How to (Re)Invent Synthetic Tait Computability

PLunch
February 19, 2025

Runming Li

Metatheory for programming languages

- **Canonicity**
- Normalization
- Parametricity
- $\ldots$

Metatheory for programming languages

- **Canonicity**
- Normalization
- Parametricity
- . . .

Theorem (Canonicity)

Every closed term of type bool *is (or evaluates to) either* true *or* false.

Tait's Computability Method: Logical Relations

Tait's Computability Method: Logical Relations

- Step 1: Define a **computability** predicate $[\![A]\!]$ by induction on types.
  - $M \in [\![\mathsf{bool}]\!]$ if $M = \mathsf{yes}$ or $M = \mathsf{no}$.

Tait's Computability Method: Logical Relations

- Step 1: Define a **computability** predicate $[\![A]\!]$ by induction on types.
  - $M \in [\![\mathsf{bool}]\!]$ if $M = \mathsf{yes}$ or $M = \mathsf{no}$.
  - $M \in [\![A \to B]\!]$ if for all $N \in [\![A]\!]$, $M\,N \in [\![B]\!]$.
  - $M \in [\![A \times B]\!]$ if $\pi_1 M \in [\![A]\!]$ and $\pi_2 M \in [\![B]\!]$.

Tait's Computability Method: Logical Relations

- Step 1: Define a **computability** predicate $[\![A]\!]$ by induction on types.
  - $M \in [\![\mathsf{bool}]\!]$ if $M = \mathsf{yes}$ or $M = \mathsf{no}$.
  - $M \in [\![A \to B]\!]$ if for all $N \in [\![A]\!]$, $M\ N \in [\![B]\!]$.
  - $M \in [\![A \times B]\!]$ if $\pi_1 M \in [\![A]\!]$ and $\pi_2 M \in [\![B]\!]$.
- Step 2: Prove that all well-typed terms are **computable**.

See Bob's 15-413/713 lecture notes for more details.

Let’s do it differently today!

I will use category theory . . .

I will use category theory …

- A **category** is a collection of **objects** and **morphisms** between objects.

$$\begin{tikzcd}
A \arrow[r, "f"] \arrow[dr, "h"'] & B \arrow[d, "g"] \\
& C
\end{tikzcd}$$

I will use category theory . . .

- An **initial object** is an object that has a unique morphism to every other object in the category.

$$\begin{tikzcd}
 & A \\
0_{\mathcal{C}} \arrow[ur] \arrow[r] \arrow[dr] & B \\
 & C
\end{tikzcd}$$

I will use category theory …

- A **terminal object** is an object that has a unique morphism from every other object in the category, usually written as $1_{\mathcal{C}}$.

$$\begin{tikzcd}
A \arrow[dr] & \\
B \arrow[r] & 1_{\mathcal{C}} \\
C \arrow[ur] &
\end{tikzcd}$$

I will use category theory . . .

- A **functor** is a “function” between two categories.

I will use category theory . . .

- A **functor** is a “function” between two categories.
- A $\mathrm{Hom}(A, B)$ is the set of morphisms from object A to object B.

I will use dependent type theory …

- A Π type is a dependent function type (think “for all” quantifier):

I will use dependent type theory . . .

- A Π type is a dependent function type (think “for all” quantifier):

Example

$$\Pi_{n:\mathbb{N}}\mathsf{even}(n) + \mathsf{odd}(n)$$

read as “for all n of type $\mathbb{N}$, n is either even or odd.”

I will use dependent type theory . . .

- A Σ type is a dependent pair type (think existential quantifier):

I will use dependent type theory …

- A Σ type is a dependent pair type (think existential quantifier):

Example

$$\Sigma_{n:\mathbb{N}}(n = 42)$$

read as “there exists an n of type $\mathbb{N}$ such that $n = 42$.”

A motivating example

Theorem

Every natural number is either even or odd, i.e., a term of type $\Pi_{n:\mathbb{N}} \mathsf{even}(n) + \mathsf{odd}(n)$.

