Mechanizing Synthetic Tait Computability in Istari

MURI Meeting
August 5, 2025

Runming Li

j.w.w. Yue Yao, Robert Harper

Mechanizing *Synthetic Tait Computability* in *Istari*

A categorical gluing approach to metatheory of programming languages using a modal dependent type theory

Mechanizing *Synthetic Tait Computability* in *Istari*

A categorical gluing approach to metatheory of programming languages using a modal dependent type theory

Mechanizing *Synthetic Tait Computability* in *Istari*

A proof assistant based on Computational Type Theory, in the tradition of NuPRL

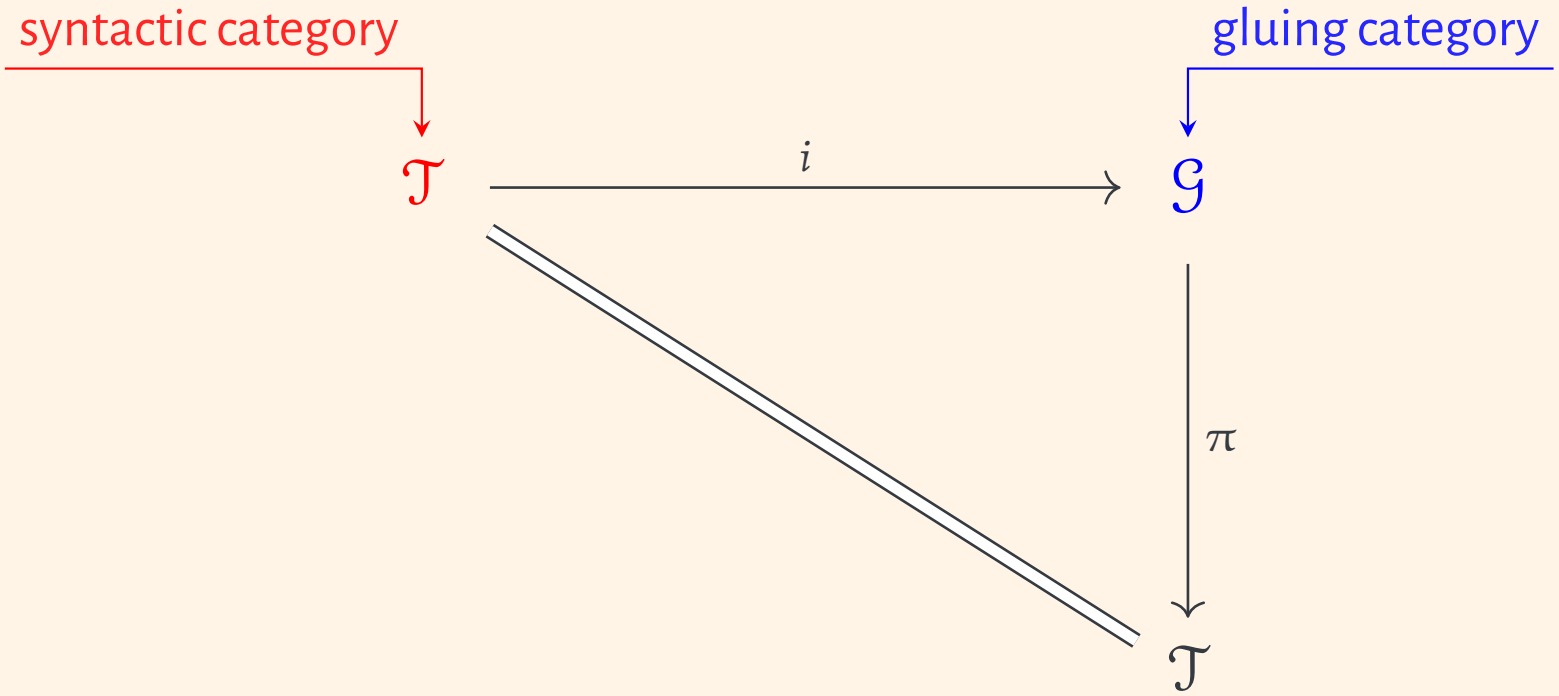
Canonicity via gluing

Start with a syntactic category $\mathcal{T}$.

Theorem (Canonicity)

For every $b : 1_{\mathcal{T}} \to tm_{\mathcal{T}}(bool_{\mathcal{T}})$, it must be the case that either $b = true_{\mathcal{T}}$ or $b = false_{\mathcal{T}}$.

