Formal Metatheory of Second-Order Abstract Syntax

Marcelo Fiore   Dmitrij Szamozvancev

Department of Computer Science and Technology
University of Cambridge, UK

POPL 2022

UNIVERSITY OF CAMBRIDGE

Lemma (Weakening) If $\Gamma \vdash t : \beta$, then $\Gamma, x : \alpha \vdash t : \beta$.
Proof Trivial.

Lemma (Substitution) If $\Gamma, x : \alpha \vdash t : \beta$ and $\Gamma \vdash s : \alpha$
then $\Gamma \vdash [s/x]t : \beta$.
Proof By induction on $\Gamma, x : \alpha \vdash t : \beta$, weakening, and exchange.

Theorem (Type preservation) If $\Gamma \vdash t : \alpha$ and $t \rightsquigarrow s$ then $\Gamma \vdash s : \alpha$.
Proof By induction on $t \rightsquigarrow s$ and the substitution lemma.

Lemma (Renaming) If $\Gamma \vdash t : \alpha$ and $\rho : \Gamma \rightsquigarrow \Delta$, then $\Delta \vdash \langle\rho\rangle t : \alpha$.
Proof By induction on $\Gamma \vdash t : \alpha$.

Lemma (Weakening) If $\Gamma \vdash t : \beta$, then $\Gamma, x : \alpha \vdash t : \beta$.
Proof By renaming with $\Gamma \rightsquigarrow (\Gamma, \alpha)$.

Lemma (Lifting) If $\Delta \vdash \sigma : \Gamma$, then $(\Delta, \alpha) \vdash \text{lift}\, \sigma : (\Gamma, \alpha)$.
Proof By induction on $\Delta \vdash \sigma : \Gamma$ and weakening.

Lemma (Simultaneous substitution) If $\Gamma \vdash t : \alpha$ and $\Delta \vdash \sigma : \Gamma$,
then $\Delta \vdash [\sigma]t : \alpha$.
Proof By induction on $\Gamma \vdash t : \alpha$ and lifting.

Lemma (Substitution) If $\Gamma, x : \alpha \vdash t : \beta$ and $\Gamma \vdash s : \alpha$
then $\Gamma \vdash [s/x]t : \beta$.
Proof By simultaneous substitution with $\Gamma \vdash \text{id}, t : \Gamma, \alpha$.

Theorem (Type preservation) If $\Gamma \vdash t : \alpha$ and $t \rightsquigarrow s$ then $\Gamma \vdash s : \alpha$.
Proof By induction on $t \rightsquigarrow s$ and the substitution lemma.

Lemma (Lift-id) $\mathsf{lift}_v\ \mathsf{id}\ v = v$

Lemma (Ren-id) $\langle \mathsf{id} \rangle t = t$

Lemma (Lift-var) $\mathsf{lift}\ \mathsf{var}\ v = \mathsf{var}\ v$

Theorem (Identity substitution) $[\mathsf{var}]t = t$

Lemma (Lift-ren-ren) $\mathsf{lift}_v\ (\varrho \circ \rho)\ v = \langle \mathsf{lift}_v\, \varrho \rangle (\mathsf{lift}_v\, \rho\, v)$

Lemma (Ren-ren) $\langle \varrho \rangle (\langle \rho \rangle t) = \langle \varrho \circ \rho \rangle t$

Lemma (Lift-ren-sub) $\mathsf{lift}\ (\langle \rho \rangle \circ \sigma)\ v = \langle \mathsf{lift}_v\, \rho \rangle (\mathsf{lift}\, \sigma\, v)$

Lemma (Ren-sub) $\langle \rho \rangle ([\sigma] t) = [\langle \rho \rangle \circ \sigma] t$

Lemma (Lift-sub-ren) $\mathsf{lift}\ (\sigma \circ \rho)\ x = \mathsf{lift}\ \sigma\ (\mathsf{lift}_v\ \rho)\ v$

Lemma (Sub-ren) $[\sigma](\langle \rho \rangle t) = [\sigma \circ \rho] t$

Lemma (Lift-sub-sub) $\mathsf{lift}\ ([\varsigma] \circ \sigma)\ x = [\mathsf{lift}\ \varsigma](\mathsf{lift}\ \sigma\ x)$

Theorem (Substitution associativity) $[\varsigma]([\sigma]\ t) = [[\varsigma] \circ \sigma]\ t$