A motivating example

Theorem

Every natural number is either even or odd, i.e., a term of type $\Pi_{n:\mathbb{N}}\mathsf{even}(n) + \mathsf{odd}(n)$.

Proof

By induction on n. ☐

A categorical proof

Construct the following category $\mathcal{C}$:

- Objects: terms of type $\Sigma_{X:\mathsf{Type}}(1 + X \to X)$
  i.e., a pair of $(X, f : 1 + X \to X)$.

A categorical proof

Construct the following category $\mathcal{C}$:

- Objects: terms of type $\Sigma_{X:\mathsf{Type}}(1 + X \to X)$ *i.e.*, a pair of $(X, f : 1 + X \to X)$.
- Morphisms (between objects (X, f) and (Y, g)): a function $h : X \to Y$.

What are some of the objects in $\mathcal{C}$?

What are some of the objects in $\mathcal{C}$?

$$X : \mathsf{Type}$$
$$X = \mathbb{N}$$

$$f : 1 + X \to X$$
$$f(\mathsf{inl}\,\star) = \mathsf{zero}$$
$$f(\mathsf{inr}\,x) = \mathsf{succ}\,x$$

What are some of the objects in $\mathcal{C}$?

$$X : \mathsf{Type}$$
$$X = \mathbb{N}$$

$$f : 1 + X \to X$$
$$f(\mathsf{inl}\,\star) = \mathsf{zero}$$
$$f(\mathsf{inr}\,x) = \mathsf{succ}\,x$$

In fact, this $(\mathbb{N}, f)$ is the initial object in $\mathcal{C}$.

What are some of the objects in $\mathcal{C}$?

$$X : \mathsf{Type}$$
$$X = \mathbb{N}$$

$$f : 1 + X \to X$$
$$f(\mathsf{inl}\,\star) = \mathsf{zero}$$
$$f(\mathsf{inr}\,x) = \mathsf{succ}\,x$$

In fact, this $(\mathbb{N}, f)$ is the initial object in $\mathcal{C}$.

$$Y : \mathsf{Type}$$
$$Y = \Sigma n : \mathbb{N}(\mathsf{even}(n) + \mathsf{odd}(n))$$

$$g : 1 + Y \to Y$$
$$g(\mathsf{inl}\,\star) = (\mathsf{zero}, \mathsf{zeroIsEven})$$
$$g(\mathsf{inr}\,(n, \mathsf{inl}\,p)) =$$
$$(\mathsf{succ}\,n, \mathsf{inr}\,(\mathsf{evenOdd}(p)))$$
$$g(\mathsf{inr}\,(n, \mathsf{inr}\,p)) =$$
$$(\mathsf{succ}\,n, \mathsf{inl}\,(\mathsf{oddEven}(p)))$$

What are some of the morphisms in $\mathcal{C}$?

What are some of the morphisms in $\mathcal{C}$?

- There is a morphism from (Y, g) to (X, f):
  a function $\pi_1 : \Sigma n : \mathbb{N}(\mathsf{even}(n) + \mathsf{odd}(n)) \to \mathbb{N}$ by projecting out the first component

What are some of the morphisms in $\mathcal{C}$?

- There is a morphism from (Y, g) to (X, f): a function $\pi_1 : \Sigma n : \mathbb{N}(\mathsf{even}(n) + \mathsf{odd}(n)) \to \mathbb{N}$ by projecting out the first component
- There is a morphism from (X, f) to (Y, g): a function $\iota : \mathbb{N} \to \Sigma n : \mathbb{N}(\mathsf{even}(n) + \mathsf{odd}(n))$

What does ι look like?

$$\iota : \mathbb{N} \to \Sigma n : \mathbb{N}(\mathsf{even}(n) + \mathsf{odd}(n))$$

What does ι look like?

$$\iota : \mathbb{N} \to \Sigma n : \mathbb{N}(\mathsf{even}(n) + \mathsf{odd}(n))$$

Maybe it is the case that $\iota(n) = (n, \mathsf{proof})$?