A gluing category $\mathcal{G}$



Proof strategy:

- Construct a category $\mathcal{G}$;
- Find structure-preserving functors $i : \mathcal{T} \to \mathcal{G}$ and $\pi : \mathcal{G} \to \mathcal{T}$ such that $\pi \circ i = \mathrm{Id}_{\mathcal{T}}$.

A gluing category $\mathcal{G}$

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

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

A gluing category $\mathcal{G}$

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

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

Objects:

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

A gluing category $\mathcal{G}$

$\Gamma = \mathsf{Hom}_{\mathcal{T}}(1_{\mathcal{T}}, -)$
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!

Canonicity proof

Define

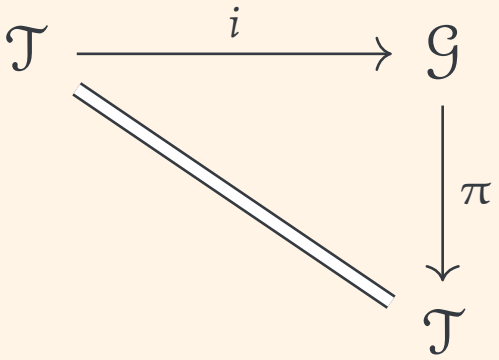
$$i : \mathcal{T} \to \mathcal{G}$$

$$i\,(\mathsf{tm}_{\mathcal{T}}(\mathsf{bool}_{\mathcal{T}})) = \mathsf{tm}_{\mathcal{G}}(\mathsf{bool}_{\mathcal{G}})$$

$$\vdots$$

$$\pi : \mathcal{G} \to \mathcal{T}$$

the forgetful functor



$$\pi \circ i = \mathsf{Id}_{\mathcal{T}}$$

Canonicity proof

Theorem (Canonicity)

For every $b : 1_{\mathcal{T}} \to tm_{\mathcal{T}}(bool_{\mathcal{T}})$, it must be the case that either $b = true_{\mathcal{T}}$ or $b = false_{\mathcal{T}}$.

Proof

By examining the morphism $i(b) : 1_{\mathcal{G}} \to \mathsf{tm}_{\mathcal{G}}(\mathsf{bool}_{\mathcal{G}})$.

□

Close $\mathcal{G}$ under connectives

- Need to close $\mathcal{G}$ under dependent products, dependent sums, equality types, and so on.
- A lot of subtle details to check such as naturality.
- No mechanization until very recently by Kaposi and Pujet [4].

Sterling's Synthetic Tait Computability

Sterling [8] identifies that the internal language of the gluing category $\mathcal{G}$ is a modal extensional dependent type theory based on a synthetic phase distinction between syntax and semantics.

c.f. Harrison's talk this morning!

Sterling's Synthetic Tait Computability

Gluing proofs become programming and type-checking obligations in this language.

Sterling [8] identifies that the internal language of the gluing category $\mathcal{G}$ is a modal extensional dependent type theory based on a synthetic phase distinction between syntax and semantics.

c.f. Harrison's talk this morning!

Synthetic Tait Computability

Parametricity for an ML module calculus	Sterling & Harper [10]
Normalization for Cartesian Cubical Type Theory	Sterling & Angiuli [9]
Normalization for a multimodal type theory	Gratzer [3]
⋮	
Canonicity for Cost-Aware Logical Framework [6]	Li & Harper [5]

Synthetic Tait Computability

Parametricity for an ML module calculus — Sterling & Harper [10]

Normalization for Cartesian Cubical Type Theory — Sterling & Angiuli [9]

Normalization for a multimodal type theory — Gratzer [3]

⋮

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

Canonicity for Cost-Aware Logical Framework [6] — Li & Harper [5]

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

Phase distinction between syntax and semantics

- A proposition syn : Prop.
  - If syn holds, then we say that we are in the syntactic phase.

Phase distinction between syntax and semantics

- A proposition syn : Prop.
  - If syn holds, then we say that we are in the syntactic phase.
- Phases induce modalities [7]:

$$\bigcirc A \triangleq \mathsf{syn} \to A$$

isolates the syntax

Phase distinction between syntax and semantics

- A proposition $\mathsf{syn} : \mathsf{Prop}$.
  - If syn holds, then we say that we are in the syntactic phase.
- Phases induce modalities [7]:

$$\bigcirc A \triangleq \mathsf{syn} \to A$$

isolates the syntax

$$\bullet A \triangleq A \sqcup \mathsf{syn}$$

isolates the semantics

$$\begin{array}{ccc} A \times \mathsf{syn} & \xrightarrow{\pi_2} & \mathsf{syn} \\ {\scriptstyle \pi_1}\downarrow & \ulcorner & \downarrow{\scriptstyle \star} \\ A & \xrightarrow{\eta} & \bullet A \end{array}$$

$$\begin{aligned} &\mathsf{data}\ \bullet\ (A : \mathcal{U}) : \mathcal{U}\ \mathsf{where} \\ &\quad \eta : A \to \bullet A \\ &\quad \star : (z : \mathsf{syn}) \to \bullet A \\ &\quad \_ : (a : A)\ (z : \mathsf{syn}) \to \eta\, a = \star\, z \end{aligned}$$

