

AN OK VERSION OF TYPE THEORY

JONATHAN STERLING

ABSTRACT. What’s the right way to write down type theory (with or without universes)? It’s unclear that there is a single right way, but there are a number of good lessons we’ve learned over time that we can use to avoid certain pitfalls and bureaucratic rat-holes. These lessons include: avoid subtyping and use coercions instead, and make presupposition admissibilities hold for obvious rather than complex reasons.

1. REMARK ON SUBSTITUTION

I am not going to treat variable binding or substitution in a formal way; let me just remark on two ways to do it:

- (1) The most invariant and well-adapted method is to eschew names, and have a single term q that stands for the last variable in the context, and then have a language of explicit substitutions which allows access to other variables via weakening. This approach scales up forever and is semantically very natural [Dyb96], and is very close to implementation as well. It is a bit hard to write down terms manually in this language, but that’s what elaboration is for.
- (2) A respectable and more traditional approach is to have pre-terms (for instance, using simple-sorted ABTs [Acz78; FPT99; FM10; Har16] or Martin-Löf’s theory of arities [Mar84]), and then have substitution be an operation first defined on raw terms. After this, the typing principle of substitution can be accounted for in one of two ways:
 - (a) The typing principle for substitution can be just added as a rule (made derivable).
 - (b) The typing principle for substitution can be made admissible.

The first option is (surprisingly) superior: there is no useful consequence of the substitution principle for dependent type theory being admissible but not derivable, so one might as well avoid the headache.

The options above span the spectrum of “very derivable” (the substitution operation is an actual generating pre-term), “semi-admissible” (substitution is an operation defined on raw terms, but its typing principle is derivable), and “very admissible” (substitution is an operation defined on raw terms *and* its typing principle follows by induction on derivations).

For the rest of this document, I will treat variables and substitution totally informally; all the approaches described above will work equally well in making this aspect of the theory precise, so it can be chosen according to taste.

2. JUDGMENTAL STRUCTURE

I have tried (and invented) many different possibilities for formulating the judgmental structure of dependent type theory, but I have ultimately come to prefer a classic version due to Martin-Löf.

- (1) **Contexts:** $\Gamma \text{ ctx}$ means that Γ is a context.
- (2) **Types:**
 - (a) $\Gamma \vdash A \text{ type}$ means that A is a type in context Γ , presupposing $\Gamma \text{ ctx}$.
 - (b) $\Gamma \vdash A \equiv B \text{ type}$ means that A and B are equal types in context Γ , presupposing $\Gamma \text{ ctx}$, $\Gamma \vdash A \text{ type}$, and $\Gamma \vdash B \text{ type}$.
- (3) **Terms:**
 - (a) $\Gamma \vdash a : A$ means that a is a term of type A in context Γ , presupposing $\Gamma \text{ ctx}$ and $\Gamma \vdash A \text{ type}$.
 - (b) $\Gamma \vdash a_0 \equiv a_1 : A$ means that a_0 and a_1 are equal terms of type A in context Γ , presupposing $\Gamma \text{ ctx}$, $\Gamma \vdash A \text{ type}$, $\Gamma \vdash a_0 : A$, and $\Gamma \vdash a_1 : A$.

2.1. Meaning of presuppositions (invariants of deduction). Each presupposition expresses a pair of invariants on a form of judgment; using the form of judgment $\Gamma \vdash a : A$ as an example, these two invariants are as follows:

- (1) If you can derive $\Gamma \vdash a : A$ then you must be able to derive $\Gamma \vdash A \text{ type}$.
- (2) If you can derive $\Gamma \vdash a : A$ and you can derive $\Gamma \vdash A \equiv B \text{ type}$, then you must be able to derive $\Gamma \vdash a : B$.