What does ι look like?

$$\iota : \mathbb{N} \to \Sigma n : \mathbb{N}(\mathsf{even}(n) + \mathsf{odd}(n))$$

Maybe it is the case that $\iota(n) = (n, \mathsf{proof})$?

But maybe it is some random function that doesn't make sense? *e.g.*, $\iota(n) = (42, \mathsf{42\text{-}isEven})$?

Fundamental theorem

$$\iota : \mathbb{N} \to \Sigma n : \mathbb{N}(\mathsf{even}(n) + \mathsf{odd}(n))$$

$$\begin{tikzcd}
(X, f) \arrow[r, "\iota"] \arrow[dr] & (Y, g) \arrow[d, "\pi_1"] \\
& (X, f)
\end{tikzcd}$$

Fundamental theorem

$$\iota : \mathbb{N} \to \Sigma n : \mathbb{N}(\mathsf{even}(n) + \mathsf{odd}(n))$$

$$\begin{tikzcd}
(X, f) \arrow[r, "\iota"] \arrow[dr] & (Y, g) \arrow[d, "\pi_1"] \\
& (X, f)
\end{tikzcd}$$

It must be the case that $\pi_1 \circ \iota = \mathsf{id}_{\mathbb{N}}$.

Fundamental theorem

$$\iota : \mathbb{N} \to \Sigma n : \mathbb{N}(\mathsf{even}(n) + \mathsf{odd}(n))$$

$$\begin{tikzcd}
(X, f) \arrow[r, "\iota"] \arrow[dr] & (Y, g) \arrow[d, "\pi_1"] \\
& (X, f)
\end{tikzcd}$$

It must be the case that $\pi_1 \circ \iota = \mathsf{id}_{\mathbb{N}}$.

It must be the case that $\iota(n) = (n, \mathsf{proof})$.

Our proof of **canonicity** would look much like this!

Simply-typed lambda calculus

We present an **equational theory** of simply-typed lambda calculus with only booleans and functions as a signature $\mathbb{SIG}$.

Simply-typed lambda calculus

We present an **equational theory** of simply-typed lambda calculus with only booleans and functions as a signature $\mathbb{SIG}$.

```
record 𝕊𝕀𝔾 where
  field
    tp : Type
    tm : tp → Type

    bool : tp
    yes : tm bool
    no : tm bool
```

Simply-typed lambda calculus

We present an **equational theory** of simply-typed lambda calculus with only booleans and functions as a signature $\mathbb{SIG}$.

record $\mathbb{SIG}$ where

field

$\mathsf{tp} : \textbf{Type}$

$\mathsf{tm} : \mathsf{tp} \to \textbf{Type}$

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

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

$\mathsf{no} : \mathsf{tm}\ \mathsf{bool}$

$\mathsf{arr} : \mathsf{tp} \to \mathsf{tp} \to \mathsf{tp}$

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

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

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

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

Example terms

Example

$$\mathsf{lam}\ (\lambda x.x) : \mathsf{tm}\ (\mathsf{arr}\ \mathsf{bool}\ \mathsf{bool})$$

which is traditionally written as $\lambda(x : \mathsf{bool}).x : \mathsf{bool} \to \mathsf{bool}$.

Example terms

Example

$$\mathsf{lam}\ (\lambda x.x) : \mathsf{tm}\ (\mathsf{arr}\ \mathsf{bool}\ \mathsf{bool})$$

which is traditionally written as $\lambda(x : \mathsf{bool}).x : \mathsf{bool} \to \mathsf{bool}$.

$$\mathsf{app}\ (\mathsf{lam}\ \lambda\_.\mathsf{yes})\ \mathsf{no} : \mathsf{tm}\ \mathsf{bool}$$

which is traditionally written as $(\lambda\_.\mathsf{yes})\ \mathsf{no} : \mathsf{bool}$.