Phase distinction between syntax and semantics

- A proposition $\mathsf{syn} : \mathsf{Prop}$.
  - If syn holds, then we say that we are in the syntactic phase.
- Phases induce modalities [7]:

$$\bigcirc A \triangleq \mathsf{syn} \to A$$

isolates the syntax

$$\bullet A \triangleq A \sqcup \mathsf{syn}$$

isolates the semantics

$$\begin{array}{ccc} A \times \mathsf{syn} & \xrightarrow{\pi_2} & \mathsf{syn} \\ {\scriptstyle \pi_1}\downarrow & \ulcorner & \downarrow{\scriptstyle \star} \\ A & \xrightarrow{\eta} & \bullet A \end{array}$$

$$\begin{aligned} &\mathsf{data}\ \bullet\ (A : \mathcal{U}) : \mathcal{U}\ \mathsf{where} \\ &\quad \eta : A \to \bullet A \\ &\quad \star : (z : \mathsf{syn}) \to \bullet A \\ &\quad \_ : (a : A)\ (z : \mathsf{syn}) \to \eta\, a = \star\, z \end{aligned}$$

$\bigcirc\bullet A$ is contractible

Syntax

record $\mathbb{SIG}$ where

tp : $\mathcal{U}$

tm : tp $\to$ $\mathcal{U}$

bool : tp

true : tm bool

false : tm bool

Syntax

record $\mathbb{SIG}$ where

$\mathsf{tp} : \mathcal{U}$

$\mathsf{tm} : \mathsf{tp} \to \mathcal{U}$

$\mathsf{bool} : \mathsf{tp}$

$\mathsf{true} : \mathsf{tm}\ \mathsf{bool}$

$\mathsf{false} : \mathsf{tm}\ \mathsf{bool}$

$\mathsf{pi} : (A : \mathsf{tp}) \to (\mathsf{tm}\ A \to \mathsf{tp}) \to \mathsf{tp}$

$\mathsf{lam} : ((a : \mathsf{tm}\ A) \to \mathsf{tm}(B\ a)) \to \mathsf{tm}\ (\mathsf{pi}\ A\ B)$

$\mathsf{app} : \mathsf{tm}\ (\mathsf{pi}\ A\ B) \to (a : \mathsf{tm}\ A) \to \mathsf{tm}\ (B\ a)$

$\mathsf{pi}_\beta : \mathsf{app}\ (\mathsf{lam}\ f)\ x = f\ x$

$\mathsf{pi}_\eta : \mathsf{lam}\ (\mathsf{app}\ f) = f$

Semantics

Assume a model $\mathsf{M} : \mathbb{SIG}$ that represents the syntax.

Semantics

Assume a model $\mathsf{M} : \mathbb{SIG}$ that represents the syntax.

Construct a model $\mathsf{M} : \{\mathbb{SIG} \mid \mathsf{syn} \hookrightarrow \mathsf{M}\}$ that represents the semantics.

Extension type makes sure syntax and semantics agree: $\bigcirc(\mathsf{M} = \mathsf{M})$.

