Implementing a category-theoretic framework for typed abstract syntax

Ralph Matthes
joint work with Benedikt Ahrens (TU Delft) and Anders Mörtberg (U Stockholm)

CNRS, Institut de Recherche en Informatique de Toulouse (IRIT),
University of Toulouse

Journées 2021 du GT Scalp, IUT Fontainebleau, France
(hybrid conference)
November 4, 2021

Abstract (1)

Recently, we described a category-theoretic construction of abstract syntax from a signature, mechanized in the UniMath library based on the Coq proof assistant, cf. “From Signatures to Monads in UniMath”. J. Autom. Reason. 63(2): 285-318 (2019).

In the present work, we describe what was necessary to generalize that work to account for simply-typed languages.

Abstract (2)

First, some definitions had to be generalized to account for the natural appearance of non-endofunctors in the simply-typed case. Second, an existing mechanized library on omega-cocontinuous functors had to be extended by constructions and theorems necessary for constructing multi-sorted syntax.

Third, and most excitingly, the theoretical framework for the semantical signatures had to be generalized from a monoidal to a bicategorical setting, again to account for non-endofunctors arising in the typed case. This uses actions of endofunctors on functors with given source, and the corresponding notion of strong functors between actions, all formalized in UniMath using the recently developed library of bicategory theory.

Abstract (3)

We explain what needed to be done to plug all of these ingredients together, modularly. The main result of our work is a general construction that, when fed with a signature for a simply-typed language, returns an implementation of that language together with suitable boilerplate code, in particular, a certified monadic substitution operation.

Outline

Introduction

Goal: represent and reason about languages with binders using category theory in type theory

Start with a simple notion of signature representing a language with binders and from this construct a monad for this language

Monads, as they fit for programming

```
-- A monad is a type family M with return and bind:
return : a → M a
(>>=) : M a → (a → M b) → M b


-- We can define Kleisli composition for any monad:
(>=>) : (a → M b) → (b → M c) → (a → M c)

-- The monad laws can be written as:
return a >>= σ = σ(a)
t >>= return = t
(t >>= σ1) >>= σ2 = t >>= (σ1 >=> σ2)
```

The monad of terms

```
-- Substitution is a monad:
var : a → Tm a
_[_] : Tm a → (a → Tm b) → Tm b


-- Kleisli composition is composition of substitutions:
_;_ : (a → Tm b) → (b → Tm c) → (a → Tm c)

-- Monad laws are rules for substitution:
(var a) [σ] = σ a
t [λx → var x] = t
(t [σ1]) [σ2] = t [σ1 ; σ2]
```

Nothing says that substitution should commute with term constructors.

This cannot even be expressed on the abstract level of monads.

Relation with binding

The term monad does not suppose any notion of binding, but the concept of term monad accommodates binding well: binding changes the parameter A that serves as "reservoir" for variable names. To represent a λ-abstraction $\lambda x.t$ with free variables taken from set A, add a generic new element to that set (thus heading for de Bruijn-style), yielding the set option$A$ and represent t relative to that extended reservoir.

Example: untyped lambda calculus as an endofunctor

```
Inductive LC (X : Type) :=
 | var : X -> LC X
 | app : LC X * LC X -> LC X
 | abs : LC (option X) -> LC X
```

This is plain Coq code and not allowed in UniMath that is lacking inductive types and families.

Relation with binding

The term monad does not suppose any notion of binding, but the concept of term monad accommodates binding well: binding changes the parameter A that serves as "reservoir" for variable names. To represent a λ-abstraction $\lambda x.t$ with free variables taken from set A, add a generic new element to that set (thus heading for de Bruijn-style), yielding the set option$A$ and represent t relative to that extended reservoir.

Example: untyped lambda calculus as an endofunctor

```
Inductive LC (X : Type) :=
 | var : X -> LC X
 | app : LC X * LC X -> LC X
 | abs : LC (option X) -> LC X
```

This is plain Coq code and not allowed in UniMath that is lacking inductive types and families.

