A Directed, Topos-Theoretic Type Theory

finegeometer

July 25, 2025

In anything at all, perfection is finally attained not when there is no longer anything to add, but when there is no longer anything to take away.
— Antoine de Saint-Exupéry; *Wind, Sand and Stars*

I describe a system that displays aspects of modal and directed type theories, at no more theoretical complexity cost than Martin Löf Type Theory.

Contents

1 Motivations

1.1 Multimode

Standard, Martin-Löf type theory can be interpreted in an arbitrary topos. By working in these toposes, and adopting the axioms they support, we get simple and beautiful "synthetic" accounts of various areas of mathematics, such as differential geometry, algebraic geometry, domain theory, or higher category theory.

In 2020, Gratzer, Kavvos, Nuyts, and Birkedal introduced *Multimode Type Theory* ([1]), which is to be interpreted not in a single topos, but a collection of them. It innovates over previous modal type theories in that it isn't tied to a specific interpretation. In fact, thanks to a followup paper by Shulman ([2]), we know that it can be interpreted in *any* finite diagram of toposes, chosen in advance.

But why should we require the toposes be chosen in advance? Why not let the *programmer* choose, on a program-by-program basis?

1.2 Directed

Homotopy type theory is an extension of Martin-Löf type theory, in which every construction provably respects isomorphism. That is, we can prove that every function from e.g. groups to sets takes isomorphic groups to isomorphic sets. This is really convenient, since otherwise we'd have to prove this manually for every function we care about; a task that quickly becomes tedious.

But just as often, we want to know that our constructions are functorial. That they preserve not just isomorphisms, but morphisms. This has inspired the search for a *directed* type theory; one in which types automatically carry the structure of a category, and functions between them are always functorial.

But there's a problem. The type $A \to B$ isn't functorial in A! It's *contravariantly* functorial. So it seems any directed type theory needs to handle multiple variances of functor. This becomes more and more complicated when we start to work with higher categories, and it is unclear how to proceed.

1.3 Simplicity

My final motivation is a strong desire for simplicity.

Martin-Löf type theory is an impressively simple language. Multimode type theory isn't *too* complicated, but it's certainly less simple. In tackling the problems described above, I am unwilling to allow the complexity of the language to continue to increase.

As an example of this philosophy, consider Multimode Type Theory. It has modalities, which allow the user to move to another topos. But abstracting over a variable means you move to a slice topos. Yet these two features are unrelated in Multimode Type Theory, so I naturally focus on trying to merge them.

2 Design Process

I began with the insight was that describing a *classifying* topos feels similar to describing a context in a type theory. Describing a geometric morphism between classifying toposes feels like a substitution.

So why not merge the concepts?

Interpret the contexts of Martin-Löf type theory as *toposes*. The substitutions as geometric morphisms. The types as the objects of the toposes, and the terms as global elements.

So I tried that. It didn't work. In the model, $(\Pi x : A.B)[\sigma]$ and $\Pi x : A[\sigma].B[\sigma[\mathbf{wk}], x]$ don't have the same interpretation.

But do we need them to?

Let's simply *remove* the commutation rules. Substitutions will no longer commute through Π-types or lambdas. To compensate, we generalize β-reduction to handle the case where a substitution is sandwiched between the lambda and the application.

Is this *too* simple? Too weak? Maybe, but maybe not. The weakened substitution rules do not block evaluation; they only weaken type equality.

And in removing these commutation rules, I seem to have unintentionally solved the variance issue for directed type theory. When we ask if $A \to B$ is functorial in A, we're no longer asking if a map $A \to A'$ yields a map $(A \to B) \to (A' \to B)$. We're asking if it yields a map $(X \to B)[A/X] \to (X \to B)[A'/X]$. Without the commutation rules, this is a different question, and there's no reason it isn't possible.

3 The Language

The language is specified as follows. Only the last bullet point is different from Martin-Löf type theory.

- We have a category of contexts.

$$\frac{}{\mathsf{Cx}} \qquad \frac{\Gamma : \mathsf{Cx} \qquad \Delta : \mathsf{Cx}}{\mathsf{Sb}(\Gamma, \Delta)}$$

$$\frac{\Gamma : \mathsf{Cx}}{\mathsf{id} : \mathsf{Sb}(\Gamma, \Gamma)} \qquad \frac{\sigma : \mathsf{Sb}(\Gamma, \Delta) \qquad \tau : \mathsf{Sb}(\Delta, \Xi)}{\tau[\sigma] : \mathsf{Sb}(\Gamma, \Xi)}$$

$$\mathsf{id}[\sigma] = \sigma \qquad \sigma[\mathsf{id}] = \sigma \qquad \upsilon[\tau[\sigma]] = \upsilon[\tau][\sigma]$$

- We have types and terms, acted on contravariantly by substitutions.

