JONATHAN STERLING

ALGEBRAIC TYPE THEORY & THE GLUING CONSTRUCTION

The syntactic metatheory of type theory comprises several components, which are brutally intertwined in the *dependent* case: closed canonicity, normalization, decidability of type checking, and initiality (soundness and completeness). While these standard results play a scientific role in type theory essentially analogous to that of cut elimination in structural proof theory, they have historically been very difficult to obtain and scale using purely syntactic methods. The purpose of this lecture is to explain a more abstract and tractable methodology for obtaining these results: generalized algebraic theories and categorical gluing.

Introduction

Type theory is not any specific artifact, to be found in the "great book of type theories", but is instead a tradition of scientific practice which is bent toward providing *useful language* for mathematical construction, and nurturing the dualities of the abstract and the concrete, and of syntax and semantics. Syntax itself is an exercise in brutality, and to study it in the concrete for too long is a form of intellectual suicide — as we will see in the following sections, the concrete view of syntax offers dozens of unforced choices, each of which is exceedingly painful to develop in a precise way.

If that is the case, what is it that type theorists are doing? How do we tell if our ideas are correct, or mere formalistic deviations? In structural proof theory, the practice of *cut elimination* separates the wheat from the chaff (Girard, *The Blind Spot: Lectures on Logic*); type theorists, on the other hand, do not have a single answer to this question, but pay careful attention to a number of (individually defeasible) syntactic properties which experience has shown to be important in practice:

1. Sound and complete interpretation into a suitable category of models ("initiality").
2. Canonicity for closed terms of base type: every term of a base type which has no free variables is definitionally equal to a constant.[1]
3. Normalization for open terms: associating to every definitional equivalence class a canonical representative, drawn from a domain for which equality is discrete or *trivially* decidable.[2]
4. Decidability of the typing relation.[3]

Unfortunately, in the case of dependent type theory, these properties are intertwined in a very painful way. The existence of interpretations of the raw syntax of type theory (as it usually appears, in a minimally annotated form) into models usually cannot be obtained except for as a corollary of a normalization theorem. But normalization theorems themselves are most perspicuously obtained as a corollary of the initiality of the syntactic model!

There is therefore a serious bootstrapping issue; type theorists have historically dealt with it by developing the syntax of type theory in a very concrete way,

[1] Viewing type theory as a logic, one obtains consistency as a corollary of canonicity.

[2] An immediate corollary is the decidability of definitional equality.

[3] Sometimes it is said that the decidability of typing is a corollary of decidability of definitional equality, but this is only the case when the syntax is highly annotated.

and then proving an even more appalling normalization theorem for this syntax, unable to exploit any algebraic abstractions. In these notes, we advocate a different approach, in which type theory is first developed in a way that guarantees *automatically* a suitable category of models with an initial object, which can be obtained from a highly-annotated version of the syntax of type theory.

Then, abstract and scalable categorical methods (principally *categorical gluing*) can be used to prove properties (2–4); these results then justify the development of an "elementary" version of the syntax of type theory, with fewer annotations and coercions,[4] operationalizing a coherent *elaboration* from the elementary syntax to the abstract (annotated) syntax.

[4] One might call this syntax the "Type Theory of the Book"; contrary to popular belief, it is quite rare to work with this style of type theory in a precise way.

Syntax of type theory in the concrete

The syntax of type theory is sometimes presented by first enumerating the *pre-terms* (often fixing an "infinite collection of variables" $\mathbb{A}$), defining a capture-avoiding substitution operation $[M/x]N$ by recursion on raw terms, and finally quotienting under injective renamings of bound variables, resulting in a notion of *α-invariant pre-term*.[5] After this, judgments which express well-formedness and definitional equality in each syntactic category (at least contexts, types and terms) are declared, and defined inductively using an informal "inference rule" notation. In most presentations, these judgments and rules are specified as ranging over the α-invariant pre-terms.

[5] Since 1984, Martin-Löf has preferred an abstract presentation of raw syntax which abstracts variable binding, but many authors have persisted in fixing this infinite collection of variables at the start, and imposing the binding discipline later. We start from this very concrete perspective mostly for the sake of illustrating the incorrect ideas which most working type theorists have repudiated.

Generating pre-syntax

Consider the type-theoretic contexts Γ. The set $\mathbf{PreCtx}_\Theta$ of pre-contexts binding the names in Θ is characterized by the following rules:[6,7]

$$\frac{}{\cdot \in \mathbf{PreCtx}_\varnothing} \qquad \frac{\Gamma \in \mathbf{PreCtx}_\Theta \quad A \in \mathbf{PreTy} \quad \mathbf{FV}(A) \subseteq \Theta \quad x \notin \Gamma}{\Gamma, x : A \in \mathbf{PreCtx}_{\Theta \cup \{x\}}}$$

Notation. We will write $\mathbf{PreTy}_\Theta$ for the subset $\{A \in \mathbf{PreTy} \mid \mathbf{FV}(A) \subseteq \Theta\}$.

Now, if the set **PreTy** of pre-types can be defined without mentioning **PreCtx**, then the pre-contexts can be defined as the least family of sets closed under these rules; but in general, the inductive definition of the pre-terms of all the sorts must be given simultaneously.[8] After this inductive definition has been completed, renaming and substitution operations are defined by recursion on the pre-terms. Because the syntax of type theory includes variable binding, it is highly error-prone to define renaming and substitution; an illustrative case is the following, which avoids incorrectly capturing a "bound" variable:

$$[N/x](\lambda y.M) = \begin{cases} \lambda y.[N/x]M & \text{if } x \equiv y \\ \lambda y.M & \text{if } x \not\equiv y \end{cases}$$

Thought 1. If it were not for binding of variables, substitution and renaming would be a trivial matter of textual replacement.

[6] Each syntactic sort $\mathcal{X}$ is equipped with a function $\mathbf{FV} : \mathcal{X} \to \mathcal{P}(\mathbb{A})$ which projects the free variables from a pre-term; this is more technical than it sounds, because it is defined simultaneously with the pre-terms themselves.

[7] Following Hofmann ("Syntax and Semantics of Dependent Types"), we make the questionable decision of requiring all variables declarations in a context to be distinct; this is in contrast with all concrete implementations of type theory, in which there is no problem with variable shadowing. However, prior to passing to a more abstract version of syntax (which we will do later), it is unclear how to cleanly lift this strange restriction.

[8] Indeed, while it is often possible to define the pre-types and pre-elements independently of pre-contexts, we will see that this is a damaging assumption in practice.

α-invariant pre-syntax

Using the renaming operation above, the pre-syntax is quotiented under the renaming of bound variables; we do not dwell on the details, but simply observe that $\lambda x.x(y) \ \sim_\alpha \ \lambda z.z(y)$ but $\lambda x.x(y) \ \not\sim_\alpha \ \lambda x.x(w)$. The result is called the *α-invariant pre-syntax*, and it is then shown to enjoy a restricted version of the induction principle of the pre-syntax.[9]

[9] Making the induction principles of α-invariant pre-syntax precise is highly technical; when variable names are drawn from some infinite collection $\mathbb{A}$, these notions can be made sense of using the machinery of *nominal sets* (Pitts, *Nominal Sets: Names and Symmetry in Computer Science*).

Well-formedness and definitional equality

Which of the α-invariant pre-contexts, α-invariant pre-types, etc. are well-formed? And when are they definitionally equal? These questions are resolved by *simultaneously* defining several families of relations in an inductive way, ranging over α-invariant pre-terms:[10]

$$\begin{aligned}
\mathbf{Ctx}_\Theta &\subseteq \mathbf{PreCtx}_\Theta/{\sim_\alpha} && \text{(contexts)}\\
\mathbf{Ty}_\Theta &\subseteq \mathbf{PreCtx}_\Theta/{\sim_\alpha} \times \mathbf{PreTy}_\Theta/{\sim_\alpha} && \text{(types)}\\
\mathbf{EqTy}_\Theta &\subseteq \mathbf{PreCtx}_\Theta/{\sim_\alpha} \times (\mathbf{PreTy}_\Theta/{\sim_\alpha})^2 && \text{(equal types)}\\
\mathbf{El}_\Theta &\subseteq \mathbf{PreCtx}_\Theta/{\sim_\alpha} \times \mathbf{PreTy}_\Theta/{\sim_\alpha} \times \mathbf{PreEl}_\Theta/{\sim_\alpha} && \text{(elements)}\\
\mathbf{EqEl}_\Theta &\subseteq \mathbf{PreCtx}_\Theta/{\sim_\alpha} \times \mathbf{PreTy}_\Theta/{\sim_\alpha} \times (\mathbf{PreEl}_\Theta/{\sim_\alpha})^2 && \text{(equal elements)}
\end{aligned}$$

[10] Definitional equality of contexts is sometimes omitted from and sometimes included in presentations of type theory.

Next, a mountain of crucial closure conditions are established for all of these relations; a few (but not nearly all) of these closure conditions are written below:

(Type presupposition) If $(\Gamma, A) \in \mathbf{Ty}_\Theta$, then $\Gamma \in \mathbf{Ctx}_\Theta$.

(Type equality presupposition) If $(\Gamma, (A_0, A_1)) \in \mathbf{EqTy}_\Theta$, then both $(\Gamma, A_i) \in \mathbf{Ty}_\Theta$.

(Type equivalence relation) Each fiber $\mathbf{EqTy}_\Theta(\Gamma, -)$ restricts to an equivalence relation on $\mathbf{Ty}_\Theta$.

(Element presupposition) If $(\Gamma, A, M) \in \mathbf{El}_\Theta$, then $(\Gamma, A) \in \mathbf{Ty}_\Theta$.

(Element equality presupposition) If $(\Gamma, A, (M_0, M_1)) \in \mathbf{El}_\Theta$, then both $(\Gamma, A, M_i) \in \mathbf{El}_\Theta$.

(Conversion) If $(\Gamma, (A_0, A_1)) \in \mathbf{EqTy}_\Theta$, then each fiber $\mathbf{El}(\Gamma, A_0, -) \subseteq \mathbf{El}(\Gamma, A_1, -)$.

(Type Weakening) If $(\Gamma, B) \in \mathbf{Ty}_\Theta$ and $(\Gamma \ldots \Delta, A) \in \mathbf{Ty}_{\Theta'}$ and $x \notin \Theta$, then $((\Gamma, x : B \ldots \Delta), A) \in \mathbf{Ty}_{\Theta' \cup \{x\}}$.

