A Unified Framework for Initial Semantics, and It's 2-Functorial!

Thomas Lamiaux, with Benedikt Ahrens and Ambroise Lafont

Genova, April 2025

Introduction

Which Initial Semantics?

Model Syntax and Substitution

We want a reusable framework to:

1. Model the syntax of PL with variable binding,
2. With its substitution structure,
3. With a recursion principle preserving its properties

Prove an Initiality Theorem

We want to prove an initiality theorem asserting under reasonable conditions, whether a signature has an initial model or not. Follows:

1. The existence provides us with syntax and its properties
2. The initiality provides us with a recursion principle for it

Which Initial Semantics?

We Focused On

Three Traditions:

1. Σ-monoids
2. Modules over monads
3. Heteregenous Substitution Systems

But Not On

- Meta-variables, "second-order" syntax
- Semantics like equations or reduction rules
- Any other traditions, like nominal syntax

Important Papers

Some basic papers on the subject:

1999 "Abstract Syntax and Variable Binding", Fiore, Plotkin, and Turi

2003 "Semantic analysis of normalisation by evaluation for typed lambda calculus", Fiore

2004 "Substitution in non-wellfounded syntax with variable binding", Matthes and Uustalu

2010 "Modules over monads and initial semantics", Hirschowitz and Maggesi

2010 "Typed Abstract Syntax", Zsido

2015 "Heterogeneous Substitution Systems Revisited", Ahrens and Matthes

2017 "List Objects with Algebraic Structure", Fiore and Saville

2018 "High-Level Signatures and Initial Semantics", Ahrens, Hirschowitz, Lafont, and Maggesi

2020 "A Cellular Howe Theorem", Borthelle, Hirschowitz, and Lafont

2022 "Implementing a category-theoretic framework for typed abstract syntax", Ahrens, Matthes, and Mörtberg

Goals, and Challenges

Goals

1. Understand the links between the different notions of signatures
2. Understand the links between the different notions of models
3. How do different representation of models relates in terms of models, for instance, $[\mathbb{F}, \mathrm{Set}]$ vs $[\mathrm{Set}, \mathrm{Set}]$

Goals, and Challenges

Goals

1. Understand the links between the different notions of signatures
2. Understand the links between the different notions of models
3. How do different representation of models relates in terms of models, for instance, $[\mathbb{F}, \mathrm{Set}]$ vs $[\mathrm{Set}, \mathrm{Set}]$

Challenges

1. A lot of (small) conference papers published without annexes:
   - Very little fully written proof, often technical
   - Information is spread out in a bunch of papers
   - No real bibliographic work on the subject
2. A lot of small technical variations in the literature

Our work

For 1. and 2. : A Unified Framework

We appropriately generalized and combined many different ideas:

1. Monoidal categories provide us with a reusable framework
2. Modules over monoids provide us with an abstract framework
3. Signatures with strength naturally appears as well-behaved signatures to prove an initiality theorem.
4. Heterogeneous Substitution Systems enables us to prove the initiality theorem modularly

We used it as a cornerstone and wrote a 13 pages related work!

Our work

For 1. and 2. : A Unified Framework

We appropriately generalized and combined many different ideas:

1. Monoidal categories provide us with a reusable framework
2. Modules over monoids provide us with an abstract framework
3. Signatures with strength naturally appears as well-behaved signatures to prove an initiality theorem.
4. Heterogeneous Substitution Systems enables us to prove the initiality theorem modularly

We used it as a cornerstone and wrote a 13 pages related work!

For 3. : 2-Functoriality and Applications

We prove, and use to 2-functorial of models to lift relations on the base categories to the models

Preview

What to Get Out of This Talk?

1. An understanding of the different frameworks
2. An understanding of how they are related, and assemble
3. Why 2-functoriality is important and useful

Abstract Framework

Modules over Monoids

Abstract Framework

Modules over Monoids

Modeling Substitution

Monoidal Categories

Different Instances

Model languages for different kind of instances

- Endofunctor categories $[C, C]$, e.g. $[\mathrm{Set}, \mathrm{Set}]$ or $[\mathrm{Set}^T, \mathrm{Set}^T]$
- Some functor categories, e.g. $[\mathbb{F}, \mathrm{Set}]$
- More complicated stuff: $\oint_T [Set^T, Set^T]$

Monoidal Categories

Different Instances

Model languages for different kind of instances

- Endofunctor categories $[C, C]$, e.g. $[\mathrm{Set}, \mathrm{Set}]$ or $[\mathrm{Set}^T, \mathrm{Set}^T]$
- Some functor categories, e.g. $[\mathbb{F}, \mathrm{Set}]$
- More complicated stuff: $\oint_T [Set^T, Set^T]$

Monoidal Categories

A monoidal category is a tuple $(C, \otimes, I, \alpha, \lambda, \rho)$, where

- C is a category,
- $\_ \otimes \_ : C \times C \to C$ is a bifunctor called the monoidal product,
- $I : C$ is an object called the unit,
- α, λ, ρ are natural isomorphisms such that $\alpha_{X,Y,Z} : (X \otimes Y) \otimes Z \cong X \otimes (Y \otimes Z)$, $\lambda_X : I \otimes X \cong X$, $\rho_X : X \otimes I \cong X$

Monoids

Substitution on Monoidal Categories

Model substitution over a monoidal category

Monoids

A monoid over a monoidal category C is a tuple (R, μ, η) where:

- R is an object of C
- A multiplication $\mu : R \otimes R \to R$
- The unit $\eta : I \to R$ are morphisms of C
- Satisfying the monoid laws: associativity and unit-law