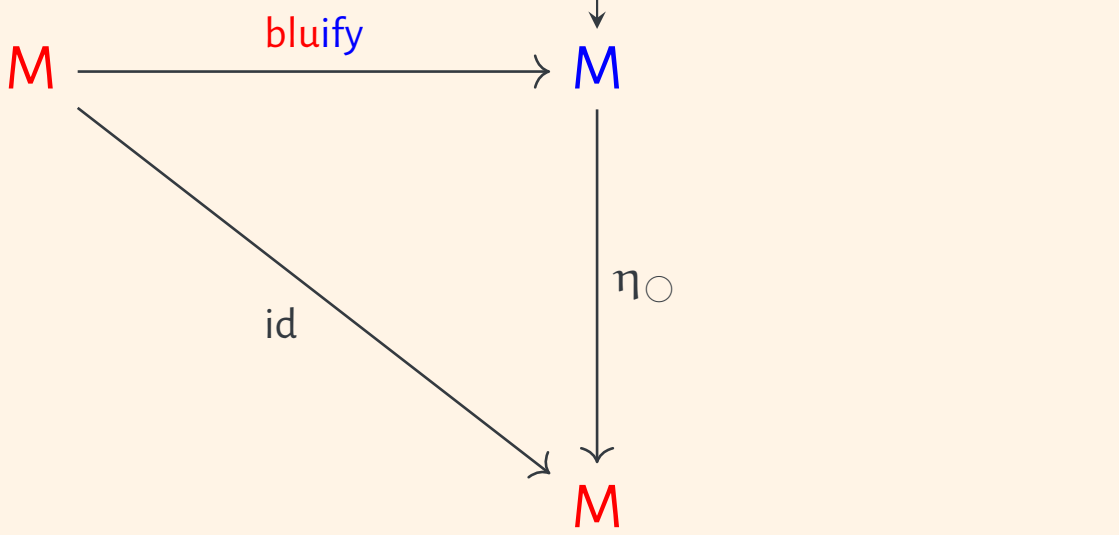
Semantics

Assume a model $\mathsf{M} : \mathbb{SIG}$ that represents the syntax.

Construct a model $\mathsf{M} : \{\mathbb{SIG} \mid \mathsf{syn} \hookrightarrow \mathsf{M}\}$ that represents the semantics.

Extension type makes sure syntax and semantics agree: $\bigcirc(\mathsf{M} = \mathsf{M})$.

defined in the internal language of $\mathcal{G}$



Constructing M

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

$$\mathsf{tp} = \Sigma_{A:\mathsf{tp}} \{\mathcal{U} \mid \mathsf{syn} \hookrightarrow \mathsf{tm}\, A\}$$

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

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

Semantics of booleans is that it is either true or false.

$$\mathsf{bool} : \{\mathsf{tp} \mid \mathsf{syn} \hookrightarrow \mathsf{bool}\}$$

$$\mathsf{bool} = (\mathsf{bool}, \Sigma_{b:\mathsf{tm}(\mathsf{bool})} \bigcirc((b = \mathsf{true}) + (b = \mathsf{false})))$$

$$\mathsf{true} : \{\mathsf{tm}(\mathsf{bool}) \mid \mathsf{syn} \hookrightarrow \mathsf{true}\}$$

$$\mathsf{true} = (\mathsf{true}, \eta_{\bigcirc}(\mathsf{inl}(\checkmark)))$$

Constructing M

$$\mathsf{tp} : \{\mathcal{U} \mid \mathsf{syn} \hookrightarrow \mathsf{tp}\}$$
$$\mathsf{tp} = \Sigma_{A:\mathsf{tp}} \{\mathcal{U} \mid \mathsf{syn} \hookrightarrow \mathsf{tm}\, A\}$$

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

Semantics of booleans is that it is either true or false.

$$\mathsf{bool} : \{\mathsf{tp} \mid \mathsf{syn} \hookrightarrow \mathsf{bool}\}$$
$$\mathsf{bool} = (\mathsf{bool}, \Sigma_{b:\mathsf{tm}(\mathsf{bool})} \bullet((b = \mathsf{true}) + (b = \mathsf{false})))$$

$$\mathsf{true} : \{\mathsf{tm}(\mathsf{bool}) \mid \mathsf{syn} \hookrightarrow \mathsf{true}\}$$
$$\mathsf{true} = (\mathsf{true}, \eta_\bullet(\mathsf{inl}(\checkmark)))$$

Constructing M

$$\mathsf{tp} : \{\mathcal{U} \mid \mathsf{syn} \hookrightarrow \mathsf{tp}\}$$
$$\mathsf{tp} = \Sigma_{A:\mathsf{tp}} \{\mathcal{U} \mid \mathsf{syn} \hookrightarrow \mathsf{tm}\, A\}$$

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

Semantics of booleans is that it is either true or false.

$$\mathsf{bool} : \{\mathsf{tp} \mid \mathsf{syn} \hookrightarrow \mathsf{bool}\}$$
$$\mathsf{bool} = (\mathsf{bool}, \Sigma_{b:\mathsf{tm}(\mathsf{bool})} \bigcirc((b = \mathsf{true}) + (b = \mathsf{false})))$$

$$\mathsf{true} : \{\mathsf{tm}(\mathsf{bool}) \mid \mathsf{syn} \hookrightarrow \mathsf{true}\}$$
$$\mathsf{true} = (\mathsf{true}, \eta_{\bigcirc}(\mathsf{inl}(\checkmark)))$$

