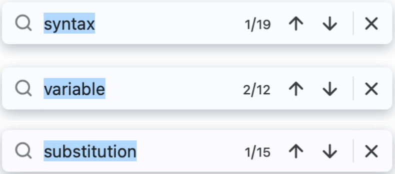
Functorial models of scope-safe syntax

Dima Szamozvancev
Department of Computer Science and Technology
University of Cambridge, UK

EuroProofNet WG6 Meeting
KU Leuven, 4 April 2024

UNIVERSITY OF CAMBRIDGE



WG6 meeting in Leuven: Schedule and Abstracts

To restate the obvious:
syntax formalisation is hard!

Challenges and choices

Encoding of variables
Atoms, numerals, indices, parameters

Representation of binding
Atom equality, de Bruijn, meta-level

Definition of substitution
Single-variable, simultaneous, explicit, nominal, de Bruijn

Formalisation of syntax
Intrinsic, extrinsic, higher-order, least fixed point

Semantic models

Axiomatisation of syntactic structure
Must account for constructors, variables, and substitution

Initiality proof
Syntax is the initial model

Semantic interpretations
Denotational semantics in any model of the syntax

Recursion and induction principles
Define operations and prove properties
on the syntax by instantiating a model

Example: natural numbers

$$n \quad ::= \quad \mathsf{Z} \quad | \quad \mathsf{S}\, n \qquad \in \mathsf{N}$$

Model is a set A with element $z \in A$ and function $s \colon A \to A$

$(\mathsf{N}, \mathsf{Z}, \mathsf{S})$ is the initial model: $[\![\mathsf{Z}]\!] = z$ and $[\![\mathsf{S}\, n]\!] = s[\![n]\!]$

Interpretations in semantic models
$(\mathbb{N}, 0, (-) + 1)$ induces $\mathsf{N} \to \mathbb{N}$, $[\![\mathsf{S}(\mathsf{S}\,\mathsf{Z})]\!] = 2 \in \mathbb{N}$
$(\mathbf{Set}, \mathbb{1}, \mathsf{Maybe})$ induces $\mathsf{N} \to \mathbf{Set}$, $[\![\mathsf{S}(\mathsf{S}\,\mathsf{Z})]\!] = \mathsf{Maybe}\,(\mathsf{Maybe}\,\mathbb{1})$

Recursion and induction principles
$(\mathsf{N} \to \mathsf{N}, \mathrm{id}, \mathsf{S} \circ -)$ induces $\mathsf{N} \to (\mathsf{N} \to \mathsf{N})$, $[\![\mathsf{S}^m\,\mathsf{Z}]\!] = \mathsf{S}^n\,\mathsf{Z} \mapsto \mathsf{S}^{m+n}\,\mathsf{Z}$
$(\mathsf{Bool}, \mathsf{true}, \mathsf{not})$ induces $\mathsf{N} \to \mathsf{Bool}$, $[\![\mathsf{S}^m\,\mathsf{Z}]\!] \iff m$ is even

Example(?): simply-typed λ-calculus

$$\begin{aligned} \alpha, \beta &::= \mathsf{B} \mid \alpha \to \beta \\ s, t &::= x \mid \lambda x : \alpha.\, b \mid t\, s \end{aligned}$$

Environment model in sets

Types are sets, contexts are cartesian products,
terms are functions $[\![\Gamma]\!] \longrightarrow [\![\alpha]\!]$

$$\begin{aligned} [\![\Gamma \vdash t : \alpha]\!] &: [\![\Gamma]\!] \longrightarrow [\![\alpha]\!] \\ [\![x_i]\!](\gamma) &= \gamma_i \\ [\![\lambda x : \alpha.\, b]\!](\gamma) &= a \mapsto [\![b]\!](\gamma, a) \\ [\![t\, s]\!](\gamma) &= [\![t]\!](\gamma)\,([\![s]\!](\gamma)) \end{aligned}$$

This is just a particular model of the STLC!

What are models of syntax?

The signature of the syntax is captured as an *endofunctor*

Sum-of-products encoding of the constructor argument

The algebraic datatype is the *initial algebra*

Initiality induces to semantic interpretations

Natural numbers

$$\mathsf{Z} : 1 \to \mathsf{N}$$
$$\mathsf{S} : \mathsf{N} \to \mathsf{N}$$

$$[\mathsf{Z}, \mathsf{S}] : (1 + \mathsf{N}) \to \mathsf{N}$$
$$[\mathsf{Z}, \mathsf{S}] : F_{\mathbb{N}}(\mathsf{N}) \to \mathsf{N}$$

$$F_{\mathbb{N}} : \mathbf{Set} \to \mathbf{Set}$$
$$F_{\mathbb{N}} \triangleq X \mapsto 1 + X$$

Binary trees

$$\mathsf{Lf} : A \to \mathsf{Tr}_A$$
$$\mathsf{Br} : \mathsf{Tr}_A \times \mathsf{Tr}_A \to \mathsf{Tr}_A$$

$$[\mathsf{Lf}, \mathsf{Br}] : (A + (\mathsf{Tr}_A \times \mathsf{Tr}_A)) \to \mathsf{Tr}_A$$
$$[\mathsf{Lf}, \mathsf{Br}] : F_{\mathsf{Tr}A}(\mathsf{Tr}_A) \to \mathsf{Tr}_A$$

$$F_{\mathsf{Tr}A} : \mathbf{Set} \to \mathbf{Set}$$
$$F_{\mathsf{Tr}A} \triangleq X \mapsto A + (X \times X)$$

Does this extend to endofunctors other than $\mathbf{Set} \to \mathbf{Set}$?

