Strictifying Categories with Families

Ambrus Kaposi, Loïc Pujet

21 february 2025

Today's menu

Formalising normalisation proofs for dependent type theory

Today's menu

Formalising normalisation proofs for dependent type theory

~ with a side of gluing ~

I.

Introduction

Why bother proving normalisation?

Why bother proving normalisation?

Dependent type theory is a popular foundation for proof assistants: Agda, Coq/Rocq, Lean...

Why bother proving normalisation?

Dependent type theory is a popular foundation for proof assistants: Agda, Coq/Rocq, Lean...

It incorporates computation within the logical foundations

Why bother proving normalisation?

Dependent type theory is a popular foundation for proof assistants: Agda, Coq/Rocq, Lean...

It incorporates computation within the logical foundations

- Mathematical objects are considered up to $\beta\eta$-equality
- Mathematical constructions are programs!

Why bother proving normalisation?

Dependent type theory is a popular foundation for proof assistants: Agda, Coq/Rocq, Lean...

It incorporates computation within the logical foundations

- Mathematical objects are considered up to $\beta\eta$-equality
- Mathematical constructions are programs!

→ The problem of type-checking is now intrinsically linked with computation.

Why bother proving normalisation?

Normalisation property:

Every well-typed terms reduce to a normal form, and the $\beta\eta$-equality of terms corresponds to the syntactical equality of normal forms.

Road to normalisation

Road to normalisation

- First, we define a calculus of untyped terms:

$$\Lambda \quad := \quad x \mid t\,u \mid \lambda x.t \mid \ldots$$

Road to normalisation

- First, we define a calculus of untyped terms:

$$\Lambda \quad := \quad x \mid t\,u \mid \lambda x.t \mid \ldots$$

- Then we define a family of typing judgments

$$\Gamma \vdash t : A \qquad \Gamma \vdash t \equiv u : A \qquad \Gamma \vdash t \rightsquigarrow u : A \qquad \ldots$$

Road to normalisation

- First, we define a calculus of untyped terms:

$$\Lambda \quad := \quad x \mid t\, u \mid \lambda x.t \mid \ldots$$

- Then we define a family of typing judgments

$$\Gamma \vdash t : A \qquad \Gamma \vdash t \equiv u : A \qquad \Gamma \vdash t \rightsquigarrow u : A \qquad \ldots$$

which are inductively generated by typing rules:

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : \Pi (x : A).B} \qquad \frac{\Gamma \vdash t \equiv u : A}{\Gamma \vdash u \equiv t : A} \qquad \ldots$$

Proving normalisation

- If we want to prove a metatheoretical property of well-typed terms, our only option is induction on typing derivations

Proving normalisation

- If we want to prove a metatheoretical property of well-typed terms, our only option is induction on typing derivations
- For complex properties, a naive induction will not go through: we must strengthen the induction hypothesis

Proving normalisation

- If we want to prove a metatheoretical property of well-typed terms, our only option is induction on typing derivations
- For complex properties, a naive induction will not go through: we must strengthen the induction hypothesis
- The standard tool for this is logical relations

Logical relations in two seconds

- A reducible integer is a normalising term of type $\mathbb{N}$

Logical relations in two seconds

- A reducible integer is a normalising term of type $\mathbb{N}$
- A reducible function of type $A \to B$ sends reducible terms of type A to reducible terms of type B

Logical relations in two seconds

- A reducible integer is a normalising term of type $\mathbb{N}$
- A reducible function of type $A \to B$ sends reducible terms of type A to reducible terms of type B
- A reducible term of type $A \times B$ has a first projection that is reducible of type A and a second projection that is reducible of type B

Logical relations in two seconds

- A reducible integer is a normalising term of type $\mathbb{N}$
- A reducible function of type $A \to B$ sends reducible terms of type A to reducible terms of type B
- A reducible term of type $A \times B$ has a first projection that is reducible of type A and a second projection that is reducible of type B
- A reducible term of type U...

The devil is in the details

- Reducibility must be generalised to contexts and substitutions

The devil is in the details

- Reducibility must be generalised to contexts and substitutions
- We must account for conversion
  → Reducibility predicates should really be partial equivalence relations (PERs) on terms

The devil is in the details

- Reducibility must be generalised to contexts and substitutions
- We must account for conversion
  → Reducibility predicates should really be **partial equivalence relations** (PERs) on terms
- We must account for free variables
  → Reducibility PERs should contain the PER of neutral terms

The devil is in the details

- Reducibility must be generalised to contexts and substitutions
- We must account for conversion
  → Reducibility predicates should really be partial equivalence relations (PERs) on terms
- We must account for free variables
  → Reducibility PERs should contain the PER of neutral terms
- We must account for weakening
  → Reducibility PERs should really be presheaves of PERs

Taming the devil with boilerplate

All these subtleties result in long, technical and error-prone proofs.