Relative monads?

"Monads need not be endofunctors" (Altenkirch, Chapman, Uustalu), and there is the question why all types or all sets should be considered as valid parameters for the reservoir of free variables. Anyway, for the sake of λ-abstraction, only one extra variable is supposed, and we could do with the natural numbers.
This could be expressed by monads relative to the functor $J_f : \mathrm{Fin} \to \mathrm{Set}$. But if we already have a "normal" monad $T : \mathrm{Set} \to \mathrm{Set}$, then $T \cdot J_f$ is a monad relative to J_f.

If we are able to get monads and not just relative monads, we construct the monads.

Overall structure

Multi-sorted binding signature

(Small) set of descriptions of term constructors with a 1-type of sorts S

Signature with strength

$H : [\mathcal{C}^{\mathsf{S}}, \mathcal{C}^{\mathsf{S}}] \to [\mathcal{C}^{\mathsf{S}}, \mathcal{C}^{\mathsf{S}}]$ with strength θ for $\mathcal{C}^{\mathsf{S}} := [\mathsf{S}, \mathcal{C}]$

Heterogeneous substitution system

$(\underline{\mathrm{Id}} + H)$-algebra with structure

Monad on $\mathcal{C}^{\mathsf{S}}$

Typically, $\mathcal{C} =$ Set (the sets of univalent foundations, not those of `Coq`).

Formalized in UniMath: https://github.com/UniMath/UniMath

A simple notion of signature for variable binding

Binding signature:

- A set I ("constructors") and
- a function $\mathrm{ar} : I \to [\mathtt{nat}]$

Example: untyped lambda calculus (to the left in vanilla Coq)

```
Inductive LC (X : Type) :=
 | var : X -> LC X
 | app : LC X * LC X -> LC X
 | abs : LC (option X) -> LC X
```

$$I := \{\mathsf{app}, \mathsf{abs}\}$$
$$\mathrm{ar}(\mathsf{app}) = [0, 0]$$
$$\mathrm{ar}(\mathsf{abs}) = [1]$$

variables are special and do not enter the signature (anyway, substitution is not homomorphic on variables)

Simply-typed lambda-calculus

Assume two families of term constructors, both indexed over two sorts s, t, with "arities"

- $\left\langle \left[\langle [\,], s \Rightarrow t\rangle, \langle [\,], s\rangle\right] \;,\; t \right\rangle$—for application
- $\left\langle \left[\langle [s], t\rangle\right] \;,\; s \Rightarrow t \right\rangle$—for λ-abstraction

That is, for any sorts s, t, there is an application constructor that takes terms of sorts $s \Rightarrow t$ and s as arguments to yield a term of sort t. No binding is involved in the arguments, thus the use of the empty lists $[\,]$.

For any sorts s, t, there is a λ-abstraction constructor taking one term of sort t with an extra potentially bound variable of sort s, which yields a term of sort $s \Rightarrow t$.

multi-sorted binding signature

Definition

A **multi-sorted binding signature** is given by a (small) set I together with a map $\mathrm{ar} : I \to \mathrm{list}(\mathrm{list}(\mathsf{S}) \times \mathsf{S}) \times \mathsf{S}$.

STLC is obviously an instance as soon as we have a binary operation $\Rightarrow$ on S:

$$I = \mathsf{S} \times \mathsf{S} + \mathsf{S} \times \mathsf{S}$$
$$\mathrm{ar}(\mathrm{inl}\langle s, t\rangle) = \big\langle [\langle [], s \Rightarrow t\rangle, \langle [], s\rangle], t\big\rangle$$
$$\mathrm{ar}(\mathrm{inr}\langle s, t\rangle) = \big\langle [\langle [s], t\rangle], s \Rightarrow t\big\rangle$$

Easily extensible to PCF (with natural numbers and booleans and operations on them).

