Sigma(\mathcal{T}) \arrow[d, "con"] \\
\mathcal{T} \bullet \mathcal{T} \arrow[rr, "sub"] & & \mathcal{T}
\end{tikzcd}
$$

Thm [Hamana, F.]

$$
\begin{tikzcd}
\Sigma\text{-Mon} \arrow[d, shift left] \\
\mathcal{G}\left(\mathbf{Set}^{\mathbb{F}[T]}\right)^{T} \arrow[u, bend left, shift left] \arrow[u, phantom, "\dashv"]
\end{tikzcd}
$$

$\mathcal{M}$

where the free Σ-monoid on $\mathcal{X}$ is an initial:

$$\mathcal{M}(\mathcal{X}) \cong V + \mathcal{X} \bullet \mathcal{M}(\mathcal{X}) + P_{\Sigma}(\mathcal{M}\mathcal{X})$$

- Provides inductively defined abstract syntax with variable binding operators and parameterised metavariables.

$$\mathcal{M}(\mathcal{X}) \cong \mathcal{V} + \mathcal{X} \bullet \mathcal{M}(\mathcal{X}) + P_{\Sigma}(\mathcal{M}\mathcal{X})$$

$$t \ ::= \ x \mid M[t_1, \ldots, t_n] \mid f(\ldots, \vec{x}.t', \ldots)$$

- Provides a provably-correct inductively-defined substitution operation, automatically preserved by semantic interpretations

Generalises type-theoretic/semantic practices

Free Algebras with Substitution in Families

- Skew substitution tensor product [Berthelle & Hirschowitz & Lafont]

$$(\mathcal{G} \cdot \mathcal{F})_\alpha(\Delta) = \sum_{\Gamma} \mathcal{G}_\alpha(\Gamma) \times \prod_{\gamma \in \Gamma} \mathcal{F}_\gamma(\Delta)$$

(with unit the family of indices)

NB: Monoids are, equivalently, abstract clones

Syntax with Substitution in Families

$$[\mathcal{F},\mathcal{G}]_\alpha(\Gamma) = \prod_{\Delta} \left(\prod_{\gamma \in \Gamma} \mathcal{F}_\gamma(\Delta) \right) \Rightarrow \mathcal{G}_\alpha(\Delta)$$

$$\begin{tikzcd}
 & P_\Sigma(\mathcal{F}) \arrow[r, "P_\Sigma(\underline{sub})"] \arrow[d, "\underline{con}"'] & P_\Sigma[\mathcal{F},\mathcal{F}] \arrow[r] & {[\mathcal{F}, P_\Sigma \mathcal{F}]} \arrow[d, "{[id, \underline{con}]}"] \\
I \arrow[r, "\underline{var}"] & \mathcal{F} \arrow[rr, "\underline{sub}"] & & {[\mathcal{F},\mathcal{F}]}
\end{tikzcd}$$

pointed strength

monoid (in closed form)

Thm: For every family $\mathcal{X} \in (\underline{\underline{Set}}^{T^*})^T$, the initial algebra $\mathcal{M}_\Sigma(\mathcal{X})$ with structure

$$\begin{cases} \underline{var} : I \to \mathcal{M}_\Sigma(\mathcal{X}) \\ \underline{mvar} : \mathcal{X} \to [\mathcal{M}_\Sigma(\mathcal{X}), \mathcal{M}_\Sigma(\mathcal{X})] \\ \underline{con} : P_\Sigma(\mathcal{X}) \to \mathcal{X} \end{cases}$$

is the free Σ-monoid on $\mathcal{X}$.

```
syntax STLC | Λ

type
  N   : 0-ary
  _↣_ : 2-ary

term

  app :  α ↣ β  α   ->   β   | _$_

  lam :  α.β   ->  α ↣ β     | ƛ_
```

```
module Λ:Terms (𝔛 : Familyₛ) where

  data Λ : Familyₛ where

    var  : ℐ ⇾ Λ

    mvar : 𝔛 α Π → Sub Λ Π Γ → Λ α Γ

    _$_  : Λ (α ↣ β) Γ → Λ α Γ → Λ β Γ

    ƛ_   : Λ β (α • Γ) → Λ (α ↣ β) Γ
```

automatically generated intrincically-typed encoding with metavaribles

proof ingredients:

- substitution operation

  $\underline{\text{sub}} : \mathcal{M}_\Sigma(\mathcal{X}) \longrightarrow [\mathcal{M}_\Sigma(\mathcal{X}), \mathcal{M}_\Sigma(\mathcal{X})]$

  induced by initiality

  requires $\Box$-coalgebra structure on $\mathcal{M}_\Sigma(\mathcal{X})$

  [recall the initial-algebra lifting thm]

proof ingredients:

- substitution operation

$$\mathcal{M}_\Sigma(\mathcal{X}) \to [\mathcal{M}_\Sigma(\mathcal{X}), \mathcal{M}_\Sigma(\mathcal{X})]$$

induced by initiality

requires $\Box$-coalgebra structure on $\mathcal{M}_\Sigma(\mathcal{X})$
[recall the initial-algebra lifting thm]

$$\begin{tikzcd}
I \bullet \mathcal{M}(\mathcal{X}) \arrow[d] \arrow[dr, bend left] & \\
\mathcal{M}(\mathcal{X}) \bullet \mathcal{M}(\mathcal{X}) \arrow[r, "sub"'] & \mathcal{M}(\mathcal{X})
\end{tikzcd}$$

$$\begin{tikzcd}
P(\mathcal{M}\mathcal{X}) \bullet \mathcal{M}(\mathcal{X}) \arrow[r] \arrow[d] & P(\mathcal{M}\mathcal{X} \bullet \mathcal{M}\mathcal{X}) \arrow[r, "Psub"] & P(\mathcal{M}\mathcal{X}) \arrow[d] \\
\mathcal{M}(\mathcal{X}) \bullet \mathcal{M}(\mathcal{X}) \arrow[rr, "sub"] & & \mathcal{M}\mathcal{X}
\end{tikzcd}$$

$$\begin{tikzcd}
(\mathcal{X} \bullet \mathcal{M}\mathcal{X}) \bullet \mathcal{M}\mathcal{X} \arrow[r] \arrow[d] & \mathcal{X} \bullet (\mathcal{M}\mathcal{X} \bullet \mathcal{M}\mathcal{X}) \arrow[r, "\mathcal{X} \bullet sub"] & \mathcal{X} \bullet \mathcal{M}(\mathcal{X}) \arrow[d] \\
\mathcal{M}\mathcal{X} \bullet \mathcal{M}\mathcal{X} \arrow[rr, "sub"] & & \mathcal{M}\mathcal{X}
\end{tikzcd}$$

proof ingredients:

- substitution operation

$$\mathcal{M}_\Sigma(\mathcal{X}) \longrightarrow [\mathcal{M}_\Sigma(\mathcal{X}), \mathcal{M}_\Sigma(\mathcal{X})]$$

induced by initiality

requires $\square$-coalgebra structure on $\mathcal{M}_\Sigma(\mathcal{X})$
[recall the initial-algebra lifting thm]

[McBride et al]

derived from general traversals

$$\mathcal{M}_\Sigma(\mathcal{X}) \longrightarrow [\mathcal{P}, \mathcal{A}]$$

parameterised semantic models

- substitution laws by initiality

$$
\begin{tikzcd}
 & {[\mathcal{M}_\Sigma(\mathcal{X}), \mathcal{M}_\Sigma(\mathcal{X})]} \arrow[d] \\
\mathcal{M}_\Sigma(\mathcal{X}) \arrow[ur, "\underline{sub}"] \arrow[d, "\underline{sub}"] & {[[\mathcal{M}_\Sigma(\mathcal{X}), \mathcal{M}_\Sigma(\mathcal{X})], [\mathcal{M}_\Sigma(\mathcal{X}), \mathcal{M}_\Sigma(\mathcal{X})]]} \arrow[d, "{[\underline{sub}, id]}"] \\
{[\mathcal{M}_\Sigma(\mathcal{X}), \mathcal{M}_\Sigma(\mathcal{X})]} \arrow[r, "{[id, \underline{sub}]}"'] & {[\mathcal{M}_\Sigma(\mathcal{X}), [\mathcal{M}_\Sigma(\mathcal{X}), \mathcal{M}_\Sigma(\mathcal{X})]]}
\end{tikzcd}
$$