Constructing M

$$\mathsf{pi} : \{(A : \mathsf{tp}) \to (\mathsf{tm}\, A \to \mathsf{tp}) \to \mathsf{tp} \mid \mathsf{syn} \hookrightarrow \mathsf{pi}\}$$

$$\mathsf{pi}\, A\, B = (\mathsf{pi}\, A\, B, \Sigma_{e:\mathsf{tm}(\mathsf{pi}\, A\, B)} \{(a : \mathsf{tm}\, A) \to \mathsf{tm}(B\, a) \mid \mathsf{syn} \hookrightarrow \mathsf{app}\, e\})$$

$$\mathsf{lam} : \{((a : \mathsf{tm}\, A) \to \mathsf{tm}(B\, a)) \to \mathsf{tm}(\mathsf{pi}\, A\, B) \mid \mathsf{syn} \hookrightarrow \mathsf{lam}\}$$

$$\mathsf{lam}\, f = (\mathsf{lam}\, f, f)$$

$$\mathsf{app} : \{\mathsf{tm}(\mathsf{pi}\, A\, B) \to (a : \mathsf{tm}\, A) \to \mathsf{tm}(B\, a) \mid \mathsf{syn} \hookrightarrow \mathsf{app}\}$$

$$\mathsf{app}\, e\, a = (\pi_2\, e)\, a$$

Constructing M

$$\mathsf{pi} : \{(A : \mathsf{tp}) \to (\mathsf{tm}\, A \to \mathsf{tp}) \to \mathsf{tp} \mid \mathsf{syn} \hookrightarrow \mathsf{pi}\}$$
$$\mathsf{pi}\, A\, B = (\mathsf{pi}\, A\, B, \Sigma_{e:\mathsf{tm}(\mathsf{pi}\ A\ B)} \{(a : \mathsf{tm}\, A) \to \mathsf{tm}(B\, a) \mid \mathsf{syn} \hookrightarrow \mathsf{app}\, e\})$$

$$\mathsf{lam} : \{((a : \mathsf{tm}\, A) \to \mathsf{tm}(B\, a)) \to \mathsf{tm}(\mathsf{pi}\, A\, B) \mid \mathsf{syn} \hookrightarrow \mathsf{lam}\}$$
$$\mathsf{lam}\, f = (\mathsf{lam}\, f, f)$$

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

Proof is short, but can be deceiving!

The proof is short because . . .

- Equality reflection, so reasoning about equality takes place judgmentally.

The proof is short because . . .

- Equality reflection, so reasoning about equality takes place judgmentally.

$$\mathsf{lam} : \{((a : \mathsf{tm}\, A) \to \mathsf{tm}(B\, a)) \to \mathsf{tm}(\mathsf{pi}\, A\, B) \mid \mathsf{syn} \hookrightarrow \mathsf{lam}\}$$

$$\mathsf{lam}\, f = (\mathsf{lam}\, f, f)$$

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

The proof is short because . . .

- Equality reflection, so reasoning about equality takes place judgmentally.

$$\mathsf{lam} : \{((a : \mathsf{tm}\, A) \to \mathsf{tm}(B\, a)) \to \mathsf{tm}(\mathsf{pi}\, A\, B) \mid \mathsf{syn} \hookrightarrow \mathsf{lam}\}$$

$$\mathsf{lam}\, f = (\mathsf{lam}\, f, f)$$

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

- Function extensionality, which also takes place judgmentally.
- Implicit coercion of extension types

Mechanizing this proof in any intensional proof assistant quickly gets out of hand!

"transport hell"

Need a proof assistant with actual equality reflection!

Friendship ended with MUBASIR

TRANSPORT

Now

EQUALITY

REFLECTION

is my

best friend

Istari

https://istarilogic.org

A NuPRL-style [2] proof assistant based on computational semantics developed by Karl Crary.

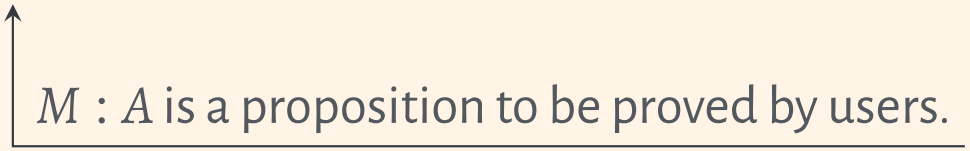
PER (modest set) model over operational semantics, à la logical relations.