Another example towards type theory: the two sorts of types and elements in the presentation of the calculus of constructions in the style of Streicher.

multi-sorted binding signature

Definition

A **multi-sorted binding signature** is given by a (small) set I together with a map $\mathrm{ar} : I \to \mathrm{list}(\mathrm{list}(\mathsf{S}) \times \mathsf{S}) \times \mathsf{S}$.

STLC is obviously an instance as soon as we have a binary operation $\Rightarrow$ on S:

$$I = \mathsf{S} \times \mathsf{S} + \mathsf{S} \times \mathsf{S}$$
$$\mathrm{ar}(\mathrm{inl}\langle s, t\rangle) = \big\langle [\langle [], s \Rightarrow t\rangle, \langle [], s\rangle], t\big\rangle$$
$$\mathrm{ar}(\mathrm{inr}\langle s, t\rangle) = \big\langle [\langle [s], t\rangle], s \Rightarrow t\big\rangle$$

Easily extensible to PCF (with natural numbers and booleans and operations on them).

Another example towards type theory: the two sorts of types and elements in the presentation of the calculus of constructions in the style of Streicher.

A categorical notion of signature for variable binding

We have seen notions of binding signature that are syntactic descriptions of syntactic systems—they are not the syntax itself.

A signature with strength is a more general, and more semantic notion of signature. It consists of an endofunctor H on endofunctors (on a suitable base category) together with extra data θ that specifies information on "how to do substitution" on H-algebras, basically prescribing in what sense substitution is homomorphic on all the user-defined constructors.

The untyped lambda calculus as a signature with strength:

- $H(F) := F \times F + F \cdot \mathrm{option}$
- $\theta := \ldots$

This assumes very little of the base category.

A categorical notion of signature for variable binding

We have seen notions of binding signature that are syntactic descriptions of syntactic systems—they are not the syntax itself.

A signature with strength is a more general, and more semantic notion of signature. It consists of an endofunctor H on endofunctors (on a suitable base category) together with extra data θ that specifies information on "how to do substitution" on H-algebras, basically prescribing in what sense substitution is homomorphic on all the user-defined constructors.

The untyped lambda calculus as a signature with strength:

- $H(F) := F \times F + F \cdot \mathrm{option}$
- $\theta := \ldots$

This assumes very little of the base category.

How to deal with sorts

We instantiate the base category $\mathcal{C}$ to the functor category $[\mathsf{S}, \mathsf{Set}]$ for a given set S, seen as discrete category (and Set can be replaced by a well-behaved other base category).
Objects of this categories are simply functions $X : \mathsf{S} \to \mathsf{Set}$. These hence correspond to sets of sorted variables analogously to the `X` parameter of the inductive STLC example seen before (for each sort $s : \mathsf{S}$ there is a set $X\, s$ of variables of that sort that can freely occur in a term).

We can now reuse the previous implementation in `UniMath` of the framework of signatures with strength, but there is still work to do.

From multi-sorted binding signatures to signatures with strength: first conceptual challenge

We work with general $\mathcal{C}^{\mathsf{S}}$ instead of only $[\mathsf{S}, \mathsf{Set}]$.
Let (I, ar) be a multi-sorted binding signature and $i : I$.
We associate with $\mathsf{ar}(i)$ an endofunctor on the endofunctors on $\mathcal{C}^{\mathsf{S}}$. For this, we have to go back and forth between $\mathcal{C}$ and $\mathcal{C}^{\mathsf{S}}$ via an adjunction for each individual sort, and the addition of a variable with a given sort to the context is dealt with by a suitable adaptation of the option functor.

$$\mathsf{option}_s\ X\ t :\equiv X\ t + \coprod_{(s=t)} 1$$

for sorts s and t

The functor associated to the multi-sorted binding signature (I, ar) is then obtained as the coproduct of the functors associated to each arity.

Getting an initial algebra for the syntax described by H

The next step: construct an initial algebra for $\underline{\mathsf{Id}} + H$, the sum of H with the functor constantly the identity on the base category. The summand $\underline{\mathsf{Id}}$ encodes "variables as terms", whereas the functor H specifies the constructors given via the multi-sorted binding signature.