Proof assistants can demand an intimidating amount of rigour

Proof assistants can demand an intimidating amount of ~~rigour~~ boilerplate benefit from

Proof assistants can ~~demand an~~

~~intimidating~~ amount of ~~rigour~~

a tasteful ~~boilerplate~~

code generation

Syntax description file

```
syntax Λ

type
  N   : 0-ary
  _≻_ : 2-ary

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

$\Longrightarrow$

Data type of types and terms

$$
\begin{aligned}
&\mathsf{data}\ \Lambda\mathsf{T} : \mathsf{Set}\ \mathsf{where}\\
&\quad \mathsf{N} \quad : \Lambda\mathsf{T}\\
&\quad \_\succ\_ : \Lambda\mathsf{T} \to \Lambda\mathsf{T} \to \Lambda\mathsf{T}
\end{aligned}
$$

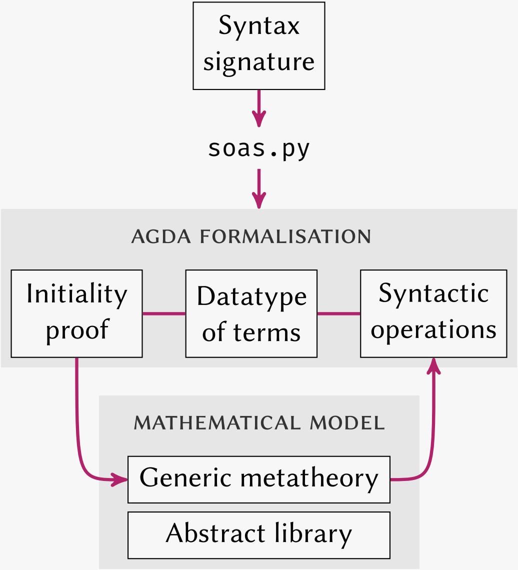
$$
\begin{aligned}
&\mathsf{data}\ \Lambda : \Lambda\mathsf{T} \to \mathsf{Ctx}\ \Lambda\mathsf{T} \to \mathsf{Set}\ \mathsf{where}\\
&\quad \mathsf{var} \ : \mathcal{I}\,\alpha\,\Gamma \to \Lambda\,\alpha\,\Gamma\\
&\quad \mathsf{app} : \Lambda\,(\alpha \succ \beta)\,\Gamma \to \Lambda\,\alpha\,\Gamma \to \Lambda\,\beta\,\Gamma\\
&\quad \mathsf{lam} : \Lambda\,\beta\,(\alpha \cdot \Gamma) \to \Lambda\,(\alpha \succ \beta)\,\Gamma
\end{aligned}
$$

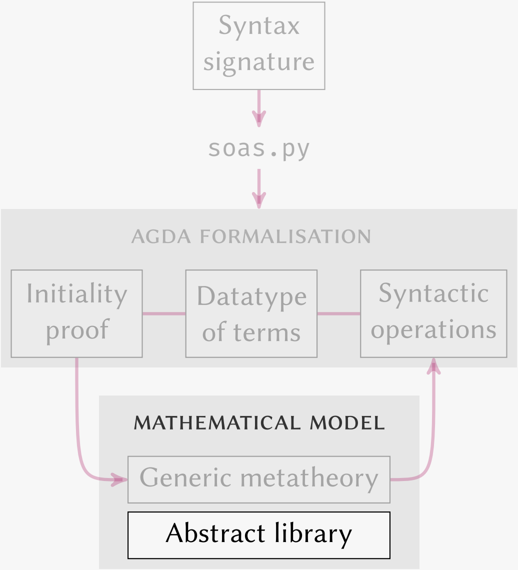
Syntactic and semantic operations

$$
\begin{aligned}
&\mathsf{wk} \quad : \Lambda\,\alpha\,\Gamma \to \Lambda\,\alpha\,(\beta \cdot \Gamma)\\
&[\_/] : \Lambda\,\alpha\,\Gamma \to \Lambda\,\beta\,(\alpha \cdot \Gamma) \to \Lambda\,\beta\,\Gamma\\
&[\![\_]\!] \quad : \Lambda\,\alpha\,\Gamma \to \mathcal{M}\,\alpha\,\Gamma
\end{aligned}
$$

Correctness laws

$$
\begin{aligned}
&\mathsf{syn\text{-}sub\text{-}lemma} \ : [r/]\ ([s/]\ t) \equiv \big[\ [r/]\ s\ /\big]\ ([r/]\ t)\\
&\mathsf{sem\text{-}sub\text{-}lemma} : [\![\ [s/]\ t\ ]\!] \equiv \mathcal{M}.sub\ [\![s]\!]\ [\![t]\!]
\end{aligned}
$$


Syntax signature
soas.py
AGDA FORMALISATION
Initiality proof
Datatype of terms
Syntactic operations
MATHEMATICAL MODEL
Generic metatheory
Abstract library


Syntax signature
soas.py
AGDA FORMALISATION
Initiality proof
Datatype of terms
Syntactic operations
MATHEMATICAL MODEL
Generic metatheory
Abstract library

The universe of discourse of abstract syntax is typed- and scoped- sets – *sorted families*

Objects and morphisms

$$\mathsf{Family}_s : \mathsf{Set}_1$$
$$\mathsf{Family}_s = T \to \mathsf{Ctx} \to \mathsf{Set}$$

$$\_\!\Rightarrow\!\_ : \mathsf{Family}_s \to \mathsf{Family}_s \to \mathsf{Set}$$
$$\mathcal{X} \Rightarrow \mathcal{Y} = \{\alpha : T\}\{\Gamma : \mathsf{Ctx}\} \to \mathcal{X}\;\alpha\;\Gamma \to \mathcal{Y}\;\alpha\;\Gamma$$

Example (Variables)

