Mechanizing Synthetic Tait Computability in Istari

CPP 2026
Jan 13th, 2026

Runming Li

j.w.w. Yue Yao, Robert Harper

Mechanizing *Synthetic Tait Computability* in *Istari*

An abstract categorical gluing approach to metatheory of type theory

Mechanizing *Synthetic Tait Computability* in *Istari*

An abstract categorical gluing approach to metatheory of type theory

Mechanizing *Synthetic Tait Computability* in *Istari*

A NuPRL-style proof assistant based on extensional type theory

Theorem (Canonicity)

Every closed term of type bool is equal to either true or false.

Proof Idea:

- equip every type A with a computability structure
- equip every element of type A with an evidence of the corresponding computability structure

$$\text{true} \quad \vdots \quad \text{inl}(\checkmark : \text{true} = \text{true}) \qquad\qquad e : \text{bool} \quad \vdots \quad (e = \text{true}) \vee (e = \text{false})$$

Proof Idea:

- equip every type A with a computability structure
- equip every element of type A with an evidence of the corresponding computability structure

$$\underset{\mathsf{inl}(\checkmark : \mathsf{true} = \mathsf{true})}{\mathsf{true}} \quad \vdots \quad \underset{(e = \mathsf{true}) \vee (e = \mathsf{false})}{e : \mathsf{bool}}$$

Proof Idea:

- equip every type A with a computability structure
- equip every element of type A with an evidence of the corresponding computability structure

$$\begin{array}{c} \mathsf{true} \\ \mathsf{inl}(\checkmark : \mathsf{true} = \mathsf{true}) \end{array} \quad \vdots \quad \begin{array}{c} e : \mathsf{bool} \\ (e = \mathsf{true}) \vee (e = \mathsf{false}) \end{array}$$

Proof Idea:

- equip every type A with a computability structure
- equip every element of type A with an evidence of the corresponding computability structure

$$\begin{array}{c} \mathsf{true} \\ \mathsf{inl}(\checkmark : \mathsf{true} = \mathsf{true}) \end{array} \quad \bullet \quad \begin{array}{c} e : \mathsf{bool} \\ (e = \mathsf{true}) \vee (e = \mathsf{false}) \end{array}$$

TRUE : BOOL

$$\begin{array}{c}\mathsf{true}\\ \mathsf{inl}(\checkmark : \mathsf{true} = \mathsf{true})\end{array} \;\vdots\; \begin{array}{c}e : \mathsf{bool}\\ (e = \mathsf{true}) \vee (e = \mathsf{false})\end{array}$$

$$\mathsf{IF} : \mathsf{BOOL} \to A \to A \to A$$

$$\lambda \begin{pmatrix} b \\ b \end{pmatrix} \begin{pmatrix} t \\ t \end{pmatrix} \begin{pmatrix} f \\ f \end{pmatrix} . \; \mathsf{case}\, b \,\{\mathsf{inl\_} \hookrightarrow t \mid \mathsf{inr\_} \hookrightarrow f\}$$

$$\mathsf{if}\ b\ t\ f$$

The blue terms work *exactly* like the red terms.

$$\text{IF TRUE}\ t\ f = t$$

$$\begin{array}{l}\text{if true } t\ f\\ \text{case}\ (\text{inl}(\checkmark))\ \{\text{inl\_} \hookrightarrow t \mid \text{inr\_} \hookrightarrow f\}\end{array} = \begin{array}{l}t\\ t\end{array}$$

The blue terms work *exactly* like the red terms.

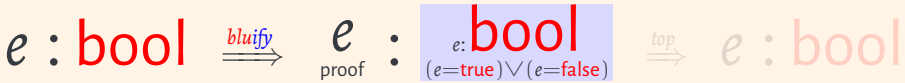
$$\mathsf{IF\ TRUE}\ t\ f = t$$

$$\begin{array}{l}\mathsf{if\ true}\ t\ f\\ \mathsf{case}\ (\mathsf{inl}(\checkmark))\ \{\mathsf{inl\_} \hookrightarrow t \mid \mathsf{inr\_} \hookrightarrow f\}\end{array} = \begin{array}{l}t\\ t\end{array}$$

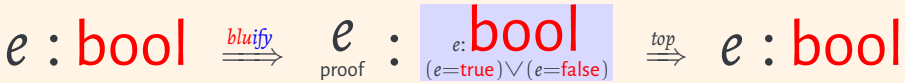
Suppose I have e : bool.