Example terms

Example

$$\mathsf{lam}\ (\lambda x.x) : \mathsf{tm}\ (\mathsf{arr}\ \mathsf{bool}\ \mathsf{bool})$$

which is traditionally written as $\lambda(x : \mathsf{bool}).x : \mathsf{bool} \to \mathsf{bool}$.

$$\mathsf{app}\ (\mathsf{lam}\ \lambda\_.\mathsf{yes})\ \mathsf{no} : \mathsf{tm}\ \mathsf{bool}$$

which is traditionally written as $(\lambda\_.\mathsf{yes})\ \mathsf{no} : \mathsf{bool}$.
By using arr_β, we can show that the above term is equal to yes.

$\mathbb{SIG}$ induces a category $\mathcal{C}$

$$\begin{tikzcd}
 & \mathsf{tm}(\mathsf{bool}) \arrow[r] \arrow[d] \arrow[dr, phantom, "\lrcorner", very near start] & \Sigma_{A:\mathsf{tp}}\mathsf{tm}(A) \arrow[d, "\pi_1"'] & \\
\mathsf{tm}(\mathsf{arr}\ \mathsf{bool}\ \mathsf{bool}) \arrow[ur, "\mathsf{app}(-,\mathsf{no})"] & 1 \arrow[l, "\mathsf{lam}(\lambda\_.\mathsf{yes})"'] \arrow[u, bend left=45, "\mathsf{yes}"] \arrow[u, bend left=20, "\mathsf{no}"] \arrow[r, "\mathsf{bool}"] \arrow[dr, "\mathsf{arr}"] & \mathsf{tp} \arrow[r, "\mathsf{arr}"] & \mathsf{tp} \to \mathsf{tp} \\
 & & \mathsf{tp} \to \mathsf{tp} \to \mathsf{tp} &
\end{tikzcd}$$

Some special morphisms

- Closed terms of type A are morphisms from $\mathsf{1}$ to $\mathsf{tm}(A)$.

$$\mathsf{yes} : \mathsf{1} \to \mathsf{tm}(\mathsf{bool})$$

$$\mathsf{no} : \mathsf{1} \to \mathsf{tm}(\mathsf{bool})$$

$$\mathsf{app}(-, \mathsf{no}) \circ \mathsf{lam}(\lambda\_.\mathsf{yes}) : \mathsf{1} \to \mathsf{tm}(\mathsf{bool})$$

Some special morphisms

- Closed terms of type A are morphisms from 1 to $\mathsf{tm}(A)$.

$$\begin{aligned}&\mathsf{yes} : 1 \to \mathsf{tm}(\mathsf{bool})\\&\mathsf{no} : 1 \to \mathsf{tm}(\mathsf{bool})\\&\mathsf{app}(-, \mathsf{no}) \circ \mathsf{lam}(\lambda\_.\mathsf{yes}) : 1 \to \mathsf{tm}(\mathsf{bool})\end{aligned}$$

Theorem (Canonicity)

For any morphism $b : 1 \to \mathsf{tm}(\mathsf{bool})$, it must be the case that $b = \mathsf{yes}$ or $b = \mathsf{no}$.

Category of Computability Structures

Construct a category $\mathcal{E}$ as follows:

- Objects: computability structures $(A \in \mathcal{C}, S \in \mathbf{Set}, f : S \to \mathsf{Hom}_{\mathcal{C}}(1, A))$.

Category of Computability Structures

Construct a category $\mathcal{E}$ as follows:

- Objects: computability structures $(A \in \mathcal{C}, S \in \mathbf{Set}, f : S \to \mathsf{Hom}_{\mathcal{C}}(1, A))$. Think as: for each morphism $e : 1 \to A$ in $\mathcal{C}$, we have a set S_e of proofs that e is computable at type A.

