Objective Metatheory of (Cubical) Type Theory

Jonathan Sterling

August 31, 2020
The implementation and semantics of dependent type theories can be studied in a syntax-independent way. Using the semantic techniques of the objective metatheory, type theorists can obtain succinct and conceptual proofs of formerly intractable results.
Who is a type theorist?
Who is a type theorist?

Someone who studies the *non-type-theoretic aspects* of type theory.
Who is a type theorist?

Someone who studies the *non-type-theoretic aspects* of type theory.
**Definition.** A statement is called *type theoretic* when it is preserved by morphisms of models of type theory: these hold in all models iff they hold in the syntax of type theory.
Definition. A statement is called type theoretic when it is preserved by morphisms of models of type theory: these hold in all models iff they hold in the syntax of type theory.

Examples. Anything expressible as a judgment of type theory; for instance, “There is a function that computes the gcd.”
**Definition.** A statement is called *type theoretic* when it is preserved by morphisms of models of type theory: these hold in all models iff they hold in the syntax of type theory.

**Examples.** Anything expressible as a judgment of type theory; for instance, “There is a function that computes the gcd.”

**Non-examples.** “Every closed term of type $\mathbb{N}$ is equal to a numeral”;
**Definition.** A statement is called *type theoretic* when it is preserved by morphisms of models of type theory: these hold in all models iff they hold in the syntax of type theory.

**Examples.** Anything expressible as a judgment of type theory; for instance, “There is a function that computes the gcd.”

**Non-examples.** “Every closed term of type \( \mathbb{N} \) is equal to a numeral”; “These two queue implementations are observationally equivalent”;...
**Definition.** A statement is called *type theoretic* when it is preserved by morphisms of models of type theory: these hold in all models iff they hold in the syntax of type theory.

**Examples.** Anything expressible as a judgment of type theory; for instance, “There is a function that computes the gcd.”

**Non-examples.** “Every closed term of type $\mathbb{N}$ is equal to a numeral”; “These two queue implementations are observationally equivalent”; “It is decidable whether two types are judgmentally equal.”
**DERIVABILITY**

About the syntax of type theory. *But, by soundness, also about any model of type theory.*

**DEDUCTION**

\[ \Gamma \vdash A \equiv A' \quad \text{type} \quad \Gamma \vdash B \equiv B' \quad \text{type} \]

\[ \Gamma \vdash (A \rightarrow B) \equiv (A' \rightarrow B') \quad \text{type} \]

A statement is called *type theoretic* when it is preserved by morphisms of models of type theory: these hold in all models iff they hold in the syntax of type theory.

Examples:
- Anything expressible as a judgment of type theory; for instance, “There is a function that computes the gcd.”
- A closed term of type \( \mathbb{N} \) is equal to another; it is decidable whether two queue implementations are observationally equivalent”; “It is decidable whether two types are judgmentally equal.”
Who is a type theorist?
Someone who studies the non-type-theoretic aspects of type theory.

**Definition.** A statement is called type theoretic when it is preserved by morphisms of models of type theory: these hold in all models iff they hold in the syntax of type theory.

**Examples.** Anything expressible as a judgment of type theory; for instance, "There is a function that computes the gcd."

**Non-examples.** "Every closed term of type $\mathbb{N}$ is equal to a numeral"; "These two queue implementations are observationally equivalent"; "It is decidable whether two types are judgmentally equal."

**DERIVABILITY**
About the syntax of type theory.
But, by soundness, also about any model of type theory.

**DEDUCTION**
$$
\Gamma \vdash A \equiv A' \text{ type} \quad \Gamma \vdash B \equiv B' \text{ type}
$$
$$
\Gamma \vdash (A \to B) \equiv (A' \to B') \text{ type}
$$

**ADMISSIBILITY**
About the syntax of type theory. (Only the syntax of type theory!)

**INVERSION**
$$
\Gamma \vdash (A \to B) \equiv (A' \to B') \text{ type}
$$
$$
\Gamma \vdash A \equiv A' \text{ type} \quad \Gamma \vdash B \equiv B' \text{ type}
$$
DERIVABILITY

About the syntax of type theory. But, by soundness, also about any model of type theory.

DEDUCTION

\[
\Gamma \vdash A \equiv A' \text{ type} \quad \Gamma \vdash B \equiv B' \text{ type}
\]

\[
\Gamma \vdash (A \to B) \equiv (A' \to B') \text{ type}
\]

ADMISSIBILITY

About the syntax of type theory. (Only the syntax of type theory!)

INVERSION

\[
\Gamma \vdash (A \to B) \equiv (A' \to B') \text{ type}
\]

\[
\Gamma \vdash A \equiv A' \text{ type} \quad \Gamma \vdash B \equiv B' \text{ type}
\]

Why do we care?

Admissibilities like this are why it is even possible to implement type checkers!
Most metatheorems important for implementation are consequences of *normalization*.

Even for basic type theory, normalization is hard to prove rigorously: 100-200 pages(*) of single-use technical lemmas that seem to have nothing to do with the matter at hand.
Someone should stop us
from making new type theories

Wouldn’t be so bad if we only had to do it once; but type theory is in its infancy and we keep making better ones.
Someone should stop us from making new type theories

Wouldn’t be so bad if we only had to do it once; but type theory is in its infancy and we keep making better ones. **Today: cubical type theory.**
Why not stick with ITT, ETT, or Nuprl?
Why not stick with ITT, ETT, or Nuprl?

**Function extensionality** and **quotient types** either break computation or they break type checking.
Why not stick with ITT, ETT, or Nuprl?