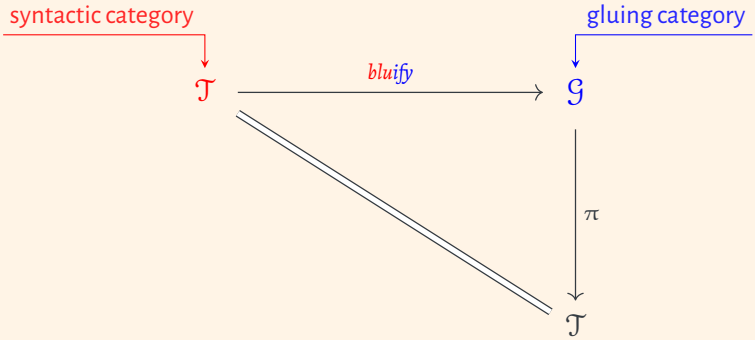
$$e : \mathsf{bool} \xrightarrow{\mathit{bluify}} \underset{\text{proof}}{e} : \underset{(e=\mathsf{true})\vee(e=\mathsf{false})}{e:\mathsf{bool}} \xrightarrow{\mathit{top}} e : \mathsf{bool}$$

Suppose I have e : bool.



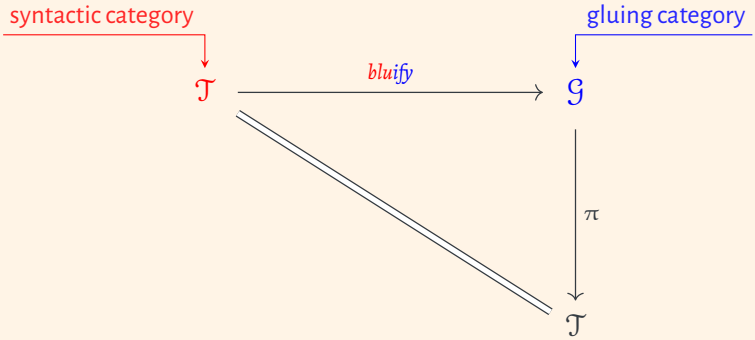
Suppose I have e : bool.





$$\pi \circ \textit{bluify} = \mathrm{id}_{\mathcal{T}}$$

"fundemental theorem of logical relations"



$$\pi \circ \mathit{bluify} = \mathrm{id}_{\mathcal{T}}$$

"fundemental theorem of logical relations"

Artin Gluing

need to mechanize them!

- It's a large proof with *a lot of* proof obligations
- Some proof obligations are *trivial* but *tedious*

e.g. naturality conditions:
all operations are stable under substitutions

Artin Gluing

need to mechanize them!

- It's a large proof with *a lot of* proof obligations
- Some proof obligations are *trivial* but *tedious*

e.g. naturality conditions:
all operations are stable under substitutions

This is the *analytic* story of Tait's computability!

The *synthetic* story: describe the functor *bluify* in a (modal) type theory!

The type theory of synthetic Tait computability

Axiom (synthetic phase distinction)

There is an intuitionistic proposition $\mathrm{syn} : \mathrm{Prop}$ *that controls syntactic v.s. semantic aspects.*

Definition

- ○ open modality isolates syntax
- ● closed modality isolates semantics

○●$A \cong 1$

have the ability to isolate syntax from semantics, but not vice versa!

The type theory of synthetic Tait computability

Axiom (synthetic phase distinction)

There is an intuitionistic proposition $\mathrm{syn} : \mathrm{Prop}$ *that controls syntactic v.s. semantic aspects.*

Definition

- ○ open modality isolates syntax
- ● closed modality isolates semantics

$\bigcirc\bullet A \cong 1$

have the ability to isolate syntax from semantics, but not vice versa!

The type theory of synthetic Tait computability

Axiom (synthetic phase distinction)

There is an intuitionistic proposition $\mathrm{syn} : \mathrm{Prop}$ *that controls syntactic v.s. semantic aspects.*

Definition

- ○ open modality isolates syntax
- ● closed modality isolates semantics

$\bigcirc\bullet A \cong \mathbf{1}$

have the ability to isolate syntax from semantics, but not vice versa!

Synthetic Tait Computability

Parametricity for an ML module calculus — Sterling & Harper

Normalization for Cartesian Cubical Type Theory — Sterling & Angiuli

Normalization for a multimodal type theory — Gratzer

⋮

If $e : \mathsf{F}(\mathsf{nat})$, then $e = \mathsf{step}^c(\mathsf{ret}(\mathsf{suc}^n(\mathsf{zero})))$.