(Type Substitution) If $(\Gamma, B, N) \in \mathbf{El}_\Theta$ and $((\Gamma, x : B \ldots \Delta), A) \in \mathbf{Ty}_{\Theta'}$, then $(\Gamma \ldots [N/x]\Delta, [N/x]A) \in \mathbf{Ty}_{\Theta' \setminus \{x\}}$.

There are numerous other closure conditions which must be established, which we omit here. In different presentations of type theory, these closure conditions can be realized as either derivable rules or as admissible rules.[11]

[11] A *derivable* rule is one which can be obtained by repeated application of generating rules; an *admissible* rule is any closure condition which holds of the relation associated to the inductive definition of the rules. As such, every derivable rule is admissible, but not the other way around.

Model theory

Once the syntax of type theory has been defined in this way, it is necessary to verify that it generates a category of models and homomorphisms, for which it is sound and complete. In modern language, this is to say that an *initial object* in the category of models and homomorphisms can be obtained from the syntax of type theory itself. This is highly technical, and is almost never worked out in full detail; the main example of such a result appears in Streicher (*Semantics of Type Theory: Correctness, Completeness, and Independence Results*), but many details are left out.

Towards abstract syntax for type theory

The definitions of the pre-syntax, the α-invariant pre-syntax, and then the judgments of type theory are extremely technical. Defining them in a precise and rigorous way, if done in the style of the previous section, can easily take hundreds of pages of "morally obvious" but technically defeasible and fragile development—all of which is essentially impossible to check.

While some logicians and mathematicians have historically preferred this very concrete style, perhaps because it is familiar from the way that propositional and first-order logic are taught to undergraduates, type theorists have been forced to seek more abstract methods for defining their objects of study. Defining the syntax was already nearly intractable using the concrete style, and more difficult results such as completeness, canonicity, and normalization are essentially out of reach without the introduction of more abstract tools.

In these lectures, I am teaching *just one* of the optimal ways to present type theories abstractly, but I caution the reader against mistakenly assuming that other superior ways will not be obtained, or that there are not already reasonable alternatives.

Abstract treatment of binding and variables Some of the most subtle and unsatisfactory aspects of the syntax of type theory have to do with variable binding; substitution and renaming are highly technical, and it is practically impossible to work directly with α-equivalence classes in a precise way without introducing extremely technical machinery such as nominal sets (Pitts, *Nominal Sets: Names and Symmetry in Computer Science*).

An alternative to dealing with these technical issues is to use a more canonical representation of variable binding, in which variables are not drawn from an infinite collection of names $\mathbb{A}$, but are instead written as "pointers" to their binding site (De Bruijn indices). This approach is often characterized as being "more abstract" than using names, but I stress that it is in fact far more concrete in a certain sense: syntax, properly construed, is a language of certain graphs, in which back-edges indicate the binding site of variable nodes.[12]

While it is very hard to develop type theory in a precise way in the presence of variable names, it is extremely easy to translate back and forth between variable names and De Bruijn indices, even supporting things like name shadowing effort-

[12] Rather than thinking of De Bruijn indices as a technical device to obtain canonical representatives of α-equivalence classes, it is better to think of syntax with names as a technical device to obtain readable linear notation for term graphs!

lessly. Therefore, De Bruijn indices are a superior primitive, and variable names are explained through a trivial elaboration of concrete syntax to abstract syntax.

Explicit substitutions The next change that most type theorists immediately make when passing to the abstract is to delete the notion of substitution and renaming on pre-terms, and instead include a new sort $\Gamma \to \Delta$ for simultaneous substitutions, and new generators and equations in each sort for applying them. Equational laws are imposed which commute substitutions γ past the constructors of type theory; for instance, one adds rules like $(M, N)[\gamma] = (M[\gamma], N[\gamma])$.[13] In the presence of explicit substitutions, it is possible to use a simpler treatment of De Bruijn variables, in which the only generator is for the "last" variable in the context, and the others are obtained through explicit weakenings (which are instances of the explicit substitution discipline).

Explicit substitutions are often used in concrete implementations of type theory, so explaining these directly brings the metatheory of type theories closer to concrete applications. It is, however, a common mistake to think that the explicit substitutions are beneficial mostly in regard to implementation, or worse, that they are an instance of conflating the concrete with the abstract. Adding explicit substitutions to the syntax of type theory is primarily useful for the following reasons:

1. With explicit substitutions, it becomes possible to consider a version of the syntax of type theory which has no pre-terms. This is a massive improvement, and can save dozens of pages of work that had previously been required to develop each individual type theory.

2. The categories of models and homomorphisms generated by type theories defined with explicit substitutions are more restricted than otherwise, ruling out certain pathological models which express nothing but downsides; with implicit substitution, it is possible to consider categories of models in which only the *initial* model is required to enjoy substitution, whereas others may not.[14]

3. It becomes possible to compare *weak type theory* and *strict type theory* within the same discipline, as in Curien ("Substitution Up to Isomorphism") and Curien, Garner, and Hofmann ("Revisiting the categorical interpretation of dependent type theory")

Closure conditions and annotations Consider the closure conditions of the form "If $(\Gamma, A) \in \mathbf{Ty}_\Theta$ then $\Gamma \in \mathbf{Ctx}_\Theta$"; these are very subtle and bureaucratic to prove, and if the designer of a type theory is not careful, they may actually turn out to be impossible or extremely difficult to prove prior to establishing a normalization result — and, of course, normalization results are very difficult to obtain prior to establishing such closure conditions!

Establishing conversion principles (if A, B are equal types, then they have the same elements, etc.) is also subtle; in most cases, these must be added directly as

[13] It is well-known that the *most* naïve rewriting theory for λ-calculi with explicit substitutions is not confluent (Curien, Hardin, and Lévy, "Confluence Properties of Weak and Strong Calculi of Explicit Substitutions"). The ship of confluent rewriting, however, had already set sail the moment η-laws for dependent sum types were taken as standard; however, if η-laws are omitted, the appropriate *confluent* rewriting system is easily obtained using a very slight modification of the generators for substitutions (ibid.).

[14] An example of a semantic type theory which fails to have general substitution is Nuprl's Computational Type Theory (CTT), a consequence of the non-standard *pointwise functionality* employed in the interpretation of the type-theoretic judgments (Kopylov, "Type Theoretical Foundations for Data Structures, Classes and Objects"; Angiuli, "Computational Semantics of Cartesian Cubical Type Theory").

Categorical models, like locally Cartesian closed categories, also fail to model substitutions directly, for a quite different reason; this is easy to "fix" using simple strictification (Lumsdaine and Warren, "The Local Universes Model: An Overlooked Coherence Construction for Dependent Type Theories"; Hofmann, "On the interpretation of type theory in locally cartesian closed categories").

generating rules in the inductive definition, but sometimes they are admissible; choosing generators is, therefore, a bit of a black art. Zooming out, it is rather silly to deal with all these bureaucratic issues, because at a high level we want to define the following:

1. A collection of contexts, identified up to definitional equality.
2. A family of collections of types relative to contexts, identified up to definitional equality.
3. A family of collections of elements, relative to types, relative to contexts, identified up to definitional equality.

The difficulty of the above is to see that such a definition is in fact inductive: proving that a *specific thing* written in this style is well-defined will amount to re-staging the definition with pre-terms and complicated closure conditions, as above — so it seems like little is gained. Is there a way out?

In fact, it is possible to do all of this bureaucratic work *just once* for an extremely simple type theory without variable binding, and obtain from that a discipline that can be used to define extremely complicated type theories (which of course have binding structure) with very little effort. This discipline is called *generalized algebra*.

Generalized algebraic theories

Generalized algebraic theories are a notion of first-order algebraic theory (i.e. algebraic theory without binding) which has sorts that depend on terms;[15] depending on whom you ask, equations can be imposed between only terms (Taylor, *Practical Foundations of Mathematics*),[16] or between both sorts and terms (Cartmell, "Generalised Algebraic Theories and Contextual Categories"). For the sake of simplicity, we will focus on the latter notion, in which sort equations can be imposed.

The main benefit of generalized algebraic theories (GATs) is that each signature automatically gives rise to a category of algebras and homomorphisms, in a *functorial way* relative to (the opposite of) a category of generalized algebraic signatures and translations; the restrictions of categories of algebras induced by translations of signatures always have left adjoints, establishing the existence of *free algebras* for each theory. Moreover, these free algebras coincide with the Lindenbaum-Tarski algebras constructed directly out of the raw syntax of GATs.

This constitutes a general *initiality theorem* for the entire class of languages which can be defined in a generalized algebraic way, which includes *all* type theories which are of mathematical interest, subject to some terms and conditions[17] — not only "Book" Homotopy Type Theory (Univalent Foundations Program, *Homotopy Type Theory: Univalent Foundations of Mathematics*), but also more exotic type theories, such as cubical type theory (Cohen et al., "Cubical Type Theory: a constructive interpretation of the univalence axiom"; Angiuli et al., "Syntax and Models of Cartesian Cubical Type Theory").

[15] These are very similar to, and interchangeable in practice with, *essentially algebraic theories*, in which operations can be partially defined.

[16] Because sorts depend on terms, sort equality is still required — but one can forbid *generating* equations between sorts.

[17] What one must swallow is a brutal *annotation regime*: if type theory is formulated in a generalized algebraic style, operations are annotated by every premise of their formation rule! Furthermore, subtyping is not directly supported; this is in fact desirable, since at the semantic level, one wishes to consider models of subtyping via coercions (often coherent).

Generalized algebraic theories do not support object-level binding operators directly; instead, we give the syntax of type theory with explicit substitutions and a generic variable, simulating variable binding in a way that is very close to both semantics and concrete implementation.

We will not dwell on the technical details of how generalized algebraic theories and their universal algebra are developed: instead, we will learn from practice to work informally with these theories, in the next chapter.

Algebraic Type Theory

This section contains joint work with Daniel Gratzer. We rely heavily on notions developed in Kaposi, Kovács, and Thorsten Altenkirch ("Constructing Quotient Inductive-inductive Types") in the context of quotient inductive-inductive types *(QIITs), a modernized presentation of generalized algebraic theories.*

We will define a version of Martin-Löf's type theory in a generalized algebraic way, supporting a single universe of small types; it is very easy to extend our construction to an entire (algebraic) cumulative hierarchy of universes, which is explained in a paper under preparation.