**Function extensionality** and **quotient types** either break computation or they break type checking.

Worse: well-behaved notions of **equivalence relation** and **quotient type** are inconsistent with standard(†) PER semantics of type theory. *Non-starter for mathematical applications!*
Why not stick with ITT, ETT, or Nuprl?

Function extensionality and quotient types either break computation or they break type checking.

Worse: well-behaved notions of equivalence relation and quotient type are inconsistent with standard PER semantics of type theory. Non-starter for mathematical applications!

Universes don’t have universal properties (unlike every other type connective).
Why not stick with ITT, ETT, or Nuprl?

Function extensionality and quotient types either break computation or they break type checking.

Worse: well-behaved notions of equivalence relation and quotient type are inconsistent with standard(+) PER semantics of type theory. All these are solved by cubical type theory (*).

Universes don’t have universal properties (unlike every other type connective).

(*) Angiuli, Hou (Favonia), and Harper [AHH17], Angiuli, Brunerie, Coquand, Hou (Favonia), Harper, and Licata [Ang+17], Awodey [Awo18a], Cohen, Coquand, Huber, and Mörtberg [Coh+17], Huber [Hub18], and Orton and Pitts [OP16], ...
Implementations of cubical type theory!

Several variants of cubical type theory have been implemented.

<table>
<thead>
<tr>
<th>Gen.</th>
<th>Style</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>evaluator</td>
<td>cubical</td>
</tr>
<tr>
<td>1</td>
<td>typechecker PRL</td>
<td>cubicaltt, yacctl, RedPRL</td>
</tr>
<tr>
<td>2</td>
<td>proof assistant</td>
<td>Cubical Agda, redtt, cooltt</td>
</tr>
</tbody>
</table>

[Ang+18b; Coh+15; Coh+18; MA18; Red18; Red20; VMA19]

Ours: RedPRL, redtt, cooltt.
Our implementations: **RedPRL**, **redtt**, **cooltt**

Each implementation was tied to a scientific experiment!

<table>
<thead>
<tr>
<th>Premise</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RedPRL</strong></td>
<td>The PRL methodology benefits HTT implementation.</td>
</tr>
<tr>
<td><strong>redtt</strong></td>
<td>Interactive cubical refinement + decidable(?) jdg.eq. increases usability.</td>
</tr>
<tr>
<td><strong>cooltt</strong></td>
<td>LF formulation + more extensional equality on $\mathbb{F}$ &amp; systems suitable for efficient implementation.</td>
</tr>
</tbody>
</table>

**My contributions:** generalization of interactive proof with holes to account for cubical boundaries; more efficient algorithms for cubical evaluation.
What is cubical type theory?

An extension of Martin-Löf type theory!

1. Interval object $\mathbb{I}$, classifying “dimensions”:

$$
\begin{array}{c}
0 : \mathbb{I} \\
1 : \mathbb{I} \\
\end{array}
\quad J
$$

**Idea:** A term $a : \prod_{i : \mathbb{I}} A(i)$ is an *identification* between $a(0)$ and $a(1)$.

2. Universe $\mathbb{F}$ of propositions closed under at least:

- extensional equality ($r =_\mathbb{I} s$) of dimensions $r, s : \mathbb{I}$
- conjunction $\phi \land \psi$ and (extensional) disjunction $\phi \lor \psi$
- universal quantification over the interval $\forall i : \mathbb{I}. \phi(i)$

**Idea:** A term $a : \phi \rightarrow A$ is a *partial element* of $A$, defined only when $\phi$ is true.
New computations! :-)  

Cubical type theory extends MLTT with new generic functions

\[
A : \mathbb{I} \to U \quad \phi : F \quad r, s : \mathbb{I} \quad f : \prod_{i : \mathbb{I}} \prod_{p : (i = r) \lor \phi A(i)}
\]

\[
\text{com}^r_s A(f) : \{\tilde{f}_s : A(s) \mid \forall p. \tilde{f}_s = f(s, p)\}
\]

Coercion/transport, symmetry, and transitivity are all special cases of \(\text{com}_A\).

Resulting theory of equality much easier to use than ITT+J!
New computations! :-(

New computation rules branch on *non-equality* of $r, s$ as well as the value of $\Lambda(i)$ for a fresh dimension $i : \mathbb{I}$. 
New computations! :-(

New computation rules branch on non-equality of $r, s$ as well as the the value of $A(i)$ for a fresh dimension $i : \mathbb{I}$.

- Neither $M \text{ val}$ nor $M \mapsto^* N$ closed under substitution!
New computations! :-(

New computation rules branch on \textit{non-equality} of \(r, s\) as well as the the value of \(A(i)\) for a fresh dimension \(i : \mathbb{I}\).

- Neither \(M\, \text{val} \) nor \(M \rightarrow^* N\) closed under substitution!
- Classic head expansion principle for operational model doesn’t make sense, needs to be generalized significantly.
New computations! :-(

New computation rules branch on *non-equality* of \( r, s \) as well as the value of \( A(i) \) for a fresh dimension \( i : \mathbb{I} \).

- Neither \( M \text{ val} \) nor \( M \mapsto^* N \) closed under substitution!
- Classic head expansion principle for operational model doesn’t make sense, needs to be generalized significantly.

\( \implies \) Operational metatheory very hard: possible, with great exertion, using *coherent expansion* principle contributed independently by Angiuli [Ang19] and Huber [Hub18].
New computations! :-(

New computation rules branch on *non-equality* of $r, s$ as well as the value of $A(i)$ for a fresh dimension $i : \mathbb{I}$.

- Neither $M \text{val}$ nor $M \rightarrow^\ast N$ closed under substitution!
- Classic head expansion principle for operational model doesn’t make sense, needs to be generalized significantly.