Canonicity for Cost-Aware Logical Framework — Li & Harper

A modal dependent type theory with a phase distinction between cost and behavior based on call-by-push-value.

Synthetic Tait Computability

Parametricity for an ML module calculus — Sterling & Harper

Normalization for Cartesian Cubical Type Theory — Sterling & Angiuli

Normalization for a multimodal type theory — Gratzer

⋮

If $e : \mathsf{F}(\mathsf{nat})$, then $e = \mathsf{step}^c(\mathsf{ret}(\mathsf{suc}^n(\mathsf{zero})))$.

Canonicity for Cost-Aware Logical Framework — Li & Harper

A modal dependent type theory with a phase distinction between cost and behavior based on call-by-push-value.

$$\mathsf{TRUE} : \mathsf{TM}\left(\mathsf{BOOL}\right)$$

$$[\mathsf{syn} \hookrightarrow \mathsf{true}, \eta_{\bullet}(\mathsf{inl}(\checkmark))] : (b : \mathsf{tm}(\mathsf{bool})) \ltimes \bullet((b = \mathsf{true}) + (b = \mathsf{false}))$$

Modality framework guarantees that $\bigcirc(\mathsf{TRUE} = \mathsf{true})$ by $\bigcirc\bullet A \cong \mathbb{1}$!

$$\mathsf{TRUE} : \mathsf{TM}\left(\mathsf{BOOL}\right)$$

$$[\mathsf{syn} \hookrightarrow \mathsf{true}, \eta_{\bullet}(\mathsf{inl}(\checkmark))] \quad : \quad (b : \mathsf{tm}(\mathsf{bool})) \ltimes \bullet((b = \mathsf{true}) + (b = \mathsf{false}))$$

Modality framework guarantees that $\bigcirc(\mathsf{TRUE} = \mathsf{true})$ by $\bigcirc\bullet A \cong \mathbf{1}$!

Want $\mathsf{TRUE} : \mathsf{TM}(\mathsf{BOOL})$ and $\bigcirc(\mathsf{TRUE} = \mathsf{true})$. Write this as $\mathsf{TRUE} : \{\mathsf{TM}(\mathsf{BOOL}) \mid \mathsf{syn} \hookrightarrow \mathsf{true}\}$.

To prove FTLR, it suffices to construct

$$\mathsf{BOOL} : \{\mathsf{TP} \mid \mathsf{syn} \hookrightarrow \mathsf{bool}\}$$
$$\mathsf{TRUE} : \{\mathsf{TM}(\mathsf{BOOL}) \mid \mathsf{syn} \hookrightarrow \mathsf{true}\}$$
$$\mathsf{FALSE} : \{\mathsf{TM}(\mathsf{BOOL}) \mid \mathsf{syn} \hookrightarrow \mathsf{false}\}$$
$$\mathsf{IF} : \{(b : \mathsf{TM}(\mathsf{BOOL})) \to \mathsf{TM}(C(\mathsf{TRUE})) \to \mathsf{TM}(C(\mathsf{FALSE})) \to \mathsf{TM}(C(b)) \mid \mathsf{syn} \hookrightarrow \mathsf{if}\}$$
$$\vdots$$

Want $\mathsf{TRUE} : \mathsf{TM}(\mathsf{BOOL})$ and $\bigcirc(\mathsf{TRUE} = \mathsf{true})$. Write this as $\mathsf{TRUE} : \{\mathsf{TM}(\mathsf{BOOL}) \mid \mathsf{syn} \hookrightarrow \mathsf{true}\}$.

To prove FTLR, it suffices to construct

$$\mathsf{BOOL} : \{\mathsf{TP} \mid \mathsf{syn} \hookrightarrow \mathsf{bool}\}$$
$$\mathsf{TRUE} : \{\mathsf{TM}(\mathsf{BOOL}) \mid \mathsf{syn} \hookrightarrow \mathsf{true}\}$$
$$\mathsf{FALSE} : \{\mathsf{TM}(\mathsf{BOOL}) \mid \mathsf{syn} \hookrightarrow \mathsf{false}\}$$
$$\mathsf{IF} : \{(b : \mathsf{TM}(\mathsf{BOOL})) \to \mathsf{TM}(C(\mathsf{TRUE})) \to \mathsf{TM}(C(\mathsf{FALSE})) \to \mathsf{TM}(C(b)) \mid \mathsf{syn} \hookrightarrow \mathsf{if}\}$$
$$\vdots$$