The monadic approach

The monadic approach

Bellegarde and Hook (1994): syntax is a monad

Convenient substitution operation on numeric de Bruijn indices

$$\begin{aligned}
&\textsf{data Tm} : \textsf{Set} \to \textsf{Set where} \\
&\quad \textsf{var} : X \to \textsf{Tm}\ X \\
&\quad \textsf{lam} : \textsf{Tm}\ X \to \textsf{Tm}\ X \\
&\quad \textsf{app} : \textsf{Tm}\ X \to \textsf{Tm}\ X \to \textsf{Tm}\ X
\end{aligned}$$

Bird and Paterson (1999): syntax is a scope-safe monad

Nested datatypes allow for "type-level" de Bruijn indices

Monadic structure derived via a generalised fold

$$\begin{aligned}
&\textsf{data Tm} : \textsf{Set} \to \textsf{Set where} \\
&\quad \textsf{var} : X \to \textsf{Tm}\ X \\
&\quad \textsf{lam} : \textsf{Tm}\ (1 + X) \to \textsf{Tm}\ (X) \\
&\quad \textsf{app} : \textsf{Tm}\ X \to \textsf{Tm}\ X \to \textsf{Tm}\ X
\end{aligned}$$

Altenkirch and Reus (1999): syntax is initial algebra in $\mathbf{Set}^{\mathbf{Set}}$

Monadic structure derived by structural or well-founded recursion

Intrinsic scoping

The parameter X exposes the *variable scope* of a term

$\mathsf{Tm}\,\emptyset$ is the set of closed terms

$\mathsf{Tm}\,X \to \mathsf{Tm}\,(1 + X)$ is term weakening

Ill-scoped terms can be eliminated

Avoids issues with out-of-scope de Bruijn indices

$$\begin{aligned} \mathsf{lam}\,(\mathsf{lam}\,(\mathsf{app}\,(\mathsf{var}\,(\mathsf{some}\,\mathsf{none}))\,(\mathsf{var}\,\mathsf{none}))) &\in \mathsf{Tm}\,\emptyset \\ \mathsf{app}\,(\mathsf{var}\,(\mathsf{some}\,\mathsf{none}))\,(\mathsf{var}\,\mathsf{none}) &\in \mathsf{Tm}\,(1 + (1 + \emptyset)) \end{aligned}$$

Flexibility over X allows for some strange terms

Scope safety only works if we start from the empty set

$$\mathsf{lam}\,(\mathsf{app}\,(\mathsf{var}\,\mathsf{none})(\mathsf{var}\,(\mathsf{some}\,[\mathsf{var}\,[],\mathsf{lam}\,(\mathsf{var}\,(\mathsf{some}\,(-0.381i)))]))) \\ \in \mathsf{Tm}\,(\mathsf{List}\,(\mathsf{Tm}\,\mathbb{C}))$$

Monadic structure

Tm can be shown to have monad structure

Variable embedding $X \to \mathsf{Tm}\ X$ is the unit

Nested term collapsing $\mathsf{Tm}\ (\mathsf{Tm}\ X) \to \mathsf{Tm}\ X$ is the join

Kleisli extension acts as simultaneous substitution

$$\mathsf{sub} : (X \to \mathsf{Tm}\ Y) \to \mathsf{Tm}\ X \to \mathsf{Tm}\ Y$$

Defining join or sub directly is not possible

Cannot simply recurse under a binder, as the set is extended

Definition requires functoriality and a *lifting* operation

$$\begin{aligned}\mathsf{map} &: (X \to Y) \to \mathsf{Tm}\ X \to \mathsf{Tm}\ Y \\ \mathsf{lift} &: (X \to \mathsf{Tm}\ Y) \to (1 + X) \to (1 + \mathsf{Tm}\ Y)\end{aligned}$$

Lifting can itself be derived from *swapping*

$$\mathsf{swap} : (1 + \mathsf{Tm}\ X) \to \mathsf{Tm}\ (1 + X)$$

$$
\begin{aligned}
&\mathsf{map} : (X \longrightarrow Y) \longrightarrow \mathsf{Tm}\ X \longrightarrow \mathsf{Tm}\ Y \\
&\mathsf{map}\ f\ (\mathsf{var}\ x) \quad = \mathsf{var}\ (f\ x) \\
&\mathsf{map}\ f\ (\mathsf{lam}\ b) \quad = \mathsf{lam}\ (\mathsf{map}\ (1 + f)\ b) \\
&\mathsf{map}\ f\ (\mathsf{app}\ g\ a) = \mathsf{app}\ (\mathsf{map}\ f\ g)\ (\mathsf{map}\ f\ a)
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{swap} : 1 + \mathsf{Tm}\ X \longrightarrow \mathsf{Tm}\ (1 + X) \\
&\mathsf{swap}\ \mathsf{none} \qquad = \mathsf{var}\ \mathsf{none} \\
&\mathsf{swap}\ (\mathsf{some}\ t) = \mathsf{map}\ \mathsf{some}\ \mathrm{t}
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{lift} : (X \longrightarrow \mathsf{Tm}\ Y) \longrightarrow (1 + X) \longrightarrow (1 + \mathsf{Tm}\ Y) \\
&\mathsf{lift}\ f = \mathsf{swap} \circ \mathsf{map}\ f
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{sub} : (X \longrightarrow \mathsf{Tm}\ Y) \longrightarrow \mathsf{Tm}\ X \longrightarrow \mathsf{Tm}\ Y \\
&\mathsf{sub}\ f\ (\mathsf{var}\ x) \quad = f\ x \\
&\mathsf{sub}\ f\ (\mathsf{lam}\ b) \quad = \mathsf{lam}\ (\mathsf{sub}\ (\mathsf{lift}\ f)\ t) \\
&\mathsf{sub}\ f\ (\mathsf{app}\ g\ a) = \mathsf{app}\ (\mathsf{sub}\ f\ g)\ (\mathsf{sub}\ f\ a)
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{join} : \mathsf{Tm}\ (\mathsf{Tm}\ X) \longrightarrow \mathsf{Tm}\ X \\
&\mathsf{join} = \mathsf{sub}\ \mathsf{id}
\end{aligned}
$$