Whether the presupposition admissibilities above are proper admissibilities or derivabilities depends on how you set up the theory. Generally speaking, it is quite perverse for (1) to be a rule of type theory, whereas it is usually necessary for (2) to be a rule of type theory. For (1) there are a number of options:

- (1) One convention is that in each rule of inference, every “metavariable” should have a single premise expressing the appropriate presupposition. Then, this presupposition admissibility will follow by *dependent case analysis* on derivations without using any sophisticated induction. This is the most well-adapted and scalable approach.
- (2) Another convention is to try and minimize the premises to the rules, including extra premises only at leaf nodes (like typing principles for constants). With a great deal of work, a specific theory formulated in this way can be shown to exhibit the required presupposition admissibilities, but this is extremely fragile and there is essentially zero pay-off to working in this way. Adding a single rule to such a type theory will usually break these admissibilities (as noted by Streicher [Str91]), which is why it is always better to just include all the premises systematically.

2.2. Congruence rules. We do not write down any congruence rules! We assume enough rules to ensure that definitional equality is a congruence. This is one of the main motivations for working in an equational logical framework (to rule out any perverse kind of “equality” that is not a congruence!) [Car78; NPS90; Uem19].

2.3. The substitution principle. As I mentioned above, we will not be treating substitutions formally in order to support some flexibility in preferred presentation. But let me state what is not negotiable: the most well-adapted notion of substitution for dependent type theory is a *simultaneous* substitution $\Delta \xrightarrow{Y} \Gamma$. So, a definition of when such a simultaneous substitution is well-typed should be given (either by rules, or by induction on contexts), and as well as a definition of when two simultaneous substitutions are equal,

and then we must arrange for the following substitution principles to hold in one way or another:

$$\frac{\text{TYPE SUBSTITUTION} \quad \Gamma, \Delta \text{ ctx} \quad \Delta \xrightarrow{\gamma} \Gamma \quad \Gamma \vdash A \text{ type}}{\Delta \vdash \gamma^* A \text{ type}}$$

$$\frac{\text{TERM SUBSTITUTION} \quad \Gamma, \Delta \text{ ctx} \quad \Delta \xrightarrow{\gamma} \Gamma \quad \Gamma \vdash A \text{ type} \quad \Gamma \vdash a : A}{\Delta \vdash \gamma^* a : \gamma^* A}$$

There are additional associativity and unit laws that one must arrange to hold, as well as naturality conditions for every constructor of type theory; if you define substitution as a meta-operation, these laws can be seen to hold by induction, or you can just add them as rules. I like to add them as rules (or, better yet, work in a logical framework where these principles are automatic). The need to answer this question in one way or another is another one of the main motivations for logical frameworks [HHP93].

3. THE RULES OF TYPE THEORY WITHOUT UNIVERSES

Now, I will describe the rules of type theory without universes; in the next section, I will show how to add universes in a well-adapted way. Please note how I am purposefully not adding any special premises for variable freshness, because I intend the use of named variables in this presentation to be a *mere notation* for any one of a thousand different precise accounts of variables.

$$\frac{\text{EMPTY CONTEXT}}{\cdot \text{ ctx}} \quad \frac{\text{CONTEXT EXTENSION} \quad \Gamma \text{ ctx} \quad \Gamma \vdash A \text{ type}}{\Gamma, x : A \text{ ctx}}$$

We must add conversion rules (for both term typing and term equality); note how we *systematically* add all possible premises here to avoid any question of presupposition admissibility.

$$\frac{\text{CONVERSION (TERM)} \quad \Gamma \text{ ctx} \quad \Gamma \vdash A, B \text{ type} \quad \Gamma \vdash A \equiv B \text{ type} \quad \Gamma \vdash a : A}{\Gamma \vdash a : B}}{\Gamma \vdash a : B} \quad \frac{\text{CONVERSION (EQUALITY)} \quad \Gamma \text{ ctx} \quad \Gamma \vdash A, B \text{ type} \quad \Gamma \vdash A \equiv B \text{ type} \quad \Gamma \vdash a_0 \equiv a_1 : A}{\Gamma \vdash a_0 \equiv a_1 : B}}$$

As an example, we will add the rules for dependent product types.