$\mathsf{TP} : \{\mathcal{U} \mid \mathsf{syn} \hookrightarrow \mathsf{tp}\}$

$\mathsf{TP} = (A : \mathsf{tp}) \ltimes \{\mathcal{U} \mid \mathsf{syn} \hookrightarrow \mathsf{tm}\, A\}$

$\mathsf{TM} : \{\mathsf{TP} \to \mathcal{U} \mid \mathsf{syn} \hookrightarrow \mathsf{tm}\}$

$\mathsf{TM}\, A = \pi_2 A$

$\mathsf{BOOL} : \{\mathsf{TP} \mid \mathsf{syn} \hookrightarrow \mathsf{bool}\}$

$\mathsf{BOOL} = [\mathsf{syn} \hookrightarrow \mathsf{bool}, (b : \mathsf{tm}(\mathsf{bool})) \ltimes \bullet((b = \mathsf{true}) + (b = \mathsf{false}))]$

$\mathsf{TRUE} : \{\mathsf{TM}(\mathsf{BOOL}) \mid \mathsf{syn} \hookrightarrow \mathsf{true}\}$

$\mathsf{TRUE} = [\mathsf{syn} \hookrightarrow \mathsf{true}, \eta_{\bullet}(\mathsf{inl}(\checkmark))]$

$$\mathsf{TP} : \{\mathcal{U} \mid \mathsf{syn} \hookrightarrow \mathsf{tp}\}$$
$$\mathsf{TP} = (A : \mathsf{tp}) \ltimes \{\mathcal{U} \mid \mathsf{syn} \hookrightarrow \mathsf{tm}\, A\}$$

$$\mathsf{TM} : \{\mathsf{TP} \to \mathcal{U} \mid \mathsf{syn} \hookrightarrow \mathsf{tm}\}$$
$$\mathsf{TM}\, A = \pi_2 A$$

$$\mathsf{BOOL} : \{\mathsf{TP} \mid \mathsf{syn} \hookrightarrow \mathsf{bool}\}$$
$$\mathsf{BOOL} = [\mathsf{syn} \hookrightarrow \mathsf{bool}, (b : \mathsf{tm}(\mathsf{bool})) \ltimes \bullet((b = \mathsf{true}) + (b = \mathsf{false}))]$$

$$\mathsf{TRUE} : \{\mathsf{TM}(\mathsf{BOOL}) \mid \mathsf{syn} \hookrightarrow \mathsf{true}\}$$
$$\mathsf{TRUE} = [\mathsf{syn} \hookrightarrow \mathsf{true}, \eta_\bullet(\mathsf{inl}(\checkmark))]$$

$$\mathsf{PI} : \{(A : \mathsf{TP}) \to (\mathsf{TM}\, A \to \mathsf{TP}) \to \mathsf{TP} \mid \mathsf{syn} \hookrightarrow \mathsf{pi}\}$$
$$\mathsf{PI}\, A\, B = [\mathsf{syn} \hookrightarrow \mathsf{pi}\, A\, B, (e : \mathsf{tm}(\mathsf{pi}\, A\, B)) \ltimes \{(a : \mathsf{TM}\, A) \to \mathsf{TM}(B\, a) \mid \mathsf{syn} \hookrightarrow \mathsf{app}\, e\}]$$

$$\mathsf{LAM} : \{((a : \mathsf{TM}\, A) \to \mathsf{TM}(B\, a)) \to \mathsf{TM}(\mathsf{PI}\, A\, B) \mid \mathsf{syn} \hookrightarrow \mathsf{lam}\}$$
$$\mathsf{LAM}\, f = [\mathsf{syn} \hookrightarrow \mathsf{lam}\, f, f]$$

$$\mathsf{APP} : \{\mathsf{TM}(\mathsf{PI}\, A\, B) \to (a : \mathsf{TM}\, A) \to \mathsf{TM}(B\, a) \mid \mathsf{syn} \hookrightarrow \mathsf{app}\}$$
$$\mathsf{APP}\, e\, a = (\pi_2\, e)\, a$$

$$\mathsf{PI}_\beta : \{\mathsf{APP}\, (\mathsf{LAM}\, f)\, a = f\, a \mid \mathsf{syn} \hookrightarrow \mathsf{pi}_\beta\}$$
$$\mathsf{PI}_\beta = \checkmark$$