We have done this before in general, based on ω-cocontinuity, see

https://arxiv.org/abs/1612.00693

(appeared in the Journal of Automated Reasoning in 2019)

Substitution on the initial algebra: second conceptual challenge—but already solved more generally

The previous step gives, in particular, an endofunctor T on $\mathcal{C}^{\mathsf{S}}$, and a natural transformation return : Id $\to T$. We complement the pair (T, return) to a monad by constructing a suitable "substitution" operation.

Two smaller steps:

- construct a heterogeneous substitution system (M. & Uustalu 2004) from (T, return), employing "Generalized Mendler Iteration"
- apply an already defined map from substitution systems to monads on the substitution systems thus obtained

Notice that the main benefit of the notion of substitution systems is that they serve as intermediate goal in this approach.

Substitution on the initial algebra: second conceptual challenge—but already solved more generally

The previous step gives, in particular, an endofunctor T on $\mathcal{C}^{\mathsf{S}}$, and a natural transformation return : Id $\to T$. We complement the pair (T, return) to a monad by constructing a suitable "substitution" operation.

Two smaller steps:

- construct a heterogeneous substitution system (M. & Uustalu 2004) from (T, return), employing "Generalized Mendler Iteration"
- apply an already defined map from substitution systems to monads on the substitution systems thus obtained

Notice that the main benefit of the notion of substitution systems is that they serve as intermediate goal in this approach.

The missing piece of work

Most of the technical work in the formalization went into showing that the functor H that we construct for a given multi-sorted binding signature

- comes with a strength θ and
- is ω-cocontinuous.

This is made more difficult by the fact that the constructions are not homogeneous: in the construction appear endofunctors on $\mathcal{C}^{\mathsf{S}}$, but also functors from $\mathcal{C}^{\mathsf{S}}$ to $\mathcal{C}$.

On the side of strengths, this required a wider notion of strength (see the next section on its understanding), and concerning ω-cocontinuity, this asked for the identification of adjunctions, and also for the construction in `UniMath` of exponentials in $[\mathcal{D}, \mathsf{Set}]$, for any category $\mathcal{D}$.

What is strength for a functor that is not an endofunctor?

Definition (prestrength)

For categories $\mathcal{C}$, $\mathcal{D}$, $\mathcal{D}'$ and a functor $H : [\mathcal{C}, \mathcal{D}'] \to [\mathcal{C}, \mathcal{D}]$, a **prestrength** for H is a natural transformation

$$\theta : (H-) \cdot U\sim \to H(- \cdot U\sim)$$

between bifunctors $[\mathcal{C}, \mathcal{D}'] \times \mathrm{Ptd}(\mathcal{C}) \to [\mathcal{C}, \mathcal{D}]$.

$\mathrm{Ptd}(\mathcal{C})$ is the category of pointed endofunctors on $\mathcal{C}$: its objects are functors $F : \mathcal{C} \to \mathcal{C}$ together with a natural transformation from Id to F. And U is the forgetful functor from $\mathrm{Ptd}(\mathcal{C})$ to $[\mathcal{C}, \mathcal{C}]$ that forgets the "point".

In our construction, we do not always have that $\mathcal{C}$ and $\mathcal{D}$ are the same category.

Definition (strength)

Given a prestrength θ for a functor H, θ is called a **strength** for H if it is "homomorphic" in the second argument $\sim$, in the following sense. The source and target bifunctors applied to a pair of objects $(X, (Z, e))$ with $X : [\mathcal{C}, \mathcal{D}']$ and $(Z, e) : \mathrm{Ptd}(\mathcal{C})$ (X for the argument symbolized by $-$ and (Z, e) for the argument symbolized by $\sim$) yield $HX \cdot Z$ and $H(X \cdot Z)$, thus $\theta_{X,(Z,e)} : HX \cdot Z \to H(X \cdot Z)$ in $[\mathcal{C}, \mathcal{D}]$. Being "homomorphic" of θ in the second argument means satisfying the equations $\theta_{X,\mathrm{id}} = \mathrm{id}_{HX}$ and