Category of Computability Structures

Construct a category $\mathcal{E}$ as follows:

- Objects: computability structures $(A \in \mathcal{C}, S \in \mathbf{Set}, f : S \to \mathrm{Hom}_{\mathcal{C}}(1, A))$.
  Think as: for each morphism $e : 1 \to A$ in $\mathcal{C}$, we have a set S_e of proofs that e is computable at type A.
- Morphisms: a morphism $b : A \to A'$ and a function $h : S \to S'$ such that:

$$\begin{tikzcd}
S \arrow[r, "h"] \arrow[d, "f"'] & S' \arrow[d, "f'"] \\
\mathrm{Hom}(1, A) \arrow[r, "\mathrm{Hom}(1{,}b)"] & \mathrm{Hom}(1, A')
\end{tikzcd}$$

What are some of the objects in $\mathcal{E}$?

- $\mathsf{tm}(\mathsf{bool}) = (\mathsf{tm}(\mathsf{bool}), \{\spadesuit, \clubsuit\}, f)$ where:
  $f(\spadesuit) = \mathsf{yes}$
  $f(\clubsuit) = \mathsf{no}$

What are some of the objects in $\mathcal{E}$?

- $\mathsf{tm}(\mathsf{bool}) = (\mathsf{tm}(\mathsf{bool}), \{\spadesuit, \clubsuit\}, f)$ where:
  $f(\spadesuit) = \mathsf{yes}$
  $f(\clubsuit) = \mathsf{no}$

Define a functor $F : \mathcal{C} \to \mathcal{E}$ such that (in particular)
$F(\mathsf{tm}(\mathsf{bool})) = \mathsf{tm}(\mathsf{bool})$.

Fundamental Theorem of Logical Relations

- Objects in $\mathcal{C}$: A.
- Objects in $\mathcal{E}$: $A = (A, S, f)$.

Fundamental Theorem of Logical Relations

- Objects in $\mathcal{C}$: A.
- Objects in $\mathcal{E}$: $A = (A, S, f)$.

$$\begin{tikzcd}
\mathcal{C} \arrow[r, "F"] \arrow[dr, "\mathsf{Id}_{\mathcal{C}}"'] & \mathcal{E} \arrow[d, "\pi_1"] \\
& \mathcal{C}
\end{tikzcd}$$

By construction, it must be the case that $\pi_1 \circ F = \mathsf{Id}_{\mathcal{C}}$.

Canonicity

Proof

Suppose we have a morphism $b : \mathtt{1} \to \mathsf{tm}(\mathsf{bool})$ in $\mathcal{C}$.

Canonicity

Proof

Suppose we have a morphism $b : 1 \to \mathsf{tm}(\mathsf{bool})$ in $\mathcal{C}$.
Compute $F(b) : 1 \to \mathsf{tm}(\mathsf{bool})$.

Canonicity

Proof

Suppose we have a morphism $b : \mathbf{1} \to \mathsf{tm}(\mathsf{bool})$ in $\mathcal{C}$.
Compute $F(b) : \mathbf{1} \to \mathsf{tm}(\mathsf{bool})$.
This means that $F(b)$ consists of a morphism $b' : \mathbf{1} \to \mathsf{tm}(\mathsf{bool})$ and a function $h : \mathbf{1}_{\mathbf{Set}} \to \{\spadesuit, \clubsuit\}$ such that:

$$\begin{tikzcd}
\mathbf{1}_{\mathbf{Set}} \arrow[r, "h"] \arrow[d, "!"'] & \{\spadesuit, \clubsuit\} \arrow[d, "f"] \\
\mathbf{1}_{\mathbf{Set}} = \mathsf{Hom}(\mathbf{1}, \mathbf{1}) \arrow[r, "b'"] & \mathsf{Hom}(\mathbf{1}, \mathsf{tm}(\mathsf{bool}))
\end{tikzcd}$$

□

Canonicity

Proof (Cont.)

Moreover, $\pi_1(F(b)) = b'$. By the fundamental theorem, $b' = b$.

Canonicity

Proof (Cont.)

Moreover, $\pi_1(F(b)) = b'$. By the fundamental theorem, $b' = b$.

$$\begin{tikzcd}
1_{\mathbf{Set}} \arrow[r, "h"] \arrow[d, "!"'] & \{\spadesuit, \clubsuit\} \arrow[d, "f"] \\
1_{\mathbf{Set}} = \mathsf{Hom}(1, 1) \arrow[r, "b"] & \mathsf{Hom}(1, \mathsf{tm}(\mathsf{bool}))
\end{tikzcd}$$

Canonicity

Proof (Cont.)