$$\mathsf{PI} : \{(A : \mathsf{TP}) \to (\mathsf{TM}\, A \to \mathsf{TP}) \to \mathsf{TP} \mid \mathsf{syn} \hookrightarrow \mathsf{pi}\}$$
$$\mathsf{PI}\, A\, B = [\mathsf{syn} \hookrightarrow \mathsf{pi}\, A\, B, (e : \mathsf{tm}(\mathsf{pi}\, A\, B)) \ltimes \{(a : \mathsf{TM}\, A) \to \mathsf{TM}(B\, a) \mid \mathsf{syn} \hookrightarrow \mathsf{app}\, e\}]$$

$$\mathsf{LAM} : \{((a : \mathsf{TM}\, A) \to \mathsf{TM}(B\, a)) \to \mathsf{TM}(\mathsf{PI}\, A\, B) \mid \mathsf{syn} \hookrightarrow \mathsf{lam}\}$$
$$\mathsf{LAM}\, f = [\mathsf{syn} \hookrightarrow \mathsf{lam}\, f, f]$$

$$\mathsf{APP} : \{\mathsf{TM}(\mathsf{PI}\, A\, B) \to (a : \mathsf{TM}\, A) \to \mathsf{TM}(B\, a) \mid \mathsf{syn} \hookrightarrow \mathsf{app}\}$$
$$\mathsf{APP}\, e\, a = (\pi_2\, e)\, a$$

$$\mathsf{PI}_\beta : \{\mathsf{APP}\, (\mathsf{LAM}\, f)\, a = f\, a \mid \mathsf{syn} \hookrightarrow \mathsf{pi}_\beta\}$$
$$\mathsf{PI}_\beta = \checkmark$$

Proof is short, but can be deceiving!

The proof is short because . . .

Hidden transport and equality reasoning

$$\mathsf{LAM} : \{((a : \mathsf{TM}\,A) \to \mathsf{TM}(B\,a)) \to \mathsf{TM}(\mathsf{PI}\,A\,B) \mid \mathsf{syn} \hookrightarrow \mathsf{lam}\}$$
$$\mathsf{LAM}\,f = [\mathsf{syn} \hookrightarrow \mathsf{lam}\,f, f]$$

need to cite $\mathsf{pi}_\beta : \mathsf{app}\,(\mathsf{lam}\,f)\,x = f\,x$ to make it type-check

need to "transport" along
$\bigcirc((a : \mathsf{TM}\,A) \to \mathsf{TM}(B\,a) = (a : \mathsf{tm}\,A) \to \mathsf{tm}(B\,a))$
which itself needs to "transport" along $\bigcirc((A : \mathsf{TP}) = (A : \mathsf{tp}))$

The proof is short because ...

Hidden transport and equality reasoning

$$\mathsf{LAM} : \{((a : \mathsf{TM}\,A) \to \mathsf{TM}(B\,a)) \to \mathsf{TM}(\mathsf{PI}\,A\,B) \mid \mathsf{syn} \hookrightarrow \mathsf{lam}\}$$
$$\mathsf{LAM}\,f = [\mathsf{syn} \hookrightarrow \mathsf{lam}\,f, f]$$

need to cite $\mathsf{pi}_\beta : \mathsf{app}\,(\mathsf{lam}\,f)\,x = f\,x$ to make it type-check

need to "transport" along
$\bigcirc((a : \mathsf{TM}\,A) \to \mathsf{TM}(B\,a) = (a : \mathsf{tm}\,A) \to \mathsf{tm}(B\,a))$
which itself needs to "transport" along $\bigcirc((A : \mathsf{TP}) = (A : \mathsf{tp}))$

In many proof assistant, this leads to “transport hell”

The proof is short because . . .

- Subtypes: if $a : \{A \mid \mathsf{syn} \hookrightarrow a_\circ\}$ then $a : A$
- Function extensionality
- Cumulative universes
- . . .

None of your favorite proof assistants support all of these features!
Except for one . . .

Istari

https://istarilogic.org

A NuPRL-style extensional proof assistant based on computational semantics developed by Karl Crary.

PER model over operational semantics, à la logical relations.

Allows: "ill-typed" terms, terms with multiple types, untyped reduction, etc[1].

[1] A great tutorial on computational semantics is Angiuli's thesis.

Istari

- Typing judgment is internal to the language.

  $M : A$ is a proposition to be proved by users.

- If $M : A(N)$ and $N = N'$, then $M : A(N')$.

  Propositional equality! Equality reflection in action.

- Type-checking is undecidable!

  If typechecker is stuck, user can manually take over!