$$\begin{array}{l} \mathsf{data}\ \mathcal{I} : \mathsf{Family}_s\ \mathsf{where} \\ \quad \mathsf{new} : \mathcal{I}\;\alpha\;(\alpha \cdot \Gamma) \\ \quad \mathsf{old}\ \ : \mathcal{I}\;\beta\;\Gamma \to \mathcal{I}\;\beta\;(\alpha \cdot \Gamma) \end{array}$$

Example (Syntactic terms)

$$\begin{array}{l} \mathsf{data}\ \Lambda : \mathsf{Family}_s\ \mathsf{where} \\ \quad \mathsf{var}\ \ : \mathcal{I}\;\alpha\;\Gamma \to \Lambda\;\alpha\;\Gamma \\ \quad \mathsf{app} : \Lambda\;(\alpha \succ \beta)\;\Gamma \to \Lambda\;\alpha\;\Gamma \to \Lambda\;\beta\;\Gamma \\ \quad \mathsf{lam} : \Lambda\;\beta\;(\alpha \cdot \Gamma) \to \Lambda\;(\alpha \succ \beta)\;\Gamma \end{array}$$

Context maps assign terms of a family to variables in a type-preserving way

$$
\begin{array}{ccccccc}
\Gamma = & \alpha & \beta & \gamma & \delta & : \mathsf{Ctx} \\
 & \big\downarrow & \big\downarrow & \big\downarrow & \big\downarrow & \\
\sigma = & t_1 : \mathcal{X}\,\alpha\,\Delta & t_2 : \mathcal{X}\,\beta\,\Delta & t_3 : \mathcal{X}\,\gamma\,\Delta & t_4 : \mathcal{X}\,\delta\,\Delta & : \Gamma \; {-}[\mathcal{X}]{\rightarrow} \; \Delta
\end{array}
$$

Simultaneous substitutions represented as a function space

$$
\begin{aligned}
&\_{-}[\_]{\rightarrow}\_ : \mathsf{Ctx} \longrightarrow \mathsf{Family}_s \longrightarrow \mathsf{Ctx} \longrightarrow \mathsf{Set} \\
&\Gamma \; {-}[\; \mathcal{X} \;]{\rightarrow} \; \Delta = \{\alpha : T\} \longrightarrow \mathcal{I}\;\alpha\;\Gamma \longrightarrow \mathcal{X}\;\alpha\;\Delta
\end{aligned}
$$

Renamings are variable-valued context maps

Contexts and renamings form a cocartesian category

$$
\begin{aligned}
&\_\rightsquigarrow\_ : \mathsf{Ctx} \longrightarrow \mathsf{Ctx} \longrightarrow \mathsf{Set} \\
&\Gamma \rightsquigarrow \Delta = \Gamma \; {-}[\; \mathcal{I} \;]{\rightarrow} \; \Delta
\end{aligned}
$$

Families can be parametrised by context maps

Make $\mathcal{Y}$ dependent on a $\mathcal{X}$-valued context map into an arbitrary Δ

Internal hom of $\mathcal{X}$ and $\mathcal{Y}$ in the skew-closed category of families

$$\langle\!\langle \_,\_ \rangle\!\rangle : \mathsf{Family}_s \longrightarrow \mathsf{Family}_s \longrightarrow \mathsf{Family}_s$$
$$\langle\!\langle \mathcal{X} , \mathcal{Y} \rangle\!\rangle \; \alpha \; \Gamma = \{\Delta : \mathsf{Ctx}\} \longrightarrow (\Gamma \; \text{-}[\; \mathcal{X} \;]\!\rightarrow \Delta) \longrightarrow \mathcal{Y} \; \alpha \; \Delta$$

"Renamable" terms are elements of $\Box\,\mathcal{X} \triangleq \langle\!\langle \mathcal{I}, \mathcal{X} \rangle\!\rangle$

A term $t \in (\Box\,\mathcal{X})\;\alpha\;\Gamma$ applied to $\rho : \Gamma \rightsquigarrow \Delta$ gives $t\;\rho : \mathcal{X}\;\alpha\;\Delta$