Generalized algebraic theories

A generalized algebraic theory is defined by a collection of *generating rules* for sort operator formation, term operator formation, sort equality, and term equality; now, whether an extension to $\mathbb{T}$ by a rule is well-defined depends on the entire language generated by $\mathbb{T}$.

(premises)	Φ, Ψ	::=	$\cdot \mid \Phi, \mathfrak{x} : \mathcal{A}$
(instantiations)	ϕ, ψ	::=	$\cdot \mid \phi, \mathcal{M}/\mathfrak{x}$
(kinds)	$\mathbf{K}, \mathbf{L}$	::=	$\mathbf{sort} \mid \mathcal{A}$
(terms)	$\mathcal{M}, \mathcal{N}$	::=	$\mathfrak{x} \mid \mathsf{op}(\psi)$
(sorts)	$\mathcal{A}, \mathcal{B}$	::=	$\mathsf{op}(\psi)$
(rules)	$\mathcal{R}, \mathcal{S}$	::=	$[\mathsf{op} : \Psi \longrightarrow \mathbf{K}] \mid [\Psi \vdash \mathcal{M} = \mathcal{N} : \mathbf{K}]$
(theories)	$\mathbb{T}$	::=	$\varnothing \mid \mathbb{T}, \mathcal{R}$

Figure 1: The grammar of generalized algebraic theory. Metavariables $\mathfrak{x}$ stand for premises of inference rules.

All the syntactic categories of Figure 1 come equipped with meta-substitution actions for instantiations ϕ; for instance $\phi^* \mathcal{M}$ is the meta-substitution of the term $\mathcal{M}$ by the instantiation ϕ. We do not dwell on the details of meta-substitution, remarking only that (following **Thought 1**) it is easy to define as textual substitution due to the lack of binding structure.

Below we summarize the judgments and rules for forming GATs and their terms; we have not been totally fastidious about including every necessary rule, since we are more interested in cultivating informal intuition.

$\mathbb{T}$ *theory*

"$\mathbb{T}$ is a theory"

$$\frac{}{\mathbb{T}\ \textit{theory}} \qquad \frac{\mathbb{T}\ \textit{theory} \quad \mathcal{R}\ \textit{rule}_{\mathbb{T}}}{\mathbb{T}, \mathcal{R}\ \textit{theory}}$$

$\mathbb{T}$ *theory*

$\mathcal{R}\ \textit{rule}_{\mathbb{T}}$

"$\mathcal{R}$ is a well-formed rule in $\mathbb{T}$"

$$\frac{\Psi\ \textit{premises}_{\mathbb{T}} \quad \mathsf{op} \notin \mathbb{T} \quad \Psi \vdash_{\mathbb{T}} \mathbf{K}\ \textit{kind}}{[\mathsf{op} : \Psi \longrightarrow \mathbf{K}]\ \textit{rule}_{\mathbb{T}}} \qquad \frac{\Psi\ \textit{premises}_{\mathbb{T}} \quad \Psi \vdash_{\mathbb{T}} \mathbf{K}\ \textit{kind} \quad \Psi \vdash_{\mathbb{T}} \mathcal{M} : \mathbf{K} \quad \Psi \vdash_{\mathbb{T}} \mathcal{N} : \mathbf{K}}{[\Psi \vdash \mathcal{M} = \mathcal{N} : \mathbf{K}]\ \textit{rule}_{\mathbb{T}}}$$

$\mathbb{T}$ *theory*

$\Psi\ \textit{premises}_{\mathbb{T}}$

"Ψ are well-formed premises in theory $\mathbb{T}$"

$$\frac{}{\cdot\ \textit{premises}_{\mathbb{T}}} \qquad \frac{\Psi\ \textit{premises}_{\mathbb{T}} \quad \mathfrak{x} \notin \Psi \quad \Psi \vdash_{\mathbb{T}} \mathcal{A} : \mathbf{sort}}{\Psi, \mathfrak{x} : \mathcal{A}\ \textit{premises}_{\mathbb{T}}}$$

$\Psi\ \textit{premises}_{\mathbb{T}}$

$\Psi \vdash_{\mathbb{T}} \mathbf{K}\ \textit{kind}$

"$\mathbf{K}$ is a well-formed kind under premises Ψ"

$\Psi \vdash_{\mathbb{T}} \mathbf{L}\ \textit{kind}$ $\quad$ $\Psi \vdash_{\mathbb{T}} \mathbf{K}\ \textit{kind}$

$\Psi \vdash_{\mathbb{T}} \mathbf{L} = \mathbf{K}\ \textit{kind}$

"$\mathbf{L}$ and $\mathbf{K}$ are equal kinds"

$$\frac{\mathbb{T}\ \textit{theory} \quad \Psi\ \textit{premises}_{\mathbb{T}}}{\Psi \vdash_{\mathbb{T}} \mathbf{sort}\ \textit{kind}} \qquad \frac{\mathbb{T}\ \textit{theory} \quad \Psi\ \textit{premises}_{\mathbb{T}} \quad \Psi \vdash_{\mathbb{T}} \mathcal{A} : \mathbf{sort}}{\Psi \vdash_{\mathbb{T}} \mathcal{A}\ \textit{kind}}$$

$$\frac{\mathbb{T}\ \textit{theory} \quad \Psi\ \textit{premises}_{\mathbb{T}} \quad \Psi \vdash_{\mathbb{T}} \mathcal{A} = \mathcal{B} : \mathbf{sort}}{\Psi \vdash_{\mathbb{T}} \mathcal{A} = \mathcal{B}\ \textit{kind}}$$

$$\frac{\Psi \vdash_{\mathbb{T}} \mathbf{K}\ \mathit{kind}}{\Psi \vdash_{\mathbb{T}} \mathcal{M} : \mathbf{K}}$$

"$\mathcal{M}$ is a term of kind $\mathbf{K}$"

$$\frac{\Psi \vdash_{\mathbb{T}} \mathcal{M} : \mathbf{K} \qquad \Psi \vdash_{\mathbb{T}} \mathcal{N} : \mathbf{K}}{\Psi \vdash_{\mathbb{T}} \mathcal{M} = \mathcal{N} : \mathbf{K}}$$

"$\mathcal{M}$ and $\mathcal{N}$ are equal terms of kind $\mathbf{K}$"

$$\frac{[\mathsf{op} : \Phi \longrightarrow \mathbf{K}] \in \mathbb{T} \quad \Psi \vdash_{\mathbb{T}} \phi : \Phi}{\Psi \vdash_{\mathbb{T}} \mathsf{op}(\phi) : \mathbf{K}} \qquad \frac{\Psi \vdash_{\mathbb{T}} \mathbf{L} = \mathbf{K}\ \mathit{kind} \quad \Psi \vdash_{\mathbb{T}} \mathcal{M} : \mathbf{L}}{\Psi \vdash_{\mathbb{T}} \mathcal{M} : \mathbf{K}} \qquad \frac{\Phi \vdash_{\mathbb{T}} \mathcal{M} : \mathbf{K} \quad \Psi \vdash_{\mathbb{T}} \phi : \Phi}{\Psi \vdash_{\mathbb{T}} \psi^* \mathcal{M} : \phi^* \mathbf{K}}$$

$$\frac{\Psi \vdash_{\mathbb{T}} \mathbf{L} = \mathbf{K}\ \mathit{kind} \quad \Psi \vdash_{\mathbb{T}} \mathcal{M} = \mathcal{N} : \mathbf{L}}{\Psi \vdash_{\mathbb{T}} \mathcal{M} = \mathcal{N} : \mathbf{K}} \qquad \frac{[\Phi \vdash \mathcal{M} = \mathcal{N} : \mathbf{K}] \in \mathbb{T} \quad \Psi \vdash_{\mathbb{T}} \phi : \Phi}{\Psi \vdash_{\mathbb{T}} \phi^* \mathcal{M} = \phi^* \mathcal{N} : \phi^* \mathbf{K}}$$

$$\frac{\Psi\ \mathit{premises}_{\mathbb{T}} \qquad \Phi\ \mathit{premises}_{\mathbb{T}}}{\Psi \vdash_{\mathbb{T}} \phi : \Phi}$$

"ϕ is an instantiation Φ in context Ψ"

$$\frac{\Psi \vdash_{\mathbb{T}} \phi : \Phi \qquad \Psi \vdash_{\mathbb{T}} \phi' : \Phi}{\Psi \vdash_{\mathbb{T}} \phi = \phi' : \Phi}$$

"ϕ and ϕ' are equal instantiations of Φ in context Ψ"

$$\frac{}{\Psi \vdash_{\mathbb{T}} \cdot : \cdot} \qquad \frac{\Psi \vdash_{\mathbb{T}} \phi : \Phi \quad \Psi \vdash_{\mathbb{T}} \mathcal{M} : \phi^* \mathcal{A}}{\Psi \vdash_{\mathbb{T}} (\phi, \mathcal{M}/\mathfrak{x}) : \Phi, \mathfrak{x} : \mathcal{A}}$$

Notation for theories

As a warm-up, we will begin to define the generalized algebraic theory of (strict) categories:

$$\begin{aligned}
&\varnothing, [\mathsf{ob} : \cdot \longrightarrow \mathbf{K}], [\mathsf{hom} : (\cdot, \mathfrak{s} : \mathsf{ob}(\cdot), \mathfrak{t} : \mathsf{ob}(\cdot)) \longrightarrow \mathbf{sort}],\\
&[\mathsf{id} : (\cdot, \mathfrak{x} : \mathsf{ob}(\cdot)) \longrightarrow \mathsf{hom}(\cdot, \mathfrak{x}/\mathfrak{s}, \mathfrak{x}/\mathfrak{t})],\\
&[\mathsf{cmp} : (\cdot, \mathfrak{x} : \mathsf{ob}(\cdot), \mathfrak{y} : \mathsf{ob}(\cdot), \mathfrak{z} : \mathsf{ob}(\cdot), \mathfrak{g} : \mathsf{hom}(\cdot, \mathfrak{y}/\mathfrak{s}, \mathfrak{z}/\mathfrak{t}), \mathfrak{f} : \mathsf{hom}(\cdot, \mathfrak{x}/\mathfrak{s}, \mathfrak{y}/\mathfrak{t})) \longrightarrow \mathsf{hom}(\cdot, \mathfrak{x}/\mathfrak{s}, \mathfrak{z}/\mathfrak{t})],\\
&[\cdot, \mathfrak{x} : \mathsf{ob}(\cdot), \mathfrak{y} : \mathsf{ob}(\cdot), \mathfrak{f} : \mathsf{hom}(\cdot, \mathfrak{x}/\mathfrak{s}, \mathfrak{y}/\mathfrak{t}) \vdash \mathsf{cmp}(\cdot, \mathfrak{x}/\mathfrak{x}, \mathfrak{x}/\mathfrak{y}, \mathfrak{y}/\mathfrak{z}, \mathfrak{f}/\mathfrak{g}, \mathsf{id}(\cdot, \mathfrak{x}/\mathfrak{x})/\mathfrak{f}) = \mathfrak{f} : \mathsf{hom}(\cdot, \mathfrak{x}/\mathfrak{s}, \mathfrak{y}/\mathfrak{t})],\\
&\ldots
\end{aligned}$$