⇒ Operational metatheory very hard: possible, with great exertion, using *coherent expansion* principle contributed independently by Angiuli [Ang19] and Huber [Hub18].

**Canonicity at the limits of operational tractability:** normalization will require new techniques (this thesis!).
New computations! (-: 

Simpler alternative to operational semantics + PERs with coherent expansion: **equational theory + Artin gluing**, as proposed by Awodey in 2015.


S., Angiuli, and Gratzer [SAG19] present an easy and complete proof of canonicity for a version of cubical type theory in less than 30 pages. *Trial run of the “objective metatheory.”*
What is the objective metatheory?

objective metatheory = local invariance + global invariance + proof relevance.
What is the objective metatheory?

objective metatheory = local invariance + global invariance + proof relevance.

1. local invariance:
   - raw syntax / op-sem $\implies$ typed syntax in equational LF
   - work invariantly over equiv. classes of typed terms
What is the objective metatheory?

objective metatheory = local invariance + global invariance + proof relevance.

1. **local invariance:**
   - raw syntax / op-sem $\implies$ typed syntax in equational LF
   - work invariantly over equiv. classes of typed terms

2. **global invariance:**
   - “objective syntax” defined only up to categorical equivalence
   - all statements respect weak equivalence of categories: freedom to choose presentations at will
What is the objective metatheory?

objective metatheory = local invariance + global invariance + proof relevance.

1. **local invariance:**
   - raw syntax / op-sem $\Rightarrow$ typed syntax in equational LF
   - work invariantly over equiv. classes of typed terms

2. **global invariance:**
   - “objective syntax” defined only up to categorical equivalence
   - all statements respect weak equivalence of categories: freedom to choose presentations at will

3. **proof relevance:**
   - “property of raw syntax” $\Rightarrow$ “structure over objective syntax”
   - generalization proof-relevant logical relations is forced!
Local invariance of objective syntax

Syntax of type theory with function types expressed as an equational LF signature:\(^1\)

\[\begin{align*}
Tp &: \text{Kind} \\
Tm &: Tp \to \text{Type} \\
Fn &: Tp \times Tp \to Tp \\
\alpha_{Fn} &: \prod_{A,B:Tp} Tm(Fn(A,B)) \cong (Tm(A) \to Tm(B))
\end{align*}\]

Above: introduction, elimination, computation, and uniqueness rules bundled in \(\alpha_{Fn}\).

**Local invariance:** impossible to utter a distinction between judgmentally equal terms. (Anti-bureaucratic power move!)

---

\(^1\)Equational LF due to Uemura [Uem19] with universes \(\text{Type} \subseteq \text{Kind}\); kinds closed under dependent products along types as in Harper, Honsell, and Plotkin [HHP93].
Global invariance of objective syntax

\[ T_p : \text{Kind} \quad T_m : T_p \to \text{Type} \quad F_n : T_p \times T_p \to T_p \]
\[ \alpha_{F_n} : \prod_{A,B:T_p} T_m(F_n(A, B)) \cong (T_m(A) \to T_m(B)) \]

The signature \( \Sigma \) above involves a choice of LF encoding, but metatheorems don’t depend on how we set up the function type (which is uniquely determined up to iso).

**Global invariance:** \( \Sigma \) presents a classifying category \( \mathcal{C}_\Sigma \) of judgments and deductions, which we work with up to weak equivalence of categories.
Proof relevance in the objective metatheory

Let’s remember how logical relations work...
(Unary) logical relations on closed terms is: for each sort $A$ sort$_\Sigma$, a subset $\bar{A} \subseteq \{ a \mid \vdash_\Sigma a : A \}/ \equiv_\Sigma$ respecting all the operations of $\Sigma$. 
(Unary) logical relations on closed terms is: for each sort $A : C_\Sigma$, a subset $\tilde{A} \subseteq \text{Hom}_{C_\Sigma}(\cdot, A)$ respecting all the operations of $\Sigma$. 
(Unary) logical relations on closed terms is: for each sort $A : C_\Sigma$, a subset $\tilde{A} \subseteq \rho(A)$ respecting all the operations of $\Sigma$, where $\rho : C_\Sigma \to \text{Set}$ is the global sections functor.
(Unary) logical relations on closed terms is: for each sort $A : C_\Sigma$, a subset $\tilde{A} \subseteq \rho(A)$ respecting all the operations of $\Sigma$, where $\rho : C_\Sigma \to \text{Set}$ is the global sections functor.

Elements $a \in \tilde{A}$ are called computable.
(Unary) logical relations on closed terms is: for each sort $A : C_\Sigma$, a subset $\tilde{A} \subseteq \rho(A)$ respecting all the operations of $\Sigma$, where $\rho : C_\Sigma \rightarrow \text{Set}$ is the global sections functor.

Elements $a \in \tilde{A}$ are called computable.

Logical relations $(A, \tilde{A} \subseteq \rho(A))$ have the structure of a category $G_\Sigma$ defined by Artin gluing.
Logical relations $(\mathcal{A}, \tilde{\mathcal{A}} \subseteq \rho(\mathcal{A}))$ have the structure of a category $\mathcal{G}_\Sigma$ defined by Artin gluing.
Logical relations \((A, \tilde{A} \subseteq \rho(A))\) have the structure of a category \(\mathcal{G}_\Sigma\) defined by Artin gluing.

A morphism \((B, \tilde{B}) \rightarrow (A, \tilde{A}) : \mathcal{G}_\Sigma\) is a term/deduction \(\alpha : B \rightarrow A\) that preserves computability, i.e. sends closed terms in \(\tilde{B}\) to closed terms in \(\tilde{A}\).
Logical relations $(\mathcal{A}, \mathcal{\tilde{A}} \subseteq \rho(\mathcal{A}))$ have the structure of a category $\mathcal{G}_\Sigma$ defined by Artin gluing.