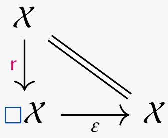
"Substitutable" terms are elements of $\langle\!\langle \mathcal{X} , \mathcal{X} \rangle\!\rangle$

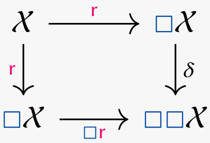
A term $t \in \langle\!\langle \mathcal{X}, \mathcal{X} \rangle\!\rangle\;\alpha\;\Gamma$ applied to $\sigma : \Gamma \; \text{-}[\mathcal{X}]\!\rightarrow \Delta$ gives $t\;\sigma : \mathcal{X}\;\alpha\;\Delta$

Renamability is coalgebra structure

Definition

A *coalgebra* for the comonad $\square$ on $\mathbf{Fam}_{\mathrm{S}}$ is an object $\mathcal{X}$ and a structure map $\mathsf{r} : \mathcal{X} \rightarrowtriangle \square\mathcal{X}$ compatible with the comonad structure:





$t \mapsto \mathsf{r}\ t\ \mathsf{id} = t$

$t \mapsto \rho, \varrho \mapsto \mathsf{r}\ (\mathsf{r}\ t\ \rho)\ \varrho = \mathsf{r}\ t\ (\varrho \circ \rho)$

The structure map $\mathsf{r} : \mathcal{X} \rightarrowtriangle \square\mathcal{X}$ acts as the *renaming operation*

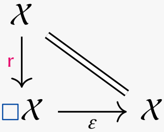
It turns a family with coalgebra structure into a renamable family

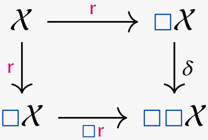
$$\mathsf{r} : \mathcal{X} \rightarrowtriangle \square\mathcal{X} = \forall\{\alpha\,\Gamma\} \rightarrow \mathcal{X}\,\alpha\,\Gamma \rightarrow (\forall\{\Delta\} \rightarrow (\Gamma \rightsquigarrow \Delta) \rightarrow \mathcal{X}\,\alpha\,\Delta)$$

Renamability is coalgebra structure

Definition

A *coalgebra* for the comonad $\square$ on $\mathbf{Fam}_{\mathsf{S}}$ is an object $\mathcal{X}$ and a structure map $\mathsf{r} : \mathcal{X} \rightarrowtriangle \square\mathcal{X}$ compatible with the comonad structure:





$t \mapsto \mathsf{r}\ t\ \mathsf{id} = t$

$t \mapsto \rho, \varrho \mapsto \mathsf{r}\ (\mathsf{r}\ t\ \rho)\ \varrho = \mathsf{r}\ t\ (\varrho \circ \rho)$

The structure map $\mathsf{r} : \mathcal{X} \rightarrowtriangle \square\mathcal{X}$ acts as the *renaming operation*

It turns a family with coalgebra structure into a renamable family

$$\mathsf{r} : \mathcal{X} \rightarrowtriangle \square\mathcal{X} = \forall\{\alpha\,\Gamma\,\Delta\} \to \mathcal{X}\,\alpha\,\Gamma \to (\Gamma \rightsquigarrow \Delta) \to \mathcal{X}\,\alpha\,\Delta$$

Substitutability is monoid structure

Definition

A *monoid* in a closed category $(\mathbf{Fam}_{\mathrm{S}}, \mathcal{I}, ⦇-,=⦈)$ is an object $\mathcal{M}$ with a unit $\eta : \mathcal{I} \rightarrow \mathcal{M}$ and multiplication $\mu : \mathcal{M} \rightarrow ⦇\mathcal{M}, \mathcal{M}⦈$ satisfying unit and associativity laws.

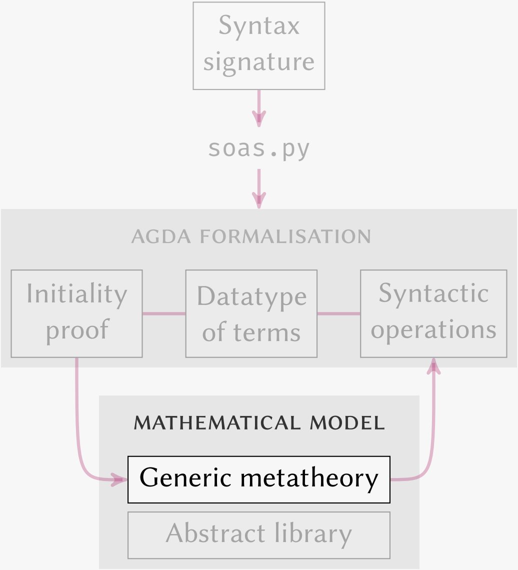
```
record Mon (𝓜 : Family_S) : Set where
  field  η : 𝓘  ⇾ 𝓜
         μ : 𝓜 ⇾ ⦇ 𝓜 , 𝓜 ⦈