Moreover, $\pi_1(F(b)) = b'$. By the fundamental theorem, $b' = b$.

$$\begin{tikzcd}
1_{\mathbf{Set}} \arrow[r, "h"] \arrow[d, "!"'] & \{\spadesuit, \clubsuit\} \arrow[d, "f"] \\
1_{\mathbf{Set}} = \mathsf{Hom}(1, 1) \arrow[r, "b"] & \mathsf{Hom}(1, \mathsf{tm}(\mathsf{bool}))
\end{tikzcd}$$

If $h = \spadesuit$, then $b = \mathsf{yes}$ (because $f(\spadesuit) = \mathsf{yes}$).
If $h = \clubsuit$, then $b = \mathsf{no}$ (because $f(\clubsuit) = \mathsf{no}$). ☐

What is this proof?

What is this proof?

- A construction of a computability structure category $\mathcal{E}$ by gluing syntax and semantics.

What is this proof?

- A construction of a computability structure category $\mathcal{E}$ by gluing syntax and semantics.
  - Follows a general construction of **Artin Gluing**.

What is this proof?

- A construction of a computability structure category $\mathcal{E}$ by gluing syntax and semantics.
    - Follows a general construction of **Artin Gluing**.
- A construction of a functor $F : \mathcal{C} \to \mathcal{E}$.

What is this proof?

- A construction of a computability structure category $\mathcal{E}$ by gluing syntax and semantics.
  - Follows a general construction of **Artin Gluing**.
- A construction of a functor $F : \mathcal{C} \to \mathcal{E}$.
  - Tedious! A lot of conditions to check.
  - F is a **functorial model** of the language.

What is a model?

What is a model?

Any implementation $\mathsf{M} : \mathbb{SIG}$ is a **model** of the language!

What is a model?

Any implementation $\mathsf{M} : \mathbb{SIG}$ is a **model** of the language!

$$\begin{aligned}
\mathsf{M.tp} &= \mathbf{Type}\\
\mathsf{M.tm}(A) &= A
\end{aligned}$$

$$\begin{aligned}
\mathsf{M.bool} &= 1 + 1\\
\mathsf{M.yes} &= \mathsf{inl}\,\star\\
\mathsf{M.no} &= \mathsf{inr}\,\star
\end{aligned}$$

$$\begin{aligned}
\mathsf{M.arr}\,A\,B &= A \to B\\
\mathsf{M.lam}\,f &= f\\
\mathsf{M.app}\,f\,x &= f\,x
\end{aligned}$$

Our Plan

- Suppose we have a model $\mathsf{M} : \mathbb{SIG}$ that corresponds to the syntax.
- Construct a model $\mathsf{M} : \mathbb{SIG}$ that corresponds to the gluing of syntax and semantics, such that

Our Plan

- Suppose we have a model $\mathsf{M} : \mathbb{SIG}$ that corresponds to the syntax.
- Construct a model $\mathsf{M} : \mathbb{SIG}$ that corresponds to the gluing of syntax and semantics, such that

$$\begin{tikzcd}
\mathsf{M} \arrow[r, "\mathsf{bluify}"] \arrow[dr, "\mathsf{id}"'] & \mathsf{M} \arrow[d] \\
& \mathsf{M}
\end{tikzcd}$$

Piecing things together

$$\begin{tikzcd}
 & & \mathcal{C} \arrow[r, "F"] \arrow[dr, "\mathrm{Id}_{\mathcal{C}}"'] & \mathcal{E} \arrow[d, "\pi_1"] \\
{\color{red}\mathsf{M}} \arrow[r, "{\color{red}\mathsf{blu}}{\color{blue}\mathsf{ify}}"] \arrow[urr, dotted] \arrow[dr, "\mathsf{id}"'] & {\color{blue}\mathsf{M}} \arrow[d] \arrow[urr, dotted] & & \mathcal{C} \\
 & {\color{red}\mathsf{M}} \arrow[urr, dotted] & &
\end{tikzcd}$$

Some machinary in the dependent type theory

- A proposition $\mathsf{syn} : \mathsf{Prop}$.
  - If syn holds, then we say that we are in the **syntactic** **phase**.