Taming the devil with boilerplate

All these subtleties result in long, technical and error-prone proofs.

→ good candidate for formalisation!

Taming the devil with boilerplate

All these subtleties result in long, technical and error-prone proofs.

→ good candidate for formalisation!

- Barras, Werner, "Coq in Coq" (1997)
- Barras, "Intuitionistic Set Theory and Type Theories with Inductive Families" (2012)
- Wieczorek, Biernacki, "A Coq Formalization of Normalization by Evaluation for Martin-Löf Type Theory" (2018)
- Abel, Öhman, Vezzosi, "Decidability of conversion for type theory in type theory" (2018)
- Adjedj, Lennon-Bertrand, Maillard, Pédrot, P., "Martin-Löf à la coq" (2023)

II.

Gluing? Quésaco?

The algebraist's approach

The algebraist's approach

Instead of reasoning on syntax, we shift our focus to a well-behaved category of models. For dependent type theory, a common option is Categories with Families (CwF)

The algebraist's approach

Instead of reasoning on syntax, we shift our focus to a well-behaved category of models. For dependent type theory, a common option is Categories with Families (CwF)

A category with families is the data of:

The algebraist's approach

Instead of reasoning on syntax, we shift our focus to a well-behaved category of models. For dependent type theory, a common option is Categories with Families (CwF)

A category with families is the data of:

- A category of contexts and subtitutions
- For every context Γ, a set of types $\mathsf{Ty}\ \Gamma$
- For every subst. $\sigma : \Delta \to \Gamma$, a function $\mathsf{Ty}\ \Gamma \to \mathsf{Ty}\ \Delta$
- For every context Γ and type A, a set of terms $\mathsf{Tm}\ \Gamma\ A$
- For every subst. $\sigma : \Delta \to \Gamma$, a function $\mathsf{Tm}\ \Gamma\ A \to \mathsf{Tm}\ \ \Delta\ \ A[\sigma]$
- Context extensions $\Gamma \triangleright A$, context projections $\mathsf{wk} : \Gamma \triangleright A \to \Gamma$ and $\mathsf{var}_0 : \mathsf{Tm}\ \ (\Gamma \triangleright A)\ \ (A[\mathsf{wk}])$

The algebraist's approach

Instead of reasoning on syntax, we shift our focus to a well-behaved category of models. For dependent type theory, a common option is Categories with Families (CwF)

A category with families is the data of:

- A category of contexts and subtitutions
- For every context Γ, a set of types $\mathsf{Ty}\ \Gamma$
- For every subst. $\sigma : \Delta \to \Gamma$, a function $\mathsf{Ty}\ \Gamma \to \mathsf{Ty}\ \Delta$
- For every context Γ and type A, a set of terms $\mathsf{Tm}\ \Gamma\ A$
- For every subst. $\sigma : \Delta \to \Gamma$, a function $\mathsf{Tm}\ \Gamma\ A \to \mathsf{Tm}\ \ \Delta\ \ A[\sigma]$
- Context extensions $\Gamma \triangleright A$, context projections $\mathsf{wk} : \Gamma \triangleright A \to \Gamma$ and $\mathsf{var}_0 : \mathsf{Tm}\ \ (\Gamma \triangleright A)\ \ (A[\mathsf{wk}])$
- and more...

Syntax without syntax

Syntax without syntax

The point is, CwFs are presented by an algebraic theory with sorts, terms and equations.

Syntax without syntax

The point is, CwFs are presented by an algebraic theory with sorts, terms and equations.

General theorems ensure the existence of an initial CwF
This initial model is the syntax!

Syntax without syntax

The point is, CwFs are presented by an algebraic theory with sorts, terms and equations.

General theorems ensure the existence of an initial CwF
This initial model is the "syntax"

(intrinsically well-typed terms quotiented by conversion.)

Moving the goalposts

We can reformulate most meta-theoretical properties to be about the initial CwF, without needing to mention raw terms.

Moving the goalposts

We can reformulate most meta-theoretical properties to be about the initial CwF, without needing to mention raw terms.