         lunit  : {σ : Γ –[ 𝓜 ]→ Δ} {v : 𝓘 α Γ} → μ (η v) σ ≡ σ v
         runit  : {t : 𝓜 α Γ} → μ t η ≡ t
         assoc : {σ : Γ –[ 𝓜 ]→ Δ} {ς : Δ –[ 𝓜 ]→ Θ} {t : 𝓜 α Γ} →
                   μ (μ t σ) ς ≡ μ t (λ v → μ (σ v) ς)
```

Simultaneous substitution μ and associativity assoc reduces to single-variable substitution and the substitution lemma, respectively

```
[_/] : 𝓜 α Γ → 𝓜 β (α · Γ) → 𝓜 β Γ
[ s /] t = μ t (λ new → s; (old v) → η v)
```

```
   [r/] ([s/] t)
≡ [ [r/] s /] ([r/] t)
```

The categorical viewpoint leads to an abstract, natural characterisation of substitution structure



Syntax signature
soas.py
AGDA FORMALISATION
Initiality proof
Datatype of terms
Syntactic operations
MATHEMATICAL MODEL
Generic metatheory
Abstract library

Initial pointed $\sum$-algebras encode syntax with variables

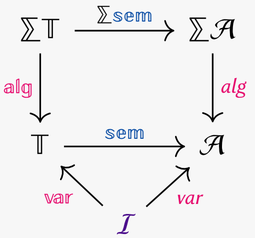
Signature endofunctor $\sum$ captures operator arities

For example, $\sum_{\text{Mon}}(\mathcal{A}) \triangleq \mathbb{1} + \mathcal{A} \times \mathcal{A}$

Algebras $\sum\mathcal{A} \to \mathcal{A}$ capture constructors of the syntax

For example, $[\text{unit}, \text{mult}] : (\mathbb{1} + \mathcal{A} \times \mathcal{A}) \to \mathcal{A} = \sum_{\text{Mon}}(\mathcal{A}) \to \mathcal{A}$

Initial pointed $\sum$-algebras $\sum\mathbb{T} \to \mathbb{T} \leftarrow \mathcal{I}$ capture structural recursion



$$\mathtt{sem}\,(\mathtt{alg}(t)) = \mathit{alg}\,(\textstyle\sum\mathtt{sem}(t))$$

$$\mathtt{sem}(\mathtt{var}\,(v)) = \mathit{var}\,(v)$$

Syntactic operations are constructed and proved correct using initiality

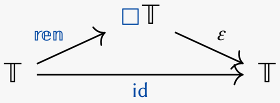
Syntactic operations can often be brought to the form $\mathbb{T} \to \mathcal{A}$

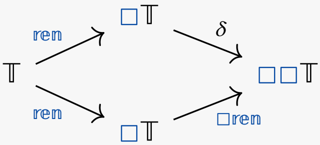
Suffices to show that $\mathcal{A}$ is a $(\Sigma + \mathcal{I})$-algebra

$$\mathsf{ren} : \mathbb{T} \to \Box\mathbb{T} \qquad \mathsf{sub} : \mathbb{T} \to \llparenthesis \mathbb{T}, \mathbb{T} \rrparenthesis$$

Correctness laws equate maps of the form $\mathbb{T} \to \mathcal{A}$

Suffices to show that edges are $(\Sigma + \mathcal{I})$-algebra homomorphisms

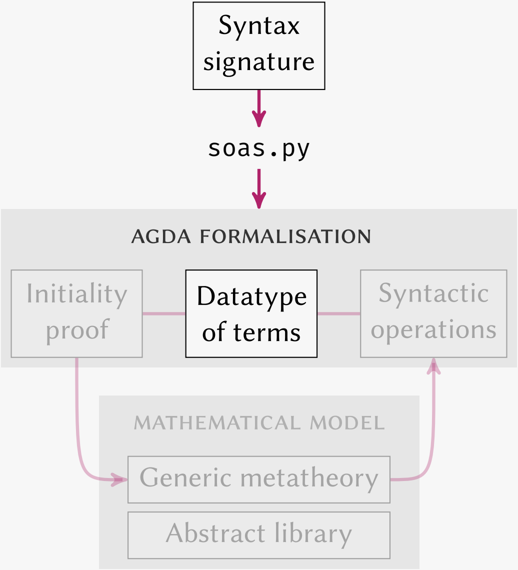




Coalgebra and monoid structure given by categorical reasoning

Compositional constructions and diagrammatic proofs

The substitution structure is defined and proved generically over any second-order signature


Syntax signature
soas.py
AGDA FORMALISATION
Initiality proof
Datatype of terms
Syntactic operations
MATHEMATICAL MODEL
Generic metatheory
Abstract library

Everything starts with a syntax description file