Some machinary in the dependent type theory

- A proposition $\mathsf{syn} : \mathsf{Prop}$.
  - If syn holds, then we say that we are in the **syntactic phase**.
- Extension type: $\{A \mid \mathsf{syn} \hookrightarrow a_\circ\}$ where $a_\circ : A$.
  - A term $a : \{A \mid \mathsf{syn} \hookrightarrow a_\circ\}$ is a term $a : A$ such that under the **syntactic phase** $a = a_\circ$.

New goal

Construct $\mathsf{M} : \{\mathbb{SIG} \mid \mathsf{syn} \hookrightarrow \mathsf{M}\}$.

$$\begin{tikzcd}
\mathsf{M} \arrow[r, "\text{bluify}"] \arrow[dr, "\text{id}"'] & \mathsf{M} : \{\mathbb{SIG} \mid \mathsf{syn} \hookrightarrow \mathsf{M}\} \arrow[d, "\text{enter syntactic phase}"] \\
 & \mathsf{M}
\end{tikzcd}$$

Constructing M

$$\mathsf{M.tp} : \{\mathbf{Type} \mid \mathsf{syn} \hookrightarrow \mathsf{M.tp}\}$$

$$\mathsf{M.tm} : \{\mathsf{M.tp} \to \mathbf{Type} \mid \mathsf{syn} \hookrightarrow \mathsf{M.tm}\}$$

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

$$\mathsf{M.yes} : \{\mathsf{M.tm}(\mathsf{M.bool}) \mid \mathsf{syn} \hookrightarrow \mathsf{M.yes}\}$$

$$\mathsf{M.no} : \{\mathsf{M.tm}(\mathsf{M.bool}) \mid \mathsf{syn} \hookrightarrow \mathsf{M.no}\}$$

$$\ldots$$

Constructing M

$$\mathsf{M.tp} : \{\mathbf{Type} \mid \mathsf{syn} \hookrightarrow \mathsf{M.tp}\}$$

$$\mathsf{M.tp} = \Sigma_{A:\mathsf{M.tp}}\{\mathbf{Type} \mid \mathsf{syn} \hookrightarrow \mathsf{M.tm}(A)\}$$

Constructing M

$$\mathsf{M.tp} : \{\mathbf{Type} \mid \mathsf{syn} \hookrightarrow \mathsf{M.tp}\}$$

$$\mathsf{M.tp} = \Sigma_{A:\mathsf{M.tp}}\{\mathbf{Type} \mid \mathsf{syn} \hookrightarrow \mathsf{M.tm}(A)\}$$

Think as: the computability structure of $\mathsf{M.tp}$ is for each syntactic type A, a collection of terms of that type and proofs that those terms are computable.

Constructing M

$$\mathsf{M.tp} : \{\mathbf{Type} \mid \mathsf{syn} \hookrightarrow \mathsf{M.tp}\}$$
$$\mathsf{M.tp} = \Sigma_{A:\mathsf{M.tp}}\{\mathbf{Type} \mid \mathsf{syn} \hookrightarrow \mathsf{M.tm}(A)\}$$

Think as: the computability structure of $\mathsf{M.tp}$ is for each syntactic type A, a collection of terms of that type and proofs that those terms are computable.