Monad laws

Monad laws established by induction

Lots of subtle helper lemmas needed

$$
\begin{aligned}
\mathsf{lift}\ \mathsf{var} &= \mathsf{id}\\
\mathsf{sub}\ \mathsf{var} &= \mathsf{id}\\
(1 + g) \circ (1 + \mathrm{f}\,) &= 1 + (g \circ f)\\
\mathsf{map}\ g \circ \mathsf{map}\ f &= \mathsf{map}\ (g \circ f)\\
\mathsf{lift}\ g \circ (1 + f) &= \mathsf{lift}\ (g \circ f)\\
\mathsf{map}\ (1 + g) \circ \mathsf{lift}\ f &= \mathsf{lift}\ (\mathsf{app}\ g \circ f)\\
\mathsf{sub}\ g \circ \mathsf{map}\ f &= \mathsf{sub}\ (g \circ f)\\
\mathsf{map}\ g \circ \mathsf{sub}\ f &= \mathsf{sub}\ (\mathsf{map}\ g \circ \mathrm{f})\\
\mathsf{lift}\ (\mathsf{sub}\ g \circ f) &= \mathsf{sub}\ (\mathsf{lift}\ g) \circ \mathsf{lift}\ f\\
\mathsf{sub}\ g \circ \mathsf{sub}\ f &= \mathsf{sub}\ (\mathsf{sub}\ g \circ f)
\end{aligned}
$$

These generalise standard properties of substitution

$$[s/x]x = x \qquad [s/x]t = t \quad \text{if } x \notin \mathrm{fv}(t)$$

$$[r/y]([s/x]t) = [[r/y]s/x]([r/y]t)$$

Models of the monadic approach

Models are built on endofunctors on **Set**

$$\mathsf{Tm} : \mathbf{Set} \to \mathbf{Set} \qquad \mathsf{Tm} \in [\mathbf{Set}, \mathbf{Set}] = \mathbf{Set}^{\mathbf{Set}}$$

Signatures are encoded as endofunctors $\Sigma : \mathbf{Set}^{\mathbf{Set}} \to \mathbf{Set}^{\mathbf{Set}}$

$$\begin{aligned} \mathsf{lam} &: \mathsf{Tm}\,(1 + X) \to \mathsf{Tm}\,X \\ \mathsf{app} &: \mathsf{Tm}\,X \to \mathsf{Tm}\,X \to \mathsf{Tm}\,X \\ \mathsf{alg} = [\mathsf{lam}, \mathsf{app}] &: \Sigma_\Lambda(\mathsf{Tm})\,X \to \mathsf{Tm}\,X \end{aligned}$$

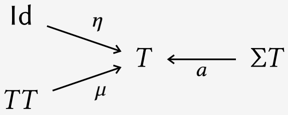
$$\begin{aligned} &\Sigma_\Lambda : \mathbf{Set}^{\mathbf{Set}} \to \mathbf{Set}^{\mathbf{Set}} \\ &\Sigma_\Lambda \triangleq S \in \mathbf{Set}^{\mathbf{Set}} \mapsto X \in \mathbf{Set} \mapsto S(1 + X) + (SX \times SX) \end{aligned}$$

Algebraic model is an $(\mathsf{Id} + \Sigma)$-algebra, syntax is the initial model

$$[\eta, a] : \mathsf{Id} + \Sigma_\Lambda S \to S \qquad [\mathsf{var}, \mathsf{alg}] : \mathsf{Id} + \Sigma_\Lambda(\mathsf{Tm}) \cong \mathsf{Tm}$$

Syntactic and substitution structure

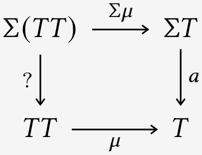
The Σ-algebra structure represents constructors,
the monad structure represents substitution

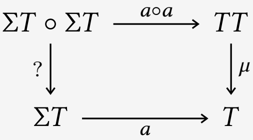


How do a and μ interact?

Is μ an Σ-algebra homomorphism?

Is a a monad morphism?





Modules over monads

In general, TT is not a Σ-algebra and ΣT is not a monad

In our case, $\Sigma T \circ \Sigma T \to \Sigma T$ is not a monad morphism

A. Hirschowitz and Maggesi (2010): *modules over monads*

Axiomatisates the relationship of constructors and substitution

Definition (Module over a monad)

Given a monad (T, η, μ) on $\mathcal{C}$, a T*-module* is a functor $S\colon \mathcal{C} \to \mathcal{C}$ and an *action* compatible with the monad structure:

$$\alpha\colon ST \to S$$

$$\begin{array}{ccc} S\mathrm{Id} & \xrightarrow{S\eta} & ST \\ & \searrow^{\mathrm{id}} & \downarrow{\scriptstyle\alpha} \\ & & S \end{array} \qquad \begin{array}{ccc} STT & \xrightarrow{S\mu} & ST \\ {\scriptstyle\alpha T}\downarrow & & \downarrow{\scriptstyle\alpha} \\ ST & \xrightarrow[\alpha]{} & S \end{array}$$

Modules over monads

Definition (Linear maps)

A *linear map* between two T-modules $(R, \alpha) \to (S, \beta)$ is a morphism $\varphi\colon R \to S$ such that

$$\begin{array}{ccc} RT & \xrightarrow{\varphi T} & ST \\ {\scriptstyle\alpha}\big\downarrow & & \big\downarrow{\scriptstyle\beta} \\ R & \xrightarrow[\varphi]{} & R \end{array}$$

T-modules and linear maps form a category $\mathbf{Mod}(T)$.

Definition (Signature and model)

A *signature* Σ is a functor mapping a monad T to a module $\Sigma(T) \in \mathbf{Mod}(T)$ for the monad. A *model* is a monad T with a module morphism $\Sigma(T) \to T$.