Example

Untyped Lambda Calulus on $[\mathrm{Set}, \mathrm{Set}]$

- $\Lambda : \mathrm{Set} \to \mathrm{Set}$ associates well-scoped terms to contexts
  - functoriality corresponds to renaming
- $\mu_\Gamma : \Lambda(\Lambda(\Gamma)) \to \Lambda(\Gamma)$ corresponds to flattening
- $\eta_\Gamma : \Gamma \to \Lambda(\Gamma)$ corresponds to variables
  - naturality corresponds to compatibility with renaming

Example

Untyped Lambda Calulus on $[\mathrm{Set}, \mathrm{Set}]$

- $\Lambda : \mathrm{Set} \to \mathrm{Set}$ associates well-scoped terms to contexts
  - functoriality corresponds to renaming
- $\mu_\Gamma : \Lambda(\Lambda(\Gamma)) \to \Lambda(\Gamma)$ corresponds to flattening
- $\eta_\Gamma : \Gamma \to \Lambda(\Gamma)$ corresponds to variables
  - naturality corresponds to compatibility with renaming

Substitution Is Modeled Indirectly

Monoids on $[\mathrm{Set}, \mathrm{Set}]$ are equivalent to (relative) monads. Given $f : \Gamma \to \Lambda(\Delta)$:

$$\sigma(f) : \Lambda(\Gamma) \xrightarrow{\Lambda(f)} \Lambda(\Lambda(\Delta)) \xrightarrow{\mu} \Lambda(\Delta)$$

Example

Untyped Lambda Calulus on $[\mathrm{Set}, \mathrm{Set}]$

- $\Lambda : \mathrm{Set} \to \mathrm{Set}$ associates well-scoped terms to contexts
  - functoriality corresponds to renaming
- $\mu_\Gamma : \Lambda(\Lambda(\Gamma)) \to \Lambda(\Gamma)$ corresponds to flattening
- $\eta_\Gamma : \Gamma \to \Lambda(\Gamma)$ corresponds to variables
  - naturality corresponds to compatibility with renaming

Substitution Is Modeled Indirectly

Monoids on $[\mathrm{Set}, \mathrm{Set}]$ are equivalent to (relative) monads. Given $f : \Gamma \to \Lambda(\Delta)$:

$$\sigma(f) : \Lambda(\Gamma) \xrightarrow{\Lambda(f)} \Lambda(\Lambda(\Delta)) \xrightarrow{\mu} \Lambda(\Delta)$$

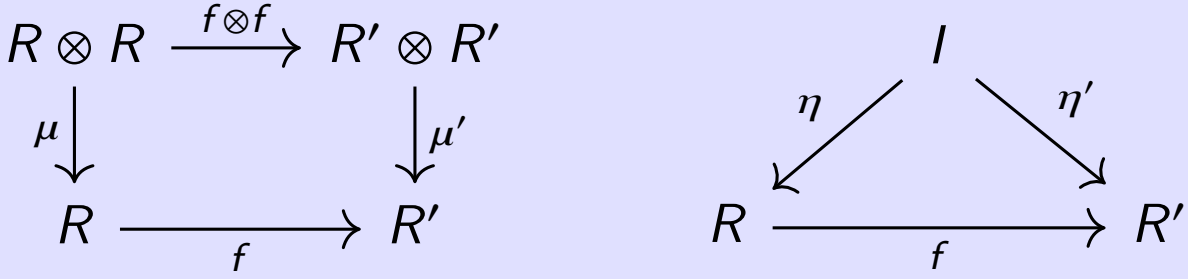
Constructors

This does not model constructors, we need an additional notion

Morphisms of Monoids Do Not Model Constructors

Morphisms of Monoids

A morphism of monoids $(R, \mu, \eta) \to (R', \mu', \eta')$ is a morphism $f : R \to R'$ preserving the multiplication and the unit:



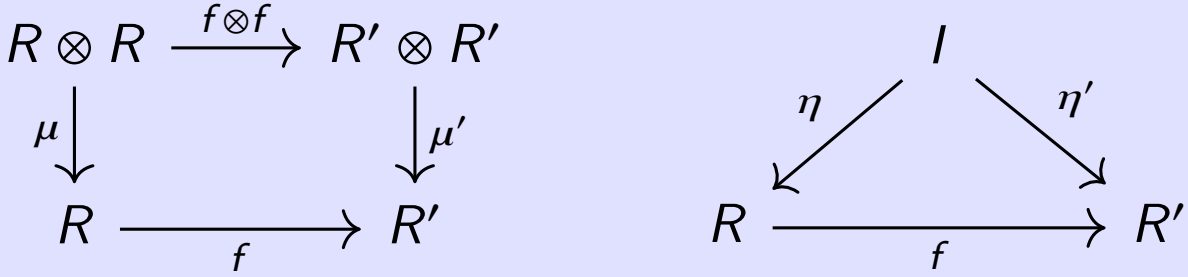
Morphisms of monoids are too strict

You can not model constructors by morphisms of monoids as they would have to preserve the unit, hence variables. Yet, $\lambda x.x \neq y$.

Morphisms of Monoids Do Not Model Constructors

Morphisms of Monoids

A morphism of monoids $(R, \mu, \eta) \to (R', \mu', \eta')$ is a morphism $f : R \to R'$ preserving the multiplication and the unit:



Morphisms of monoids are too strict

You can not model constructors by morphisms of monoids as they would have to preserve the unit, hence variables. Yet, $\lambda x.x \neq y$.

Solution?

Let us remove the unit!