$$\frac{\text{DEPENDENT PRODUCT FORMATION} \quad \Gamma \text{ ctx} \quad \Gamma \vdash A \text{ type} \quad \Gamma, x : A \vdash B(x) \text{ type}}{\Gamma \vdash \prod_{x:A} B(x) \text{ type}}$$

$$\frac{\text{DEPENDENT PRODUCT INTRODUCTION} \quad \Gamma \text{ ctx} \quad \Gamma \vdash A \text{ type} \quad \Gamma, x : A \vdash B(x) \text{ type} \quad \Gamma, x : A \vdash b(x) : B(x)}{\Gamma \vdash \lambda_{x:A.B(x)}(x.b(x)) : \prod_{x:A} B(x)}$$

$$\frac{\text{DEPENDENT PRODUCT ELIMINATION} \quad \Gamma \text{ ctx} \quad \Gamma \vdash A \text{ type} \quad \Gamma, x : A \vdash B(x) \text{ type} \quad \Gamma \vdash f : \prod_{x:A} B(x) \quad \Gamma \vdash a : A}{\Gamma \vdash \text{ap}_{x:A.B(x)}(f, a) : B(a)}$$

DEPENDENT PRODUCT COMPUTATION

$$\frac{\Gamma \text{ ctx} \quad \Gamma \vdash A \text{ type} \quad \Gamma, x : A \vdash B(x) \text{ type} \quad \Gamma, x : A \vdash b(x) : B(x) \quad \Gamma \vdash a : A}{\Gamma \vdash \text{ap}_{x:A.B(x)}(\lambda_{x:A.B(x)}(x.b(x)), a) \equiv b(a) : B(a)}$$

DEPENDENT PRODUCT UNIQUENESS

$$\frac{\Gamma \text{ ctx} \quad \Gamma \vdash A \text{ type} \quad \Gamma, x : A \vdash B(x) \text{ type} \quad \Gamma \vdash f : \prod_{x:A} B(x)}{\Gamma \vdash f \equiv \lambda_{x:A.B(x)}(x.\text{ap}_{x:A.B(x)}(f, x)) : \prod_{x:A} B(x)}$$

Other “negative” connectives (like dependent sum) will follow an analogous pattern. All such types should come equipped with an η -rule as above.

4. EXTENSION: A UNIVERSE

We will now see how to add a universe to the type theory. The most well-adapted way to add a universe is in the Tarskian style; universes à la Russell should be understood as a matter of elaboration, definitely not as a matter of semantics. It is occasionally said that the inclusion of explicit decoding of codes in the universe is an intolerable bureaucracy, but in fact it leads to a great simplification in both the syntax and semantics of dependent type theory, playing the same role that coercions have played in making sense of complex features of programming languages. It is very easy to elaborate universes à la Russell to universes à la Tarski, so it doesn't affect the user's experience.

$$\begin{array}{c} \text{UNIVERSE FORMATION} \\ \Gamma \text{ ctx} \\ \hline \Gamma \vdash U \text{ type} \end{array} \qquad \begin{array}{c} \text{DECODING FORMATION} \\ \Gamma \text{ ctx} \quad \Gamma \vdash a : U \\ \hline \Gamma \vdash \text{el}(a) \text{ type} \end{array}$$

A universe is closed under connectives by adding codes together with computation rules for their decodings. For instance, we add dependent product types to the universe in the following way:

$$\begin{array}{c} \text{PI CODE FORMATION} \\ \Gamma \text{ ctx} \quad \Gamma \vdash a : U \quad \Gamma, x : \text{el}(a) \vdash b(x) : U \\ \hline \Gamma \vdash \hat{\Pi}_{x:a} b(x) : U \end{array}$$

$$\begin{array}{c} \text{PI CODE DECODING} \\ \Gamma \text{ ctx} \quad \Gamma \vdash a : U \quad \Gamma, x : \text{el}(a) \vdash b(x) : U \\ \hline \Gamma \vdash \text{el}(\hat{\Pi}_{x:a} b(x)) \equiv \prod_{x:\text{el}(a)} \text{el}(b(x)) \text{ type} \end{array}$$

Remark 4.1. We are explaining the meaning of type codes by means of a rule of definitional equivalence decoding the type $\text{el}(a)$. An alternative is to add introduction, elimination, computation, and uniqueness rules for $\text{el}(a)$ expressing a definitional isomorphism instead; long-term this may be even better for implementations, because information may be lost when decoding a type code (this is a frequent occurrence in cubical type theories, in which the decoding of a V -type need not include the full data of the equivalence).