$$\frac{\Gamma : \mathsf{Cx}}{\mathsf{Ty}(\Gamma)} \qquad \frac{A : \mathsf{Ty}(\Gamma)}{\mathsf{Tm}(\Gamma, A)}$$

$$\frac{\sigma : \mathsf{Sb}(\Xi, \Gamma) \qquad A : \mathsf{Ty}(\Gamma)}{A[\sigma] : \mathsf{Ty}(\Xi)} \qquad \frac{\sigma : \mathsf{Sb}(\Xi, \Gamma) \qquad a : \mathsf{Tm}(\Gamma, A)}{a[\sigma] : \mathsf{Tm}(\Xi, A[\sigma])}$$

- A few specific contexts, characterized by their mapping-in behavior.

$$\frac{}{\Gamma : \mathsf{Cx}} \qquad \frac{\Gamma, \Delta : \mathsf{Cx} \qquad A : \mathsf{Ty}(\Gamma)}{(\Gamma, x : A) : \mathsf{Cx}}$$

$$\mathsf{Sb}(\Xi, \epsilon) \simeq \mathbf{1} \qquad \mathsf{Sb}(\Xi, (\Gamma, x : A)) \simeq (\sigma : \mathsf{Sb}(\Xi, \Gamma)) \times \mathsf{Tm}(\Xi, A[\sigma])$$

$$()[\sigma] = () \qquad (\tau, a)[\sigma] = \tau[\sigma], a[\sigma]$$

- In those last two equations, the computation rules for composition of substitutions, I conflate $\mathsf{Sb}(\Xi, \epsilon)$ with $\mathbf{1}$, and similarly for $\mathsf{Sb}(\Xi, (\Gamma, x : A))$. I continue to do so throughout this writeup.
- Somewhat surprisingly, the above rules implicitly introduce the concept of variables. The identity substitution on any nonempty context decomposes as $\mathsf{id} = (\mathsf{wk}, \mathsf{v}_0)$, into a weakening substitution and the variable with DeBruijn index zero. If the weakened context is still nonempty, then we further decompose $\mathsf{wk} = (\mathsf{wk}^2, \mathsf{v}_1)$. Et cetera.

Where did I learn this trick? Cite it.

(Those last two equations, the computation rules for composition of substitutions, only make sense after implicitly casting by the equivalences above them.)

- Finally, we introduce the type constructors, along with their introduction and elimination forms. The eliminators' inference rules are set up so they can act on substituted types; the introduction rules are as normal.

  Notably, we assume *nothing* about how these interact with substitution. It turns out to be provable that substitution commutes through the elimination rules anyway, but this is not true for the introduction rules or type constructors.

  Is there a nicer (categorical?) explanation of these rules? It reminds me of the "lax" behavior found in higher category theory, but I wasn't able to figure it out.

  - Π-types:

$$\frac{A : \mathsf{Ty}(\Gamma) \qquad B : \mathsf{Ty}(\Gamma, x : A)}{((x : A) \to B) : \mathsf{Ty}(\Gamma)} \qquad \frac{e : \mathsf{Tm}((\Gamma, x : A), B)}{\lambda x.e : \mathsf{Tm}(\Gamma, (x : A) \to B)}$$

$$\frac{\sigma : \mathsf{Sb}(\Xi, \Gamma) \qquad f : \mathsf{Tm}(\Xi, ((x : A) \to B)[\sigma])}{\lambda^{-1}x.f : \mathsf{Tm}((\Xi, x : A[\sigma]), B[\sigma[\mathsf{wk}], \mathsf{v}_0])}$$

$$\lambda^{-1}x.(\lambda x.e)[\sigma] \equiv e[\sigma[\mathsf{wk}], \mathsf{v}_0] \qquad f \equiv \lambda x.\lambda^{-1}x.f$$

  - Universes:

$$\frac{}{\mathcal{U} : \mathsf{Ty}(\Gamma)} \qquad \frac{A : \mathsf{Ty}(\Gamma)}{\mathsf{code}\, A : \mathsf{Tm}(\Gamma, \mathcal{U})} \qquad \frac{\sigma : \mathsf{Sb}(\Xi, \Gamma) \qquad t : \mathsf{Tm}(\Xi, \mathcal{U}[\sigma])}{\mathsf{El}\, t : \mathsf{Ty}(\Xi)}$$

$$\mathsf{El}\,(\mathsf{code}\, A)[\sigma] \equiv A[\sigma] \qquad t \equiv \mathsf{code}(\mathsf{El}\, t)$$

3.1 Topos Interpretation

If we're careful about universe levels, the type theory interprets usefully into toposes.

Interpret Cx as the set of toposes. Interpret $\mathsf{Sb}(\Gamma, \Delta)$ as the set of geometric morphisms $\Gamma \leftrightarrows \Delta$.