Modules over a Monoid

Module over a Monoid

Given a monoid R, a R-module is a tuple (M, p^M) where:

- M is an object of C
- $p^M : M \otimes R \longrightarrow M$ is a morphism called module substitution
- compatible with μ and η in some way

Modules over a Monoid

Module over a Monoid

Given a monoid R, a R-module is a tuple (M, p^M) where:

- M is an object of C
- $p^M : M \otimes R \longrightarrow M$ is a morphism called module substitution
- compatible with μ and η in some way

Basic Building Blocks

1. Every monoid (R, μ, η) is a module (R, η) over itself
2. $M \times M'$ and $M + M'$ are modules under reasonable conditions

Modules over a Monoid

Module over a Monoid

Given a monoid R, a R-module is a tuple (M, p^M) where:

- M is an object of C
- $p^M : M \otimes R \longrightarrow M$ is a morphism called module substitution
- compatible with μ and η in some way

Basic Building Blocks

1. Every monoid (R, μ, η) is a module (R, η) over itself
2. $M \times M'$ and $M + M'$ are modules under reasonable conditions

Variable Binding Is Instance Specific

On $[\mathrm{Set}, \mathrm{Set}]$, there is a module $R^{(n)}(\Gamma) := R(\Gamma + n)$

Module Morphisms Models Constructors

Module Morphism

A morphism of R-modules $(M, p^M) \to (M', p^{M'})$ is a morphism $r : M \to M'$ of C preserving the module substitutions:

$$\begin{array}{ccc} M \otimes R & \xrightarrow{r \otimes R} & M' \otimes R \\ {\scriptstyle p^M}\downarrow & & \downarrow{\scriptstyle p^{M'}} \\ M & \xrightarrow[r]{} & M' \end{array}$$

Module Morphisms Models Constructors

Module Morphism

A morphism of R-modules $(M, p^M) \to (M', p^{M'})$ is a morphism $r : M \to M'$ of C preserving the module substitutions:

$$\begin{array}{ccc} M \otimes R & \xrightarrow{r \otimes R} & M' \otimes R \\ {\scriptstyle p^M}\downarrow & & \downarrow{\scriptstyle p^{M'}} \\ M & \xrightarrow[r]{} & M' \end{array}$$

Untyped Lambda Calculus

The untyped lambda calculus is modeled on $[\mathrm{Set}, \mathrm{Set}]$ by:

- A morphism $R \times R \to R$ modeling app
- A morphism $R^{(1)} \to R$ modeling abs

Hence, by $R \times R + R^{(1)} \to R$

Abstract Framework

Modules over Monoids

Signatures, and Models

Signatures

What Do We Want?

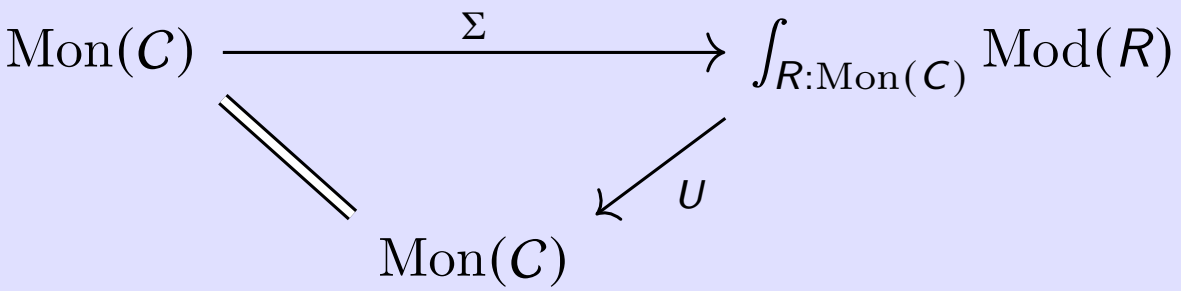
We want to represent languages by a monoid $R : \mathrm{Mon}(C)$, and a module morphism over it $X \to R$ for some $X : \mathrm{Mod}(R)$.

Signatures

Modules form a contravariant functor over monoids:

$$\mathrm{Mod} : \mathrm{Mon}(C)^{\mathrm{op}} \to \mathbf{Cat}$$

A signature is a functor $\Sigma : \mathrm{Mon}(C) \longrightarrow \int_{R:\mathrm{Mon}(C)} \mathrm{Mod}(R)$:



Models

Models

A model of a signature Σ is a tuple (R, r) where

- $R : \mathrm{Mon}(C)$ is a monoid
- $r : \Sigma(R) \to R$ is a morphism of R-modules.

Models

Models

A model of a signature Σ is a tuple (R, r) where

- $R : \mathrm{Mon}(C)$ is a monoid
- $r : \Sigma(R) \to R$ is a morphism of R-modules.

Morphism of Model

A morphism of Σ-models $(R, r) \to (R', r')$ is a morphism of monoids $f : R \to R'$ compatible with r and r':

$$\begin{array}{ccc} \Sigma(R) & \xrightarrow{r} & R \\ {\scriptstyle \Sigma(f)}\downarrow & & \downarrow{\scriptstyle f} \\ f^*\Sigma(R') & \xrightarrow[f^*r']{} & f^*R' \end{array}$$

Models

Models

A model of a signature Σ is a tuple (R, r) where

- $R : \mathrm{Mon}(C)$ is a monoid
- $r : \Sigma(R) \to R$ is a morphism of R-modules.

Morphism of Model

A morphism of Σ-models $(R, r) \to (R', r')$ is a morphism of monoids $f : R \to R'$ compatible with r and r':

$$\begin{array}{ccc} \Sigma(R) & \xrightarrow{r} & R \\ {\scriptstyle \Sigma(f)}\downarrow & & \downarrow{\scriptstyle f} \\ f^*\Sigma(R') & \xrightarrow[f^*r']{} & f^*R' \end{array}$$

There are non representable signatures

$\mathcal{P} \circ \Theta$ does not have an initial model, so we need an initiality theorem

Initiality Theorem

Signatures with Strength

Towards an Initiality Theorem

Goal

We want an initiality theorem asserting under reasonable conditions, whether a signature has an initial model or not.