A morphism $(\mathcal{B}, \mathcal{\tilde{B}}) \longrightarrow (\mathcal{A}, \mathcal{\tilde{A}}) : \mathcal{G}_\Sigma$ is a term/deduction $\alpha : \mathcal{B} \longrightarrow \mathcal{A}$ that preserves computability, i.e. sends closed terms in $\mathcal{\tilde{B}}$ to closed terms in $\mathcal{\tilde{A}}$.

Thinking of subsets as injective functions:
Logical relations $(\mathcal{A}, \tilde{\mathcal{A}} \subseteq \rho(\mathcal{A}))$ have the structure of a category $\mathcal{G}_\Sigma$ defined by **Artin gluing**.
Logical relations \((A, \tilde{A} \subseteq \rho(A))\) have the structure of a category \(\mathcal{G}_\Sigma\) defined by Artin gluing.

\[
\begin{array}{ccc}
\mathcal{G}_\Sigma & \longrightarrow & \text{Sub}(\text{Set}) \\
\downarrow & & \downarrow \\
\mathcal{C}_\Sigma & \longrightarrow & \text{Set}
\end{array}
\]
Logical relations \((\mathcal{A}, \overset{\sim}{\mathcal{A}} \subseteq \rho(\mathcal{A}))\) have the structure of a category \(\mathcal{G}_\Sigma\) defined by **Artin gluing**.

\[
\begin{array}{ccc}
\mathcal{G}_\Sigma & \longrightarrow & \text{Sub}(\text{Set}) \\
\downarrow & & \downarrow \\
\mathcal{C}_\Sigma & \longrightarrow & \text{Set}
\end{array}
\]

**Fundamental Theorem of Logical Relations:** prove that \(\mathcal{G}_\Sigma\) is a model of the theory \(\Sigma\)!
**FTLR Idea:** interpret each type into $\mathcal{G}_\Sigma$ in such a way that an element carries the proof of the desired metatheorem, e.g. canonicity at base type:

\[
\begin{array}{ccc}
\mathcal{G}_\Sigma & \xrightarrow{\text{Sub}(\text{Set})} & \{\text{yes, no}\} \\
\downarrow & & \downarrow \\
\mathcal{C}_\Sigma & \xrightarrow{\rho} & \text{Set} \\
\end{array}
\]

Connectives that have $\beta/\eta$ laws are uniquely determined! No need (or ability) to be clever.
Proof-relevant logical relations!

What is the logical relation for the collection of types $T_p$?
Ought to be a “relation of relations”, but that’s nonsense.
Proof-relevant logical relations!

What is the logical relation for the collection of types $T p$? Ought to be a “relation of relations”, but that’s nonsense.

**Old solution:** parameterize in complex inductive definition, prove dozens of technical lemmas [All87]
Proof-relevant logical relations!

What is the logical relation for the collection of types $T_p$? Ought to be a “relation of relations”, but that’s nonsense.

**Old solution:** parameterize in complex inductive definition, prove dozens of technical lemmas [All87]

**New solution:** generalize from *property* to *structure*\(^2\)

---

\(^1\)Application to universes due to Shulman [Shu15], developed further by Coquand [Coq19]; main ingredients invented by the Grothendieck school in the 1970s [AGV72].
Proof-relevant logical relations!

What is the logical relation for the collection of types $T_p$? Ought to be a “relation of relations”, but that’s nonsense.

**Old solution:** parameterize in complex inductive definition, prove dozens of technical lemmas \[\text{[All87]}\]

**New solution:** generalize from property to structure$^2$

\[
\begin{align*}
\mathcal{G}_\Sigma & \longrightarrow \text{Sub}(\text{Set}) \\
\downarrow & \\
\mathcal{C}_\Sigma & \longrightarrow \text{Set} \\
\rho & \\
\downarrow & \\
\mathcal{G}_\Sigma[[T_p]] & \quad \rho(T_p)
\end{align*}
\]

$^1$Application to universes due to Shulman [Shu15], developed further by Coquand [Coq19]; main ingredients invented by the Grothendieck school in the 1970s [AGV72].
Proof-relevant logical relations!

What is the logical relation for the collection of types $Tp$? Ought to be a “relation of relations”, but that’s nonsense.

**Old solution:** parameterize in complex inductive definition, prove dozens of technical lemmas [All87]

**New solution:** generalize from *property* to *structure*\(^2\)

\[
\begin{align*}
G_\Sigma & \longrightarrow \text{Fam}(\text{Set}) \\
\sum_{\Lambda: \rho(Tp)} (\rho(Tm)(\Lambda) \rightarrow \text{Set}_{small}) & \downarrow \quad \text{G}_\Sigma[Tp] \\
C_\Sigma & \longrightarrow \text{Set} \\
\rho & \downarrow \quad \rho(Tp)
\end{align*}
\]

\(^1\)Application to universes due to Shulman [Shu15], developed further by Coquand [Coq19]; main ingredients invented by the Grothendieck school in the 1970s [AGV72].
Abstract Artin gluing: logical relations as types!

Constructing the logical relation in $\mathcal{G}_\Sigma$ over $\mathcal{C}_\Sigma$ still very technical! **Trivialized by “synthetic Tait computability” (STC).**³

³Based on an idea of Shulman [Shu11], **STC** is an “Orton-Pitts method” [OP16] for syntactic metatheory [SG20; SH20].
Abstract Artin gluing: logical relations as types!