Since it's most traditional to use an equation rather than an isomorphism, I'm sticking to that for the moment; I just wanted to point out what the future might look like. 🍷

5. EXTENSION: A CUMULATIVE HIERARCHY OF UNIVERSES

Let \mathcal{L} be a preorder whose elements $k, l \in \mathcal{L}$ will be universe levels; it is most common to let \mathcal{L} be the set of natural numbers, but there is no reason a priori that sets with larger cardinality cannot be used. We will show how to extend type theory with a universe hierarchy parameterized in \mathcal{L} . First, for each $k \in \mathcal{L}$ we add a universe à la Tarski:

$$\begin{array}{c}
\text{UNIVERSE FORMATION} \\
\frac{\Gamma \text{ ctx}}{\Gamma \vdash \mathbb{U}_k \text{ type}} \\
\\
\text{DECODING FORMATION} \\
\frac{\Gamma \text{ ctx} \quad \Gamma \vdash a : \mathbb{U}_k}{\Gamma \vdash \text{el}_k(a) \text{ type}} \\
\\
\text{PI CODE FORMATION} \\
\frac{\Gamma \text{ ctx} \quad \Gamma \vdash a : \mathbb{U}_k \quad \Gamma, x : \text{el}_k(a) \vdash b(x) : \mathbb{U}_k}{\Gamma \vdash \hat{\Pi}_{x:a}^k b(x) : \mathbb{U}_k} \\
\\
\text{PI CODE DECODING} \\
\frac{\Gamma \text{ ctx} \quad \Gamma \vdash a : \mathbb{U}_k \quad \Gamma, x : \text{el}_k(a) \vdash b(x) : \mathbb{U}_k}{\Gamma \vdash \text{el}_k(\hat{\Pi}_{x:a}^k b(x)) \equiv \prod_{x:\text{el}_k(a)} \text{el}_k(b(x)) \text{ type}}
\end{array}$$

We must also include codes for smaller universes. Every universe \mathbb{U}_k needs to have a code $\hat{\mathbb{U}}_k^l$ in each universe \mathbb{U}_l where $l > k$.

$$\begin{array}{c}
\text{UNIVERSE CODE} \\
\frac{\Gamma \text{ ctx} \quad (k < l)}{\Gamma \vdash \hat{\mathbb{U}}_k^l : \mathbb{U}_l} \\
\\
\text{UNIVERSE CODE DECODING} \\
\frac{\Gamma \text{ ctx} \quad (k < l)}{\Gamma \vdash \text{el}_l(\hat{\mathbb{U}}_k^l) \equiv \mathbb{U}_k \text{ type}}
\end{array}$$

Likewise, the decodings $\text{el}_k(a)$ need to have codes in \mathbb{U}_l whenever $l \geq k$:

$$\begin{array}{c}
\text{(HOLD ON FOR THE OFFICIAL RULE)} \\
\frac{\Gamma \text{ ctx} \quad \Gamma \vdash a : \mathbb{U}_k \quad (l \geq k)}{\Gamma \vdash \hat{\text{el}}_k^l(a) : \mathbb{U}_l}
\end{array}$$

Squinting, this appears to be a functorial action for the level preorder on the universe hierarchy, so it will be clearer to write it as $\hat{\uparrow}_k^l a$, meaning “lift the code a from level k to level l ”. Then the remaining rules that we need will write themselves:

$$\begin{array}{c}
\text{UNIVERSE LIFT} \\
\frac{\Gamma \text{ ctx} \quad \Gamma \vdash a : \mathbb{U}_k \quad (l \geq k)}{\Gamma \vdash \hat{\uparrow}_k^l a : \mathbb{U}_l} \\
\\
\text{LIFT UNIT} \\
\frac{\Gamma \text{ ctx} \quad \Gamma \vdash a : \mathbb{U}_k}{\Gamma \vdash \hat{\uparrow}_k^k a \equiv a : \mathbb{U}_k} \\
\\
\text{LIFT ASSOC} \\
\frac{\Gamma \text{ ctx} \quad \Gamma \vdash a : \mathbb{U}_k \quad (m \geq l \geq k)}{\Gamma \vdash \hat{\uparrow}_l^m \hat{\uparrow}_k^l a \equiv \hat{\uparrow}_k^m a : \mathbb{U}_m}
\end{array}$$

