A formalization of the initiality conjecture in Agda

Guillaume Brunerie
j.w.w. Menno de Boer, Peter Lumsdaine, and Anders Mörtberg

HoTT 2019, CMU, Pittsburgh
August 16, 2019

Initiality

Initiality conjecture

Given a type theory $\mathbb{T}$, the term model $\mathsf{Syn}_{\mathbb{T}}$ (or syntactic category) is initial in the category of models of $\mathbb{T}$.

It shows that there is a canonical way to interpret type theory into a model with the appropriate structure.

Questions:

- What is a type theory?
- What is the category of models of a given type theory?
- What is the term model of a given type theory?

Initiality

Initiality conjecture

Given a type theory $\mathbb{T}$, the term model $\mathrm{Syn}_{\mathbb{T}}$ (or syntactic category) is initial in the category of models of $\mathbb{T}$.

It shows that there is a canonical way to interpret type theory into a model with the appropriate structure.

Questions:

- What is a type theory?
- What is the category of models of a given type theory?
- What is the term model of a given type theory?

Initiality

Initiality conjecture

Given a type theory $\mathbb{T}$, the term model $\mathsf{Syn}_{\mathbb{T}}$ (or syntactic category) is initial in the category of models of $\mathbb{T}$.

It shows that there is a canonical way to interpret type theory into a model with the appropriate structure.

Questions:

- What is a type theory?
- What is the category of models of a given type theory?
- What is the term model of a given type theory?

Background / related work

- Streicher proved initiality for a rather simple dependent type theory (1991).
- The extension to more complicated type theories has never been checked in detail.
- Voevodsky noticed this gap and stressed that this is something very important to prove. His (unfinished) series of papers on C-systems is going in this direction.
- The Initiality Project started by Mike Shulman aims to get a human-readable proof of initiality for a concrete type theory.
- Some work is also being done to define a general notion of type theories (Bauer–Haselwarter–Lumsdaine, Brunerie)

This talk

Goal:

- Take the type theory to be MLTT
- Give a proof of initiality for it, formalized in a proof assistant

Peter Lumsdaine suggested this project to the four of us in October 2018, but differences in opinions led to two parallel formalization projects:

- Menno de Boer and myself (in Agda, no HoTT, self contained, based on contextual categories)
- Peter Lumsdaine and Anders Mörtberg (in Coq/Unimath, based on categories with attributes)

This talk is about the Agda formalization.[1]

[1] https://github.com/guillaumebrunerie/initiality

This talk

Goal:

- Take the type theory to be MLTT
- Give a proof of initiality for it, formalized in a proof assistant

Peter Lumsdaine suggested this project to the four of us in October 2018, but differences in opinions led to two parallel formalization projects:

- Menno de Boer and myself (in Agda, no HoTT, self contained, based on contextual categories)
- Peter Lumsdaine and Anders Mörtberg (in Coq/Unimath, based on categories with attributes)

This talk is about the Agda formalization.[1]

[1] https://github.com/guillaumebrunerie/initiality

Results

The type theory we want to prove initiality for has

- Π-types
- Σ-types
- Natural numbers
- Identity types
- Infinite hierarchy of Tarski universes stable under the previous operations

We have formalized everything, except J which is only half-way formalized…[1]

[1] https://github.com/guillaumebrunerie/initiality

Meta-theory

The meta-theory used is the basic type theory of Agda 2.6.0.1 together with

- Prop (definitionally proof-irrelevant propositions[1], like `SProp` in Coq)
- function extensionality,
- propositional extensionality,
- quotients that compute.

[1] *Definitional Proof-Irrelevance without K*, G. Gilbert, J. Cockx, M. Sozeau, N. Tabareau

Prop

- If A : Prop and $u, v : A$, then u and v are definitionally equal
- Inductive families can be "squashed" to Prop and we can then only eliminate out of them to another Prop.

We use Prop everywhere where it makes sense:

- Our identity type is Prop-valued.
- Derivability of pre-judgments is an inductive family in Prop.
- An equivalence relation on a type A is $\sim : A \to A \to$ Prop which is reflexive, symmetric and transitive.

Note that we cannot define transport/subst, but we essentially never need it in this formalization.

[1] *Definitional Proof-Irrelevance without K*, G. Gilbert, J. Cockx, M. Sozeau, N. Tabareau

Contextual categories[1]

Definition

A contextual category is a category with a grading on the objects $\ell : \mathrm{Ob} \to \mathbb{N}$ and some additional structure.
For instance for each $A : \mathrm{Ob}_{n+1}$, we have $\mathrm{ft}(A) : \mathrm{Ob}_n$.