Modules over monads

Example

The endofunctor $\delta\colon [\mathcal{C},\mathcal{C}] \to [\mathcal{C},\mathcal{C}]$, $\delta(A)(X) \triangleq A(1+X)$ lifts to modules: given $(S, \alpha\colon ST \to S)$, we have

$$(\delta(S)\circ T)(A) = S(1{+}TA) \xrightarrow{S\mathsf{swap}} S(T(1{+}A)) \xrightarrow{\alpha_{1+A}} S(1{+}A) = \delta(S)(A)$$

Example

The endofunctor Σ_Λ is a signature that maps a monad T to $\delta T + T \times T$. A model is a monad T with module morphism $[l, a]\colon \delta T + (T \times T) \to T$:

$$\begin{array}{ccc} \delta T \circ T & \xrightarrow{l\circ T} & T \circ T \\ {\scriptstyle \sigma^\delta}\downarrow & & \\ \delta(T \circ T) & & \downarrow{\scriptstyle \mu} \\ {\scriptstyle \delta\mu}\downarrow & & \\ \delta T & \xrightarrow[l]{} & T \end{array} \qquad \begin{array}{ccc} (T \times T) \circ T & \xrightarrow{a\circ T} & T \circ T \\ {\scriptstyle \sigma^\times}\downarrow & & \\ (T \circ T) \times (T \circ T) & & \downarrow{\scriptstyle \mu} \\ {\scriptstyle \mu\times\mu}\downarrow & & \\ T \times T & \xrightarrow[a]{} & T \end{array}$$

Signatures with strength

Structure map $\Sigma T \circ T \Longrightarrow \Sigma T$ often given via $\mu \colon TT \Longrightarrow T$

Corresponds to recursive multiplication of subterms

Definition

Signature with strength A *signature with strength* Σ, σ is an endofunctor $\Sigma \colon [\mathcal{C}, \mathcal{C}] \to [\mathcal{C}, \mathcal{C}]$ with natural transformation

$$\sigma_{A,(B,p)} \colon \Sigma A \circ B \to \Sigma(A \circ B) \colon [\mathcal{C}, \mathcal{C}] \times \mathsf{Id}/[\mathcal{C}, \mathcal{C}] \to [\mathcal{C}, \mathcal{C}]$$

satisfying unit and associativity axioms.

A signature with strength is a signature in the previous sense

$$\Sigma(T) \circ T \xrightarrow{\sigma_{T,(T,\eta)}} \Sigma(T \circ T) \xrightarrow{\Sigma\mu} \to \Sigma T$$

Initial-algebra semantics

We can now show that (Tm, alg) is the initial model for Σ_Λ

Theorem

For all models $\left(T \in \mathbf{Mon}(\mathcal{C}), a : \Sigma T \to T \in \mathbf{Mod}(T)\right)$*, there exists a unique monad morphism* $\mathsf{sem} : \mathsf{Tm} \Longrightarrow T$ *satisfying:*

$$\begin{array}{ccc} \Sigma(\mathsf{Tm})(X) & \xrightarrow{\mathsf{alg}} & \mathsf{Tm}(X) \\ {\scriptstyle \Sigma(\mathsf{sem})_X} \downarrow & & \downarrow {\scriptstyle \mathsf{sem}_X} \\ \Sigma(T)(X) & \xrightarrow[a]{} & T(X) \end{array}$$

The map is a monad and module homomorphism

Preserves constructors and substitution

Satisfies the semantic substitution lemma

Monadic approach

Mature and well-developed theory
Work on denotational and operational semantics
equations, translations, non-wellfounded syntax, etc.[1]

Untyped setting easy to implement in functional languages
Laws or typed syntax still needs dependent types

Endofunctors allow for more variation than needed
Context extension enough for most simple syntaxes

Endofunctors on endofunctors, modules, over monads,
application vs. composition can get confusing
Loose hierarchy between levels of contexts, terms, signatures

$$(\delta(S) \circ T)(TX) \quad \text{vs} \quad (\delta(ST) \circ T)(X) \quad \text{vs} \quad \delta(S \circ TT)(X)$$

---

[1] Ahrens (2016), Ahrens, A. Hirschowitz, et al. (2021), Ahrens and Zsido (2011), A. Hirschowitz, T. Hirschowitz, and Lafont (2020), and A. Hirschowitz and Maggesi (2012)

The presheaf approach

The presheaf approach

Fiore, Plotkin, and Turi (1999): syntax lives in *presheaves*

Sets varying over a category of contexts and renamings

Definition (Presheaf)

A *(covariant) presheaf* on a small category $\mathbb{C}$ is a functor $P : \mathbb{C} \to \mathbf{Set}$. Presheaves and natural transformations form the category $\widetilde{\mathbb{C}} \triangleq \mathbf{Set}^{\mathbb{C}}$. *$S$-sorted presheaves* form the S-indexed category $\widetilde{\mathbb{C}}^S$.

Example

$\mathsf{Tm} \in \widetilde{\mathbb{F}}^S$ for $\mathbb{F}$ the category of contexts over S is the family of sets $\mathsf{Tm}_\alpha(\Gamma) \triangleq \{ t \mid \Gamma \vdash t : \alpha \}$, with the variable renaming operation

$$\mathsf{ren} : (\Gamma \to \Delta) \to \mathsf{Tm}_\alpha(\Gamma) \to \mathsf{Tm}_\alpha(\Delta)$$

Renaming structure

Like endofunctors, renaming is baked into the definition

Most often instantiated as weakening with $\Gamma \to \alpha \cdot \Gamma$

Unlike endofunctors, contexts are a lower-class object to terms

Renaming rules are not arbitrary functions between sets

This helps eliminate confusion between context-,
term- and signature-level operations