We have not got very far, but already the notation is too gruesome to continue. For this reason, we adopt an informal notation for writing the generating rules of a generalized algebraic theory:[18]

$$\frac{}{\mathsf{ob} : \mathbf{sort}}\ \boxed{\mathsf{ob}} \qquad \frac{\Delta, \Gamma : \mathsf{ob}}{\Delta \to \Gamma : \mathbf{sort}}\ \boxed{\mathsf{hom}} \qquad \frac{\Gamma : \mathsf{ob}}{\mathsf{id}_\Gamma : \Gamma \to \Gamma}\ \boxed{\mathsf{id}}$$

$$\frac{\Xi, \Delta, \Gamma : \mathsf{ob} \quad \gamma : \Delta \to \Gamma \quad \delta : \Xi \to \Delta}{\gamma \circ \delta : \Xi \to \Gamma}\ \boxed{\mathsf{cmp}} \qquad \frac{\Delta, \Gamma : \mathsf{ob} \quad \gamma : \Delta \to \Gamma}{\mathsf{id}_\Gamma \circ \gamma = \gamma : \Delta \to \Gamma}\ \boxed{=}$$

[18] Observe how these rules look very similar to ordinary inference rules, except that they are very constrained: their premises must all take the form of non-repeated declarations $\mathfrak{x} : \mathcal{A}$, etc.

$$\frac{\Delta, \Gamma : \mathsf{ob} \quad \gamma : \Delta \to \Gamma}{\gamma \circ \mathsf{id}_\Delta = \gamma : \Delta \to \Gamma}\,(=)$$

$$\frac{\mathrm{H}, \Xi, \Delta, \Gamma : \mathsf{ob} \quad \xi : \mathrm{H} \to \Xi \quad \delta : \Xi \to \Delta \quad \gamma : \Delta \to \Gamma}{(\gamma \circ \delta) \circ \xi = \gamma \circ (\delta \circ \xi) : \mathrm{H} \to \Gamma}\,(=)$$

Algebras for a generalized algebraic theory

We do not dwell on the technicalities of algebras for a theory, but only remark that it works intuitively: a closed sort $\mathcal{A}$ is interpreted as a set, and a term of sort $\mathcal{A}$ is interpreted as an element of that set; a dependent sort $\mathfrak{x} : \mathcal{A} \vdash_{\mathbb{T}} \mathcal{B} : \mathbf{sort}$ is interpreted as a family of sets indexed in the set assigned to $\mathcal{A}$, etc. The model theory of GATs is still underdeveloped: only set-theoretic models are considered, and while a functorial semantics is claimed in Cartmell ("Generalised Algebraic Theories and Contextual Categories"), it has not been developed in a precise way. We will be constructing algebras for generalized algebraic theories in an informal way only, conscious of the current limitations.

For a $\mathbb{T}$-algebra $\mathcal{M}$, we summarize the way the interpretation works in below, letting $\mathcal{V}_\alpha < \mathcal{V}_\beta$ be Grothendieck universes:

$$\begin{aligned}
\left([\mathsf{op} : \Psi \longrightarrow \mathbf{K}]\ \mathit{rule}_{\mathbb{T}}\right)^{\mathcal{M}} &\in \textstyle\prod_{\psi_0^{\mathcal{M}} \in \Psi^{\mathcal{M}}} \mathbf{K}^{\mathcal{M}}(\psi_0^{\mathcal{M}}) \\
\left(\Psi\ \mathit{premises}_{\mathbb{T}}\right)^{\mathcal{M}} &\in \mathcal{V}_\beta \\
\left(\Psi \vdash_{\mathbb{T}} \mathbf{K}\ \mathit{kind}\right)^{\mathcal{M}} &\in \Psi^{\mathcal{M}} \to \mathcal{V}_\beta \\
\left(\Psi \vdash_{\mathbb{T}} \mathcal{M} : \mathbf{K}\right)^{\mathcal{M}} &\in \textstyle\prod_{\psi_0^{\mathcal{M}} \in \Psi^{\mathcal{M}}} \mathbf{K}^{\mathcal{M}}(\psi_0^{\mathcal{M}}) \\
\left(\Psi \vdash_{\mathbb{T}} \phi : \Phi\right)^{\mathcal{M}} &\in \Psi^{\mathcal{M}} \to \Phi^{\mathcal{M}} \\
\left(\cdot\ \mathit{premises}_{\mathbb{T}}\right)^{\mathcal{M}} &= \{*\} \\
\left(\Psi, \mathfrak{x} : \mathcal{A}\ \mathit{premises}_{\mathbb{T}}\right)^{\mathcal{M}} &= \textstyle\sum_{\psi_0^{\mathcal{M}} \in \Psi^{\mathcal{M}}} \mathcal{A}^{\mathcal{M}} \psi_0^{\mathcal{M}} \\
\left(\Psi \vdash_{\mathbb{T}} \cdot : \cdot\right)^{\mathcal{M}} &= \lambda \psi_0^{\mathcal{M}}.\, * \\
\left(\Psi \vdash_{\mathbb{T}} (\phi, \mathcal{N})/\mathfrak{x} : \Phi, \mathfrak{x} : \mathcal{A}\right)^{\mathcal{M}} &= \lambda \psi_0^{\mathcal{M}}.\, (\phi^{\mathcal{M}} \psi_0^{\mathcal{M}}, \mathcal{N}^{\mathcal{M}} \psi_0^{\mathcal{M}}) \\
\left(\Psi \vdash_{\mathbb{T}} \mathbf{sort}\ \mathit{kind}\right)^{\mathcal{M}} &= \lambda \psi_0^{\mathcal{M}}.\, \mathcal{V}_\alpha \\
\left(\Psi \vdash_{\mathbb{T}} \mathcal{A}\ \mathit{kind}\right)^{\mathcal{M}} &= \left(\Psi \vdash_{\mathbb{T}} \mathcal{A} : \mathbf{sort}\right)^{\mathcal{M}} \\
\left(\Psi \vdash_{\mathbb{T}} \mathsf{op}(\phi) : \mathbf{K}\right)^{\mathcal{M}} &= \lambda \psi_0^{\mathcal{M}}.\, \mathsf{op}^{\mathcal{M}}(\phi^{\mathcal{M}}(\psi_0^{\mathcal{M}})) \\
\left(\Psi, \mathfrak{x} : \mathcal{A} \vdash_{\mathbb{T}} \mathfrak{x} : \mathcal{A}\right)^{\mathcal{M}} &= \lambda \psi_0^{\mathcal{M}}.\, \pi_2 \psi_0^{\mathcal{M}} \\
\left(\Psi, \mathfrak{y} : \mathcal{B} \vdash_{\mathbb{T}} \mathfrak{x} : \mathcal{A}\right)^{\mathcal{M}} &= \lambda \psi_0^{\mathcal{M}}.\, \mathfrak{x}^{\mathcal{M}}(\pi_1(\psi_0^{\mathcal{M}}))
\end{aligned}$$

The entire interpretation is mutual, but the only data which must be supplied is the ground interpretations of the operators $[\mathsf{op} : \Psi \longrightarrow \mathbf{K}]$.

Type theory as a generalized algebraic theory

The syntax of the current version of Martin-Löf's type theory[19] is generated in

[19] For instance, the version presented in Dybjer ("Internal type theory") and Granström (*Treatise on Intuitionistic Type Theory*).

a way that resembles a strict version of the semantic categories-with-families (cwf) construction. The generalized algebraic theory of type theory therefore begins with a category of contexts and substitutions; we therefore start with the theory of strict categories that we gave above, renaming ob to ctx. We will require a terminal context:[20]

$$\frac{}{\cdot : \mathsf{ctx}}\ \texttt{emp} \qquad \frac{\Gamma : \mathsf{ctx}}{!_\Gamma : \Gamma \to \cdot}\ \texttt{emp/subst} \qquad \frac{\Gamma : \mathsf{ctx} \quad \eta : \Gamma \to \cdot}{\eta = _\Gamma : \Gamma \to \cdot}\ \texttt{=}$$

Ordinarily, we would then proceed to define the context-dependent sort of types, and the type-dependent sort of elements; in order to support a universe, we will do something slightly more general.[21] We begin by adding to our signature the data of a partially ordered set of *sizes*:[22]

$$\frac{}{\mathsf{size} : \textbf{sort}}\ \texttt{size} \qquad \frac{i, j : \mathsf{size}}{i \le j : \textbf{sort}}\ \texttt{ord} \qquad \frac{i, j : \mathsf{size} \quad p, q : i \le j}{p = q : i \le j}\ \texttt{=}$$

$$\frac{i : \mathsf{size}}{\star : i \le i}\ \texttt{size/refl} \qquad \frac{i, j, k : \mathsf{size} \quad p : i \le j \quad q : j \le k}{\star : i \le k}\ \texttt{size/trans}$$

In our running example, we will consider only two sizes; but the method which we are presenting scales up to an arbitrarily complex hierarchy of universes, including countable and even larger transfinite hierarchies.[23]

$$\frac{}{0 : \mathsf{size}}\ \texttt{small} \qquad \frac{}{1 : \mathsf{size}}\ \texttt{large} \qquad \frac{i : \mathsf{size}}{\star : i \le 1}\ \texttt{enlarge}$$

$$\frac{i : \mathsf{size} \quad p : 1 \le i}{i = 1 : \mathsf{size}}\ \texttt{=}$$

Next, we will specify the sort of types of a particular size; we require both a substitution action and a size-lifting action, which we will define simultaneously:[24]

$$\frac{\Gamma : \mathsf{ctx} \quad i : \mathsf{size}}{\mathsf{Ty}_i(\Gamma) : \textbf{sort}}\ \texttt{ty}$$

$$\frac{\Delta, \Gamma : \mathsf{ctx} \quad i, j : \mathsf{size} \quad A : \mathsf{Ty}_i(\Gamma) \quad \gamma : \Delta \to \Gamma \quad p : i \le j}{A[\gamma]_{|j} : \mathsf{Ty}_j(\Delta)}\ \texttt{ty/act}$$

Notation. We will write $A[\gamma]$ for $A[\gamma]_{|i}$ when i can easily be inferred from context; likewise, we write $A_{|i}$ for $A[\mathsf{id}_\Gamma]_{|i}$.