Istari

- Typing judgment is internal to the language.

  $M : A$ is a proposition to be proved by users.

- If $M : A(N)$ and $N = N'$, then $M : A(N')$.

  Propositional equality! Equality reflection in action.

- Type-checking is undecidable!

  If typechecker is stuck, user can manually take over!

Istari

- Typing judgment is internal to the language.

  $M : A$ is a proposition to be proved by users.

- If $M : A(N)$ and $N = N'$, then $M : A(N')$.

  Propositional equality! Equality reflection in action.

- Type-checking is undecidable!

  If typechecker is stuck, user can manually take over!

Istari

```
define/id_vec{m n}/
/
  fn v . v
//
  forall(m n : nat). vec(m + n) → vec(n + m)
/;
  unfold /id_vec/.
  introOf /m n v/.
  typecheck.
  apply /Nat.plus_commute/.
qed();
```

a proof that $m + n = n + m$!

Istari

```
define /id_vec {m n}/
/
  fn v . v
//
  forall (m n : nat). vec (m + n) → vec (n + m)
/;
  unfold /id_vec/.
  introOf /m n v/.
  typecheck.
  apply /Nat.plus_commute/.
qed();
```

a proof that $m + n = n + m$!

Istari

```
define /id_vec {m n}/
/
  fn v . v
//
  forall (m n : nat). vec (m + n) → vec (n + m)
/;
  unfold /id_vec/.
  introOf /m n v/.
  typecheck.
  apply /Nat.plus_commute/.
qed();
```

a proof that $m + n = n + m$!

A library of phase distinction in Istari

- Phase
- Modalities
- Strict glue types
- Extension types

Thanks to Istari's subset type $\{x : A \mid P(x)\}$,
which handles the coercion implicitly: if $a : \{x : A \mid P(x)\}$, then $a : A$.

$$\mathsf{LAM} : \{((a : \mathsf{TM}\, A) \to \mathsf{TM}(B\, a)) \to \mathsf{TM}(\mathsf{PI}\, A\, B) \mid \mathsf{syn} \hookrightarrow \mathsf{lam}\}$$
$$\mathsf{LAM}\, f = [\mathsf{syn} \hookrightarrow \mathsf{lam}\, f, f]$$

Istari

```
define/LAM {A B}/
/
  fn f .glue (fn z .lam f) f
//
Ext ((forall(a : TM A).TM(B a)) → TM(PI A B)) (fn z .lam)
/;
  ...
qed();
```

$$\mathsf{LAM} : \{((a : \mathsf{TM}\, A) \to \mathsf{TM}(B\, a)) \to \mathsf{TM}(\mathsf{PI}\, A\, B) \mid \mathsf{syn} \hookrightarrow \mathsf{lam}\}$$
$$\mathsf{LAM}\, f = [\mathsf{syn} \hookrightarrow \mathsf{lam}\, f, f]$$

Istari

```
define/LAM {A B}/
/
  fn f .glue (fn z .lam f) f
//
Ext ((forall(a : TM A).TM(B a)) → TM(PI A B)) (fn z .lam)
/;
  ...
qed();
```

$$\mathsf{LAM} : \{\ ((a : \mathsf{TM}\,A) \to \mathsf{TM}(B\,a)) \to \mathsf{TM}(\mathsf{PI}\,A\,B)\ \mid \mathsf{syn} \hookrightarrow \mathsf{lam}\}$$

$$\mathsf{LAM}\,f = [\mathsf{syn} \hookrightarrow \mathsf{lam}\,f, f]$$

Istari

```
define/LAM {A B}/
/
  fn f .glue (fn z .lam f) f
//
Ext ( (forall(a : TM A).TM(B a)) → TM(PI A B) ) (fn z .lam)
/;
  ...
qed();
```

$$\mathsf{LAM} : \{((a : \mathsf{TM}\, A) \to \mathsf{TM}(B\, a)) \to \mathsf{TM}(\mathsf{PI}\, A\, B) \mid \mathsf{syn} \hookrightarrow \mathsf{lam}\}$$

$$\mathsf{LAM}\, f = [\mathsf{syn} \hookrightarrow \mathsf{lam}\, f, f]$$

Istari

```
define/LAM {A B}/
/
  fn f .glue (fn z .lam f) f
//
Ext ((forall(a : TM A).TM(B a)) → TM(PI A B)) ( fn z .lam )
/;
  ...
qed();
```