Constructing the logical relation in $\mathcal{G}_\Sigma$ over $\mathcal{C}_\Sigma$ still very technical! Trivialized by “synthetic Tait computability” (STC).³

Idea: replace “A logical relation is given by …” with “A logical relation can be used to…”.

³Based on an idea of Shulman [Shu11], STC is an “Orton-Pitts method” [OP16] for syntactic metatheory [SG20; SH20].
Abstract Artin gluing: logical relations as types!

Constructing the logical relation in $\mathcal{G}_\Sigma$ over $\mathcal{C}_\Sigma$ still very technical! **Trivialized by “synthetic Tait computability” (STC).**

**Idea:** replace “A logical relation is given by ...” with “A logical relation can be used to...”.

**Result:** a rich type theory whose types denote generalized logical relations. Syntax and semantics accessed via **modalities** à la Rijke, Shulman, and Spitters [RSS17].

---

3Based on an idea of Shulman [Shu11], **STC** is an “Orton-Pitts method” [OP16] for syntactic metatheory [SG20; SH20].
Constructing the logical relation in $\mathcal{G}_\Sigma$ over $\mathcal{C}_\Sigma$ still very technical! **Trivialized by “synthetic Tait computability” (STC).**

**Idea:** replace “A logical relation is given by ...” with “A logical relation can be used to...”.

**Result:** a rich type theory whose types denote generalized logical relations. Syntax and semantics accessed via **modalities** à la Rijke, Shulman, and Spitters [RSS17].

$\Lambda \leadsto “A\ synthetic\ logical\ relation”$

$\bigcirc \Lambda \leadsto “The\ syntactic\ part\ of\ \Lambda”\ (acts\ like\ a\ sort)$

$\bigcirc \Lambda \leadsto “The\ semantic\ part\ of\ \Lambda”\ (acts\ like\ a\ set)$

---

3 Based on an idea of Shulman [Shu11], **STC** is an “Orton-Pitts method” [OP16] for syntactic metatheory [SG20; SH20].
Synthetic Tait computability

Extensional type theory **STC** with an *open modality* $\circ A$, and a complementary *closed modality* $\bullet A$, both monadic.

$A \sim \text{“A synthetic logical relation”}$

$\circ A \sim \text{“The syntactic part of } A \text{” (acts like a sort)}$

$\bullet A \sim \text{“The semantic part of } A \text{” (acts like a set)}$
Synthetic Tait computability

Extensional type theory **STC** with an *open modality* \( \bigcirc A \), and a complementary *closed modality* \( \bullet A \), both monadic.

\[ A \sim \text{“A synthetic logical relation”} \]
\[ \bigcirc A \sim \text{“The syntactic part of } A \text{” (acts like a sort)} \]
\[ \bullet A \sim \text{“The semantic part of } A \text{” (acts like a set)} \]

**Facts/Axioms:**

1. The syntactic part of the semantic part of a logical relation is trivial: \( \bigcirc \bullet A \cong 1 \).
Synthetic Tait computability

Extensional type theory STC with an open modality $\circ A$, and a complementary closed modality $\bullet A$, both monadic.

$A \sim \text{“A synthetic logical relation”}$

$\circ A \sim \text{“The syntactic part of } A\text{” (acts like a sort)}$

$\bullet A \sim \text{“The semantic part of } A\text{” (acts like a set)}$

**Facts/Axioms:**

1. The syntactic part of the semantic part of a logical relation is trivial: $\circ \bullet A \cong 1$.

2. A logical relation can be reconstructed from its syntactic and semantic parts [AGV72; RSS17].
Synthetic Tait computability

Extensional type theory \textbf{STC} with an open modality $\circ A$, and a complementary closed modality $\bullet A$, both monadic.

\[ A \sim \text{“A synthetic logical relation”} \]
\[ \circ A \sim \text{“The syntactic part of } A \text{” (acts like a sort)} \]
\[ \bullet A \sim \text{“The semantic part of } A \text{” (acts like a set)} \]

\textbf{Facts/Axioms:}

1. The syntactic part of the semantic part of a logical relation is trivial: $\circ \bullet A \cong 1$.
2. A logical relation can be reconstructed from its syntactic and semantic parts [AGV72; RSS17].
3. A logical relation can be realigned to have different (but isomorphic) syntactic part [OP16].
Idea: axiomatize a $\Sigma$-algebra $A^\circ_\Sigma$ in the $\circ$-modal fragment of STC, then construct a $\Sigma$-algebra $A^*_\Sigma$ in STC such that $\circ(A^* = A^\circ)$ holds.
Idea: axiomatize a $\Sigma$-algebra $A^\circ_\Sigma$ in the $\circ$-modal fragment of STC, then construct a $\Sigma$-algebra $A^*_\Sigma$ in STC such that $\circ(A^* = A^\circ)$ holds.

Examples:

$$Tp^* \equiv \sum_{A^\circ:Tp^*} \{A^* : U | \circ(Tm^\circ(A^\circ) = A^*)\}$$

$$ans^* \equiv (ans^\circ, \sum_{b^\circ:Tm^\circ(ans^\circ)} \bigcirc\{b^\bullet : 2 | b^\circ = \text{if } b^\bullet \text{ then } yes^\circ \text{ else } no^\circ\})$$

Payoff: painful construction of (e.g.) dependent product in logical relations made trivial, because open modalities commute with dependent products. No more technical lemmas!
Idea: axiomatize a $\Sigma$-algebra $\mathcal{A}_\Sigma^\circ$ in the $\circ$-modal fragment of \textsc{STC}, then construct a $\Sigma$-algebra $\mathcal{A}_\Sigma^*$ in \textsc{STC} such that $\circ(\mathcal{A}^* = \mathcal{A}^\circ)$ holds.

