From natural numbers to the lambda calculus

Benedikt Ahrens

joint work with Ralph Matthes and Anders Mörtberg

Outline

1. About UniMath
2. Signatures and associated syntax

Outline

UniMath as a language

“UniMath“ stands for “Univalent Mathematics“
Goal: have a “core” **language of dependent types**

- rich enough to formalize mathematics
- simple enough to allow for a proof of (equi-)consistency

In practice, the UniMath language is a fragment of the Calculus of Inductive Constructions implemented in the Coq proof assistant.

Overview: types in UniMath

Type former	Notation	(special case)
Inhabitant	$a : A$	
Dependent type	$x : A \ \vdash \ B(x)$	
Sigma type	$\sum_{(x:A)} B(x)$	$A \times B$
Product type	$\prod_{(x:A)} B(x)$	$A \to B$
Coproduct type	$A + B$	
Identity type	$\mathsf{Id}_A(a, b)$, $a = b$	
Universe	U	
Nat, Bool, $\mathbf{1}$, $\mathbf{0}$		

- axioms: function extensionality, univalence
- universes: $\mathsf{U} : \mathsf{U}$ (inconsistency, as a way to implement resizing)

Overview: what UniMath does not have

- record types
- general inductive types
- general HITs

In this talk

we discuss the construction of some inductive types from the other type formers

UniMath as a library

General purpose libraries:

- Foundations
- Number systems
- Algebra
- Category theory

Specialized libraries:

- Categories in FOLDS style
- Substitution systems (discussed later)

Outline

What are inductive types?

Inductive types are types of tree-like data.
Usage:

- "containers" for data, e.g., lists of elements of a fixed type

```
Inductive list (X : Type) :=
 | nil : list X
 | cons : X -> list X -> list X.
```

- representations of mathematically interesting objects, e.g., $\mathbb{N}$ and lambda terms

```
Inductive LC (X : Type) :=
 | Var : X -> LC X
 | App : LC X * LC X -> LC X
 | Abs : LC (option X) -> LC X
```

Formal definition of inductive types

What are inductive types?

Two characterizations:

- **external** via inference rules
- **internal** via universal property—as initial algebra

Related work: AGS'12 compare universal property definition with internal version of external characterization

Our goal

We are interested in internally characterized inductive types, and their construction in the UniMath language.

Why the need for a systematic construction?

Some inductive types are easily constructed, e.g., lists over a given base type:

- $\mathsf{Vect}(A, n) := A^n$
- $\mathsf{List}(A) := \sum_{(n:\mathbb{N})} \mathsf{Vect}(A, n)$

But: it is **not always that easy**.

Exercise

The terms of the lambda calculus can be defined as a nested inductive data type, say, using Coq's `Inductive`:

```
Inductive LC (X : Type) :=
 | Var : X -> LC X
 | App : LC X * LC X -> LC X
 | Abs : LC (option X) -> LC X
```

Define an equivalent type using just the UniMath language.

Signatures and inductive types

- In order to say what an inductive type is, we need to specify a notion of **signature**.
- A signature specifies the shape of the "trees" by specifying
  - type of nodes
  - "number" of subtrees of a node

Various notions of "signature" in the literature, for different classes of inductive types:

- polynomial functors
- containers — for W-types
- "strengthened" signatures (rank 2) — for type families

Two notions of signature for **variable binding**

Binding signature

- a type A of "constructors" (nodes)
- decidable equality $\prod_{(x,y:A)}(x = y) + \neg(x = y)$
- a map $\mathsf{arity} : A \to \mathsf{List}(\mathsf{Nat})$

Signature à la Matthes & Uustalu

- a functor $H : [\mathcal{C}, \mathcal{C}] \to [\mathcal{C}, \mathcal{C}]$
- a natural transformation between bifunctors

$$\theta : (H-) \cdot U(\sim) \quad \longrightarrow \quad H(- \cdot U(\sim))$$

satisfying some axioms

  - θ explains how to do substitution
  - U is forgetful functor from pointed endofunctors to endofunctors

The lambda calculus

```
Inductive LC (X : Type) :=
 | Var : X -> LC X
 | App : LC X * LC X -> LC X
 | Abs : LC (option X) -> LC X
```

Binding signature of the lambda calculus:

- $A := \{\mathsf{app}, \mathsf{abs}\}$
- $\mathsf{app} \mapsto [0, 0] \quad \mathsf{abs} \mapsto [1]$

Signature à la M&U of the lambda calculus:

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

Remark:

- The constructor var is not mentioned explicitly in signatures, but is later dealt with in the definition of "models" of such signatures.