The functoriality of Ty is completed by the following rules:

$$\frac{\Gamma : \mathsf{ctx} \quad i : \mathsf{size} \quad A : \mathsf{Ty}_i(\Gamma)}{A[\mathsf{id}_\Gamma]_{|i} = A : \mathsf{Ty}_i(\Gamma)}\ \texttt{=}$$

[20] We note that these rules are only a textual alternative to the informal statement "The category ctx has a terminal object", which (like most categorical statements) is generalized algebraic, and therefore corresponds to an extension of the current theory by some rules.

[21] Our notion of universe lies somewhere between "à la Russell" and "à la Tarski": we avoid duplication between codes and types, but nonetheless express a lifting from small types to large types.

[22] A common trick in generalized algebraic theories is to define a predicate using a sort of proofs of the predicate, together with an equational axiom to make them coherent. This also justifies the use of the notation $\star$ for each generator, because (up to definitional equality) there is always at most one term of the sort.

[23] In other words, we extend our theory with the data of the "interval category".

[24] In mathematical terms, this corresponds to extending our theory with the data of a presheaf $\mathsf{Ty} : \mathbf{Psh}(\mathsf{ctx} \times \mathsf{size}^{\mathsf{op}})$.

$$\frac{\Xi, \Delta, \Gamma : \mathsf{ctx} \quad \gamma : \Delta \to \Gamma \quad \delta : \Xi \to \Delta \quad i, j, k : \mathsf{size} \quad p : i \leq j \quad q : j \leq k \quad A : \mathsf{Ty}_i(\Gamma)}{A[\gamma \circ \delta]_{|k} = A[\gamma]_{|j}[\delta]_{|k} : \mathsf{Ty}_j(\Xi)}\ (=)$$

Next, we define the sort of elements, dependent on the sort of *large types*![25]

$$\frac{\Gamma : \mathsf{ctx} \quad A : \mathsf{Ty}_1(\Gamma)}{\mathsf{El}(\Gamma \vdash A) : \mathbf{sort}}\ \texttt{el}$$

$$\frac{\Delta, \Gamma : \mathsf{ctx} \quad \gamma : \Delta \to \Gamma \quad A : \mathsf{Ty}_1(\Gamma) \quad M : \mathsf{El}(\Gamma \vdash A)}{M[\gamma] : A[\gamma]}\ \texttt{el/act}$$

$$\frac{\Gamma : \mathsf{ctx} \quad A : \mathsf{Ty}_1(\Gamma) \quad M : \mathsf{El}(\Gamma \vdash A)}{M[\mathsf{id}_\Gamma] = M : A}\ (=)$$

$$\frac{\Xi, \Delta, \Gamma : \mathsf{ctx} \quad \gamma : \Delta \to \Gamma \quad \delta : \Xi \to \Delta \quad A : \mathsf{Ty}_1(\Gamma) \quad M : \mathsf{El}(\Gamma \vdash A)}{M[\gamma \circ \delta] = M[\gamma][\delta] : A[\gamma \circ \delta]}\ (=)$$

[25] In mathematical terms, this is to fix a dependent presheaf $\mathsf{El} : \mathbf{Psh}(\mathsf{ctx})/\mathsf{Ty}_1$, i.e. a natural transformation $\mathsf{El} \xrightarrow{\pi} \mathsf{Ty}_1$.

Context comprehension

We next a *context comprehension* for each large type:[26]

$$\frac{\Gamma : \mathsf{ctx} \quad A : \mathsf{Ty}_1(\Gamma)}{\Gamma.A : \mathsf{ctx}}\ \texttt{ext} \qquad \frac{\Gamma : \mathsf{ctx} \quad A : \mathsf{Ty}_1(\Gamma)}{\mathbf{p}_A : \Gamma.A \to \Gamma}\ \texttt{drop}$$

$$\frac{\Gamma : \mathsf{ctx} \quad A : \mathsf{Ty}_1(\Gamma)}{\mathbf{q}_A : \mathsf{El}(\Gamma.A \vdash A[\mathbf{p}_A])}\ \texttt{var}$$

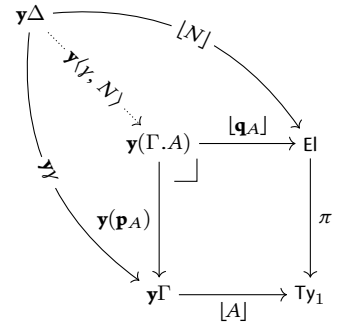
$$\frac{\Delta, \Gamma : \mathsf{ctx} \quad A : \mathsf{Ty}_1(\Gamma) \quad \gamma : \Delta \to \Gamma \quad N : \mathsf{El}(\Delta \vdash A[\gamma])}{\langle \gamma, N \rangle : \Delta \to \Gamma.A}\ \texttt{snoc}$$

$$\frac{\Delta, \Gamma : \mathsf{ctx} \quad A : \mathsf{Ty}_1(\Gamma) \quad \eta : \Delta \to \Gamma.A}{\eta = \langle \mathbf{p}_A \circ \eta, \mathbf{q}_A[\eta] \rangle : \Delta \to \Gamma.A}\ (=)$$

$$\frac{\Delta, \Gamma : \mathsf{ctx} \quad A : \mathsf{Ty}_1(\Gamma) \quad \gamma : \Delta \to \Gamma \quad N : \mathsf{El}(\Delta \vdash A[\gamma])}{\mathbf{p}_A \circ \langle \gamma, N \rangle = \gamma : \Delta \to \Gamma}\ (=)$$

$$\frac{\Delta, \Gamma : \mathsf{ctx} \quad A : \mathsf{Ty}_1(\Gamma) \quad \gamma : \Delta \to \Gamma \quad N : \mathsf{El}(\Delta \vdash A[\gamma])}{\mathbf{q}_A[\langle \gamma, N \rangle] = N : \mathsf{El}(\Delta \vdash A[\gamma])}\ (=)$$

[26] All these rules correspond to the data of a *representability structure* for the natural transformation $\mathsf{El} \xrightarrow{\pi} \mathsf{Ty}_1$ in the sense of Fiore (*Discrete generalised polynomial functors*) and Awodey ("A cubical model of homotopy type theory"). It provides a choice of representable pullback of π along any natural transformation out of a representable object:



Universe of small types

A universe of *small* types can be added easily using a sort equation:

$$\frac{\Gamma : \mathsf{ctx}}{\mathcal{U} : \mathsf{Ty}_1(\Gamma)}\ \boxed{\texttt{univ}} \qquad \frac{\Delta, \Gamma : \mathsf{ctx} \quad \gamma : \Delta \to \Gamma}{\mathcal{U}[\gamma]_{|1} = \mathcal{U} : \mathsf{Ty}_i(\Gamma)}\ \boxed{=}$$

$$\frac{\Gamma : \mathsf{ctx}}{\mathsf{El}(\Gamma \vdash \mathcal{U}) = \mathsf{Ty}_0(\Gamma) : \textbf{sort}}\ \boxed{=}$$

Our final sort equation, which equates the elements of the universe with the small types, gives the flavor a universes à la Rusell. The notion of an "element" is defined only for large types, however, so in order to consider the elements of a small type $A \in \mathsf{El}(\Gamma \vdash \mathcal{U})$, one must lift it $A_{|1}$; in this way, the level shift plays a similar role to the decoding operation in traditional presentations of universes à la Tarski. The main difference is that we do not need to provide syntax and rules for both codes and their meanings; instead, we do both simultaneously.

Dependent product types

$$\frac{\Gamma : \mathsf{ctx} \quad i : \mathsf{size} \quad A : \mathsf{Ty}_i(\Gamma) \quad B : \mathsf{Ty}_i(\Gamma.A_{|1})}{\Pi(A, B) : \mathsf{Ty}_i(\Gamma)}\ \boxed{\texttt{pi}}$$

$$\frac{\Delta, \Gamma : \mathsf{ctx} \quad i, j : \mathsf{size} \quad \gamma : \Delta \to \Gamma \quad p : i \leq j \quad A : \mathsf{Ty}_i(\Gamma) \quad B : \mathsf{Ty}_i(\Gamma.A_{|1})}{\Pi(A, B)[\gamma]_{|j} = \Pi(A[\gamma]_{|j}, B[\langle \gamma \circ \mathbf{p}_{A[\gamma]_{|j}}, \mathbf{q}_{A[\gamma]_{|j}} \rangle]_{|j}) : \mathsf{Ty}_j(\Gamma)}\ \boxed{=}$$

$$\frac{\Gamma : \mathsf{ctx} \quad A : \mathsf{Ty}_1(\Gamma) \quad B : \mathsf{Ty}_1(\Gamma.A) \quad M : \mathsf{El}(\Gamma.A \vdash B)}{\lambda(M) : \mathsf{El}(\Gamma \vdash \Pi(A, B))}\ \boxed{\texttt{lam}}$$

$$\frac{\Delta, \Gamma : \mathsf{ctx} \quad \gamma : \Delta \to \Gamma \quad A : \mathsf{Ty}_1(\Gamma) \quad B : \mathsf{Ty}_1(\Gamma.A) \quad M : \mathsf{El}(\Gamma.A \vdash B)}{\lambda(M)[\gamma] = \lambda(M[\langle \gamma \circ \mathbf{p}_{A[\gamma]}, \mathbf{q}_{A[\gamma]} \rangle]) : \mathsf{El}(\Delta \vdash \Pi(A, B)[\gamma])}\ \boxed{=}$$

$$\frac{\Gamma : \mathsf{ctx} \quad A : \mathsf{Ty}_1(\Gamma) \quad B : \mathsf{Ty}_1(\Gamma.A) \quad M : \mathsf{El}(\Gamma \vdash \Pi(A, B)) \quad N : \mathsf{El}(\Gamma \vdash A)}{\mathsf{app}(M, N) : \mathsf{El}(\Gamma \vdash \Pi(A, B))}\ \boxed{\mathsf{app}}$$

$$\frac{\begin{array}{l}\Delta, \Gamma : \mathsf{ctx} \quad \gamma : \Delta \to \Gamma \\ A : \mathsf{Ty}_1(\Gamma) \quad B : \mathsf{Ty}_1(\Gamma.A) \quad M : \mathsf{El}(\Gamma \vdash \Pi(A, B)) \quad N : \mathsf{El}(\Gamma \vdash A)\end{array}}{\mathsf{app}(M, N)[\gamma] = \mathsf{app}(M[\gamma], N[\gamma]) : \mathsf{El}(\Delta \vdash \Pi(A, B)[\gamma])}\ \boxed{=}$$

$$\frac{\Gamma : \mathsf{ctx} \quad A : \mathsf{Ty}_1(\Gamma) \quad B : \mathsf{Ty}_1(\Gamma.A) \quad M : \mathsf{El}(\Gamma.A \vdash B) \quad N : \mathsf{El}(\Gamma \vdash A)}{\mathsf{app}(\lambda(M), N) = M[\langle \mathsf{id}_\Gamma, N \rangle] : B[\langle \mathsf{id}_\Gamma, N \rangle]}\ \boxed{=}$$

$$\frac{\Gamma : \mathsf{ctx} \quad A : \mathsf{Ty}_1(\Gamma) \quad B : \mathsf{Ty}_1(\Gamma.A) \quad M : \mathsf{El}(\Gamma \vdash \Pi(A, B))}{M = \lambda(M[\mathbf{p}_A]) : \mathsf{El}(\Gamma \vdash \Pi(A, B))}\ \boxed{=}$$