Signatures Are Too General

- There is no known constructive initiality theorem for signatures
- There is no known criterion for the product or coproduct of representable signatures to still be representable

Towards an Initiality Theorem

Goal

We want an initiality theorem asserting under reasonable conditions, whether a signature has an initial model or not.

Signatures Are Too General

- There is no known constructive initiality theorem for signatures
- There is no known criterion for the product or coproduct of representable signatures to still be representable

Be More Reasonable

We want to restrict the shape of the module substitutions:

$$\Sigma(R) \otimes R \to \Sigma(R)$$

Towards Signatures with Strength

Fixpoint Theorem

Given a signature Σ, if M is a model of Σ then $\underline{I} + \Sigma(M)$ is a model of Σ. Moreover, if M is initial, then $M \cong \underline{I} + \Sigma(M)$.

Towards Signatures with Strength

Assume $H : C \to C$, and (R, η, μ) a monoid, we want to build a C morphism that is a module morphism:

$$H(R) \otimes R \dashrightarrow^{???} H(R)$$

Towards Signatures with Strength

Fixpoint Theorem

Given a signature Σ, if M is a model of Σ then $\underline{I} + \Sigma(M)$ is a model of Σ. Moreover, if M is initial, then $M \cong \underline{I} + \Sigma(M)$.

Towards Signatures with Strength

Assume $H : C \to C$, and (R, η, μ) a monoid, we want to build a C morphism that is a module morphism:

$$H(R) \otimes R \overset{???}{\dashrightarrow} H(R \otimes R) \xrightarrow{H(\mu)} H(R)$$

Towards Signatures with Strength

Fixpoint Theorem

Given a signature Σ, if M is a model of Σ then $\underline{I} + \Sigma(M)$ is a model of Σ. Moreover, if M is initial, then $M \cong \underline{I} + \Sigma(M)$.

Towards Signatures with Strength

Assume $H : C \to C$, and (R, η, μ) a monoid, we want to build a C morphism that is a module morphism:

$$H(R) \otimes R \overset{???}{\dashrightarrow} H(R \otimes R) \xrightarrow{H(\mu)} H(R)$$

Signatures with Strength

Unfolding the required properties leads to signatures with strength!

Signatures with Strength

Signatures with Strength

A signature with strength is a pair (H, θ) where:

- $H : C \to C$ is an endofunctor,
- A natural transformation called strength θ, for all $A : C$ and pointed object $b : I \to B$:

$$\theta_{A,b} : H(A) \otimes B \longrightarrow H(A \otimes B)$$

- Such that θ it is compatible with α and ρ

Basic Building Blocks

- There is a trivial signature with strength Θ
- $M \times M'$ and $M + M'$ are signatures under reasonable conditions
- On $[\mathrm{Set}, \mathrm{Set}]$, there is a signature with strength $\Theta^{(n)}$

Links with Signatures

Signatures with Strength Are Signatures

1. There is a functor from signatures with strength to signatures
2. It preserves the basic building blocks under reasonable conditions

Models Are The Same

The notion of Σ-monoids used by Fiore et al. is exactly the notion of model specialized to signatures with strength.

Links with Signatures

Signatures with Strength Are Signatures

1. There is a functor from signatures with strength to signatures
2. It preserves the basic building blocks under reasonable conditions

Models Are The Same

The notion of Σ-monoids used by Fiore et al. is exactly the notion of model specialized to signatures with strength.

Consequences

1. You can think in terms of signatures and modules,
2. But use signatures with strength to specify your language, and use the initiality theorem

Initiality Theorem

Initiality Theorem

Let $\mathcal{C}$ be a monoidal category, such that:

1. It has initial object, binary coproducts, ω-colimits,
2. They are preserves by $\_ \otimes Z$, for all $Z : \mathcal{C}$

Then, any ω-cocontinuous signature with strength (H, θ):

1. The associated signature has an initial model $\overline{H}$,
2. Its underlying object is the initial algebra $\mu A.(I + H(A))$

Initiality Theorem

Initiality Theorem

Let $\mathcal{C}$ be a monoidal category, such that:

1. It has initial object, binary coproducts, ω-colimits,
2. They are preserves by $\_ \otimes Z$, for all $Z : \mathcal{C}$

Then, any ω-cocontinuous signature with strength (H, θ):

1. The associated signature has an initial model $\overline{H}$,
2. Its underlying object is the initial algebra $\mu A.(I + H(A))$

Untyped Lambda Calculus

1. We work with $[\mathrm{Set}, \mathrm{Set}]$ which satisfies all the hypotheses
2. The signature is $\Theta \times \Theta + \Theta^{(1)}$ which is ω-cocontinuous modularly