Examples:

$$\text{Tp}^* \equiv \sum_{A^\circ : \text{Tp}^\circ} \{ A^* : \mathcal{U} \mid \circ(\text{Tm}^\circ(A^\circ) = A^*) \}$$

$$\text{ans}^* \equiv (\text{ans}^\circ, \sum_{b^\circ : \text{Tm}^\circ(\text{ans}^\circ)} \bullet \{ b^\bullet : \{2 \mid b^\circ = \text{if } b^\bullet \text{ then yes}^\circ \text{ else no}^\circ \} \}$$

Payoff: painful construction of (e.g.) dependent product in logical relations made trivial, because open modalities commute with dependent products. \textit{No more technical lemmas!}
Show $\circ T_p^* \simeq T_p^\circ$. 
Show $\circ T_p^* \simeq T_p^\circ$. 
Show $\bigcirc T_p^* \cong T_p^\circ$.

$$\bigcirc \sum_{A^\circ: T_p^\circ} \{ A^* : U \mid \bigcirc (T_m^\circ(A^\circ) = A^*) \}$$
Show $\circ T_p^* \cong T_p^\circ$.

$$\sum_{A^\circ : \circ T_p^\circ} \{ A^* : \mathcal{U} | \circ (T_m^\circ (A^\circ)) = A^* \}$$
Show \( \circ T_p^* \simeq T_p^\circ \).

\[
\sum_{A^\circ : \circ T_p^\circ \circ \{A^* : \mathcal{U} \mid T_{m^\circ}(A^\circ) = A^*\}}
\]
Show $\circ T_p^* \cong T_p^\circ$.
Show $\bigcirc T_p^* \cong T_p^\circ$.

$$\sum_{A^\circ : \bigcirc T_p^\circ 1}$$
Show $\circ T_p^* \cong T_p^\circ$.
Show $⊙ Tp^* \cong Tp^\circ$. 
Show $\circ T_p^* \cong T_p^\circ$. 

$T_p^\circ$
Closure under dependent products.

Construct $\Pi^* : (A : Tp^*, B : A \rightarrow Tp^*) \rightarrow Tp^*$ with
$
\bigcirc \Pi^*(A, B) = \Pi^\circ(A^\circ, \lambda x.B^\circ(x)).
$
Closure under dependent products.

Fix $A : Tp^*$, $B : A \to Tp^*$ to construct $\Pi^*(A, B)$ with syntactic part $\Pi^°(A^°, \lambda x.B^°(x))$. 

Closure under dependent products.

Fix $A : Tp^*$, $B : A \to Tp^*$ to construct $\Pi^*(A, B)$ with syntactic part $\Pi^\circ(A^\circ, \lambda x.B^\circ(x))$.

Recall: $Tp^* \equiv \sum_{A^\circ : Tp^\circ} \{ A^* : U \mid \circ(Tm^\circ(A^\circ) = A^*) \}$
Closure under dependent products.

Fix $A : Tp^*$, $B : A \to Tp^*$ to construct $\Pi^*(A, B)$ with syntactic part $\Pi^\circ(A^\circ, \lambda x. B^\circ(x))$.

Recall: $Tp^* \equiv \sum_{A^\circ : Tp^\circ} \{ A^* : \mathcal{U} \mid \circ(Tm^\circ(A^\circ) = A^*) \}$

1. Choose first component: $\Pi^\circ(A^\circ, \lambda x. B^\circ(x))$. 
Closure under dependent products.

Fix \( A : Tp^* \), \( B : A \to Tp^* \) to construct \( \Pi^*(A, B) \) with syntactic part \( \Pi^\circ(A^\circ, \lambda x. B^\circ(x)) \).

Recall: \( Tp^* \cong \sum_{A^\circ : Tp^\circ} \{ A^* : U \mid \circ(Tm^\circ(A^\circ) = A^*) \} \)

1. Choose first component: \( \Pi^\circ(A^\circ, \lambda x. B^\circ(x)) \).

2. Choose second component: \( (x : Tm^*(A)) \to Tm^*(B(x)) \).
Closure under dependent products.

Fix $A : Tp^*$, $B : A \to Tp^*$ to construct $\Pi^*(A, B)$ with syntactic part $\Pi^*(A', \lambda x.B'(x))$.

Recall: $Tp^* \cong \sum_{A_0 : Tp^*} \{ A^* : U \mid \circ(Tm^*(A)) = A^* \}$

1. Choose first component: $\Pi^*(A', \lambda x.B'(x))$.
2. Choose second component: $(x : Tm^*(A)) \to Tm^*(B(x))$, realigned by the following isomorphism:

$$\circ((x : Tm^*(A)) \to Tm^*(B(x))) \cong Tm^*(\Pi^*(A', B'(x)))$$
\( \bigcirc \left( (\chi : Tm^*(A)) \rightarrow Tm^*(B(\chi)) \right) \)
\[(\chi : \bigcirc \text{TM}^*(A)) \rightarrow \bigcirc \text{TM}^*(B(\chi))\]
\((x : \text{Tm}^\circ(A^\circ)) \to \text{Tm}^\circ(B^\circ(x))\)
(\chi : Tm^\circ(A^\circ)) \rightarrow Tm^\circ(B^\circ(\chi))
S. & Harper introduced **STC** to give a trivial proof of a non-trivial parametricity result for effectful ML modules:


**This thesis**: use **STC** to prove normalization of Cartesian cubical type theory / **TT**.
Proposed work

Prove normalization for Cartesian cubical type theory (TT).
Proposed work

**Prove normalization for Cartesian cubical type theory (TT).**

1. Construct a model of STC extended by the syntax of TT and its normal forms.
Proposed work

Prove normalization for Cartesian cubical type theory (TT\textsuperscript{•}•).