- The initial CwF satisfies canonicity for booleans if any closed inhabitant of $\mathbb{B}$ is convertible to either $true$ or $false$

Moving the goalposts

We can reformulate most meta-theoretical properties to be about the initial CwF, without needing to mention raw terms.

- The initial CwF satisfies canonicity for booleans if any closed inhabitant of $\mathbb{B}$ is convertible to either $true$ or $false$
- In order to talk about normalisation, we first need to define the set of normal forms of type A for any type A.

Moving the goalposts

We can reformulate most meta-theoretical properties to be about the initial CwF, without needing to mention raw terms.

- The initial CwF satisfies canonicity for booleans if any closed inhabitant of $\mathbb{B}$ is convertible to either $true$ or $false$
- In order to talk about normalisation, we first need to define the set of normal forms of type A for any type A.

  Then, the initial CwF satisfies normalisation if any (possibly open) term of A is convertible to a normal form of of type A.

Moving the goalposts

We can reformulate most meta-theoretical properties to be about the initial CwF, without needing to mention raw terms.

- The initial CwF satisfies canonicity for booleans if any closed inhabitant of $\mathbb{B}$ is convertible to either $true$ or $false$
- In order to talk about normalisation, we first need to define the set of normal forms of type A for any type A.

  Then, the initial CwF satisfies normalisation if any (possibly open) term of A is convertible to a normal form of of type A.

  Assuming the equality of normal forms is decidable and the proof is constructive, this is enough to obtain decidability of typechecking.

The algebraist revisits reducibility

In order to prove properties of the initial CwF, we ditch logical relations for a newer and fancier tool: gluing

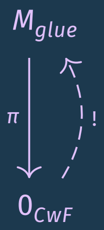
The algebraist revisits reducibility

In order to prove properties of the initial CwF, we ditch logical relations for a newer and fancier tool: gluing

$$\begin{array}{c} M_{glue} \\ \pi \big\downarrow \quad \big\uparrow ! \\ 0_{CwF} \end{array}$$

The algebraist revisits reducibility

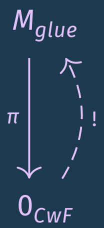
In order to prove properties of the initial CwF, we ditch logical relations for a newer and fancier tool: gluing



M_{glue} is the glued model, whose types are pairs of a type A of 0_{CwF} and a "reducibility structure" $A \to Set$

The algebraist revisits reducibility

In order to prove properties of the initial CwF, we ditch logical relations for a newer and fancier tool: gluing

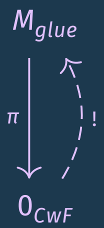


M_{glue} is the glued model, whose types are pairs of a type A of 0_{CwF} and a "reducibility structure" $A \to Set$

π is the first projection

The algebraist revisits reducibility

In order to prove properties of the initial CwF, we ditch logical relations for a newer and fancier tool: gluing



M_{glue} is the glued model, whose types are pairs of a type A of 0_{CwF} and a "reducibility structure" $A \to Set$

π is the first projection

Initiality ensures that π has a section, which associates a proof of reducibility to any object of 0_{CwF}.

The algebraist revisits reducibility

In the end, this is still a form of induction.

The algebraist revisits reducibility

In the end, this is still a form of induction.

But instead of using PERs on raw syntax, we use proof-relevant predicates on well-typed syntax quotiented by conversion.

The algebraist revisits reducibility

In the end, this is still a form of induction.

But instead of using PERs on raw syntax, we use proof-relevant predicates on well-typed syntax quotiented by conversion.

The resulting proof is arguably more principled and cleaner

The algebraist revisits reducibility

In the end, this is still a form of induction.

But instead of using PERs on raw syntax, we use proof-relevant predicates on well-typed syntax quotiented by conversion.

The resulting proof is arguably more principled and cleaner…at least on paper!

III.

Gluing in a proof assistant

Extensionality in type theory

Extensionality in type theory

First obstacle:

In order to formalise a proof of normalisation by gluing, we need quotient types, and function extensionality

Extensionality in type theory

First obstacle:

In order to formalise a proof of normalisation by gluing, we need quotient types, and function extensionality...both of which are problematic in dependent type theory.

Extensionality in type theory

First obstacle:

In order to formalise a proof of normalisation by gluing, we need quotient types, and function extensionality...both of which are problematic in dependent type theory.

Postulating function extensionality blocks computation

match (funext *e*) with
| refl => *t* $\rightsquigarrow$???

Extensionality in type theory

First obstacle:

In order to formalise a proof of normalisation by gluing, we need quotient types, and function extensionality...both of which are problematic in dependent type theory.