$$\mathsf{LAM} : \{((a : \mathsf{TM}\, A) \to \mathsf{TM}(B\, a)) \to \mathsf{TM}(\mathsf{PI}\, A\, B) \mid \mathsf{syn} \hookrightarrow \mathsf{lam}\}$$

$$\mathsf{LAM}\, f = [\, \mathsf{syn} \hookrightarrow \mathsf{lam}\, f\, , f]$$

Istari

```
define/LAM {A B}/
/
  fn f .glue ( fn z .lam f ) f
//
Ext ((forall(a : TM A).TM(B a)) → TM(PI A B)) (fn z .lam)
/;
  ...
qed();
```

$$\mathsf{LAM} : \{((a : \mathsf{TM}\, A) \to \mathsf{TM}(B\, a)) \to \mathsf{TM}(\mathsf{PI}\, A\, B) \mid \mathsf{syn} \hookrightarrow \mathsf{lam}\}$$

$$\mathsf{LAM}\, f = [\mathsf{syn} \hookrightarrow \mathsf{lam}\, f, f\,]$$

Istari

```
define/LAM {A B}/
/
  fn f .glue (fn z .lam f) f
//
Ext ((forall(a : TM A).TM(B a)) → TM(PI A B)) (fn z .lam)
/;
  ...
qed();
```

Our mechanized proof is almost verbatim from the on-paper one!

We mechanized · · ·

- Dependent type theory: booleans with large eliminations and Π-types, *etc.*
- Cost-Aware Logical Framework: call-by-push-value adjunction $\mathsf{F} \dashv \mathsf{U}$, cost-behavior phase distinction, cost effects, *etc.*

	Intensional	Extensional
Where is the proof?	In the term	In the typing derivation
Typechecking	Automatic	Mostly automatic but sometimes manual
Mechanized gluing	Kaposi & Pujet, ICFP 25	This work

	Intensional	Extensional
Where is the proof?	In the term	In the typing derivation
Typechecking	Automatic	Mostly automatic but sometimes manual
Mechanized gluing	Kaposi & Pujet, ICFP 25	This work

	Intensional	Extensional
Where is the proof?	In the term	In the typing derivation
Typechecking	Automatic	Mostly automatic but sometimes manual
Mechanized gluing	Kaposi & Pujet, ICFP 25	This work

	Intensional	Extensional
Where is the proof?	In the term	In the typing derivation
Typechecking	Automatic	Mostly automatic but sometimes manual
Mechanized gluing	Kaposi & Pujet, ICFP 25	This work

Lines of tactics	Definitions
0–100	TM, TP, TRUE, FALSE
100–200	PI, LAM, PI_β, BOOL
400–500	APP, PI_η
1200–1500	IF, IF_{β_1}, IF_{β_2}

Accordingly, these proofs [gluing] have a very extensional flavor, and as such are less amenable to implementation in a proof assistant based on intensional type theory. Moreover, the more sophisticated iterations [STC] rely on (multi)modal type theories as internal languages for feature-rich categories, for which mechanization is still in its infancy.

— Adjedj et al. Martin-Löf à la Coq. CPP 24

Bonus slides

[W]hereas in the past the synthetic and analytic viewpoints were opposed to each other, today they must be viewed as two parts of a whole whose interplay leads to new and useful developments in mathematics.

— Jon Sterling, Naïve logical relations in synthetic Tait computability

$$
\begin{aligned}
\mathsf{tp} &: \mathcal{U} \\
\mathsf{tm} &: \mathsf{tp} \to \mathcal{U} \\
\mathsf{bool} &: \mathsf{tp} \\
\mathsf{true}\ \mathsf{false} &: \mathsf{tm}(\mathsf{bool}) \\
\mathsf{if} &: (b : \mathsf{tm}(\mathsf{bool})) \to \mathsf{tm}(C(\mathsf{true})) \to \mathsf{tm}(C(\mathsf{false})) \to \mathsf{tm}(C(b)) \\
\mathsf{if}_{\beta_1} &: \mathsf{if}\ \mathsf{true}\ t\ f = t \\
\mathsf{if}_{\beta_2} &: \mathsf{if}\ \mathsf{false}\ t\ f = f \\
\mathsf{pi} &: (A : \mathsf{tp}) \to (\mathsf{tm}(A) \to \mathsf{tp}) \to \mathsf{tp} \\
\mathsf{lam} &: ((x : \mathsf{tm}(A)) \to \mathsf{tm}(B(x))) \to \mathsf{tm}(\mathsf{pi}\ A\ B) \\
\mathsf{app} &: \mathsf{tm}(\mathsf{pi}\ A\ B) \to (x : \mathsf{tm}(A)) \to \mathsf{tm}(B(x)) \\
\mathsf{pi}_\beta &: \mathsf{app}\ (\mathsf{lam}\ f)\ a = f\ a \\
\mathsf{pi}_\eta &: \mathsf{lam}\ (\mathsf{app}\ e) = e
\end{aligned}
$$

Axiom

Syntactically, there exists syntax tp, tm, bool, true, false, if, pi, lam, app, ⋯

Second-order syntax remove the burden of substitution and naturality!