Interpret *small* types as objects of Γ, and their terms as global elements. Interpret *large* types as toposes over Γ, and their terms as sections. The inclusion of small types into large types becomes the map from objects $A \in \Gamma$ to their corresponding slice toposes $\Gamma_{/A}$.

Substitutions act on small types by their inverse images, and on large types by pullback in **Topos**. Similarly for terms.

The empty context, of course, is the topos of sets. The interpretation of a larger context $\Gamma, x : A$ is the same as the interpretation of A, viewed as a large type. The equations describing substitutions follow after a bit of diagram chasing.

For Π-types, we need to be careful about size. Let $A : \mathsf{Ty}(\Gamma)$ and $B : \mathsf{Ty}(\Gamma, A)$.

- When B is small, allow A to be large. Then $(x : A) \to B$ is a small type, interpreted as the direct image of B under the geometric morphism $\mathsf{wk} : \mathsf{Sb}((\Gamma, x : A), \Gamma)$.
- When B is large, so is $(x : A) \to B$. We interpret it as the dependent product in **Topos** along the topos map $A \to \Gamma$, applied to B.

  This exists when the morphism $A \to \Gamma$ is exponentiable. In particular, since slice toposes are exponentiable, we are able to interpret $(x : A) \to B$ whenever A is a small type.

Finally, the universe $\mathcal{U} : \mathsf{Ty}(\Gamma)$ is a large type. We interpret it as the product of Γ and the classifying topos for objects, equipped with the product projection back to Γ. Sections of this projection correspond exactly to small types, as required.

4 Consequences

4.1 Substitution And Eliminators

By design, substitution does not commute through Π-types or λ-expressions. However, it *does* commute through λ^{-1}, as a consequence of the β and η rules.

$$(\lambda^{-1}x.f)[\sigma[\mathsf{wk}], \mathsf{v}_0] = \lambda^{-1}x.(\lambda x.(\lambda^{-1}x.f))[\sigma] = \lambda^{-1}x.f[\sigma]$$

When we derive function application from λ^{-1} in the usual way, we see that substitution commutes through it too.

$$fa = (\lambda^{-1}x.f)[\mathsf{id}, a]$$

$$\begin{aligned}(fa)[\tau] &= (\lambda^{-1}x.f)[\mathsf{id}, a][\tau] \\ &= (\lambda^{-1}x.f)[\tau, a[\tau]] \\ &= (\lambda^{-1}x.f)[\tau[\mathsf{wk}], \mathsf{v}_0][\mathsf{id}, a[\tau]] \\ &= (\lambda^{-1}x.f[\tau])[\mathsf{id}, a[\tau]] \\ &= f[\tau]a[\tau]\end{aligned}$$

By similar logic, substitution commutes through El.

4.2 Functoriality of Substitution

The first thing to check is the functoriality of substitution. If we can build a B from an A, and we have an $A[\sigma]$, we want a $B[\sigma]$.

More formally, if we have $\sigma : \mathsf{Sb}(\Gamma, \Delta)$, and a type $B : \mathsf{Ty}((\Delta, x : A))$ and a term $e : \mathsf{Tm}((\Delta, x : A), B)$, then we want to form a type $B' : \mathsf{Ty}((\Gamma, x : A[\sigma]))$ and a term $e' : \mathsf{Tm}((\Gamma, x : A[\sigma]), B')$.

It took me embarassingly long to figure this out, but we're simply being asked for a substitution $\mathsf{Sb}((\Gamma, x : A[\sigma]), (\Delta, x : A))$. And we have one. It's given by $(\sigma[\mathsf{wk}], \mathsf{v}_0)$.

Todo list

Where did I learn this trick? Cite it. 4
Is there a nicer (categorical?) explanation of these rules? It reminds me of the "lax" behavior found in higher category theory, but I wasn't able to figure it out. 4
And more:

- Directed Univalence
- Do Church-encoded datatypes work? Assuming directed univalence, do we get the full induction principles, or just the recursion principles?
    - If the latter, explore adding datatypes in the typical way. Substitutions should commute past many of them.
- Just generally explore the system.
- Are there good higher-categorical models?
- Citations

References

[1] Daniel Gratzer et al. "Multimodal Dependent Type Theory". In: *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*. LICS '20. Saarbrücken, Germany: Association for Computing Machinery, 2020, pp. 492–506. ISBN: 9781450371049. DOI: 10.1145/3373718.3394736. URL: https://doi.org/10.1145/3373718.3394736.

[2] Michael Shulman. "Semantics of multimodal adjoint type theory". In: *Electronic Notes in Theoretical Informatics and Computer Science* Volume 3-Proceedings of... (Nov. 2023). ISSN: 2969-2431. DOI: 10.46298/entics.12300. URL: http://dx.doi.org/10.46298/entics.12300.