A base type

We add a base type with two constants; our reason for doing this is to have some constants with which to state our canonicity theorem.

$$\frac{\Gamma : \mathsf{ctx} \quad i : \mathsf{size}}{\mathsf{base} : \mathsf{Ty}_i(\Gamma)}\ \boxed{\mathsf{base}} \qquad \frac{\Delta, \Gamma : \mathsf{ctx} \quad \gamma : \Delta \to \Gamma \quad i, j : \mathsf{size} \quad p : i \leq j}{\mathsf{base}[\gamma]_{|j} = \mathsf{base} : \mathsf{Ty}_j(\Delta)}\ \boxed{=}$$

$$\frac{\Gamma : \mathsf{ctx}}{\mathsf{c0} : \mathsf{El}(\Gamma \vdash \mathsf{base})}\ \boxed{\mathsf{c0}} \qquad \frac{\Gamma : \mathsf{ctx}}{\mathsf{c1} : \mathsf{El}(\Gamma \vdash \mathsf{base})}\ \boxed{\mathsf{c1}}$$

$$\frac{\Delta, \Gamma : \mathsf{ctx} \quad \gamma : \Delta \to \Gamma}{\mathsf{c0}[\gamma] = \mathsf{c0} : \mathsf{El}(\Delta \vdash \mathsf{base})}\ \boxed{=} \qquad \frac{\Delta, \Gamma : \mathsf{ctx} \quad \gamma : \Delta \to \Gamma}{\mathsf{c1}[\gamma] = \mathsf{c1} : \mathsf{El}(\Delta \vdash \mathsf{base})}\ \boxed{=}$$

We do not impose any elimination rule or equational rules.

The gluing construction

How do you express a closure property for the syntax of type theory? Most closure properties of interest (like canonicity, normalization, etc.) are too complex to prove directly on the syntax of type theory, but they can be proved by "threading the syntax of type theory through its semantics", which is precisely what older proof theoretic techniques (like logical predicates, or the method of computability) do.

Categorical gluing is a more general and more abstract version of logical predicates which is very harmonious with the algebraic perspective that we have developed here. Whereas the fundamental theorem of logical relations is something quite involved to prove, the equivalent of this theorem in the context of categorical gluing is usually automatic, and the only thing that matters is the definitions.[27]

[27] This might appear to violate the *Law of Conservation of Effort*, but it doesn't: all the effort has been factored into the very general algebraic framework and the categories of models which it generates.

Dependent algebras for a theory

Previously, we summarized the data of an ordinary algebra for a theory. Relative to such an ordinary algebra $\mathcal{M}^{\LEFTcircle}$, we now specify some of the data of a *dependent algebra* $\mathcal{M}^{\RIGHTcircle}$ over $\mathcal{M}^{\LEFTcircle}$, simultaneously defining the *total algebra* $\mathcal{M}^{\bullet}$.[28] We assume $(\Psi \vdash_{\mathbb{T}} \textbf{sort}\ \mathit{kind})^{\LEFTcircle}\psi^{\LEFTcircle} = \mathcal{V}_\alpha$. Only some of the details are given below.

[28] The notion of a *dependent algebra* is due to Kaposi, Kovács, and Thorsten Altenkirch ("Constructing Quotient Inductive-inductive Types"), who use the name *displayed algebra*. Dependent algebras are well-suited for type theorists, who generally prefer to work fiber-wise.

$$
\begin{aligned}
([\mathsf{op} : \Psi \longrightarrow \mathbf{K}]\ \mathit{rule}_{\mathbb{T}})^{\RIGHTcircle} &\in \textstyle\prod_{\psi^{\bullet}\in\Psi^{\bullet}} \mathbf{K}^{\RIGHTcircle}\psi^{\bullet}(\mathsf{op}^{\LEFTcircle}\psi^{\LEFTcircle}) \\
\left(\Psi\ \mathit{premises}_{\mathbb{T}}\right)^{\RIGHTcircle} &\in \Psi^{\LEFTcircle} \to \mathcal{V}_\beta \\
(\Psi \vdash_{\mathbb{T}} \mathbf{K}\ \mathit{kind})^{\RIGHTcircle} &\in \textstyle\prod_{\psi^{\bullet}\in\Psi^{\bullet}} (\mathbf{K}^{\LEFTcircle}\psi^{\LEFTcircle} \to \mathcal{V}_\beta) \\
(\Psi \vdash_{\mathbb{T}} \mathcal{M} : \mathbf{K})^{\RIGHTcircle} &\in \textstyle\prod_{\psi^{\bullet}\in\Psi^{\bullet}} \mathbf{K}^{\RIGHTcircle}\psi^{\bullet}(\mathcal{M}^{\LEFTcircle}\psi^{\LEFTcircle}) \\
(\Psi \vdash_{\mathbb{T}} \phi : \Phi)^{\RIGHTcircle} &\in \textstyle\prod_{\psi^{\bullet}\in\Psi^{\bullet}} \Phi^{\RIGHTcircle}(\phi^{\LEFTcircle}\psi^{\LEFTcircle}) \\
(\Psi \vdash_{\mathbb{T}} \textbf{sort}\ \mathit{kind})^{\RIGHTcircle}\psi^{\bullet}\mathcal{A}^{\LEFTcircle} &= A^{\LEFTcircle} \to \mathcal{V}_\alpha \\
(\Psi \vdash_{\mathbb{T}} \mathcal{A}\ \mathit{kind})^{\RIGHTcircle}\psi^{\bullet} &= (\Psi \vdash_{\mathbb{T}} \mathcal{A} : \textbf{sort})^{\RIGHTcircle}\psi^{\bullet} \\
(\Psi \vdash_{\mathbb{T}} \mathsf{op}(\phi) : \mathbf{K})^{\RIGHTcircle}\psi^{\bullet} &= \mathsf{op}^{\RIGHTcircle}(\phi^{\bullet}(\psi^{\bullet})) \\
\left(\Psi\ \mathit{premises}_{\mathbb{T}}\right)^{\bullet} &= \textstyle\sum_{\psi^{\LEFTcircle}\in\Psi^{\LEFTcircle}} \Psi^{\RIGHTcircle}\psi^{\LEFTcircle} \\
(\Psi \vdash_{\mathbb{T}} \mathbf{K}\ \mathit{kind})^{\bullet}\psi^{\bullet} &= \textstyle\sum_{\mathcal{M}^{\LEFTcircle}\in\mathbf{K}^{\LEFTcircle}\psi^{\LEFTcircle}} \mathbf{K}^{\RIGHTcircle}\psi^{\bullet}\mathcal{M}^{\LEFTcircle} \\
(\Psi \vdash_{\mathbb{T}} \mathcal{M} : \mathbf{K})^{\bullet}\psi^{\bullet} &= (\mathcal{M}^{\LEFTcircle}\psi^{\LEFTcircle}, \mathcal{M}^{\RIGHTcircle}\psi^{\bullet}) \\
(\Psi \vdash_{\mathbb{T}} \phi : \Phi)^{\bullet}\psi^{\bullet} &= (\phi^{\LEFTcircle}\psi^{\LEFTcircle}, \phi^{\RIGHTcircle}\psi^{\bullet})
\end{aligned}
$$

Canonicity for type theory

Proposition 1 (Canonicity)**.** *If* $M : \mathsf{El}(\cdot \vdash \mathsf{base})$*, then either* $M = \mathsf{c0}$ *or* $M = \mathsf{c1}$.

Let $\mathbb{T}$ be the theory that we constructed in the previous chapter; its category of algebras and homomorphisms is written $\mathbf{Alg}[\mathbb{T}]$. We will construct the gluing algebra $\mathcal{M}^\bullet$ generically in another algebra $\mathcal{M}^\circ : \mathbf{Alg}[\mathbb{T}]$; concretely, we will always instantiate $\mathcal{M}^\circ$ as the *initial algebra*, but we never depend on that during the construction.[29]

[29] We choose this half-filled circle notation because we will be "gluing" $\mathcal{M}^\circ$ together with a *dependent algebra* $\mathcal{M}^\circ$ lying over $\mathcal{M}^\circ$, writing the resulting total algebra $\mathcal{M}^\bullet$.

Notation. For each piece of syntax $\mathcal{M}$, we will write $\mathcal{M}^\circ$ for its interpretation in $\mathcal{M}^\circ$ and $\mathcal{M}^\bullet$ for its interpretation in the gluing algebra $\mathcal{M}^\bullet$.