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

[1] A great resource on computational semantics is Angiuli [1].

Istari

- Typing judgment is internal to the language.



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.

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 mannually take over!

Istari example

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 example

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 $m + n = n + m$!

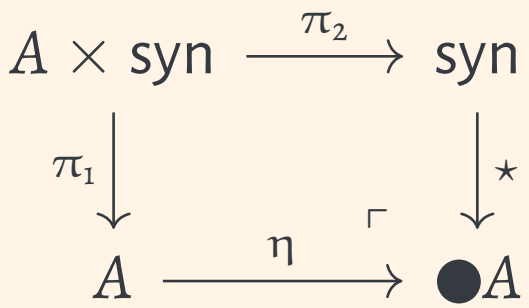
Istari example

Istari

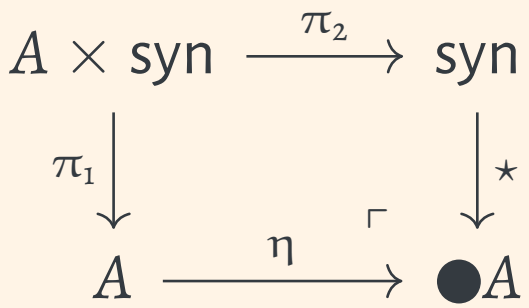
```
define/id_vec {m n}/
/
  fn v . v
//
  forall(m n : nat). vec(m + n) → vec(n + m)
/;
  unfold /id_vec/.
  introOf /m n v/.
  typecheck.
  apply /Nat.plus_commute/.
qed().
```

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

● modality



● modality



(Non-cubical) Agda

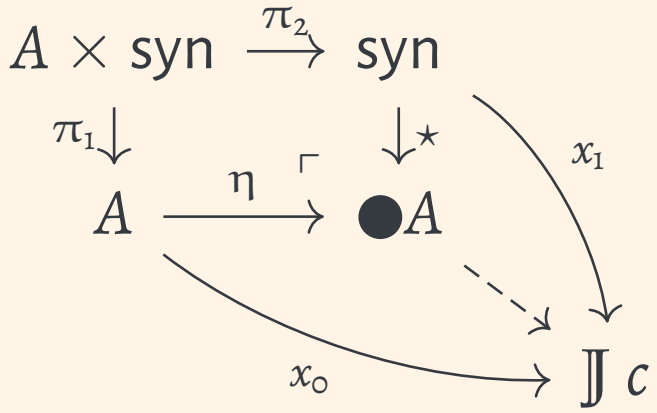
$\bullet : \mathsf{Type} \to \mathsf{Type}$

$\eta : A \to \bullet\, A$

$\star : \mathsf{syn} \to \bullet\, A$

$\eta{\equiv}\star : (a : A)\ (z : \mathsf{syn}) \to \eta\, a \equiv \star\, z$

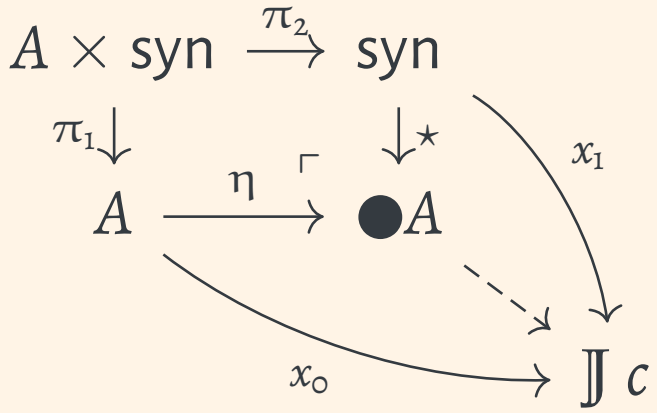
● modality



(Non-cubical) Agda

```
ind : (c : ● A)
      (𝕁 : ● A → Type)
      (x0 : (a : A) → 𝕁 (η a))
      (x1 : (z : syn) → 𝕁 (⋆ z))
      ((a : A) → (z : syn) → (subst 𝕁 (η=⋆ a z) (x0 a)) ≡ x1 z) →
      𝕁 c
```