```
syntax STLC | Λ

type
  N   : 0-ary
  _↣_ : 2-ary | r30

term
  app : (α ↣ β)  α  →  β     | _$_ l20
  lam : α.β         →  α ↣ β | ƛ_  r10
```

Type syntax

data ΛT : Set where
  N : ΛT
  _↣_ : $\Lambda T \to \Lambda T \to \Lambda T$

Term syntax

data $\Lambda : \Lambda T \to \mathsf{Ctx}\ \Lambda T \to$ Set where
  var : $\mathcal{I}\ \alpha\ \Gamma \to \Lambda\ \alpha\ \Gamma$
  _\$_ : $\Lambda\ (\alpha \succ \beta)\ \Gamma \to \Lambda\ \alpha\ \Gamma \to \Lambda\ \beta\ \Gamma$
  ƛ_ : $\Lambda\ \beta\ (\alpha \cdot \Gamma) \to \Lambda\ (\alpha \succ \beta)\ \Gamma$

Operator symbols

data Λ_o : Set where
  $\mathsf{app}_o : \{\alpha\ \beta : \Lambda T\} \to \Lambda_o$
  $\mathsf{lam}_o : \{\alpha\ \beta : \Lambda T\} \to \Lambda_o$

Signature

$\Lambda{:}\mathsf{Sig} : \Lambda_o \to \mathsf{List}\ (\mathsf{Ctx} \times \Lambda T) \times \Lambda T$
$\Lambda{:}\mathsf{Sig} = \lambda\ \{\ \mathsf{app}_o \to [([],\alpha \succ \beta),([],\alpha)],\ \beta$
$\quad ;\ \mathsf{lam}_o \to [([\alpha],\beta)],\ \alpha \succ \beta\ \}$

The signature determines the endofunctor $\mathbb{\Sigma}$ and its algebras

$$\mathbb{\Sigma} : \mathsf{Family}_s \to \mathsf{Family}_s$$
$$\mathbb{\Sigma}\ \mathcal{X}\ \alpha\ \Gamma = \Sigma[\ o \in \Lambda_o\ ]\ (\alpha \equiv \mathsf{Sort}\ o \times \mathsf{Arg}\ (\mathsf{Arity}\ o)\ \mathcal{X}\ \Gamma)$$

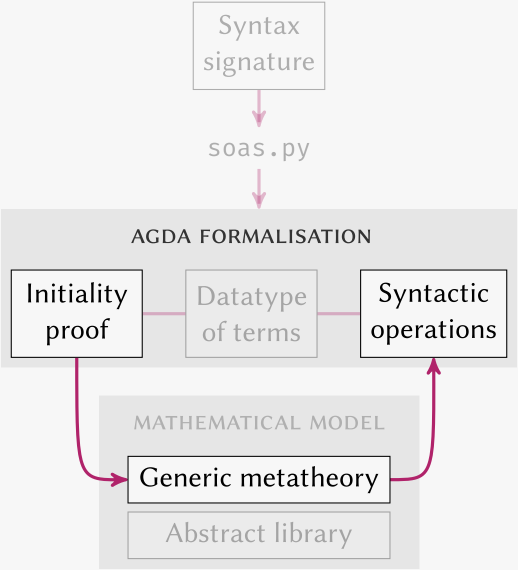
Associate an operator symbol with a tuple of $\mathcal{X}$-terms

The operator arity determines the type and context of subterms

$$f : \mathcal{X}\ (\alpha \rightarrowtail \beta)\ \Gamma,\ a : \mathcal{X}\ \alpha\ \Gamma \vdash \quad (\mathsf{app}_o, \mathsf{refl}, (f, a)) \quad : \mathbb{\Sigma}\ \mathcal{X}\ \beta\ \Gamma$$
$$b : \mathcal{X}\ \beta\ (\alpha \cdot \Gamma) \vdash \quad (\mathsf{lam}_o, \mathsf{refl}, b) \quad : \mathbb{\Sigma}\ \mathcal{X}\ (\alpha \rightarrowtail \beta)\ \Gamma$$

Pointed $\mathbb{\Sigma}$-algebras $\mathbb{\Sigma}\mathcal{X} \rightharpoonup \mathcal{X} \leftharpoondown \mathcal{I}$ carry the syntactic structure

The initial such algebra will also have substitution structure


Syntax signature
soas.py
AGDA FORMALISATION
Initiality proof
Datatype of terms
Syntactic operations
MATHEMATICAL MODEL
Generic metatheory
Abstract library

The inductive type of terms is the initial pointed $\sum$-algebra

Translation of Λ to any algebra $(\mathcal{A}, \mathit{var} : \mathcal{I} \rightarrow \mathcal{A}, \mathit{alg} : \sum\mathcal{A} \rightarrow \mathcal{A})$
Captures all forms of recursive definitions on the syntax