We are now ready to construct the gluing model of type theory to prove canonicity. Because $\mathcal{M}^\circ$ is a $\mathbb{T}$-algebra, the interpretation ctx° has the structure of a strict category with a terminal object. In fact, we will freely use the categorical notation that we defined in the margins to refer to objects from $\mathcal{M}^\circ$. When $\Gamma^\circ \in \mathsf{ctx}^\circ$, we will write $|\Gamma| = \mathbf{Hom}_{\mathsf{ctx}^\circ}(\mathbf{1}, \Gamma)$ for the *global sections* of Γ°. We assume Grothendieck universes $\mathcal{V}_0 < \mathcal{V}_1 < \mathcal{V}_\alpha$. We will by giving the interpretations of the sorts in the dependent algebra $\mathcal{M}^\circ$:

$$
\begin{aligned}
\mathsf{ctx}^\circ(*) &= \lambda\Gamma^\circ.\ |\Gamma^\circ| \to \mathcal{V}_\alpha \\
\mathsf{hom}^\circ(\Delta^\bullet, \Gamma^\bullet) &= \lambda\gamma^\circ.\ \textstyle\prod_{\delta_0^\bullet \in |\Delta^\bullet|} \Gamma^\circ(\gamma^\circ \circ \delta_0^\circ) \\
\mathsf{size}^\circ(*) &= \lambda i^\circ.\ \{j \in \{0,1\} \mid i = j^\circ\} \\
\mathsf{ord}^\circ(i^\bullet, j^\bullet) &= \{* \mid i^\circ \le j^\circ\} \\
\mathsf{ty}^\circ(i^\bullet, \Gamma^\bullet) &= \lambda A^\circ.\ \textstyle\prod_{\gamma_0^\bullet \in |\Gamma^\bullet|} \mathsf{El}(\cdot \vdash A^\circ[\gamma_0^\circ])^\circ \to \mathcal{V}_{i^\circ} \\
\mathsf{el}^\circ(\Gamma^\bullet, A^\bullet) &= \lambda M^\circ.\ \textstyle\prod_{\gamma_0^\bullet \in |\Gamma^\bullet|} \mathsf{Ty}_{A^\bullet}(1^\bullet)^\circ \gamma_0^\bullet(M^\circ[\gamma_0^\circ]) \\
&\text{where } |\Gamma^\bullet| = \textstyle\sum_{\gamma^\circ \in |\Gamma^\circ|} \Gamma^\circ \gamma^\circ
\end{aligned}
$$

Next, we need to give the interpretations of all the operations. We will write the left-hand sides of our definitions in the fully-annotated abstract notation, feeling free to use the convenient notation we have imposed on the right. We begin by interpreting sizes and their order:

$$
\begin{aligned}
\mathsf{small}^\circ(*) &= 0 && \text{(size)} \\
\mathsf{large}^\circ(*) &= 1 && \text{(size)} \\
\mathsf{enlarge}^\circ(i^\bullet) &= * && (- \le -)
\end{aligned}
$$

Next, we interpret the main judgmental structure of dependent type theory: contexts, substitutions, context extension, types and elements.

$$
\begin{aligned}
\mathsf{id}^\circ(\Gamma^\bullet)\gamma_0^\bullet &= \gamma_0^\circ && (- \to -) \\
\mathsf{cmp}^\circ(\Xi^\bullet, \Delta^\bullet, \Gamma^\bullet, \gamma^\bullet, \delta^\bullet)\xi_0^\bullet &= \gamma^\circ(\delta^\circ \circ \xi_0^\circ, \delta^\circ \xi_0^\bullet) && (- \to -) \\
\mathsf{emp}^\circ(*)\eta_0^\circ &= \{*\} && \text{(ctx)}
\end{aligned}
$$

$$\begin{aligned}
\mathtt{emp/subst}^{\bullet}(\Gamma^{\bullet})\eta_0^{\circ} &= * \\
\mathtt{ext}^{\bullet}(\Gamma^{\bullet}, A^{\bullet})\eta_0^{\circ} &= \textstyle\sum_{\gamma^{\bullet} \in \Gamma^{\bullet}(\mathbf{p}_{A^{\circ}} \circ \eta_0^{\circ})} A^{\bullet}(\mathbf{p}_{A^{\circ}} \circ \eta_0^{\circ}, \gamma^{\bullet})\mathbf{q}_{A^{\circ}} && \text{(ctx)} \\
\mathtt{drop}^{\bullet}(\Gamma^{\bullet}, A^{\bullet})\eta_0^{\bullet} &= \pi_1(\eta_0^{\bullet}) && (- \to -) \\
\mathtt{snoc}^{\bullet}(\Delta^{\bullet}, \Gamma^{\bullet}, A^{\bullet}, \gamma^{\bullet}, N^{\bullet})\delta_0^{\bullet} &= (\gamma^{\bullet}\delta_0^{\bullet}, N^{\bullet}\delta_0^{\bullet}) && (- \to -) \\
\mathtt{var}^{\bullet}(\Gamma^{\bullet}, A^{\bullet})\eta_0^{\bullet} &= \pi_2(\eta_0^{\bullet}) && (\mathsf{El}(- \vdash -)) \\
\mathtt{ty/act}^{\bullet}(\Delta^{\bullet}, \Gamma^{\bullet}, i^{\bullet}, j^{\bullet}, A^{\bullet}, \gamma^{\bullet}, p^{\bullet})\delta_0^{\bullet} M^{\circ} &= A^{\bullet}(\gamma^{\bullet} \circ \delta_0^{\bullet})M^{\circ} && (\mathsf{Ty}_{-}(-)) \\
\mathtt{el/act}^{\bullet}(\Delta^{\bullet}, \Gamma^{\bullet}, \gamma^{\bullet}, A^{\bullet}, M^{\bullet})\delta_0^{\bullet} &= M^{\bullet}(\gamma^{\bullet} \circ \delta_0^{\bullet}) && (\mathsf{El}(- \vdash -))
\end{aligned}$$

Dependent product types are particularly natural to interpret.

$$\begin{aligned}
\mathtt{pi}^{\bullet}(\Gamma^{\bullet}, i^{\bullet}, A^{\bullet}, B^{\bullet})\gamma_0^{\bullet} M^{\circ} &= \textstyle\prod_{N^{\bullet} \in \mathsf{El}(\cdot \vdash A^{\bullet}[\gamma^{\bullet}])^{\bullet}} B^{\bullet}\langle \gamma^{\bullet}, N^{\bullet} \rangle (M^{\circ} N^{\circ}) && (\mathsf{Ty}_{-}(-)) \\
\mathtt{lam}^{\bullet}(\Gamma^{\bullet}, A^{\bullet}, B^{\bullet}, M^{\bullet})\gamma_0^{\bullet} N^{\bullet} &= M^{\bullet}\langle \gamma_0^{\bullet}, N^{\bullet} \rangle && (\mathsf{El}(- \vdash -)) \\
\mathtt{app}^{\bullet}(\Gamma^{\bullet}, A^{\bullet}, B^{\bullet}, M^{\bullet}, N^{\bullet})\gamma_0^{\bullet} &= M^{\bullet}\gamma^{\bullet} N^{\bullet} && (\mathsf{El}(- \vdash -))
\end{aligned}$$

Finally, we interpret the universe and the base type. The family assigned to the universe "ties the knot" of the gluing construction, equipping type (codes) with their own predicates. The family assigned to the base type is the most important part of the entire development! Because the base type is not given by a universal property, we could have chosen any number of predicates; ours builds in a *choice* of whether an element is equal to the first constant or to the second.

$$\begin{aligned}
\mathtt{univ}^{\bullet}(\Gamma^{\bullet})\gamma_0^{\bullet} A^{\circ} &= A^{\circ} \to \mathcal{V}_0 \\
\mathtt{base}^{\bullet}(\Gamma^{\bullet}, i^{\bullet})\gamma_0^{\bullet} M^{\circ} &= (M^{\circ} = \mathsf{c0}^{\circ}) + (M^{\circ} = \mathsf{c1}^{\circ}) \\
\mathtt{c0}^{\bullet}(\Gamma^{\bullet})\gamma_0^{\bullet} &= \mathbf{inl}(\mathbf{refl}_{\mathsf{c0}^{\circ}}) \\
\mathtt{c1}^{\bullet}(\Gamma^{\bullet})\gamma_0^{\bullet} &= \mathbf{inr}(\mathbf{refl}_{\mathsf{c1}^{\circ}})
\end{aligned}$$

It is a simple exercise to check that the equational axioms are modeled by the interpretation above.

Lemma 2. *The projection* $\mathcal{M}^{\bullet} \xrightarrow{p} \mathcal{M}^{\circ}$ *is a homomorphism of* $\mathbb{T}$*-algebras.*

Proof. Immediate. □

Theorem 3 (Canonicity)**.** *If* $M : \mathsf{El}(\cdot \vdash \mathsf{base})$*, then either* $M = \mathsf{c0}$ *or* $M = \mathsf{c1}$*.*

Proof. In the above, let $\mathcal{M}^{\circ}$ be the *initial* $\mathbb{T}$-algebra. Then, we are trying to show that if $M^{\circ} \in \mathsf{El}(\cdot \vdash \mathsf{base}^{\circ})^{\circ}$, then either $M^{\circ} = \mathsf{c0}^{\circ}$ or $M^{\circ} = \mathsf{c1}^{\circ}$. Because $\mathcal{M}^{\circ}$ is the initial algebra, we have some glued element $N^{\bullet} \in \mathsf{El}(\cdot \vdash \mathsf{base}^{\bullet})^{\bullet}$, which is to say, an element $N^{\circ} \in \mathsf{El}(\cdot \vdash \mathsf{base}^{\circ})^{\circ}$ together with $N^{\bullet}(*) \in (N^{\circ} = \mathsf{c0}^{\circ}) + (N^{\circ} = \mathsf{c1}^{\circ})$. Therefore, it suffices to show that $N^{\circ} = M^{\circ}$; but this follows from the universal property of the initial $\mathbb{T}$-alebra, because $M^{\circ} = p(N^{\bullet})$.

□

Bibliography

Altenkirch, T., P. Dybjer, M. Hofmann, and P. Scott. "Normalization by Evaluation for Typed Lambda Calculus with Coproducts". In: *Proceedings of the 16th Annual IEEE Symposium on Logic in Computer Science*. LICS '01. Washington, DC, USA: IEEE Computer Society, 2001, pp. 303–. URL: http://dl.acm.org/citation.cfm?id=871816.871869.

Altenkirch, Thorsten, Martin Hofmann, and Thomas Streicher. "Categorical reconstruction of a reduction free normalization proof". In: *Category Theory and Computer Science*. Ed. by David Pitt, David E. Rydeheard, and Peter Johnstone. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, pp. 182–199. ISBN: 978-3-540-44661-3.

Angiuli, Carlo. "Computational Semantics of Cartesian Cubical Type Theory". To appear. PhD thesis. Pittsburgh, PA, USA: Carnegie Mellon University, 2019 (cit. on p. 7).