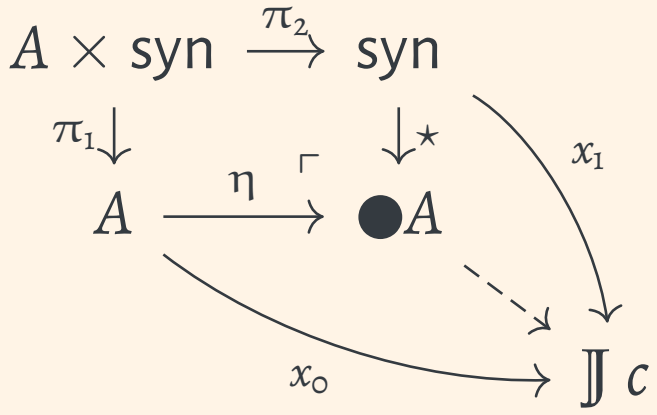
● modality



(Non-cubical) Agda

```
ind : (c : ● A)
      (𝕁 : ● A → Type)
      (x0 : (a : A) → 𝕁 (η a))
      (x1 : (z : syn) → 𝕁 (⋆ z))
      ((a : A) → (z : syn) → (subst 𝕁 (η≡⋆ a z) (x0 a)) ≡ x1 z) →
      𝕁 c
```

● modality



(Non-cubical) Agda

```
ind : (c : ● A)
      (𝕁 : ● A → Type)
      (x0 : (a : A) → 𝕁 (η a))
      (x1 : (z : syn) → 𝕁 (⋆ z))
      ((a : A) → (z : syn) → (subst 𝕁 (η≡⋆ a z) (x0 a)) ≡ x1 z) →
      𝕁 c
```

modality in Istari

Istari

$$
\begin{aligned}
&\mathit{ind} : \mathsf{forall}(c : \mathit{closed}\ A) \\
&\quad (J : \mathit{closed}\ A \to \mathsf{type}) \\
&\quad (x0 : \mathsf{forall}(a : A).\ J(\mathit{eta}\ a)) \\
&\quad (x1 : \mathsf{forall}(z : \mathit{syn}).\ J(\mathit{star}\ z))\ . \\
&\quad (\mathsf{forall}(a : A)(z : \mathit{syn}).\ (x0\ a) = (x1\ z) : \_) \xrightarrow{g} \\
&\quad J\ c
\end{aligned}
$$

modality in Istari

Istari's interative panel

$$
\begin{aligned}
&\mathit{closed} : \mathsf{type} \to \mathsf{type}\\
&\mathit{eta} : A \to \mathit{closed}\ A\\
&\mathit{star} : \mathit{syn} \to \mathit{closed}\ A\\
&\mathit{law} : \mathsf{forall}(z : \mathit{syn})(a : A).\ \mathit{eta}\ a = \mathit{star}\ z : \mathit{closed}\ A\\
&J : \mathit{closed}\ A \to \mathsf{type}\\
&c : \mathit{closed}\ A\\
&x_0 : \mathsf{forall}(a : A).\ J(\mathit{eta}\ a)\\
&x_1 : \mathsf{forall}(z : \mathit{syn}).\ J(\mathit{star}\ z)\\
&z : \mathit{syn}\\
&a : A\\
&|\!-\\
&\mathit{eta}\ a = \mathit{star}\ z : \mathit{closed}\ A\\
&1\ \mathsf{goal}(\mathsf{depth}\ 0)
\end{aligned}
$$

modality in Istari

Istari's interative panel

$$
\begin{aligned}
&\mathit{closed} : \mathsf{type} \to \mathsf{type}\\
&\mathit{eta} : A \to \mathit{closed}\ A\\
&\mathit{star} : \mathit{syn} \to \mathit{closed}\ A\\
&\mathit{law} : \mathsf{forall}(z : \mathit{syn})(a : A).\ \mathit{eta}\ a = \mathit{star}\ z : \mathit{closed}\ A\\
&J : \mathit{closed}\ A \to \mathsf{type}\\
&c : \mathit{closed}\ A\\
&x_0 : \mathsf{forall}(a : A).\ J(\mathit{eta}\ a)\\
&x_1 : \mathsf{forall}(z : \mathit{syn}).\ J(\mathit{star}\ z)\\
&z : \mathit{syn}\\
&a : A\\
&\vdash\\
&\mathit{eta}\ a = \mathit{star}\ z : \mathit{closed}\ A\\
&1\ \mathsf{goal}(\mathsf{depth}\ 0)
\end{aligned}
$$

Use either apply */law/* or auto tactics to fill in the goal.

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 $a : A$.