Idea:

- Objects represents contexts (of the given length)
- Morphisms represent context morphisms/total substitutions
- A type $\Gamma \vdash A$ is represented as the context (Γ, A)
- A term $\Gamma \vdash u : A$ is represented as the morphism $\Gamma \vdash (\mathrm{id}_\Gamma, u) : (\Gamma, A)$

We represent contextual categories as the models of an essentially algebraic theory with sorts Ob_n and $\mathrm{Mor}_{n,m}$ (for all $n, m \in \mathbb{N}$) and all the operations and equations needed.

[1] `contextualcat.agda#CCat`

Structured contextual categories[1]: type formers

For every type former we add one new operation and one new equation. For instance for Π-types we add

$$\begin{aligned} \mathsf{PiStr} &: (B : \mathsf{Ob}_{n+2}) \to \mathsf{Ob}_{n+1} \\ \mathsf{PiStr}_{\mathsf{ft}} &: (B : \mathsf{Ob}_{n+2}) \to \mathsf{ft}(\mathsf{PiStr}(B)) = \mathsf{ft}(\mathsf{ft}(B)) \end{aligned}$$

or more uniformly:

$$\begin{aligned} \mathsf{PiStr} &: (\Gamma : \mathsf{Ob}_n)(A : \mathsf{Ob}_{n+1})(A_{\mathsf{ft}} : \mathsf{ft}(A) = \Gamma) \\ &\qquad (B : \mathsf{Ob}_{n+2})(B_{\mathsf{ft}} : \mathsf{ft}(B) = A) \to \mathsf{Ob}_{n+1} \\ \mathsf{PiStr}_{\mathsf{ft}} &: (\Gamma\ A\ A_{\mathsf{ft}}\ B\ B_{\mathsf{ft}} : [\cdots]) \to \mathsf{ft}(\mathsf{PiStr}(\Gamma, A, A_{\mathsf{ft}}, B, B_{\mathsf{ft}})) = \Gamma \end{aligned}$$

[1] `contextualcat.agda#StructuredCCat`

Structured contextual categories[1]: term formers

For every term former we add one new operation and two new equations. For instance for the successor $\mathsf{suc} : \mathbb{N} \to \mathbb{N}$, we add

$$\begin{aligned}\mathsf{sucStr} &: (\Gamma : \mathsf{Ob}_n)\ (u : \mathsf{Mor}_{n,n+1})\ (u_s : \text{is-section}(u))\\ &\qquad (u_1 : \partial_1(u) = \mathsf{NatStr}(\Gamma)) \to \mathsf{Mor}_{n,n+1}\\ \mathsf{sucStr}_s &: (\Gamma\ u\ u_s\ u_1 : [\cdots]) \to \text{is-section}(\mathsf{sucStr}(\Gamma, u, u_s, u_1))\\ \mathsf{sucStr}_1 &: (\Gamma\ u\ u_s\ u_1 : [\cdots]) \to \partial_1(\mathsf{sucStr}(\Gamma, u, u_s, u_1)) = \mathsf{NatStr}(\Gamma)\end{aligned}$$

where is-section(u) is the equality

$$\mathsf{comp}(\mathsf{pp}(\partial_1(u)), u) = \mathsf{id}(\partial_0(u)).$$

[1] `contextualcat.agda#StructuredCCat`

Structured contextual categories[1]: naturality and equalities

For every type/term former, we need one additional equation (naturality). For instance:

$$\begin{aligned}\mathrm{PiStrNat} : (g : \mathrm{Mor}_{m,n})(B : \mathrm{Ob}_{n+2})(p : \mathrm{ft}(\mathrm{ft}(B)) = \partial_1(g))\\ \to \mathrm{star}(g, \mathrm{PiStr}(B), \_) = \mathrm{PiStr}(\mathrm{star}^{+}(g, B, \_))\\ \mathrm{sucStrNat} : (g : \mathrm{Mor}_{m,n})(u\ \ u_s\ \ u_1\ \ : [\dots])(p : \partial_0(u) = \partial_1(g))\\ \to \mathrm{starTm}(g, \mathrm{sucStr}(u, u_s, u_1), \_) = \mathrm{sucStr}(\mathrm{starTm}(g, u, \_), \_, \_)\end{aligned}$$

(the operations star^{+} and starTm are derived from the structure of contextual category)

Finally for equations (e.g. β/η-equality), we add the appropriate equalities, replacing uses of substitution by star/starTm.

[1] `contextualcat.agda#StructuredCCat`

Syntax[1] and typing rules[2]

- Two syntactic classes of pre-types and pre-terms
- Variables are de Bruijn indices
- Syntax is well-scoped (e.g. `TmExpr n` is the type of pre-terms with n variables) and fully annotated
- We use Agda's reflection mechanism to prove most of the syntactic lemmas
- We do not assume the substitution rules, but we prove that they are admissible (and many other admissible rules)

[1] `typetheory.agda` and `syntax.agda`
[2] `rules.agda`

Quotients[1]

We postulate quotients as higher inductive types.