Proving the Initiality Theorem

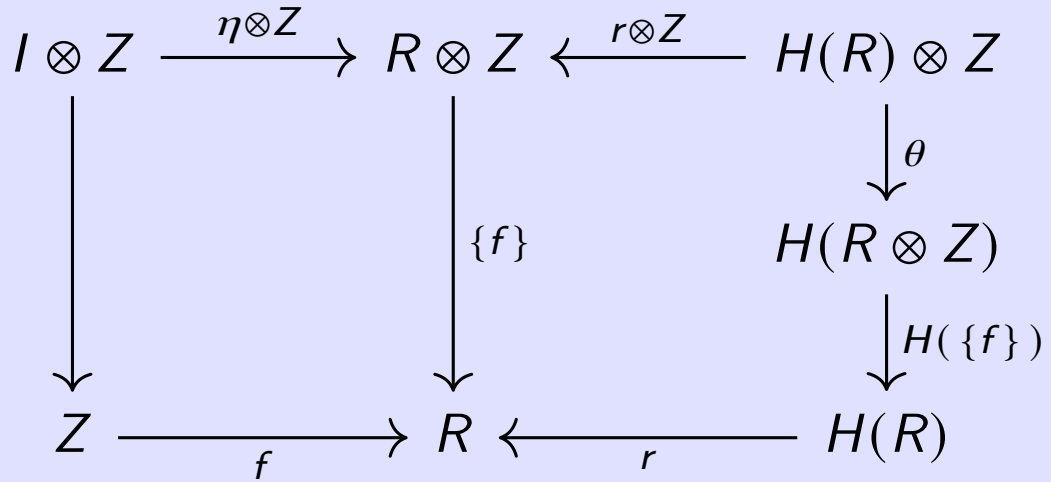
Heteregoneous Substitution Systems

Heterogeneous Substitution System

Heterogeneous Substitution System

A heterogeneous substitution system (hss) for a signature with strength (H, θ) is a tuple (R, η, r) where:

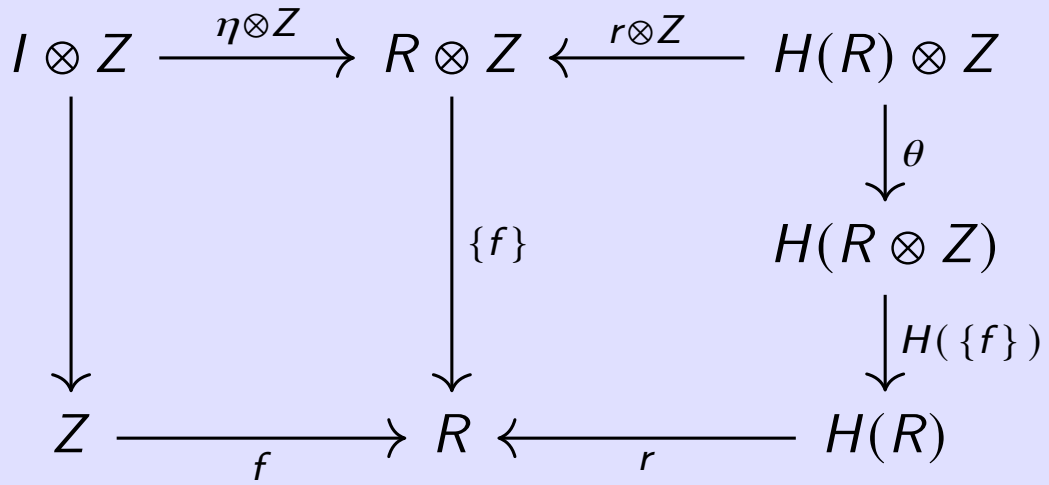
- An algebra (R, η, r) for $\underline{I} + H$,
- Such that for all $(Z, e) : \mathrm{Ptd}(C)$ and $f : Z \to R$, there is a unique morphism $\{f\} : R \otimes Z \to R$ such that:



Hss to Models

Hss induces Models

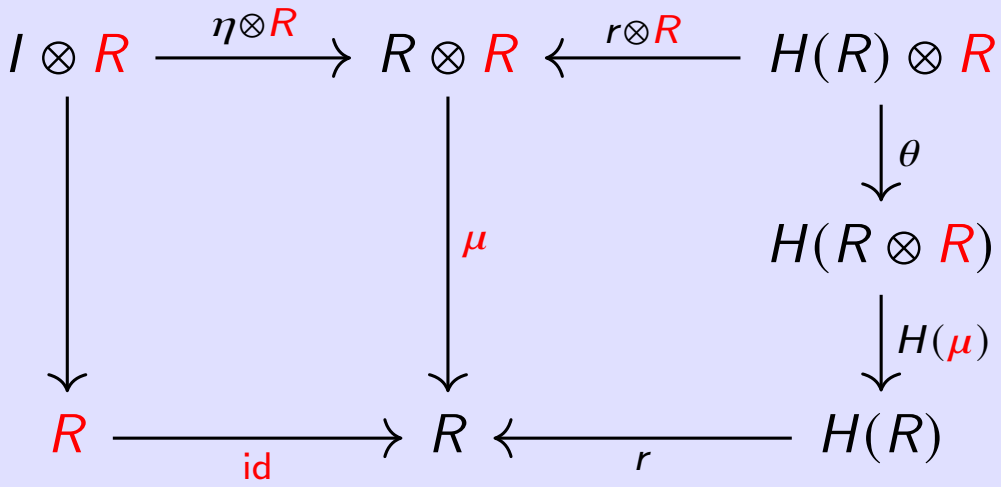
You instantiate the Hss to build $\mu : R \otimes R \to R$, and prove the different properties using uniqueness of $\{f\}$:



Hss to Models

Hss induces Models

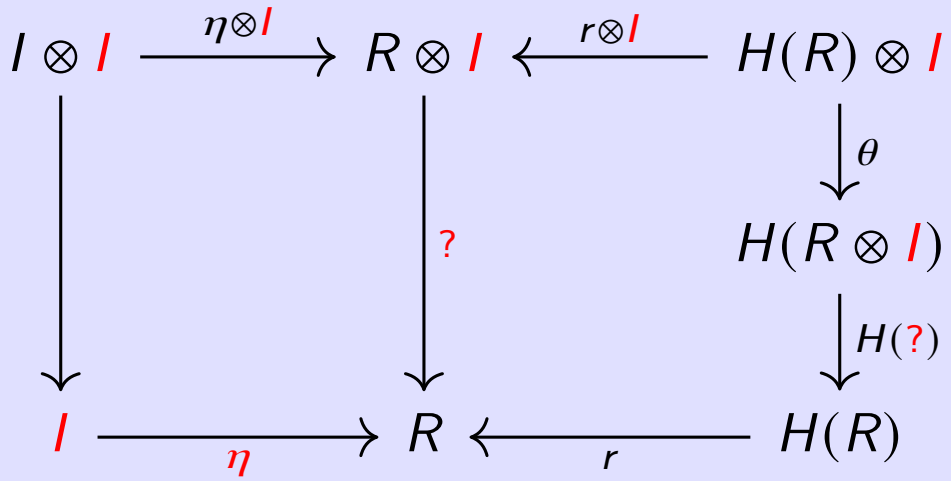
You instantiate the Hss to build $\mu : R \otimes R \to R$, and prove the different properties using uniqueness of $\{f\}$:



Hss to Models

Hss induces Models

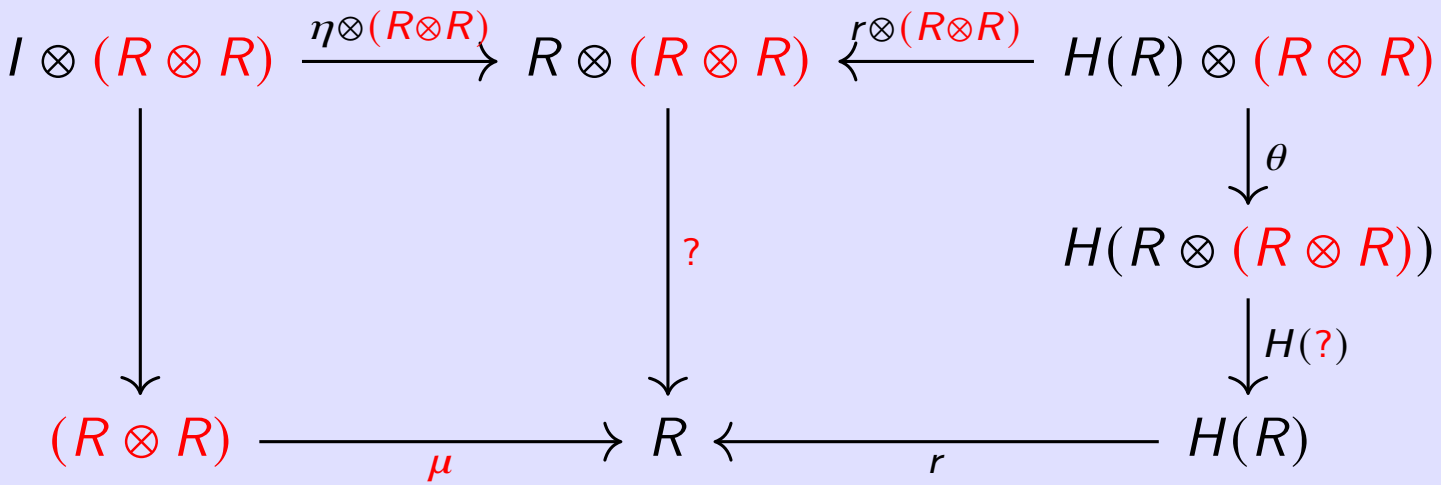
You instantiate the Hss to build $\mu : R \otimes R \to R$, and prove the different properties using uniqueness of $\{f\}$:



Hss to Models

Hss induces Models

You instantiate the Hss to build $\mu : R \otimes R \to R$, and prove the different properties using uniqueness of $\{f\}$:



Hss to Models

Hss induces Models

You instantiate the Hss to build $\mu : R \otimes R \to R$, and prove the different properties using uniqueness of $\{f\}$:

Proof Sketch

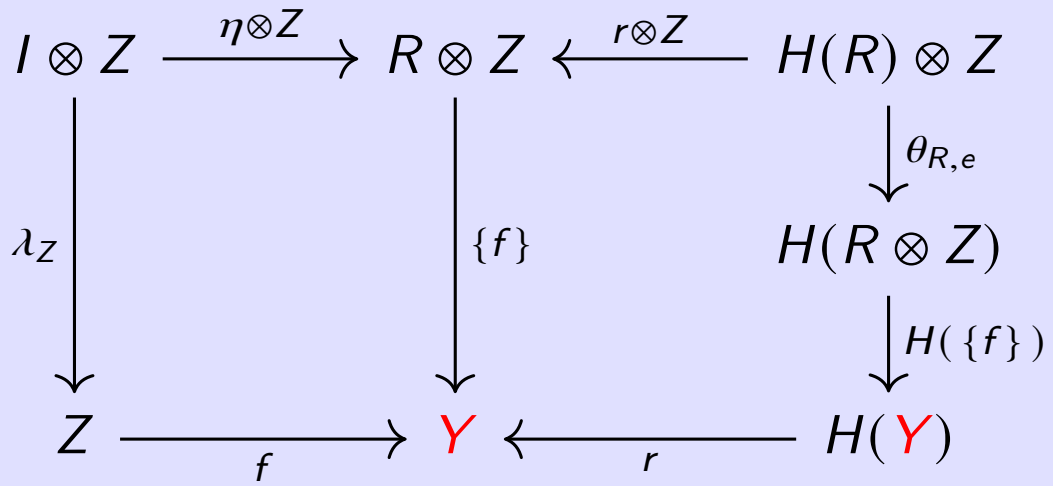
1. You use "Adamek Theorem" to build an initial algebra
2. You use "Generalized Mendler Iterations" to turn it into a Hss
3. You use a "Fusion Law" to prove it is initial as a model

Comparison with Fiore et al's Work

Parametrised Initiality v.s. Hss

They use "Parametrised Initiality" which once instantiated gives a variant of hss:

- Forall $f : Z \to Y$, there is a unique morphism $\{f\} : R \otimes Z \to Y$ such that:



Comparison with Fiore et al's Work

Parametrised Initiality v.s. Hss

They use "Parametrised Initiality" which once instantiated gives a variant of hss:

- It makes it easier to prove initiality
- It can not be applied to non-well-founded syntax

Comparison with Fiore et al's Work

Parametrised Initiality v.s. Hss

They use "Parametrised Initiality" which once instantiated gives a variant of hss:

- It makes it easier to prove initiality
- It can not be applied to non-well-founded syntax

They Use the Same Theorem

They use [5, Theorem 4.8] to prove "parametrised initiality":

1. It looks very different from "Generalized Mendler Iterations",
2. It can be split into a theorem and a corrolary
3. Once split, both theorems are direction application of each others

2-Functoriality, and Applications

2-Functoriality, and Applications

2-Functoriality of Models

What Do We Want?

Relating different kind of contexts

We would to be able to relate different representation of contexts, for instance $[\mathbb{F}, \mathrm{Set}]$ and $[\mathrm{Set}, \mathrm{Set}]$.

A generalized recursion principle

Recursion provided by initiality is limited to model over the same type system T as it is hardcoded in the base category $[\mathrm{Set}^T, \mathrm{Set}^T]$

Goal

We want a generic method to relate models over different base categories

Computing Models is 2-Functorial

Parametric Signatures

There is a 2-category of parametric signatures:

- An object consists of a monoidal category C and a signature Σ
- A 1-cell between $(C, \Sigma) \to (D, \Sigma')$ consists of a monoidal functor $F : C \to D$ and a natural transformation $\alpha : \Sigma' \circ F \to F \circ \Sigma$
- c.f. the paper

Models

There is a 2-functor Model : **ModSig** $\to$ **Cat** that computes the category of models of any module signature.

Proof: Not so trivial, c.f the paper.

Side Note on Initiality

Restricting to Nice Signatures with Strength

You can restrict the 2-category **ModSig** to **NicePSigStrength** of monoidal categories and signature with strength satisfying the initiality theorem.

Initiality Theorem

The category of models of a nice signature with strength, as computed by the following 2-functor, always has an initial object:

$$\textbf{NicePSigStrength} \longrightarrow \textbf{ModSig} \xrightarrow{\text{Model}} \textbf{Cat}$$

2-Functoriality, and Applications

Relating Different Representations of Contexts

Interpreting Binding Signatures

Untyped Binding Signatures

An untyped binding signature S is a family of list of natural numbers $[n_1, ...n_p]_I$.

Binding-Friendly Monoidal Categories

A monoidal category C is said binding-friendly if it has

- Finite products left-preserved by the tensor;
- Non-empty coproducts left-preserved by the tensor;
- An exponentiable unit I.

It is a 2-Category

It forms a 2-category **BindMonCat** with monoidal functors preserving the structure, and monoidal natural transformation.

Relating Different Representations of Context

Interpreting Binding Signatures

Any binding signature S induces a 2-functor **BindMonCat** $\to$ **Cat** computing its associated category of models:

$$\mathbf{BindMonCat} \xrightarrow{\mathrm{Sem}_S} \mathbf{ModSig} \xrightarrow{\mathrm{Model}} \mathbf{Cat}$$

Relating Different Representations of Context

Interpreting Binding Signatures

Any binding signature S induces a 2-functor **BindMonCat** $\to$ **Cat** computing its associated category of models:

$$\textbf{BindMonCat} \xrightarrow{\mathrm{Sem}_S} \textbf{ModSig} \xrightarrow{\mathrm{Model}} \textbf{Cat}$$

Relating [$\mathbb{F}$,Set] and [Set,Set]

1. There is a coreflection $[\mathbb{F}, \mathrm{Set}] \simeq [\mathrm{Set}, \mathrm{Set}]_f \leftrightarrows [\mathrm{Set}, \mathrm{Set}]$,
2. This form a coreflection in **BindMonCat** (c.f paper)
3. For any binding signature S, there is a coreflection:

$$\mathrm{Model}_S([\mathbb{F}, \mathrm{Set}]) \simeq \mathrm{Model}_S([\mathrm{Set}, \mathrm{Set}]_f) \leftrightarrows \mathit{Model}_S([\mathrm{Set}, \mathrm{Set}])$$

2-Functoriality, and Applications

A Generalized Recursion Principles for Simply-Typed languages

Intuition

A generalized recursion principle

Recursion provided by initiality is limited to model over the same type system T as it is hard-coded in the base category $[\mathrm{Set}^T, \mathrm{Set}^T]$

Solution?

1. Recursion between languages with different type system should rely on a translating of the type systems $g : T \to T'$

Intuition

A generalized recursion principle

Recursion provided by initiality is limited to model over the same type system T as it is hard-coded in the base category $[\mathrm{Set}^T, \mathrm{Set}^T]$

Solution?

1. Recursion between languages with different type system should rely on a translating of the type systems $g : T \to T'$
2. Any function $g : T \to T'$ generates an adjunction, and the functor $[\mathrm{Set}^{T'}, \mathrm{Set}^{T'}] \to [\mathrm{Set}^T, \mathrm{Set}^T]$ is monoidal

$$\mathrm{Set}^T \underset{\longleftarrow}{\overset{\longrightarrow}{\bot}} \mathrm{Set}^{T'} \qquad [\mathrm{Set}^T, \mathrm{Set}^T] \underset{\longleftarrow}{\overset{\longrightarrow}{\bot}} [\mathrm{Set}^{T'}, \mathrm{Set}^{T'}]$$

Intuition

A generalized recursion principle

Recursion provided by initiality is limited to model over the same type system T as it is hard-coded in the base category $[\mathrm{Set}^T, \mathrm{Set}^T]$

Solution?