Mechanizing Synthetic Tait Computability

$$\mathsf{lam} : \{((a : \mathsf{tm}\, A) \to \mathsf{tm}(B\, a)) \to \mathsf{tm}(\mathsf{pi}\, A\, B) \mid \mathsf{syn} \hookrightarrow \mathsf{lam}\}$$

$$\mathsf{lam}\, f = (\mathsf{lam}\, f, f)$$

Mechanizing Synthetic Tait Computability

$$\mathsf{lam} : \{((a : \mathsf{tm}\, A) \to \mathsf{tm}(B\, a)) \to \mathsf{tm}(\mathsf{pi}\, A\, B) \mid \mathsf{syn} \hookrightarrow \mathsf{lam}\}$$
$$\mathsf{lam}\, f = (\mathsf{lam}\, f, f)$$

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().
```

Mechanizing Synthetic Tait Computability

$$\mathsf{lam} : \{\ ((a : \mathsf{tm}\ A) \to \mathsf{tm}(B\ a)) \to \mathsf{tm}(\mathsf{pi}\ A\ B)\ \mid \mathsf{syn} \hookrightarrow \mathsf{lam}\}$$

$$\mathsf{lam}\ f = (\mathsf{lam}\ f, f)$$

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().
```

Mechanizing Synthetic Tait Computability

$$\mathsf{lam} : \{((a : \mathsf{tm}\, A) \to \mathsf{tm}(B\, a)) \to \mathsf{tm}(\mathsf{pi}\, A\, B) \mid \mathsf{syn} \hookrightarrow \mathsf{lam}\}$$

$$\mathsf{lam}\, f = (\mathsf{lam}\, f, f)$$

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().
```

Mechanizing Synthetic Tait Computability

$$\mathsf{lam} : \{((a : \mathsf{tm}\, A) \to \mathsf{tm}(B\, a)) \to \mathsf{tm}(\mathsf{pi}\, A\, B) \mid \mathsf{syn} \hookrightarrow \mathsf{lam}\}$$

$$\mathsf{lam}\, f = (\ \mathsf{lam}\, f\ , f)$$

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().
```

Mechanizing Synthetic Tait Computability

$$\mathsf{lam} : \{((a : \mathsf{tm}\, A) \to \mathsf{tm}(B\, a)) \to \mathsf{tm}(\mathsf{pi}\, A\, B) \mid \mathsf{syn} \hookrightarrow \mathsf{lam}\}$$

$$\mathsf{lam}\, f = (\mathsf{lam}\, f, f)$$

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 the same as the on-paper one!

https://github.com/runmingl/istari-stc

Istari features I didn't talk about

- Quotient types
- Guarded recursive types
- Future modality
- Impredicative quantification
- . . .

Future work

All mechanized in Coq by Karl Crary

Justify phase distinction computationally in Istari's metatheory.

Future work

All mechanized in Coq by Karl Crary

Justify phase distinction computationally in Istari's metatheory.

Kripke logical relations over operational semantics

References I

Angiuli, C.

Computational Semantics of Cartesian Cubical Type Theory.

PhD thesis, Carnegie Mellon University, Sept. 2019.

Constable, R. L., Allen, S. F., Bromley, H. M., Cleaveland, W. R., Cremer, J. F., Harper, R. W., Howe, D. J., Knoblock, T. B., Mendler, N. P., Panangaden, P., Sasaki, J. T., and Smith, S. F.

Implementing Mathematics with the Nuprl Proof Development System.

Prentice-Hall, NJ, 1986.

References II

Gratzer, D.

Normalization for multimodal type theory.

In *Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science* (Haifa Israel, Aug. 2022), ACM, p. 1–13.

Kaposi, A., and Pujet, L.

Type theory in type theory using a strictified syntax.

Proc. ACM Program. Lang., ICFP (Aug. 2025).

Li, R., and Harper, R.

Canonicity for cost-aware logical framework via synthetic tait computability.

arXiv:2504.12464 [cs].

References III

Niu, Y., Sterling, J., Grodin, H., and Harper, R.

A cost-aware logical framework.

Proc. ACM Program. Lang. 6, POPL (Jan. 2022).

Rijke, E., Shulman, M., and Spitters, B.

Modalities in homotopy type theory.

Logical Methods in Computer Science Volume 16, Issue 1 (Jan 2020).

References IV

Sterling, J.

First Steps in Synthetic Tait Computability: The Objective Metatheory of Cubical Type Theory.

PhD thesis, Carnegie Mellon University, Oct. 2021.

Version 1.1, revised May 2022.

Sterling, J., and Angiuli, C.

Normalization for cubical type theory.

In *2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)* (june 2021), p. 1–15.

References V

Sterling, J., and Harper, R.

Logical relations as types: Proof-relevant parametricity for program modules.

Journal of the ACM 68, 6 (Dec. 2021), 1–47.