Presheaves cannot be composed or applied to each other

Presheaves over $\mathbb{F}$ are equivalent to finitary endofunctors

$$\mathbf{Set}^{\mathbb{F}} \simeq [\mathbf{Set}, \mathbf{Set}]_f$$

Intrinsic typing and scoping

Presheaves conveniently capture intrinsic typing and scoping

A term $t \in T_\alpha\Gamma$ is well-scoped in context Γ and has type α

There is a distinguished presheaf of *variables*

The set is inhabited if τ appears in Γ

$$V_\alpha\Gamma \triangleq \text{よ}[\alpha](\Gamma) = \mathbb{F}([\alpha], \Gamma) \qquad [\alpha] \xrightarrow{\text{new}} \alpha \cdot \Gamma \xleftarrow{\text{old}} \Gamma$$

Context extension is equivalently presheaf exponentiation by V

Evaluation corresponds to strengthening

$$\delta_\tau(P)_\alpha\Gamma \triangleq P_\alpha(\tau \cdot \Gamma) \cong P_\alpha^{V_\tau}(\Gamma) \qquad \delta_\tau(P)_\alpha \times V_\tau \cong P_\alpha^{V_\tau} \times V_\tau \longrightarrow P_\alpha$$

Signatures and models

Constructors combine into *signature endofunctor* $\Sigma \colon \widetilde{\mathbb{F}}^S \longrightarrow \widetilde{\mathbb{F}}^S$

Matching input and output sorts introduces some complexity

$$\Sigma_\Lambda P_\tau \triangleq \Big[\sum_{\alpha,\beta \in S} \delta_\alpha P_\beta \times (\tau = (\alpha \to \beta)) \Big] + \Big[\sum_{\alpha \in S} P_{\alpha \to \tau} \times P_\tau \Big]$$

Algebraic model is a $V + \Sigma$-algebra, syntax is the initial model

$$[v, a] \colon V + \Sigma_\Lambda(A) \longrightarrow A \qquad [\mathsf{var}, \mathsf{alg}] \colon V + \Sigma_\Lambda(\mathsf{Tm}) \cong \mathsf{Tm}$$

$$\begin{aligned}
\mathsf{var} &\colon V_\alpha\Gamma \longrightarrow \mathsf{Tm}_\alpha\Gamma \\
\mathsf{alg} = [\mathsf{lam} &\colon \mathsf{Tm}_\beta(\alpha \cdot \Gamma) \longrightarrow \mathsf{Tm}_{\alpha\to\beta}(\Gamma), \\
\mathsf{app} &\colon \mathsf{Tm}_{\alpha\to\beta}(\Gamma) \times \mathsf{Tm}_\alpha(\Gamma) \longrightarrow \mathsf{Tm}_\alpha(\Gamma)]
\end{aligned}$$

Substitution structure

Like endofunctors, substitution amounts to additional structure

Analogous to monad multiplication or bind

Unlike endofunctors, a presheaf cannot be a monad

$\mathcal{A} \circ \mathcal{A} \to \mathcal{A}$ is not defined, since $\mathcal{A}$ is not an endofunctor

First solution: a V-relative monad structure[2]

Definition (Relative monad)

For functors $J, F \colon \mathcal{C} \to \mathcal{D}$, F is a *J-relative monad* if it comes with a unit and extension operator satisfying unit and associativity laws:

$$\eta_A \colon JA \to FA \qquad (-)^\dagger \colon \mathcal{D}(JA, FB) \to \mathcal{D}(FA, FB)$$

Example

A presheaf with substitution structure is a V-relative monad:

$$v \colon V_\alpha\Gamma \to P_\alpha\Gamma \qquad (-)^\dagger \colon \mathbf{Set}(V_\alpha\Gamma, P_\alpha\Delta) \to \mathbf{Set}(P_\alpha\Gamma, P_\alpha\Delta)$$

---

[2] Altenkirch, Chapman, and Uustalu (2010)

Substitution structure

Second solution: monoid for the substitution tensor product[3]

Definition (Monoidal category)

A *monoidal category* $\mathcal{C}$ has a *unit object* $I \in \mathcal{C}$ and a *tensor product* $(-) \otimes (=) \colon \mathcal{C} \times \mathcal{C} \to \mathcal{C}$ with natural isomorphisms

$$\lambda \colon I \otimes B \cong B \qquad \rho \colon A \cong A \otimes I \qquad \alpha \colon (A \otimes B) \otimes C \cong A \otimes (B \otimes C)$$

satisfying two coherence laws.

Example

Presheaves have a monoidal structure with unit V and tensor

$$(P \otimes Q)_\alpha(\Delta) \triangleq \int^{\Gamma \in \mathbb{F}} P_\alpha \Gamma \times {}^{\Gamma}Q_\Delta$$

where ${}^{\Gamma}Q_\Delta = \prod_{\alpha \in S} \mathbf{Set}(V_\alpha \Gamma, Q_\alpha \Delta)$.

[3] Fiore, Plotkin, and Turi (1999)

Substitution structure

$$(P \otimes Q)_\alpha(\Delta) \triangleq \int^{\Gamma \in \mathbb{F}} P_\alpha\Gamma \times {}^\Gamma Q_\Delta \qquad {}^\Gamma Q_\Delta = \prod_{\alpha \in S} \mathbf{Set}(V_\alpha\Gamma, Q_\alpha\Delta)$$

$$\left(\Gamma, t \in P_\alpha\Gamma, \sigma \colon {}^\Gamma Q_\Delta\right) \in (P \otimes Q)_\alpha(\Delta)$$

The coend performs a quotienting on the tuples
Enforces an internal renaming-invariance

$$(\Gamma, t, \sigma \circ \rho) = (\Delta, P(\rho)(t), \sigma) \in (P \otimes Q)_\alpha\Theta \quad \text{for } \rho \colon \Gamma \to \Delta, \sigma \colon {}^\Delta Q_\Theta$$

Essential for the invertibility of structure maps

$$(\Gamma, t, \rho) \mapsto P(\rho)(t) \mapsto (\Delta, P(\rho)(t), \mathrm{id}) = (\Gamma, t, \rho)$$

Substitution structure

Definition

A *monoid* in a monoidal category $(C, I, \otimes)$ is an object M with unit $\eta\colon I \to M$ and multiplication $M \otimes M \to M$ satisfying unit and associativity laws.