$$\theta_{X,(Z' \cdot Z, e' \cdot e)} = H(\alpha^{-1}_{X,Z',Z}) \circ \theta_{X \cdot Z',(Z,e)} \circ (\theta_{X,(Z',e')} \cdot Z) \ ,$$

using the inverse of the canonical isomorphism $\alpha_{X,Y,Z} : X \cdot (Y \cdot Z) \simeq (X \cdot Y) \cdot Z$ for composable functors.

We wanted to understand better the general principle behind this definition and in what sense this is a notion of "homomorphism".

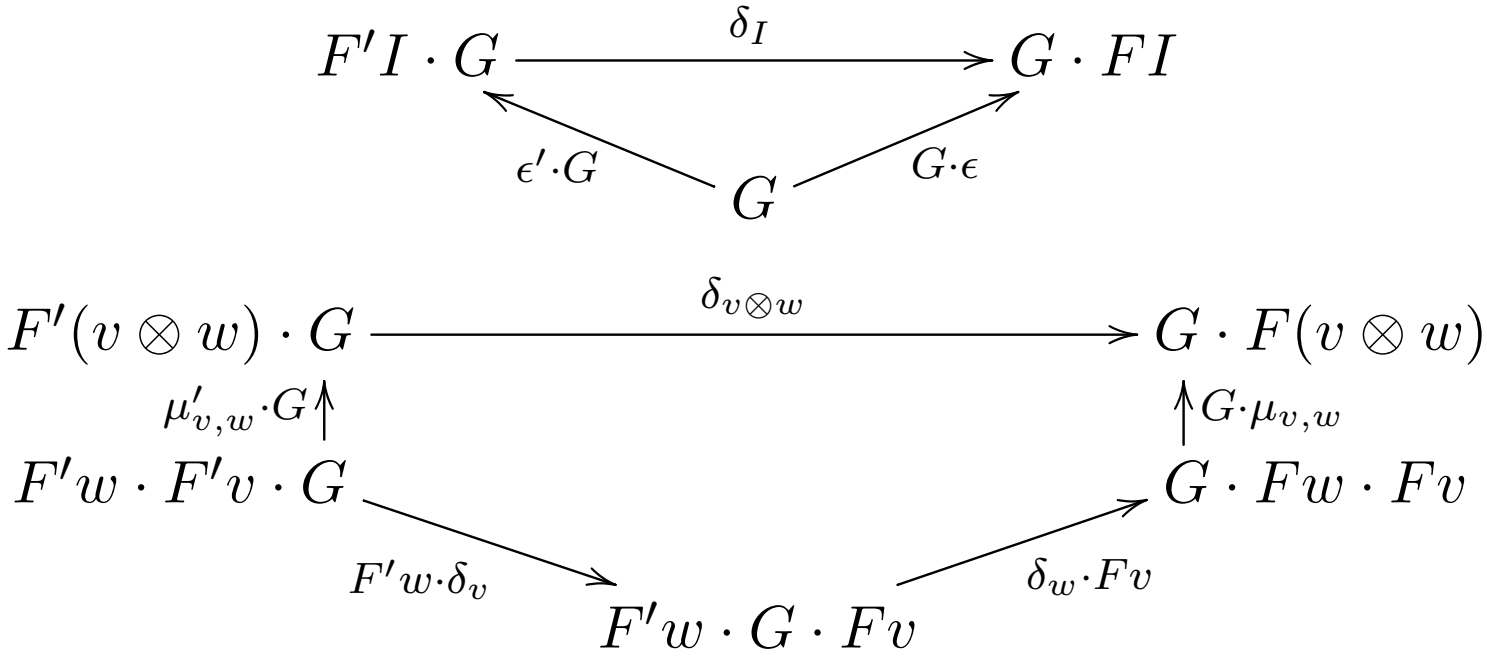
action-based strength

Consider a monoidal category $\mathcal{V}$ with tensor product $\otimes$, categories $\mathcal{A}$ and $\mathcal{A}'$ and actions of $\mathcal{V}$ on both, expressed as strong monoidal functors $F : \mathcal{V} \to [\mathcal{A}, \mathcal{A}]$ and $F' : \mathcal{V} \to [\mathcal{A}', \mathcal{A}']$ (where the monoidal structure on the endofunctor categories is given by composition in diagrammatic order).