1. Recursion between languages with different type system should rely on a translating of the type systems $g : T \to T'$
2. Any function $g : T \to T'$ generates an adjunction, and the functor $[\mathrm{Set}^{T'}, \mathrm{Set}^{T'}] \to [\mathrm{Set}^T, \mathrm{Set}^T]$ is monoidal

$$\mathrm{Set}^T \underset{\longleftarrow}{\overset{\longrightarrow}{\bot}} \mathrm{Set}^{T'} \qquad [\mathrm{Set}^T, \mathrm{Set}^T] \underset{\longleftarrow}{\overset{\longrightarrow}{\bot}} [\mathrm{Set}^{T'}, \mathrm{Set}^{T'}]$$

3. This should induce a functor on model up to "retying" by g, giving us a generalized recursion principle somehow

Simply-Typed Binding Signatures, but Formally

Simply-Typed Binding Signatures

Given a set of types T, a simply-typed binding signatures is family of object of $(T^* \times T)^* \times T$ specifying the types of the constructor:

$$t_1^{(\vec{u}_1)} \times \ldots \times t_n^{(\vec{u}_n)} \to \tau$$

Retyping

Given a translation of type system $g : T \to T'$, retyping by g is:

$$g(t_1)^{(g(\vec{u}_1))} \times \ldots \times g(t_n)^{(g(\vec{u}_n))} \to g(\tau)$$

Intuition, but Formally

The category of simply-typed binding signatures

We define the 1-category **STSig** of simply-typed binding signatures:

- Objects are pairs of a set T and a signature $(\alpha_i)_{i:I}$ over T
- Morphisms $(T, (\alpha_i)_{i:I}) \to (T', (\alpha'_i)_{i:I'})$ are pairs of a translation of the type system $T \xrightarrow{g} T'$, and a mapping of constructors $I \xrightarrow{h} I'$ compatible with retyping $\forall i : I, \alpha'_{h(i)} = g^* \alpha_i$

Computing Models

There is a 1-functor computing the categories of models of any simply-typed binding signature over the base category $[\mathrm{Set}^T, \mathrm{Set}^T]$, and its always has an initial object:

$$\mathbf{STSig}^{\mathrm{op}} \longrightarrow \mathbf{ModSig} \xrightarrow{\mathrm{Model}} \mathbf{Cat}$$

A Generalized Recursion Principle

Given $(T, \alpha) : \mathbf{STSig}$, we want a generalized recursion principle.

Next Step?

How do we get a generalized recursion principle out of:

$$\mathbf{STSig}^{\mathrm{op}} \longrightarrow \mathbf{ModSig} \xrightarrow{\mathrm{Model}} \mathbf{Cat}$$

A Generalized Recursion Principle

Given (T, α) : **STSig**, we want a generalized recursion principle:

Restricting to the Coslice of (T,α)

We can restrict the functor computing models to the coslice of (T, α); which objects are signatures (T', α') with a morphism $(T, \alpha) \to (T', \alpha')$:

$$((T, \alpha)/\mathbf{STSig})^{\mathrm{op}} \longrightarrow \mathbf{STSig}^{\mathrm{op}} \longrightarrow \mathbf{ModSig} \xrightarrow{\mathrm{Model}} \mathbf{Cat}$$

A Generalized Recursion Principle

Given (T, α) : **STSig**, we want a generalized recursion principle:

Restricting to the Coslice of (T,α)

$$((T, \alpha)/\mathbf{STSig})^{\mathrm{op}} \longrightarrow \mathbf{STSig}^{\mathrm{op}} \longrightarrow \mathbf{ModSig} \xrightarrow{\mathrm{Model}} \mathbf{Cat}$$

The Extended Category of Models

There is a category **ExtModel**$((T, \alpha))$ defined as:

- An object is a model M of simply-typed signature (T', α) equipped with a morphism $(T, \alpha) \xrightarrow{(g,h)} (T', \alpha)$
- A morphism consist of a morphism of models up to retying

A Generalized Recursion Principle

Given (T, α) : **STSig**, we want a generalized recursion principle:

Restricting to the Coslice of (T,α)

$$((T, \alpha)/\mathbf{STSig})^{\mathrm{op}} \longrightarrow \mathbf{STSig}^{\mathrm{op}} \longrightarrow \mathbf{ModSig} \xrightarrow{\mathrm{Model}} \mathbf{Cat}$$

The Extended Category of Models

There is a category **ExtModel**$((T, \alpha))$ defined as:

- An object is a model M of simply-typed signature (T', α) equipped with a morphism $(T, \alpha) \xrightarrow{(g,h)} (T', \alpha)$
- A morphism consist of a morphism of models up to retying

A Generalized Recursion Principle

The initial model of (T, α) is initial in **ExtModel**$((T, \alpha))$

Conlusion

Recap

1. Modules over monoids provides us an abstract framework
2. Signatures with strength provides us with an initiality theorem
3. Hss enables us to prove it modularly

Recap

1. Modules over monoids provides us an abstract framework
2. Signatures with strength provides us with an initiality theorem
3. Hss enables us to prove it modularly

2-functoriality enables to relate different instances nicely

Recap

1. Modules over monoids provides us an abstract framework
2. Signatures with strength provides us with an initiality theorem
3. Hss enables us to prove it modularly

2-functoriality enables to relate different instances nicely

Goals

With this work, we hope to:

1. Make this field more accessible to newcommers,
2. Set a basis to investigate and unify more complex notions of initial semantics, such as equations or reduction rules.