Angiuli, Carlo, Guillaume Brunerie, Thierry Coquand, Kuen-Bang Hou (Favonia), Robert Harper, and Daniel R. Licata. "Syntax and Models of Cartesian Cubical Type Theory". Preprint. Feb. 2019. URL: https://github.com/dlicata335/cart-cube (cit. on p. 8).

Awodey, Steve. "A cubical model of homotopy type theory". In: *Annals of Pure and Applied Logic* 169.12 (2018). Logic Colloquium 2015, pp. 1270–1294. ISSN: 0168-0072. DOI: 10.1016/j.apal.2018.08.002 (cit. on p. 16).

Buisse, Alexandre and Peter Dybjer. "Towards formalizing categorical models of type theory in type theory". In: *Electronic Notes in Theoretical Computer Science* 196 (2008), pp. 137–151.

Cartmell, John. "Generalised Algebraic Theories and Contextual Categories". PhD thesis. Oxford University, Jan. 1978 (cit. on pp. 8, 14).

– "Generalised algebraic theories and contextual categories". In: *Annals of Pure and Applied Logic* 32 (1986), pp. 209–243. ISSN: 0168-0072.

Castellan, Simon, Pierre Clairambault, and Peter Dybjer. "Undecidability of Equality in the Free Locally Cartesian Closed Category (Extended version)". In: *Logical Methods in Computer Science* 13.4 (2017).

Cohen, Cyril, Thierry Coquand, Simon Huber, and Anders Mörtberg. "Cubical Type Theory: a constructive interpretation of the univalence axiom". In: *IfCoLog Journal of Logics and their Applications* 4.10 (Nov. 2017), pp. 3127–3169. URL: http://www.collegepublications.co.uk/journals/ifcolog/?00019 (cit. on p. 8).

Coquand, Thierry. *Canonicity and normalization for Dependent Type Theory*. Oct. 2018. arXiv: 1810.09367.

Coquand, Thierry and Peter Dybjer. "Intuitionistic Model Constructions and Normalization Proofs". In: *Mathematical. Structures in Comp. Sci.* 7.1 (Feb. 1997), pp. 75–94. ISSN: 0960-1295. DOI: 10.1017/S0960129596002150. URL: http://dx.doi.org/10.1017/S0960129596002150.

Crole, R. L. *Categories for Types*. Cambridge Mathematical Textbooks. New York: Cambridge University Press, 1993. ISBN: 978-0-521-45701-9.

Curien, Pierre-Louis. "Substitution Up to Isomorphism". In: *Fundam. Inf.* 19.1-2 (Sept. 1993), pp. 51–85. ISSN: 0169-2968. URL: http://dl.acm.org/citation.cfm?id=175469.175471 (cit. on p. 7).

Curien, Pierre-Louis, Richard Garner, and Martin Hofmann. "Revisiting the categorical interpretation of dependent type theory". In: *Theoretical Computer Science* 546 (2014). Models of Interaction: Essays in Honour of Glynn Winskel, pp. 99–119. ISSN: 0304-3975. DOI: 10.1016/j.tcs.2014.03.003 (cit. on p. 7).

Curien, Pierre-Louis, Thérèse Hardin, and Jean-Jacques Lévy. "Confluence Properties of Weak and Strong Calculi of Explicit Substitutions". In: *J. ACM* 43.2 (Mar. 1996), pp. 362–397. ISSN: 0004-5411. DOI: 10.1145/226643.226675. URL: http://doi.acm.org/10.1145/226643.226675 (cit. on p. 7).

Dybjer, Peter. "Internal type theory". In: *Types for Proofs and Programs: International Workshop, TYPES '95 Torino, Italy, June 5–8, 1995 Selected Papers*. Ed. by Stefano Berardi and Mario Coppo. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 120–134. ISBN: 978-3-540-70722-6 (cit. on p. 14).

Fiore, Marcelo. *Discrete generalised polynomial functors*. Slides from talk given at ICALP 2012. 2012. URL: https://www.cl.cam.ac.uk/~mpf23/talks/ICALP2012.pdf (cit. on p. 16).

– "Semantic Analysis of Normalisation by Evaluation for Typed Lambda Calculus". In: *Proceedings of the 4th ACM SIGPLAN International Conference on Principles and Practice of Declarative Programming*. PPDP '02. Pittsburgh, PA, USA: ACM, 2002, pp. 26–37. ISBN: 1-58113-528-9. DOI: 10.1145/571157.571161. URL: http://doi.acm.org/10.1145/571157.571161.

Girard, Jean-Yves. *The Blind Spot: Lectures on Logic*. European Mathematical Society, Sept. 15, 2011. ISBN: 3-03719-088-4 (cit. on p. 3).

Granström, Johan G. *Treatise on Intuitionistic Type Theory*. Springer Publishing Company, Incorporated, 2013. ISBN: 94-007-3639-8 (cit. on p. 14).

Hofmann, Martin. "On the interpretation of type theory in locally cartesian closed categories". In: *Computer Science Logic*. Ed. by Leszek Pacholski and Jerzy Tiuryn. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, pp. 427–441. ISBN: 978-3-540-49404-1 (cit. on p. 7).

– "Syntax and Semantics of Dependent Types". In: *Semantics and Logics of Computation*. Cambridge University Press, 1997, pp. 79–130 (cit. on p. 4).

Hofmann, Martin and Thomas Streicher. "Lifting Grothendieck Universes". Unpublished note. 1997. URL: https://www2.mathematik.tu-darmstadt.de/~streicher/NOTES/lift.pdf.

Johnstone, Peter T. *Sketches of an Elephant: A Topos Theory Compendium: Volumes 1 and 2*. Oxford Logical Guides 43. Oxford Science Publications, 2002.

Kaposi, Ambrus, András Kovács, and Thorsten Altenkirch. "Constructing Quotient Inductive-inductive Types". In: *Proc. ACM Program. Lang.* 3.POPL (Jan. 2019), 2:1–2:24. ISSN: 2475-1421. DOI: 10.1145/3290315. URL: http://doi.acm.org/10.1145/3290315 (cit. on pp. 11, 19).

Kopylov, Alexei. "Type Theoretical Foundations for Data Structures, Classes and Objects". PhD thesis. Cornell University, 2004 (cit. on p. 7).

Lumsdaine, Peter Lefanu and Michael A. Warren. "The Local Universes Model: An Overlooked Coherence Construction for Dependent Type Theories". In: *ACM Trans. Comput. Logic* 16.3 (July 2015), 23:1–23:31. ISSN: 1529-3785. DOI: 10.1145/2754931. URL: http://doi.acm.org/10.1145/2754931 (cit. on p. 7).

Martin-Löf, Per. "An Intuitionistic Theory of Types: Predicative Part". In: *Logic Colloquium '73*. Ed. by H. E. Rose and J. C. Shepherdson. Vol. 80. Studies in Logic and the Foundations of Mathematics. Elsevier, 1975, pp. 73–118. DOI: 10.1016/S0049-237X(08)71945-1.

– "Constructive Mathematics and Computer Programming". In: *6th International Congress for Logic, Methodology and Philosophy of Science*. Published by North Holland, Amsterdam. 1982. Hanover, Aug. 1979, pp. 153–175.

– *Intuitionistic type theory. Notes by Giovanni Sambin*. Vol. 1. Studies in Proof Theory. Bibliopolis, 1984, pp. iv+91. ISBN: 88-7088-105-9 (cit. on p. 4).

Martin-Löf, Per. *Substitution calculus*. Notes from a lecture given in Göteborg. 1992.

Mitchell, John C. and Andre Scedrov. "Notes on sconing and relators". In: *Computer Science Logic*. Ed. by E. Börger, G. Jäger, H. Kleine Büning, S. Martini, and M. M. Richter. Berlin, Heidelberg: Springer Berlin Heidelberg, 1993, pp. 352–378. ISBN: 978-3-540-47890-4.

Nordström, Bengt, Kent Peterson, and Jan M. Smith. *Programming in Martin-Lof's Type Theory*. Vol. 7. International Series of Monographs on Computer Science. NY: Oxford University Press, 1990.

Palmgren, E. and S. J. Vickers. "Partial Horn logic and cartesian categories". In: *Annals of Pure and Applied Logic* 145.3 (2007), pp. 314–353. ISSN: 0168-0072. DOI: 10.1016/j.apal.2006.10.001.

Pitts, Andrew M. *Nominal Sets: Names and Symmetry in Computer Science*. New York, NY, USA: Cambridge University Press, 2013. ISBN: 1-107-01778-5 (cit. on pp. 5, 6).

Shulman, Michael. *Categorical Logic from a Categorical Point of View*. Lecture notes. 2017. URL: https://github.com/mikeshulman/catlog.

– *Scones, Logical Relations, and Parametricity*. Blog. 2006. URL: https://golem.ph.utexas.edu/category/2013/04/scones_logical_relations_and_p.html.

– "Univalence for inverse diagrams and homotopy canonicity". In: *Mathematical Structures in Computer Science* 25.5 (2015), pp. 1203–1277. DOI: 10.1017/S0960129514000565.

Sterling, Jonathan. *Algebraic Type Theory and Universe Hierarchies*. Dec. 2018. arXiv: 1902.08848 [math.LO].

Sterling, Jonathan and Bas Spitters. *Normalization by gluing for free λ-theories*. Sept. 2018. arXiv: 1809.08646 [cs.LO].

Streicher, Thomas. "Categorical intuitions underlying semantic normalisation proofs". In: *Preliminary Proceedings of the APPSEM Workshop on Normalisation by Evaluation*. Ed. by O. Danvy and P. Dybjer. Department of Computer Science, Aarhus University, 1998.

– *Investigations Into Intensional Type Theory*. Habilitationsschrift, Universität München. 1994.

– *Semantics of Type Theory: Correctness, Completeness, and Independence Results*. Cambridge, MA, USA: Birkhauser Boston Inc., 1991. ISBN: 0-8176-3594-7 (cit. on p. 6).

Taylor, Paul. *Practical Foundations of Mathematics*. Cambridge studies in advanced mathematics. Cambridge, New York (N. Y.), Melbourne: Cambridge University Press, 1999. ISBN: 0-521-63107-6 (cit. on p. 8).

Univalent Foundations Program, The. *Homotopy Type Theory: Univalent Foundations of Mathematics*. Institute for Advanced Study: https://homotopytypetheory.org/book, 2013 (cit. on p. 8).