Example

A monoid M in the category of presheaves comes with a variable embedding $\eta\colon V \to M$ and a substitution operation

$$\mu\colon M \otimes M \to M \qquad (M \otimes M)_\alpha \Delta = \{M_\alpha \Gamma \times {}^{\Gamma}M_\Delta \to M_\alpha \Gamma\}_{\alpha \in S, \Gamma, \Delta \in \mathbb{F}}$$

natural in Δ and dinatural in Γ:

$$\mu(\Gamma, t, M(\rho) \circ \sigma) = M(\rho)(\mu(\Gamma, t, \sigma)) \quad \mu(\Gamma, t, \sigma \circ \rho) = \mu(\Delta, M(\rho)(t), \sigma)$$

Models in presheaves

Presheaves with compatible algebra and monoid structures are semantic models

Definition (Σ-monoids)

Given a strong endofunctor $\Sigma\colon \widetilde{\mathbb{F}}^S \to \widetilde{\mathbb{F}}^S$, a Σ-*monoid* is a monoid (M, η, μ) with Σ-algebra structure $a\colon \Sigma M \to M$ satisfying

$$\begin{array}{ccccc}
\Sigma M \otimes M & \xrightarrow{\sigma_{M,M}} & \Sigma(M \otimes M) & \xrightarrow{\Sigma\mu} & \Sigma M \\
{\scriptstyle a\otimes M}\downarrow & & & & \downarrow{\scriptstyle a} \\
M \otimes M & & \xrightarrow[\mu]{\hspace{6em}} & & M
\end{array}$$

The pointed strength $\sigma_{P,Q}\colon \Sigma P \otimes Q \to \Sigma(P \otimes Q)$ pushes substitutions into subterms and under binders

Initial-algebra semantics

We may again show that Tm is the initial Σ_Λ-monoid

Involves:

- Equipping Tm with a renaming operation
- Defining the strength $\Sigma\mathsf{Tm} \otimes \mathsf{Tm} \to \Sigma(\mathsf{Tm} \otimes \mathsf{Tm})$
- Deriving the substitution operation $\mathsf{Tm} \otimes \mathsf{Tm} \to \mathsf{Tm}$
- Proving functoriality, strength, and substitution laws
- Inducing generic semantics $\mathsf{Tm} \to M$ into any Σ-monoid M
- Proving the semantics preserves Σ-monoid structure

Presheaf approach

Widely extensible mathematical framework

Polymorphism, equational logic, second-order algebraic theories, linearity, metavariable calculi, etc.[4]

Contexts, naturality, monoids, etc. easier to keep straight

Clear hierarchy of concepts and properties

Limited work on reduction and operational semantics

No obvious way to incorporate with current models

Mathematically involved and hard/impossible to formalise fully

Complex nesting of categorical structures, quotienting

[4] Fiore (2008), Fiore and Hamana (2013), Fiore and Hur (2010), Fiore and Mahmoud (2010), Power (2007), and Tanaka (2000)

The family approach

Presheaf model as formalisation framework

The presheaf model is not amenable to faithful formalisation
Abstract categorical concepts not always constructive

Complex hierarchy of structures computationally expensive
Agda grinds to a halt when checking functoriality and naturality

Requiring presheaf actions everywhere is overkill
Only needed for weakening in capture-avoiding substitution

In some places, renaming is undesirable
Metavariables should not be renamed, but need to conform to setting

The family approach

Fiore and Sz. (2022): indexed families of sets almost enough
Where renaming is needed, it can be requested explicitly

Mathematical basis for common formalisation methods
Puts previously ad-hoc techniques on a formal foundation

Works around the need for quotienting
Weaker structures, more general definitions

Retains the initiality property of syntax
Practically usable framework based on a sound theory

Intrinsically-typed syntax

Instead of presheaves, we work with indexed families of sets

Direct to represent in proof assistants

$$\mathsf{Fam} : S \longrightarrow S^* \longrightarrow \mathsf{Set}$$

Family of variables and terms are inductive datatypes

Standard dependently-typed formalisation technique

```
data S : Set where
  B   : S
  _→_ : S → S → S

data Ctx : Set where
  ∅   : Ctx
  _·_ : S → Ctx → Ctx
```

```
data I : Fam where
  new :           I α (α · Γ)
  old : I β Γ → I β (α · Γ)

data Tm : Fam where
  var : I α Γ → Tm α Γ
  app : Tm (α → β) Γ → Tm α Γ → Tm β Γ
  lam : Tm β (α · Γ) → Tm (α → β) Γ
```

Renaming structure

Families cannot be renamed a priori

A family is fully determined by its elements

If renaming is needed, it's axiomatised as a co/algebra structure

Free presheaf monad and cofree presheaf comonad

$$\Diamond X_\alpha \Delta \triangleq \sum_{\Gamma \in S^*} X_\alpha \Gamma \times (\Gamma \to \Delta) \qquad \Box X_\alpha \Gamma \triangleq \prod_{\Delta \in S^*} (\Gamma \to \Delta) \longrightarrow X_\alpha \Delta$$

$$\mathsf{ren} : \prod_{\Gamma, \Delta \in S^*} (\Gamma \to \Delta) \longrightarrow \mathsf{Tm}_\alpha \Gamma \longrightarrow \mathsf{Tm}_\alpha \Delta \cong \Diamond \mathsf{Tm} \longrightarrow \mathsf{Tm} \cong \mathsf{Tm} \longrightarrow \Box \mathsf{Tm}$$