Postulating function extensionality blocks computation

match (funext e) with
| refl => t $\rightsquigarrow$???

However, we really wanted to extract an algorithm from our proof!

Extensionality in type theory

No reason to panic: in 2025, this is not an insurmountable problem anymore. We have several options:

- Cubical type theory (Coquand, Cohen, Huber, Mörtberg '16)
- Observational type theory (Altenkirch, McBride, Swierstra '07)

Extensionality in type theory

No reason to panic: in 2025, this is not an insurmountable problem anymore. We have several options:

- Cubical type theory (Coquand, Cohen, Huber, Mörtberg '16)
- Observational type theory (Altenkirch, McBride, Swierstra '07)

By changing the behaviour of equality, these theories support function extensionality and quotient types while retaining all the metatheoretical properties of Martin-Löf type theory

Extensionality in type theory

No reason to panic: in 2025, this is not an insurmountable problem anymore. We have several options:

- Cubical type theory (Coquand, Cohen, Huber, Mörtberg '16)
- Observational type theory (Altenkirch, McBride, Swierstra '07)

By changing the behaviour of equality, these theories support function extensionality and quotient types while retaining all the metatheoretical properties of Martin-Löf type theory

Furthermore, OTT is available in Coq/Rocq (P., Leray, Tabareau) and can be implemented in Agda using rewriting rules.

Gluing in a proof assistant

Gluing in a proof assistant

The first step is a definition of the initial CwF
Most natural option: some form of inductive type

Gluing in a proof assistant

The first step is a definition of the initial CwF
Most natural option: some form of inductive type

More specifically, the initial CwF is best described as a quotient inductive-inductive type

Gluing in a proof assistant

The first step is a definition of the initial CwF
Most natural option: some form of inductive type

More specifically, the initial CwF is best described as a quotient inductive-inductive type

```
Con    : Set i
Sub    : Con → Con → Set j
_∘_    : ∀{Γ Δ} → Sub Δ Γ → ∀{Θ} → Sub Θ Δ → Sub Θ Γ
ass    : ∀{Γ Δ}{γ : Sub Δ Γ}{Θ}{δ : Sub Θ Δ}{Ξ}{θ : Sub Ξ Θ} → ((γ ∘ δ) ∘ θ) ~ (γ ∘ (δ ∘ θ))
id     : ∀{Γ} → Sub Γ Γ
idl    : ∀{Γ Δ}{γ : Sub Δ Γ} → (id ∘ γ) ~ γ
idr    : ∀{Γ Δ}{γ : Sub Δ Γ} → (γ ∘ id) ~ γ
◇      : Con
ε      : ∀{Γ} → Sub Γ ◇
◇η     : ∀{Γ}{σ : Sub Γ ◇} → σ ~ (ε {Γ})

Ty     : Con → Set k
_[_]T : ∀{Γ} → Ty Γ → ∀{Δ} → Sub Δ Γ → Ty Δ
[∘]T   : ∀{Γ}{A : Ty Γ}{Δ}{γ : Sub Δ Γ}{Θ}{δ : Sub Θ Δ} → A [ γ ∘ δ ]T ~ A [ γ ]T [ δ ]T
[id]T : ∀{Γ}{A : Ty Γ} → A [ id ]T ~ A

Tm     : (Γ : Con) → Ty Γ → Set l
_[_]t : ∀{Γ}{A : Ty Γ} → Tm Γ A → ∀{Δ}(γ : Sub Δ Γ) → Tm Δ (A [ γ ]T)
[∘]t   : ∀{Γ}{A : Ty Γ}{a : Tm Γ A}{Δ}{γ : Sub Δ Γ}{Θ}{δ : Sub Θ Δ} → a [ γ ∘ δ ]t ~ a [ γ ]t [ δ ]t
[id]t : ∀{Γ}{A : Ty Γ}{a : Tm Γ A} → a [ id ]t ~ a

_▹_    : (Γ : Con) → Ty Γ → Con
_,[_]_: ∀{Γ Δ}(γ : Sub Δ Γ) → ∀ {A A'} → A [ γ ]T ~ A' → Tm Δ A' → Sub Δ (Γ ▹ A)
p      : ∀{Γ A} → Sub (Γ ▹ A) Γ
q      : ∀{Γ A} → Tm (Γ ▹ A) (A [ p ]T)
▹β₁    : ∀{Γ Δ}{γ : Sub Δ Γ}{A}{a : Tm Δ (A [ γ ]T)} → p ∘ (γ ,[ ~refl ] a) ~ γ
▹β₂    : ∀{Γ Δ}{γ : Sub Δ Γ}{A}{a : Tm Δ (A [ γ ]T)} → q [ γ ,[ ~refl ] a ]t ~ a
▹η     : ∀{Γ Δ A}{γa : Sub Δ (Γ ▹ A)} → ((p ∘ γa) ,[ [∘]T ] (q [ γa ]t)) ~ γa
```