A more elementary description of actions was suggested by Pareigis in the 70ies (which is essentially equivalent to the present one). The present one has been known for about 20 years.

Let $G : \mathcal{A} \to \mathcal{A}'$ be a functor and let δ be a natural transformation between functors from $\mathcal{V}$ to $[\mathcal{A}, \mathcal{A}']$ that are defined on objects v of $\mathcal{V}$ as $F'v \cdot G$ and $G \cdot Fv$, respectively. The transformation δ is a **parameterized distributivity**—or a strength—for G if the following diagrams commute.

action-based strength laws



The authors are unaware of occurrences of this notion in the literature. However, there is the essentially equivalent notion of Pareigis.

Do these laws express "homomorphism"? Is strength for signatures an instance of this notion?

Is strength for signatures an instance of this notion?

Yes, it is.

The problem to overcome is that the parameter categories $\mathcal{C}$, $\mathcal{D}$, $\mathcal{D}'$ of strength for signatures need not coincide. The way out is the rather straightforward observation that the endomorphisms of a bicategory act on any hom-category which agrees with the endos in the source type. And this is then used for the bicategory of (small) categories, functors and natural transformations.

Why bicategories and not 2-categories? The latter can be viewed as a mathematical idealization of the former, and bicategories are the more "categorical" concept. The not necessarily strict categories only form a bicategory, and this notion has been implemented in `UniMath` together with a rich library of tools for working with bicategories—where computer help is very much appreciated given the many axioms.

Hence, we were able to prove in `UniMath` that strength for signatures is an action-based strength.

Do these laws express "homomorphism"?

This question is mostly settled, but the formalization is not complete. The formalization efforts showed the need for some refactoring in the library on category theory in `UniMath`, mainly for a better interplay between monoidal cagories, displayed categories (probably currently a unique feature of `UniMath`) and bicategories.

The main idea is to characterize parameterized distributivity by having a strong monoidal functor into a suitable category (obtained through a displayed category in `UniMath`). This extends the correspondence between natural transformations between functors into a functor category with functors into a suitably crafted category.

Such a strong monoidal functor is the ultimate embodiment of having "homomorphic" laws.

Summary

We have formalized:

- Construction of a (term) monad from a multi-sorted binding signature
- Examples: simply-typed lambda calculus, PCF and CoC in the style of Streicher
- method: work in "towers" of functor categories to take into account typed contexts

We have used function extensionality and univalence for propositions which both had to be added as axioms to Coq. We also rely on set quotients in `UniMath` and thereby on the resizing rule (for propositional truncation).

Future goals

- Signatures dealing with type substitution
- Show that the datatype together with the constructed substitution operation is initial in a category of "algebras with substitution"
- Finish our analysis of strength notions that makes precise in which sense strength means being homomorphic (this question goes far beyond our intended application of representation of typed syntax with binding)—we suspect the appropriate abstraction level for this result to be bicategories (so as to allow for an elegant formalization)

Thank you for your attention!

Future goals

- Signatures dealing with type substitution
- Show that the datatype together with the constructed substitution operation is initial in a category of "algebras with substitution"
- Finish our analysis of strength notions that makes precise in which sense strength means being homomorphic (this question goes far beyond our intended application of representation of typed syntax with binding)—we suspect the appropriate abstraction level for this result to be bicategories (so as to allow for an elegant formalization)

Thank you for your attention!