Check: under the syntactic phase (assuming syn),

$$\begin{aligned} &\mathsf{M.tp} \\ =&\Sigma_{A:\mathsf{M.tp}}\{\mathbf{Type} \mid \mathsf{syn} \hookrightarrow \mathsf{M.tm}(A)\} \\ \cong&\Sigma_{A:\mathsf{M.tp}}\mathbf{1} \\ \cong&\mathsf{M.tp} \end{aligned}$$

Constructing M

$$\mathsf{M.tp} : \{\mathbf{Type} \mid \mathsf{syn} \hookrightarrow \mathsf{M.tp}\}$$
$$\mathsf{M.tp} = \Sigma_{A:\mathsf{M.tp}}\{\mathbf{Type} \mid \mathsf{syn} \hookrightarrow \mathsf{M.tm}(A)\}$$

$$\mathsf{M.tm} : \{\mathsf{M.tp} \to \mathbf{Type} \mid \mathsf{syn} \hookrightarrow \mathsf{M.tm}\}$$
$$\mathsf{M.tm}(A) = \pi_2 A$$

Constructing M

$$\mathsf{M.tp} : \{\mathbf{Type} \mid \mathsf{syn} \hookrightarrow \mathsf{M.tp}\}$$

$$\mathsf{M.tp} = \Sigma_{A:\mathsf{M.tp}}\{\mathbf{Type} \mid \mathsf{syn} \hookrightarrow \mathsf{M.tm}(A)\}$$

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

$$\mathsf{M.bool} = (\mathsf{M.bool}, \Sigma_{b:\mathsf{tm}(\mathsf{bool})}(b = \mathsf{M.yes}) + (b = \mathsf{M.no}))^1$$

$$\mathsf{M.yes} : \{\mathsf{M.tm}(\mathsf{M.bool}) \mid \mathsf{syn} \hookrightarrow \mathsf{M.yes}\}$$

$$\mathsf{M.yes} = (\mathsf{M.yes}, \mathsf{inl}(\checkmark))$$

---

[1] Well, I lied slightly.

Constructing M

Everything else is just a routine programming exercise in a dependently typed language.

In almost all cases, there is only one way that makes the type-checker happy.

Constructing M

Everything else is just a routine programming exercise in a dependently typed language.

In almost all cases, there is only one way that makes the type-checker happy.

Just like in traditional Logical Relations, there is no creativity beyond the base types.

And that is Synthetic Tait's Computability!

What is this proof?

What is this proof?

- A programming exercise to construct $\textcolor{blue}{\mathsf{M}} : \{\mathbb{SIG} \mid \mathsf{syn} \hookrightarrow \textcolor{red}{\mathsf{M}}\}$ in a dependently typed language.
- Everything else can be black-boxed if you don't want to deal with category theory.

$$\begin{tikzcd}
 & & \mathcal{C} \arrow[r, "F"] \arrow[ddr, "\mathsf{Id}_{\mathcal{C}}"'] & \mathcal{E} \arrow[dd, "\pi_1"] \\
\textcolor{red}{\mathsf{M}} \arrow[r, "\textcolor{red}{\mathsf{blu}}\textcolor{blue}{\mathsf{ify}}"] \arrow[ddr, "\mathsf{id}"'] \arrow[urr, dotted] & \textcolor{blue}{\mathsf{M}} \arrow[dd] \arrow[urr, dotted] & & \\
 & & & \mathcal{C} \\
 & \textcolor{red}{\mathsf{M}} \arrow[urr, dotted] & &
\end{tikzcd}$$

Synthetic Tait Computability in Real World

Parametricity for an ML module calculus	Sterling & Harper
Normalization for Cartesian Cubical Type Theory	Sterling & Angiuli
Normalization for a multimodal type theory	Gratzer
⋮	

Synthetic Tait Computability in Real World

Parametricity for an ML module calculus	Sterling & Harper
Normalization for Cartesian Cubical Type Theory	Sterling & Angiuli
Normalization for a multimodal type theory	Gratzer
⋮	
Canonicity for Cost-Aware Logical Framework	☺

Conclusion

- Syntax and semantics of a programming language displays a **phase distinction** that can be manipulated synthetically.
- **Synthetic Tait Computability** exploits this by **gluing** syntax and semantics together.
- Proving meta-theoretic properties by Logical Relations can be reduced to a programming exercise in a dependently typed language.