Gluing in a proof assistant

Then, we want to define indexed CwFs over the initial CwF
→ another, even more complex, list of fields

Gluing in a proof assistant

Then, we want to define indexed CwFs over the initial CwF
→ another, even more complex, list of fields

Finally, we want to define the glued model as an indexed CwF
→ welcome to transport hell!

Gluing in a proof assistant

The transport hell is much worse than what we are used to:

Gluing in a proof assistant

The transport hell is much worse than what we are used to:

In traditional proofs, terms are a first order object and substitutions are defined by recursion on terms.
Most substitution laws become definitional equalities

$$(\Pi\ A\ B)[\sigma] \equiv \Pi\ (A[\sigma])\ (B[\sigma \uparrow])$$

Gluing in a proof assistant

The transport hell is much worse than what we are used to:

In traditional proofs, terms are a first order object and substitutions are defined by recursion on terms.
Most substitution laws become definitional equalities

$$(\Pi\ A\ B)[\sigma] \equiv \Pi\ \ (A[\sigma])\ \ (B[\sigma \uparrow])$$

But in our QIIT formulation, substitutions are part of the algebra signature, and we only get propositional equalities

$$(\Pi\ A\ B)[\sigma] = \Pi\ \ (A[\sigma])\ \ (B[\sigma \uparrow])$$

Gluing in a proof assistant

In conclusion, normalisation by gluing is even less tractable than old fashioned normalisation proofs.

IV.

Strictification

From propositional to definitional

Point of today's talk:
give an alternative definition of the initial CwF, for which almost all of the administrative equations become definitional equalities.

Strictifying groups

Suppose G is a group:

$$
\begin{array}{lll}
G & : & Set \\
\_\times\_ & : & G \to G \to G \\
inv & : & G \to G \\
e & : & G
\end{array}
\qquad
\begin{array}{lll}
unit_l & : & \forall x,\ e \times x = x \\
unit_r & : & \forall x,\ x \times e = x \\
inv_l & : & \forall x,\ (inv\ x) \times x = e \\
inv_r & : & \forall x,\ x \times (inv\ x) = e
\end{array}
$$

$$assoc : \forall x\ y\ z,\ (x \times y) \times z = x \times (y \times z)$$

Strictifying groups

Suppose G is a group:

$$\begin{array}{lll} G & : & Set \\ \_\times\_ & : & G \to G \to G \\ inv & : & G \to G \\ e & : & G \end{array} \qquad \begin{array}{lll} unit_l & : & \forall x,\ e \times x = x \\ unit_r & : & \forall x,\ x \times e = x \\ inv_l & : & \forall x,\ (inv\ x) \times x = e \\ inv_r & : & \forall x,\ x \times (inv\ x) = e \end{array}$$

$$assoc : \forall x\ y\ z,\ (x \times y) \times z = x \times (y \times z)$$

Then G embeds in the group of permutations of G (Cayley's theorem)

$$Perm(G) := \{\, f : G \to G \mid \text{isBijective } f \,\}$$

Strictifying groups

Suppose G is a group:

$$
\begin{array}{lcl}
G & : & Set \\
\_ \times \_ & : & G \to G \to G \\
inv & : & G \to G \\
e & : & G \\
assoc & : & \forall x\ y\ z,\ (x \times y) \times z = x \times (y \times z)
\end{array}
\qquad
\begin{array}{lcl}
unit_l & : & \forall x,\ e \times x = x \\
unit_r & : & \forall x,\ x \times e = x \\
inv_l & : & \forall x,\ (inv\ x) \times x = e \\
inv_r & : & \forall x,\ x \times (inv\ x) = e
\end{array}
$$

Then G embeds in the group of permutations of G (Cayley's theorem)

$$Perm(G) := \{\ f : G \to G \mid \text{isBijective}\ f\ \}$$

Essential point: the group law on Perm(G) is given by function composition, which is definitionally associative and unital!