Achieved goals

We construct

1. a signature à la M&U from a binding signature
2. the data type (functor on Set) specified by a binding signature
3. a model (“substitution system”) of a signature à la M&U on the data type constructed in item (2)
   - equips the datatype with a “substitution“ operation
4. a monad from any substitution system
   - shows that the substitution constructed in (3) satisfies monadic laws

- Items 3 and 4 were done on paper in [MU’04].
- Item 2 is a well-known category-theoretic construction.

Construction of initial algebras in UniMath

Initial algebra of $F : \mathcal{C} \to \mathcal{C}$ (Adámek)

If F is ω-cocontinuous, then the colimit of

$$0 \to F0 \to F^2 0 \to \dots$$

is an initial F-algebra.

- Can construct colimits from coproducts and coequalizers
- in plain type theory we have coproducts
- in univalent type theory, additionally have set quotients a.k.a. coequalizers in Set

Restriction

This approach only allows construction of inductive **sets**.

Construction of a substitution operation

via **Generalized Mendler Iteration**

- asserts the unique existence of a morphism making some diagram commute—the *iterator*
- comes with a suitable **fusion law**: when is a composition of an iterator with a function again an iterator
- see work by Mendler, Bird & Paterson

Example: binding signature of MLTT

```
Definition PiSig : GenSig :=
  mkGenSig (isdeceqstn 3) (three_rec [0,1] [1] [0,0]).
Definition SigmaSig : GenSig :=
  mkGenSig (isdeceqstn 3) (three_rec [0,1] [0,0] [0,2]).
Definition SumSig : GenSig :=
  mkGenSig (isdeceqstn 4) (four_rec [0,0] [0] [0] [0,1,1]).
Definition IdSig : GenSig :=
  mkGenSig (isdeceqstn 3) (three_rec [0,0,0] [] [0,0]).
(* Define the arity of the eliminators for Fin by recursion *)
Definition FinSigElim (n : nat) : list nat.
Proof.
induction n as [|n ih].
- apply [0].
- apply (0 :: ih).
Defined.
(* Define the signature of the constructors for Fin by recursion *)
Definition FinSigConstructors (n : nat) : stn n -> list nat := fun _ => [].
(* Uncurried version of the FinSig family *)
Definition FinSigFun : (Sigma n : nat, unit + (stn n + unit)) -> list nat.
Proof.
induction 1 as [n p].
induction p as [_|p].
- apply [].
- induction p as [p|].
  + apply (FinSigConstructors _ p).
  + apply (FinSigElim n).
Defined.
Lemma isdeceqFinSig : isdeceq (Sigma n, unit + (stn n + unit)).
Proof.
intros [n p] [m q].
induction (isdeceqnat n m) as [h|h].
- induction h.
   + destruct (isdeceqcoprod isdecequnit
                 (isdeceqcoprod (isdeceqstn n) isdecequnit) p q) as [Hpq|Hpq]
```

Future goals

Recall:

- Constructions 1-4 yield a monad from a binding signature
- The datatype constructed from a binding signature satisfies a universal property (initial algebra)

Goal

Show that the datatype together with the constructed substitution operation is initial in a category of "algebras with substitution".

This is a rephrasing of a result by Hirschowitz & Maggesi, who prove a similar result, but rely on Coq's inductive types for constructing the datatype.

Future goals

Recall:

- Constructions 1-4 yield a monad from a binding signature
- The datatype constructed from a binding signature satisfies a universal property (initial algebra)

Goal

Show that the datatype together with the constructed substitution operation is initial in a category of "algebras with substitution".

This is a rephrasing of a result by Hirschowitz & Maggesi, who prove a similar result, but rely on Coq's inductive types for constructing the datatype.

The end — thanks for your attention!

References

- Adámek: *Free algebras and automata realizations in the language of categories*, CMUC, 1974
- Altenkirch, Reus: *Monadic presentations of lambda terms using generalized inductive types*, CSL'99
- Awodey, Gambino, Sojakova: *Inductive types in homotopy type theory*, LiCS'12
- Bird, Paterson: *Generalised folds for nested datatypes*, JFP, 1999
- Hirschowitz, Maggesi: *Initial Semantics for Strengthened Signatures*, FiCS'12
- Hirschowitz, Maggesi: *Modules over Monads and Linearity*, WoLLIC'07
- Matthes, Uustalu: *Substitution in non-wellfounded syntax with variable binding*, TCS, 2004