crucially using the theory of traversals

Metasubstitution in Presheaves

Thm [F.]: In the presheaf model,

$\mathcal{M}_\Sigma$ is an enriched monad

$\rightsquigarrow$ induces an (inductively-defined provably correct internal) metasubstitution operation

$$\underline{\mathrm{msub}} : \mathcal{M}_\Sigma(\mathcal{X}) \times \left(\mathcal{M}_\Sigma(\mathcal{Y})\right)^{\mathcal{X}} \longrightarrow \mathcal{M}_\Sigma(\mathcal{Y})$$

Metasubstitution in Families

Linear internalisation

$$\underline{\mathsf{msub}} : \mathcal{M}_\Sigma(\mathcal{X}) \longrightarrow \left(\mathcal{X} \mathbin{-\!\!\bullet} \mathcal{M}_\Sigma(\mathcal{Y})\right) \multimap \mathcal{M}_\Sigma(\mathcal{Y})$$

may be induced by initiality

In elementary terms:

$$\mathcal{M}_\Sigma(\mathcal{X})(\Gamma_1) \to \Big(\prod_{\Delta} \mathcal{X}(\Delta) \Rightarrow \mathcal{M}_\Sigma(\mathcal{Y})(\Delta+\Gamma_2)\Big) \Rightarrow \mathcal{M}_\Sigma(\mathcal{Y})(\Gamma_1+\Gamma_2)$$

Metasubstitution recursion

- $I \longrightarrow (\mathcal{X} \mathbin{-\!\!\bullet} M_\Sigma y) \multimap M_\Sigma(y)$

  $i \longmapsto \sigma \longmapsto \underline{\mathrm{var}}(i)$

- $P_\Sigma((\mathcal{X} \mathbin{-\!\!\bullet} M_\Sigma y) \multimap M_\Sigma y) \longrightarrow (\mathcal{X} \mathbin{-\!\!\bullet} M_\Sigma y) \multimap M_\Sigma y$

  linear strength

  $(\mathcal{X} \mathbin{-\!\!\bullet} M_\Sigma y) \multimap P_\Sigma(M_\Sigma y)$

- $\mathcal{X} \xrightarrow{\mathrm{rec}} [(\mathcal{X} \mathbin{-\!\!\bullet} M_\Sigma y) \multimap M_\Sigma y, (\mathcal{X} \mathbin{-\!\!\bullet} M_\Sigma y) \multimap M_\Sigma y]$

  $m \longmapsto \varepsilon \longmapsto \sigma \longmapsto \underline{\mathrm{sub}}(\sigma m, [\varepsilon\sigma, \underline{\mathrm{wk}}])$

$$\underline{\mathrm{msub}}\,(\underline{\mathrm{mvar}}(m,t))\,\sigma$$

$$= \underline{\mathrm{sub}}\,(\sigma(m),\ [\,\underline{\mathrm{msub}}\ t\ \sigma\ ,\ \underline{\mathrm{wk}}\,])$$

$$\begin{tikzcd}
x \bullet m(x) \arrow[r, "\underline{\mathrm{rec}} \,\bullet\, \underline{\mathrm{msub}}"] \arrow[d, "\underline{\mathrm{mvar}}"'] & \begin{array}{c} [(x \mathbin{-\!\bullet} my) \multimap my,\ (x \mathbin{-\!\bullet} my) \multimap my] \\ \bullet\ ((x \mathbin{-\!\bullet} my) \multimap my) \end{array} \arrow[d, "\underline{\mathrm{eval}}"] \\
m(x) \arrow[r, "\underline{\mathrm{msub}}"'] & (x \mathbin{-\!\bullet} my) \multimap my
\end{tikzcd}$$