In fact, out mechanization does not have any substitution proofs at all!

Of course, this limits the models that can be written down internally, but synthetically it gives rise to the correct category that represents the syntax.

Axiom

Syntactically, there exists syntax tp, tm, bool, true, false, if, pi, lam, app, · · ·

Second-order syntax remove the burden of substitution and naturality!

In fact, out mechanization does not have any substitution proofs at all!

Of course, this limits the models that can be written down internally, but synthetically it gives rise to the correct category that represents the syntax.

Axiom

Syntactically, there exists syntax tp, tm, bool, true, false, if, pi, lam, app, · · ·

Second-order syntax remove the burden of substitution and naturality!

In fact, out mechanization does not have any substitution proofs at all!

Of course, this limits the models that can be written down internally, but synthetically it gives rise to the correct category that represents the syntax.

Artin Gluing $\mathcal{G}$

$\Gamma = \mathrm{Hom}_{\mathcal{T}}(1_{\mathcal{T}}, -)$
the global section functor

$$\mathcal{G} = \mathbf{Set} \downarrow \Gamma$$

Objects:

$$\begin{pmatrix} S \in \mathbf{Set} \\ \downarrow f \\ \mathrm{Hom}_{\mathcal{T}}(1_{\mathcal{T}}, A) \end{pmatrix}$$

Think as: proof-relevant predicates on closed terms of type A!

Artin Gluing $\mathcal{G}$

$\Gamma = \mathsf{Hom}_{\mathcal{T}}(1_{\mathcal{T}}, -)$
the global section functor

$$\mathcal{G} = \mathbf{Set} \downarrow \Gamma$$

Objects:

$$\begin{pmatrix} S \in \mathbf{Set} \\ \downarrow f \\ \mathsf{Hom}_{\mathcal{T}}(1_{\mathcal{T}}, A) \end{pmatrix}$$

Think as: proof-relevant predicates on closed terms of type A!

Artin Gluing $\mathcal{G}$

$\Gamma = \mathsf{Hom}_{\mathcal{T}}(1_{\mathcal{T}}, -)$
the global section functor

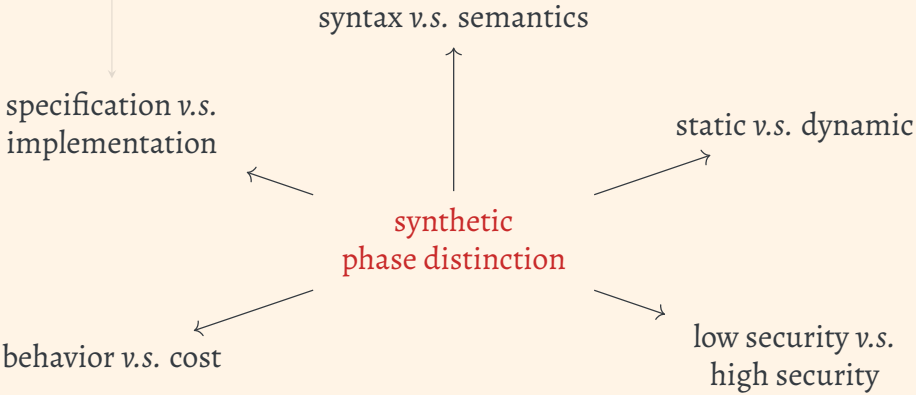
$$\mathcal{G} = \mathbf{Set} \downarrow \Gamma$$

Objects:

$$\begin{pmatrix} S \in \mathbf{Set} \\ \downarrow f \\ \mathsf{Hom}_{\mathcal{T}}(1_{\mathcal{T}}, A) \end{pmatrix}$$

Think as: proof-relevant predicates on closed terms of type A!

see Harrison Grodin's accompanying POPL talk
Abstraction Functions as Types on Friday!



see Harrison Grodin's accompanying POPL talk
Abstraction Functions as Types on Friday!