```
sem : Λ → 𝒜
sem (var v)     = var v
sem (app g a) = alg (appₒ, refl, (sem g , sem a))
sem (lam b)     = alg (lamₒ, refl, sem b)
```

Uniqueness of sem among all homomorphisms $(g, \langle \mathit{var} \rangle, \langle \mathit{alg} \rangle)$
Captures all forms of inductive equality proofs on the syntax

```
sem! : (t : Λ α Γ) → sem t ≡ g t
sem! (var v)                                   = sym ⟨var⟩
sem! (app f a) rewrite sem! f | sem! a = sym ⟨alg⟩
sem! (lam b)     rewrite sem! b             = sym ⟨alg⟩
```

The initiality proof for the syntax instantiates the generic metatheory

```
open import SOAS.Metatheory Λ:Sig Λ 𝕤𝕖𝕞 𝕤𝕖𝕞!

⟦_⟧ : Λ α Γ ⟶ (⟦ Γ ⟧ᶜ → ⟦ α ⟧ᵗ)
⟦_⟧ = ΣAlg.𝕤𝕖𝕞 record { … }

sub-lemma : {b : Λ β (α ∙ Γ)}{a : Λ α Γ} (γ : ⟦ Γ ⟧ᶜ) →
            ⟦ [a/] b ⟧ γ ≡ ⟦ b ⟧ (γ ⁺ ⟦ a ⟧)

data _⇝_ : Λ α Γ ⟶ Λ α Γ ⟶ Set where
  β-ƛ : {b : Λ β (α ∙ Γ)}     {a : Λ α Γ} → (ƛ b) $ a ⇝ [ a /] b
  ζ-$ : {f g : Λ (α ↣ β) Γ} {a : Λ α Γ} → f          ⇝ g
                                         → f $ a      ⇝ g $ a

sound : {t s : Λ α Γ} → t ⇝ s → (γ : ⟦ Γ ⟧ᶜ) → ⟦ t ⟧ γ ≡ ⟦ s ⟧ γ
sound (ζ-$ r)       γ rewrite sound r γ = refl
sound (β-ƛ {t}{b}) γ rewrite sub-lemma t b
  = cong ⟦ b ⟧ (dext λ{ new → refl ; (old v) → refl })
```

Move from language design to
valuable metatheory in seconds!

Example: first-order logic