1. Construct a model of STC extended by the syntax of TT\textsuperscript{•}• and its normal forms.

2. Define a “computability \(\Sigma\)-algebra” in STC such that each sort \(A\) exhibits Tait’s Yoga [Tai67]:

\[
\begin{align*}
A^{\text{Ne}} & \xrightarrow{\text{reflect}_A} A^* & \xrightarrow{\text{reify}_A} A^{\text{Nf}} \\
A^{\text{Ne}} \xrightarrow{\eta_{A^{\text{Ne}}}} A^* & \xrightarrow{\eta_{A^*}} A^{\text{Nf}} 
\end{align*}
\]
Proposed work

Prove normalization for Cartesian cubical type theory ($\text{TT}^{\square}$).

1. Construct a model of $\text{STC}$ extended by the syntax of $\text{TT}^{\square}$ and its normal forms.

2. Define a “computability $\Sigma^{\square}$-algebra” in $\text{STC}$ such that each sort $A$ exhibits Tait’s Yoga [Tai67]:

3. That does it! **Next:** actually construct the model.
Renamings & unstable normal forms, geometrically

Let $\mathcal{C}$ be the category of judgments and deductions.
Renamings & unstable normal forms, geometrically

Let \( \mathcal{C} \) be the category of judgments and deductions.

- By Yoneda, have a space\(^\ast\) \( \hat{\mathcal{C}} \) of (generalized) judgments and substitution functions.
Renamings & unstable normal forms, geometrically

Let \( \mathcal{C} \) be the category of judgments and deductions.

- By Yoneda, have a space\((*)\) \( \hat{\mathcal{C}} \) of (generalized) judgments and substitution functions.

- Likewise, have a space of (generalized) formal contexts and renaming functions \( \hat{\mathcal{R}} \) lying over \( \hat{\mathcal{C}} \). **Idea: Kripke logical relations varying in** \( \hat{\mathcal{R}} \rightarrow \hat{\mathcal{C}} \).
Renamings & unstable normal forms, geometrically

Let $\mathcal{C}$ be the category of judgments and deductions.

- By Yoneda, have a space(*) $\overset{*}{\mathcal{C}}$ of (generalized) judgments and substitution functions.

- Likewise, have a space of (generalized) formal contexts and renaming functions $\overset{\circ}{\mathcal{R}}$ lying over $\overset{*}{\mathcal{C}}$. Idea: \textbf{Kripke logical relations varying in} $\overset{\circ}{\mathcal{R}} \rightarrow \overset{*}{\mathcal{C}}$.

- $\overset{\circ}{\mathcal{R}}$ has monoidal closed structure $(\otimes, \multimap)$ to implement bunches of non-equal dimensions and fresh binders:

\[
\begin{align*}
\text{NEUTRAL COE} \ i & \leadsto j \\
\psi : I \otimes I \quad A : I \multimap Ne \quad M : Nf \\
\hline
\text{coe}_{\psi}^{\psi} (M) : Ne
\end{align*}
\]
Let $\mathcal{C}$ be the category of judgments and deductions.

- By Yoneda, have a space $(\ast) \hat{\mathcal{C}}$ of (generalized) judgments and substitution functions.
- Likewise, have a space of (generalized) formal contexts and renaming functions $\hat{\mathcal{R}}$ lying over $\hat{\mathcal{C}}$. **Idea: Kripke logical relations varying in** $\hat{\mathcal{R}} \rightarrow \hat{\mathcal{C}}$.
- $\hat{\mathcal{R}}$ has monoidal closed structure $(\otimes, \multimap)$ to implement bunches of non-equal dimensions and fresh binders:

\[
\begin{align*}
\text{NEUTRAL COE } i & \sim j \\
\psi : I \otimes I & \quad A : I \multimap Ne \quad M : Nf \\
\text{coe}_{A}^{\psi}(M) : Ne
\end{align*}
\]

**Foreseen difficulties:** What to do with the inconsistent context $(\Gamma, p : 0 \equiv_{\Pi} 1)$?
Kripke logical relations, geometrically

\[ S = \{ \emptyset, \{ \circ \}, \{ \circ, \bullet \} \} \] is the interface of a family.

Open point \( \circ \) / equal.osf coordinate of the base/index

Closed point \( \bullet \) / equal.osf coordinate of the total space

Sierpiński cylinder: \( \hat{R} \times S \) is the interface of families of generalized contexts and renamings.

Sierpiński cone(*)/gluing: \( K = \hat{C} \sqcup \hat{R} \) \( \hat{R} \times S \) is the interface of families of generalized contexts and renamings indexed in generalized judgments and substitutions, i.e. Kripke logical relations!

\( \hat{R} \times S \) \( \hat{R} \times \circ \) \( \hat{C} K \) Sh(\( K \)) a suitable model of STC.
Kripke logical relations, geometrically

- **Sierpiński interval:** $S = \{\emptyset, \circ, \circ, \bullet\}$ is the interface of a *family*.
  - open point $\circ = \text{coordinate of the base/index}$
  - closed point $\bullet = \text{coordinate of the total space}$
Kripke logical relations, geometrically

- **Sierpiński interval**: $S = \{\emptyset, \circ, \{\circ, \bullet\}\}$ is the interface of a *family*.
  - open point $\circ$ = coordinate of the base/index
  - closed point $\bullet$ = coordinate of the total space
- **Sierpiński cylinder**: $\hat{R} \times S$ is the interface of *families of generalized contexts and renamings*.
Kripke logical relations, geometrically

- **Sierpiński interval:** $\mathcal{S} = \{\emptyset, \{\circ\}, \{\circ, \bullet\}\}$ is the interface of a family.
  - open point $\circ$ = coordinate of the base/index
  - closed point $\bullet$ = coordinate of the total space

- **Sierpiński cylinder:** $\hat{\mathcal{R}} \times \mathcal{S}$ is the interface of families of generalized contexts and renamings.