Families with renaming structure are equivalent to presheaves

The structure is only requested when needed

Substitution structure

$$(X \oplus Y)_\alpha \Delta \triangleq \sum_{\Gamma \in S^*} X_\alpha \Gamma \times {}^\Gamma Y_\Delta$$

Substitution tensor product no longer monoidal

No quotienting to enforce renaming-invariance

Weaker *skew-monoidal* structure

Structure maps and laws are not invertible

$$\lambda \colon I \oplus Y \to Y \quad \rho \colon X \to X \oplus I \quad \alpha \colon (X \oplus Y) \oplus Z \to X \oplus (Y \oplus Z)$$

$\Diamond$-algebras are equivalently modules for I

$\Diamond X$ combines a term with a substitution of variables for variables

$$\Diamond X \cong X \oplus I \qquad\qquad (\Diamond X \to X) \cong X \otimes I \to X$$

Substitution monoids same as before

May ask for a monoid with compatible $\Diamond$-algebra structure

Signatures with pointed strength

Signatures are family endofunctors with a *pointed* $\diamond$*-strength*
Point maps variables to variables, renaming allows weakening

$$\sigma_{X,Y} : \Sigma X \oplus Y \to \Sigma(X \oplus Y) : \mathsf{Fam} \times I/\diamond\text{-}\mathbf{Alg} \to \mathsf{Fam}$$

For context extension δ, strength is defined via lift
Extends both contexts of a substitution rule

$$\begin{aligned}
&\mathsf{lift}_{(X,p,x)} : {}^{\Gamma}Y_{\Delta} \to {}^{(\tau\cdot\Gamma)}Y_{(\tau\cdot\Delta)} : I/\diamond\text{-}\mathbf{Alg} \to \mathbf{Set} \\
&\mathsf{lift}_{(X,p,x)}\ \sigma\ \mathsf{new} && \triangleq p\ \mathsf{new} \\
&\mathsf{lift}_{(X,p,x}\ \sigma\ (\mathsf{old}\, v) && \triangleq x(\sigma\, v, \mathsf{old}) \\
\\
&\sigma^{\delta}_{X,Y}(\Gamma, t, \sigma) && \triangleq (\tau \cdot \Gamma, t, \mathsf{lift}\, \sigma)
\end{aligned}$$

Signatures with pointed strength

Problem: $\Diamond$-**Alg** is not monoidal, σ is not associative

No quotienting to equate reassociated substitutions

Solution: associativity in terms of *balanced maps*

Functions $f : X \oplus Y \to Z$ that equate quotientable tuples

$$f(\Gamma, t, \sigma \circ \rho) = f(\Delta, x(t, \rho), \sigma)$$

$$\begin{array}{ccccccc}
(\Sigma W \oplus X) \oplus Y & \xrightarrow{\sigma_{X,Y} \oplus Z} & \Sigma(X \oplus Y) \oplus Z & \xrightarrow{\sigma_{X \oplus Y, Z}} & \Sigma((X \oplus Y) \oplus Z) & \xrightarrow{\Sigma\alpha_{X,Y,Z}} & \Sigma(X \oplus (Y \oplus Z)) \\
{\scriptstyle \alpha_{\Sigma W,X,Y}} \downarrow & & & & & & \downarrow {\scriptstyle \Sigma(X \oplus f)} \\
\Sigma W \oplus (X \oplus Y) & & \xrightarrow[\mathrm{id} \oplus f]{} & \Sigma W \oplus Z & \xrightarrow[\sigma_{W,Z}]{} & & \Sigma(W \oplus Z)
\end{array}$$

Associativity law for σ^{δ} generalises all the lemmas for lift

$$\mathsf{lift}\,(\varsigma \circ \rho) = \mathsf{lift}\,\varsigma \circ \mathsf{lift}_I\,\rho$$

$$\mathsf{lift}\,(\mathsf{ren}\,\varrho \circ \sigma) = \mathsf{ren}\,(\mathsf{lift}_I\,\varrho) \circ \mathsf{lift}\,\sigma$$

$$\mathsf{lift}\,(\mathsf{sub}\,\varsigma \circ \sigma) = \mathsf{sub}\,(\mathsf{lift}\,\varsigma) \circ \mathsf{lift}\,\sigma$$

Models and initiality

Models are Σ-monoids as before

Monoids are automatically $\Diamond$-algebras: renaming is substitution of variables for variables

Family of terms is the initial Σ-monoid

Both renaming and substitution is induced by initiality

The initial model in Fam is provably equivalent to the model in $\widetilde{\mathbb{F}}^{S}$

All the theory faithfully lifts to the presheaf model

The family model

First steps of adapting the presheaf model to a constructive setting
Promising and categorically motivated formulation

Simple formalisation in dependently-typed proof assistants
Code generation tool to go from a syntax description
to an intrinsically-typed metatheoretic framework

Elegantly incorporates second-order features
Metavariables, metasubstitution, equational systems

More complex type theories nontrivial to adapt
Linear substitutions, polymorphism, etc. still heavy to formalise

Syntax description file

```
syntax Λ

type
  N    : 0-ary
  _→_  : 2-ary

term
  app  : (α → β)  α  →  β
  lam  : α.β         →  (α → β)
```

Syntactic and semantic operations

$$\mathsf{wkn} : \Lambda\,\alpha\,\Gamma \longrightarrow \Lambda\,\alpha\,(\beta \cdot \Gamma)$$

$$[\_/] : \Lambda\,\alpha\,\Gamma \longrightarrow \Lambda\,\beta\,(\alpha \cdot \Gamma) \longrightarrow \Lambda\,\beta\,\Gamma$$

$$[\![\_]\!] : \Lambda\,\alpha\,\Gamma \longrightarrow M\,\alpha\,\Gamma$$

Correctness laws

$$\mathsf{syn\text{-}sub\text{-}lemma} : [r/]\,([s/]\,t) \equiv \big[\,[r/]\,s\,/\big]\,([r/]\,t)$$

$$\mathsf{sem\text{-}sub\text{-}lemma} : [\![\,[s/]\,t\,]\!] \equiv M.\mathsf{sub}\,[\![s]\!]\,[\![t]\!]$$

Conclusions

Finding models of syntax enables generic metatheory
Derivation of tedious boilerplate code for free

Functorial models make context-dependence explicit
Functoriality highlights importance of renaming

Family model weakens assumptions for the sake of practicality
Also clarifies roles of variables, weakening, etc.