```
syntax F

type
  *   : 0-ary
  N   : 0-ary


term
  false         → * | ⊥
  and : *  *    → * | _∧_
  not : *       → * | ¬_
  all : N.*     → * | ∀̲_  ...


theory
  ‘and’ commutative, idempotent
  ‘false’ annihilates ‘and’
  (DM∧)  a  b ▷ not (and (a, b)) = or (not(a), not(b))
  (∧P∀ᴸ) p : *  q : N.* ▷ and (p, all(x.q[x]))
                        = all (x. and(p,q[x]))  ...
```

data FT : Set where
  ★ : FT
  N : FT

data F : FT → Ctx FT → Set where
  var : $\mathcal{I}\ \alpha\ \Gamma \to \Lambda\ \alpha\ \Gamma$
  ⊥ : F ★ Γ
  _∧_ : F ★ Γ → F ★ Γ → F ★ Γ
  ¬_ : F ★ Γ → F ★ Γ
  ∀̲_ : F ★ (N · Γ) → F ★ Γ ...

Example: first-order logic

```
data _▷_⊢_≈ₐ_ : ∀(𝔐 Γ) → F 𝔐 α Γ → F 𝔐 α Γ → Set where
  ∧C    : ⁅ * ⁆ ⁅ * ⁆      ▷ ∅ ⊢ 𝔞 ∧ 𝔟            ≈ₐ 𝔟 ∧ 𝔞
  ⊥X∧ᴸ  : ⁅ * ⁆            ▷ ∅ ⊢ ⊥ ∧ 𝔞            ≈ₐ ⊥
  DM∧   : ⁅ * ⁆ ⁅ * ⁆      ▷ ∅ ⊢ ¬ (𝔞 ∧ 𝔟)        ≈ₐ (¬ 𝔞) ∨ (¬ 𝔟)
  ∧P∀ᴸ  : ⁅ * ⁆ ⁅ N ⊩ * ⁆  ▷ ∅ ⊢ 𝔞 ∧ (∀̲ 𝔟⟨ x₀ ⟩) ≈ₐ ∀̲ (𝔞 ∧ 𝔟⟨ x₀ ⟩)    ...

ax_with_   : {s t : F 𝔐 α Π}          → 𝔐 ▷ Π ⊢ s ≈ₐ t →
             (ζ : MSub Γ 𝔐 𝔑) → 𝔑 ▷ Π ⊢ msub s ζ ≈ msub t ζ

cong[_]in_ : {s t : F 𝔐 β (Π ∔ Γ)}       → 𝔐 ▷ (Π ∔ Γ) ⊢ s ≈ₐ t →
             (u : F (𝔐 ⁅ Π ⊩ β ⁆) α Γ) → 𝔐 ▷ Γ ⊢ msub₁ u s ≈ msub₁ u t


∧P∀ᴿ : ⁅ N ⊩ * ⁆ ⁅ * ⁆ ⊢ (∀̲ 𝔞⟨ x₀ ⟩) ∧ 𝔟 ≈ ∀̲ (𝔞⟨ x₀ ⟩ ∧ 𝔟)
∧P∀ᴿ = begin
    (∀̲ 𝔞⟨ x₀ ⟩) ∧ 𝔟    ≈⟨ ax ∧C with ⟪ (∀̲ 𝔞⟨ x₀ ⟩) ◂ 𝔟 ⟫ ⟩
    𝔟 ∧ (∀̲ 𝔞⟨ x₀ ⟩)    ≈⟨ ax ∧P∀ᴸ with ⟪ 𝔟 ◂ 𝔞⟨ x₀ ⟩ ⟫ ⟩
    ∀̲ (𝔟 ∧ 𝔞⟨ x₀ ⟩)    ≈⟨ cong[ ax ∧C with ⟪ 𝔟 ◂ 𝔞⟨ x₀ ⟩ ⟫ ]in ∀̲ (◌ᶜ⟨ x₀ ⟩) ⟩
    ∀̲ (𝔞⟨ x₀ ⟩ ∧ 𝔟)    ■
```

Conclusions

A signature captures the full syntactic structure of a language
No definition or proof requires particular, syntax-specific insight

Categorical viewpoint lends generality and deep understanding
Capture and abstract over common constructions from literature

Second-order extensions follow naturally from the theory
Further analysis of metasubstitution is ongoing work

Generality, practicality, efficiency – choose three!

Source code, documentation, and examples
can be found on the project page:

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

Give it a try!