Strictifying groups

If we have access to a sort of proof-irrelevant propositions, we can define a group that is isomorphic to G:

$$G' := \{\, f : G \to G \mid \exists (g : G), f = \tau_g \,\}$$

With G' being definitionally associative and unital:

$$((f, f_\varepsilon) \circ (g, g_\varepsilon)) \circ (h, h_\varepsilon) \equiv (f, f_\varepsilon) \circ ((g, g_\varepsilon) \circ (h, h_\varepsilon))$$
$$(f, f_\varepsilon) \circ (id, id_\varepsilon) \equiv (f, f_\varepsilon)$$
$$(id, id_\varepsilon) \circ (f, f_\varepsilon) \equiv (f, f_\varepsilon)$$

Strictifying CwFs, first attempt

Cayley's theorem is an instance of the Yoneda lemma:
any category $\mathcal{C}$ embeds into the category $\hat{\mathcal{C}} := \mathrm{Hom}(\mathcal{C}^{op}, \mathrm{Set})$.

Strictifying CwFs, first attempt

Cayley's theorem is an instance of the Yoneda lemma:
any category $\mathcal{C}$ embeds into the category $\hat{\mathcal{C}} := \mathrm{Hom}(\mathcal{C}^{op}, \mathsf{Set})$.

The Yoneda generalises to categories with families:

Given a CwF $\mathcal{C}$, the presheaf category $\hat{\mathcal{C}}$ is naturally equipped with a CwF structure inherited from Set. Additionally, there is an embedding of CwFs $\mathcal{C} \to \hat{\mathcal{C}}$.

Strictifying CwFs, first attempt

Cayley's theorem is an instance of the Yoneda lemma:
any category $\mathcal{C}$ embeds into the category $\hat{\mathcal{C}} := \mathrm{Hom}(\mathcal{C}^{op}, \mathrm{Set})$.

The Yoneda generalises to categories with families:

Given a CwF $\mathcal{C}$, the presheaf category $\hat{\mathcal{C}}$ is naturally equipped with a CwF structure inherited from Set. Additionally, there is an embedding of CwFs $\mathcal{C} \to \hat{\mathcal{C}}$.

We can thus try the same trick: define $\mathcal{C}'$ to be the image of $\mathcal{C}$ under the embedding.

Strictifying CwFs, first attempt

$\mathcal{C}'$ is thus isomorphic to $\mathcal{C}$, and enjoys more definitional eqs:

- substitutions are definitionally associative
- substitutions are definitionally unital
- wk and var_0 satisfy their equations definitionally

Strictifying CwFs, first attempt

$\mathcal{C}'$ is thus isomorphic to $\mathcal{C}$, and enjoys more definitional eqs:

- substitutions are definitionally associative
- substitutions are definitionally unital
- wk and var_0 satisfy their equations definitionally

...BUT

The commutation of substitutions with binders is not definitional

$$(\Pi\ A\ B)[\sigma] \not\equiv \Pi\ (A[\sigma])\ (B[\sigma \uparrow])$$

Strict presheaves

Unfolding the computations, the reason why substitutions do not commute with binders boils down to natural transformations not being definitional

$$F_y\,(a\,|_f) \not\equiv (F_x\,a)\,|_f$$

Strict presheaves

Unfolding the computations, the reason why substitutions do not commute with binders boils down to natural transformations not being definitional

$$F_y\,(a\,|_f) \not\equiv (F_x\,a)\,|_f$$

In "Russian constructivism in a prefascist theory" (2020), Pédrot introduces prefascist sets, an alternative definition of presheaves that is strictly natural.

Strictifying CwFs, second attempt

If we reproduce our strictification construction using Pédrot's definition, we obtain a new CwF $\mathcal{C}''$, in which all* the administrative equalities are definitional.

Strictifying CwFs, second attempt

If we reproduce our strictification construction using Pédrot's definition, we obtain a new CwF $\mathcal{C}''$, in which all* the administrative equalities are definitional.

We formalised the construction of $\mathcal{C}''$ and its isomorphism with $\mathcal{C}$ in Agda.

Strictifying CwFs, second attempt

If we reproduce our strictification construction using Pédrot's definition, we obtain a new CwF $\mathcal{C}''$, in which all* the administrative equalities are definitional.

We formalised the construction of $\mathcal{C}''$ and its isomorphism with $\mathcal{C}$ in Agda. Surprisingly doable, even when the CwF is equipped with dependent products and booleans!

Back to our original goal

Applying our strictification construction to the initial CwF, it becomes much easier to construct gluing models. We were able to define a canonicity model (which computes normal forms for closed terms) in about 200 lines!

(the strictification construction is about 4000 lines)

Thank you!