Given a type A and a Prop-valued equivalence relation $\sim$ on A, the quotient $A/\sim$ has two constructors

- $\mathrm{proj} : A \to A/\sim$
- $\mathrm{eq} : (a\ b : A)(r : a \sim b) \to \mathrm{proj}(a) = \mathrm{proj}(b)$

together with the corresponding dependent elimination rule, and the (definitional) reduction rule for proj (using rewriting rules).

[1] `quotients.agda`

Effectiveness of quotients[1]

Lemma

Given $a, b : A$, *if* $\mathrm{proj}(a) = \mathrm{proj}(b)$, *then there exists* $r : a \sim b$.

Proof (encode-decode).

Given $a : A$, we define $P : A/\!\sim\, \to \mathrm{Prop}$ by

$$P(\mathrm{proj}(b)) = a \sim b$$
$$\mathrm{ap}_P(\mathrm{eq}(r)) = [\ldots] : (a \sim b) = (a \sim c) \quad (\text{where } r : b \sim c)$$

(requires propositional extensionality)

Now we prove that given $p : \mathrm{proj}(a) = x$, then $P(x)$ holds (by induction on p).

Finally, we can apply it to $x = \mathrm{proj}(b)$. □

[1] `quotients.agda#reflect`

The term model[1]

- Ob_n is the quotient of the set of derivable contexts of length n by the equivalence relation $\Gamma \sim \Delta \iff \vdash \Gamma = \Delta$.
- $\mathrm{Mor}_{n,m}$ is the quotient of the set of derivable $\Gamma \vdash \delta : \Delta$ where $|\Gamma| = n$ and $|\Delta| = m$, by the appropriate equivalence relation.
- contextual category structure: use the corresponding syntaxic operations

$$\mathrm{comp}(\theta, \delta) = \theta[\delta] \qquad \mathrm{id}(\Gamma) = \mathrm{id}_\Gamma \qquad \mathrm{ft}((\Gamma, A)) = \Gamma$$

$$\mathrm{pp}((\Gamma, A)) = \mathrm{id}_\Gamma \qquad \mathrm{star}(\delta, (\Delta, B)) = (\Gamma, B[\delta]) \qquad \mathrm{pt} = \varnothing$$

$$\mathrm{qq}(\delta, (\Delta, B)) = (\delta, x_n) \qquad \mathrm{ss}((\delta, u)) = (\mathrm{id}_\Gamma, u) \qquad \text{pt-mor} = ()$$

and check that they are invariant w.r.t. definitional equality.
- operations for type/term formers: use the type/term former

$$\mathrm{PiStr}((\Gamma, A, B)) = (\Gamma, \Pi_A B) \qquad \mathrm{sucStr}((\mathrm{id}, u)) = (\mathrm{id}, \mathrm{suc}(u))$$

[1] `termmodel.agda`

Partial interpretation[1]

A partial function $X \rightharpoonup Y$ is defined as a map $X \to \text{Partial}(Y)$ where

$$\text{Partial}(Y) := \Sigma_{P:\text{Prop}}(P \to Y)$$

For a pre-type A, a pre-term u and an object $X : \text{Ob}_n$, we have

$$[\![A]\!]_X : \text{Partial}(\text{Ob}_{n+1})$$

$$[\![u]\!]_X : \text{Partial}(\text{Mor}_{n,n+1})$$

For variables we use the structure of contextual categories, and for type/term formers we recursively interpret the arguments and then use the appropriate function on structured contextual categories.

[1] `partialinterpretation.agda`

Partial interpretation[1]

A partial function $X \rightharpoonup Y$ is defined as a map $X \to \mathrm{Partial}(Y)$ where

$$\mathrm{Partial}(Y) := \Sigma_{P:\mathrm{Prop}}(P \to Y)$$

For a pre-type A, a pre-term u and an object $X : \mathrm{Ob}_n$, we have

$$[\![A]\!]_X : \mathrm{Partial}(\mathrm{Ob}_{n+1})$$

$$[\![u]\!]_X : \mathrm{Partial}(\mathrm{Mor}_{n,n+1})$$

For variables we use the structure of contextual categories, and for type/term formers we recursively interpret the arguments and then use the appropriate function on structured contextual categories.

[1] `partialinterpretation.agda`

Example[1]

```
⟦_⟧Ty : TyExpr n → Ob n → Partial (Ob (suc n))
⟦_⟧Tm : TmExpr n → Ob n → Partial (Mor n (suc n))

⟦ pi A B ⟧Ty Γ = do
  [A]    ← ⟦ A ⟧Ty Γ
  [A]ft  ← assume (ft [A] ≡ Γ)
  [B]    ← ⟦ B ⟧Ty [A]
  [B]ft  ← assume (ft [B] ≡ [A])
  return (PiStr Γ [A] [A]ft [B] [B]ft)

⟦ suc u ⟧Tm Γ = do
  [u]   ← ⟦ u ⟧Tm Γ
  [u]s  ← assume (is-section [u])
  [u]1  ← assume (∂1 [u] ≡ NatStr Γ)
  return (sucStr [u] [u]s [u]1)
```

[1] partialinterpretation.agda

Totality[1]

In what follows we assume that $[\![\Gamma]\!]$ is defined, and $X := [\![\Gamma]\!]$.