Second-order Equational Logic

```
variable
  α β γ : T
  Γ Δ Π : Ctx
  𝔐 𝔑 : MCtx

-- Second-order equational logic

module EqLogic ( _▹_⊢_≋ₐ_ : ∀ 𝔐 Γ {α} → (𝔐 ▷ 𝕋) α Γ → (𝔐 ▷ 𝕋) α Γ → Set ) where

data _▹_⊢_≋_ : (𝔐 : MCtx){α : T}(Γ : Ctx) → (𝔐 ▷ 𝕋) α Γ → (𝔐 ▷ 𝕋) α Γ → Set₁ where

  ax  : {t s : (𝔐 ▷ 𝕋) α Γ}
      → 𝔐 ▹ Γ ⊢ t ≋ₐ s
      -----------------
      →  𝔐 ▹ Γ ⊢ t ≋ s

  eq  : {t s : (𝔐 ▷ 𝕋) α Γ}
      → t ≡ s
      ----------------
      → 𝔐 ▹ Γ ⊢ t ≋ s

  sy  : {t s : (𝔐 ▷ 𝕋) α Γ}
      → 𝔐 ▹ Γ ⊢ t ≋ s
      ----------------
      → 𝔐 ▹ Γ ⊢ s ≋ t

  tr  : {t s u : (𝔐 ▷ 𝕋) α Γ}
      → 𝔐 ▹ Γ ⊢ t ≋ s
      → 𝔐 ▹ Γ ⊢ s ≋ u
      ----------------
      → 𝔐 ▹ Γ ⊢ t ≋ u

  □ms : {t s : (𝔐 ▷ 𝕋) α Γ}

      → 𝔐 ▹ Γ ⊢ t ≋ s
      → (ρ : Γ ↝ Δ)
      → (ζ ξ : MSub Δ 𝔐 𝔑)
      → (∀{τ Π}(𝔪 : Π ⊩ τ ∈ 𝔐) → 𝔑 ▹ (Π ∔ Δ) ⊢ (ix≀ ζ 𝔪) ≋ (ix≀ ξ 𝔪))
      -----------------------------------------------------------------
      → 𝔑 ▹ Δ ⊢ (□msub≀ t ρ ζ) ≋ (□msub≀ s ρ ξ)
```

- The laws of metasubstitution are approached by a decomposition:

$$
\begin{aligned}
&\mathcal{M}_\Sigma(x) \\
&\quad \xrightarrow{\mathcal{M}_\Sigma(\eta)} \mathcal{M}_\Sigma\big((x \mathbin{-\!\!\bullet} \mathcal{M}_\Sigma(y)) \multimap \mathcal{M}_\Sigma(y)\big) \\
&\quad\quad \xrightarrow{\mathrm{str}} (x \mathbin{-\!\!\bullet} \mathcal{M}_\Sigma(y)) \multimap \mathcal{M}_\Sigma \mathcal{M}_\Sigma(y) \\
&\quad\quad\quad \xrightarrow{\mathrm{id} \multimap \mu} (x \mathbin{-\!\!\bullet} \mathcal{M}_\Sigma(y)) \multimap \mathcal{M}_\Sigma(y)
\end{aligned}
$$

Conclusion

- Automatic generation of generic:
  - second-order abstract syntax
  - provably-correct substitution and meta substitution operations
  - algebraic models with compositional interpretations
- Agda implementation
  - for deduction and computation
  - mathematically inspired

Directions

- Application case studies
  - Simply-Typed Contextual Modal Type Theory [Nanevski & Pfenning & Pientka]
- Linearity

cartesian linear

cartesian with patterns dual linear/cartesian

- Reflection
- Polymorphism
- Type dependency