- **Sierpiński cone(\textasteriskcentered)/gluing:** $\mathcal{K} = \hat{\mathcal{C}} \sqcup \hat{\mathcal{R}} \times \mathcal{S}$ is the interface of families of generalized contexts and renamings indexed in generalized judgments and substitutions, i.e. Kripke logical relations!

```
\begin{tikzcd}
\hat{\mathcal{R}} & \hat{\mathcal{C}} \\
\hat{\mathcal{R}} \times \circ \\
\hat{\mathcal{R}} \times \mathcal{S} \arrow[r, dashed] & \mathcal{K}
\end{tikzcd}
```
Kripke logical relations, geometrically

- **Sierpiński interval**: \( S = \{\emptyset, \{\circ\}, \{\circ, \bullet\}\} \) is the interface of a family.
  - open point \( \circ = \) coordinate of the base/index
  - closed point \( \bullet = \) coordinate of the total space

- **Sierpiński cylinder**: \( \hat{\mathbb{R}} \times S \) is the interface of families of generalized contexts and renamings.

- **Sierpiński cone(*)/gluing**: \( K = \hat{\mathbb{C}} \sqcup \hat{\mathbb{R}} \times S \) is the interface of families of generalized contexts and renamings indexed in generalized judgments and substitutions, i.e. Kripke logical relations!

\[
\begin{array}{ccc}
\hat{\mathbb{R}} & \rightarrow & \hat{\mathbb{C}} \\
\downarrow & & \downarrow \\
\hat{\mathbb{R}} \times \circ & \rightarrow & \hat{\mathbb{C}} \\
\downarrow \quad \quad \quad \quad \downarrow \\
\hat{\mathbb{R}} \times S & \rightarrow & K
\end{array}
\]

\( \text{Sh}(K) \) a suitable model of \( \text{STC} \).
From now on, we work in the language of **STC**. Let $A$ be a type of $\text{TT}$. 

1. We have an object $A^{\text{Var}} : \mathcal{U}$ of variables with $\circ(A^{\text{Var}} = A^\circ)$
2. Extend it to a definition of *neutral forms* $A^{\text{Ne}}$ of type $A$
3. Extend it to a definition of *normal forms* $A^{\text{Nf}}$ of type $A$.

**Subtleties:** representation of interval dimension binders, tensors of dimensions.
From now on, we work abstractly in the (extended) STC substantiated above. Inspired by and improving on Coquand [Coq19], we define the computability algebra of types:

$$Tp^* \equiv \sum \left\{ \begin{array}{l}
A^\circ : Tp^\circ \\
\mathcal{A} : \{ \mathcal{A} : Tp^{Nf} \mid \circ (\mathcal{A} = A^\circ) \} \\
A^* : \{ A^* : U \mid \circ (A^* = Tm^\circ(A^\circ)) \} \\
\text{reflect}_A : \{ f : A^{Ne} \rightarrow A^* \mid \circ (f = \text{id}) \} \\
\text{reify}_A : \{ f : A^* \rightarrow A^{Nf} \mid \circ (f = \text{id}) \} \end{array} \right\}$$
The computability algebra

From now on, we work abstractly in the (extended) STC substantiated above. Inspired by and improving on Coquand [Coq19], we define the computability algebra of types:

\[ T_p^* \cong \sum \begin{cases} A^\circ : T_p^\circ \\ \mathcal{A} : \{ \mathcal{A} : T_p^{Nf} \mid \circ(\mathcal{A} = A^\circ) \} \\ A^* : \{ A^* : \mathcal{U} \mid \circ(A^* = T_m^\circ(A^\circ)) \} \\ \text{reflect}_A : \{ f : A^{Ne} \to A^* \mid \circ(f = \text{id}) \} \\ \text{reify}_A : \{ f : A^* \to A^{Nf} \mid \circ(f = \text{id}) \} \end{cases} \]

**Foreseen difficulties:** We must define a version of \( T_p^* \) that equips each *family* \( \mathbb{I} \to T_p^* \) with a composition operation; possible if the interval is atomic [Lic+18]. This is the main thing that could go wrong.
The goal is a proof of normalization for cubical type theory with a univalent universe. Granular prediction impossible, but I estimate the following milestones to serve as fallback positions if part of this turns out to be intractable.

- **Now + 6 months:** a proof of \( \beta/\eta \) normalization for MLTT + \( \Pi \) + \( \mathbb{F} \) + extension types, no universes or HITs.
- **Now + 8 months:** a mathematical specification of the elaboration of a redtt/cooltt-style external language to the core type theory specified above.
- **Now + 12 months:** extension of proof to include a univalent universe.
Stretch goals

The following are things that I do not promise to do, but which might happen along the way if my progress is better than expected. If I don’t do it, one of your students should!

- **Extensions of the coq implementation**: modules, higher inductive types, more sophisticated universe hierarchies, or support for **STC** modalities.
- **Extensions of synthetic Tait computability** to account for modal and substructural type theory.
- **Objective metatheory of effectful PLs**: Sterling and Harper [SH20] just a first step, more development needed!
References I


References II


<table>
<thead>
<tr>
<th>Reference</th>
<th>Author(s)</th>
<th>Title and Details</th>
</tr>
</thead>
</table>
References IV


References V


References VI


<table>
<thead>
<tr>
<th>Reference</th>
<th>Title</th>
</tr>
</thead>
</table>
References VIII


References IX


References X