Theorem

If $\Gamma \vdash A$ is derivable, then $[\![A]\!]_X$ is defined.

If $\Gamma \vdash u : A$ is derivable, then $[\![u]\!]_X$ is defined and $\partial_1([\![u]\!]_X) = [\![A]\!]_X$.

If $\Gamma \vdash A = A'$ is derivable, then $[\![A]\!]_X = [\![A']\!]_X$ (if both are defined).

If $\Gamma \vdash u = u' : A$ is derivable, then $[\![u]\!]_X = [\![u']\!]_X$ (if both are defined).

[1] `totality.agda`

Interpretation of substitutions[1]

Theorem

If $\Delta \vdash A$ *and* $\Gamma \vdash \delta : \Delta$*, then* $[\![A[\delta]]\!]_Y$ *is defined and moreover*

$$[\![A[\delta]]\!]_Y = \mathrm{star}([\![\delta]\!]_{X,Y}, [\![A]\!]_X, \_)$$

If $\Delta \vdash u : A$ *and* $\Gamma \vdash \delta : \Delta$*, then* $[\![u[\delta]]\!]_Y$ *is defined and moreover*

$$[\![u[\delta]]\!]_Y = \mathrm{starTm}([\![\delta]\!]_{X,Y}, [\![u]\!]_X, \_)$$

[1] `totality.agda`

Initiality (existence)[1]

Given an arbitrary structured contextual category $\mathcal{C}$, we want to construct a morphism from the syntactic category to $\mathcal{C}$.

- $\mathrm{Ob}_n \to \mathrm{Ob}_n^{\mathcal{C}}$: use the partial interpretation of contexts, the fact that it is actually total, and that it respects definitional equalities,
- $\mathrm{Mor}_{n,m} \to \mathrm{Mor}_{n,m}^{\mathcal{C}}$: same for context morphisms,
- contextual category structure: use the appropriate lemmas, e.g. the substitution lemma, $[\![\mathrm{id}_\Gamma]\!]_{X,X} = \mathrm{id}_X$, and so on,
- additional operations corresponding to type/term formers: use the fact that the partial interpretation function is appropriately defined.

[1] `initiality.agda#existence`

Initiality (uniqueness)[1]

Given two morphisms f, g from the syntactic category to $\mathcal{C}$, we want to prove that they are equal.

- (on objects)

$$\begin{aligned} f((\Gamma, \Pi_A B)) &= f(\mathrm{PiStr}((\Gamma, A, B))) \\ &= \mathrm{PiStr}(f((\Gamma, A, B))) \\ &= \mathrm{PiStr}(g((\Gamma, A, B))) \\ &= g(\mathrm{PiStr}((\Gamma, A, B))) \\ &= g((\Gamma, \Pi_A B)) \end{aligned}$$

Not by induction on the length, but on the number of symbols of the context (more or less. . .).

[1] `initiality.agda#uniqueness`

Initiality (uniqueness)[1]

- (on morphisms)

$$
\begin{aligned}
f((\delta, u)) &= f((\delta, x_n) \circ (\mathrm{id}, u)) \\
&= f(\mathrm{qq}(\delta) \circ (\mathrm{id}, u)) \\
&= \mathrm{qq}(f(\delta)) \circ f((\mathrm{id}, u)) \\
&= \mathrm{qq}(g(\delta)) \circ g((\mathrm{id}, u)) \\
&= g((\delta, u))
\end{aligned}
$$

For $f((\mathrm{id}, u)) = g((\mathrm{id}, u))$: by induction on u, similarly to uniqueness on objects

[1] `initiality.agda#uniqueness`

Conclusion

- We have a formalized proof of initiality for $\Pi, \Sigma, \mathbb{N}$, universes, and hopefully soon for Id.
- The most complicated parts are definitely Nat-elim and J, as their typing rules are much more complicated than for the other type/term formers. We still have to figure out how to make typechecking of this proof efficient.
- There are various tricky inductions that we could have overlooked without Agda. For instance, to prove totality for the term `J(A,P,d,a,b,p)` we need it for `Id(A,a,b)`, but it is not a subterm.
- Some admissible rules are also tricky to prove, like $\Gamma \vdash A[\delta] = A'[\delta']$ if $\Delta \vdash A = A'$ and $\Gamma \vdash \delta = \delta' : \Delta$.
- Strict propositions are very nice to use and seem quite helpful.