Paper and $\binom{\text{currently}}{\text{broken}}$ Agda library can be found at

`https://tinyurl.com/agda-soas`

Thank you!

References I

Ahrens, Benedikt (2016). "Modules over relative monads for syntax and semantics". In: *Mathematical Structures in Computer Science* 26.1, pp. 3–37. DOI: 10.1017/S0960129514000103.

Ahrens, Benedikt, André Hirschowitz, Ambroise Lafont, and Marco Maggesi (2021). "Presentable signatures and initial semantics". In: *Logical Methods in Computer Science* Volume 17, Issue 2. DOI: 10.23638/LMCS-17(2:17)2021.

Ahrens, Benedikt and Julianna Zsido (2011). *Initial Semantics for higher-order typed syntax in Coq*. arXiv: 1012.1010 [cs.LO].

Altenkirch, Thorsten, James Chapman, and Tarmo Uustalu (2010). "Monads Need Not Be Endofunctors". In: *Proceedings of the 13th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS 2010)*. Ed. by Luke Ong. Lecture Notes in Computer Science (LNCS). Springer, pp. 297–311. DOI: 10.1007/978-3-642-12032-9_21.

Altenkirch, Thorsten and Bernhard Reus (1999). "Monadic Presentations of Lambda Terms Using Generalized Inductive Types". In: *Proceedings of the 13th International Workshop on Computer Science Logic (CSL 1999)*. Vol. 1683. Lecture Notes in Computer Science (LNCS). Springer, pp. 453–468. DOI: 10.1007/3-540-48168-0_32.

Bellegarde, Françoise and James Hook (1994). "Substitution: A Formal Methods Case Study Using Monads and Transformations". In: *Science of Computer Programming* 23.2-3, pp. 287–311. DOI: 10.1016/0167-6423(94)00022-0.

Bird, Richard and Ross Paterson (1999). "De Bruijn Notation as a Nested Datatype". In: *Journal of Functional Programming* 9.1, pp. 77–91. DOI: 10.1017/S0956796899003366.

References II

Fiore, Marcelo (2008). "Second-Order and Dependently-Sorted Abstract Syntax". In: *Proceedings of the 23rd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2008)*, pp. 57–68. DOI: 10.1109/LICS.2008.38.

Fiore, Marcelo and Makoto Hamana (2013). "Multiversal Polymorphic Algebraic Theories: Syntax, Semantics, Translations, and Equational Logic". In: *Proceedings of the 28th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2013)*. IEEE Computer Society, pp. 520–529. DOI: 10.1109/LICS.2013.59.

Fiore, Marcelo and Chung-Kil Hur (2010). "Second-Order Equational Logic (Extended Abstract)". In: *Proceedings of the 24th International Workshop on Computer Science Logic (CSL 2010)*. Ed. by Anuj Dawar and Helmut Veith, pp. 320–335. DOI: 10.1007/978-3-642-15205-4_26.

Fiore, Marcelo and Ola Mahmoud (2010). "Second-Order Algebraic Theories". In: *Proceedings of the 35th International Symposium on Mathematical Foundations of Computer Science (MFCS 2010)*. Ed. by Petr Hliněný and Antonín Kučera. Vol. 6281. Lecture Notes in Computer Science (LNCS). Springer, pp. 368–380. DOI: 10.1007/978-3-642-15155-2_33.

Fiore, Marcelo, Gordon Plotkin, and Daniele Turi (1999). "Abstract Syntax and Variable Binding". In: *Proceedings of the 14th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 1999)*, pp. 193–202. DOI: 10.1109/LICS.1999.782615.

Fiore, Marcelo and Dmitrij Szamozvancev (2022). "Formal Metatheory of Second-Order Abstract Syntax". In: *Proceedings of the ACM on Programming Languages* 6.POPL, 53:1–53:29. DOI: 10.1145/3498715.

References III

Hirschowitz, André, Tom Hirschowitz, and Ambroise Lafont (2020). “Modules over Monads and Operational Semantics”. In: *Proceedings of the 5th International Conference on Formal Structures for Computation and Deduction (FSCD 2020)*. Ed. by Zena M. Ariola. Vol. 167. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 12:1–12:23. DOI: 10.4230/LIPICS.FSCD.2020.12.

Hirschowitz, André and Marco Maggesi (2010). “Modules over Monads and Initial Semantics”. In: 208.5, pp. 545–564. DOI: 10.1016/j.ic.2009.07.003.

Hirschowitz, André and Marco Maggesi (2012). “Initial Semantics for Strengthened Signatures”. In: *Proceedings of the 8th Workshop on Fixed Points in Computer Science (FICS 2012)*. Ed. by Dale Miller and Zoltán Ésik. Vol. 77. EPTCS, pp. 31–38. DOI: 10.4204/EPTCS.77.5.

Power, John (2007). “Abstract Syntax: Substitution and Binders”. In: *Electronic Notes in Theoretical Computer Science* 173, pp. 3–16. DOI: 10.1016/j.entcs.2007.02.024.

Tanaka, Miki (2000). “Abstract Syntax and Variable Binding for Linear Binders”. In: *Proceedings of the 25th International Symposium on Mathematical Foundations of Computer Science (MFCS 2000)*. Ed. by Mogens Nielsen and Branislav Rovan. Vol. 1893. Lecture Notes in Computer Science (LNCS). Springer, pp. 670–679. DOI: 10.1007/3-540-44612-5_62.