5.1. Rational cumulativity. The rational counterpart to “cumulativity” is then obtained by adding the following rule of definitional equivalence:¹

$$\begin{array}{c}
\text{CUMULATIVITY} \\
\frac{\Gamma \text{ ctx} \quad \Gamma \vdash a : \mathbb{U}_k \quad (l \geq k)}{\Gamma \vdash \text{el}_l(\hat{\uparrow}_k^l a) \equiv \text{el}_k(a) \text{ type}}
\end{array}$$

The alternative “non-cumulative” thing to do is to add introduction, elimination, computation, and uniqueness rules to each type $\text{el}_l(\hat{\uparrow}_k^l a)$ expressing the equation above as a definitional isomorphism.


Exercise 5.1. Show that, in the presence of the CUMULATIVITY rule above, the following definitional equivalence of types holds:

$$\text{el}_l(\hat{\uparrow}_k^l \hat{\Pi}_{x:a}^k b(x)) \equiv \text{el}_l(\hat{\Pi}_{x:\hat{\uparrow}_k^l a}^l \hat{\uparrow}_k^l b(x))$$

¹The adaptation of the name “cumulativity” for this rule is due to Shulman [Shu19].

Remark 5.2. It is also possible to add even stronger rules than CUMULATIVITY, which are reasonable but even harder to justify in many models. For instance, one might have the following rule (which is stronger than the equation from Exercise 5.1!):

$$\frac{(\dots)}{\Gamma \vdash \uparrow_k^l \hat{\Pi}_{x:a}^k b(x) \equiv \hat{\Pi}_{x:\uparrow_k^l a}^l \uparrow_k^l b(x) : U_l}$$

I personally recommend against including such a rule, as it is not needed in practice. But my coauthors and I have included it in the past in various presentations of type theory [SAG19; Ste18]. 

5.2. Eschewing cumulativity. Another alternative is to eschew the cumulativity equation and add introduction, elimination, computation, and uniqueness rules to each type $\text{el}_l(\uparrow_k^l a)$ expressing the equation above as a definitional isomorphism. This version of the type theory has many more useful models (Shulman [Shu19] gives a general coherence theorem for finding such models, based on an extension of the local universes method [LW15]).

6. EXTENSION: A TYPE OF BOOLEANS

We may add a type of booleans, as an example of a type that we will *not* give an η -rule.

BOOLEAN FORMATION $\Gamma \text{ ctx}$	BOOLEAN INTRODUCTION $\Gamma \text{ ctx}$
$\Gamma \vdash \text{bool type}$	$\Gamma \vdash \text{tt, ff} : \text{bool}$
BOOLEAN ELIMINATION $\Gamma \text{ ctx} \quad \Gamma, x : \text{bool} \vdash C(x) \text{ type} \quad \Gamma \vdash t : C(\text{tt}) \quad \Gamma \vdash f : C(\text{ff}) \quad \Gamma \vdash b : \text{bool}$ <hr style="width: 100%;"/> $\Gamma \vdash \text{if}_{x.C(x)}(b; t; f) : C(b)$	
BOOLEAN COMPUTATION $\Gamma \text{ ctx} \quad \Gamma, x : \text{bool} \vdash C(x) \text{ type} \quad \Gamma \vdash t : C(\text{tt}) \quad \Gamma \vdash f : C(\text{ff})$ <hr style="width: 100%;"/> $\Gamma \vdash \text{if}_{x.C(x)}(\text{tt}; t; f) \equiv t : C(\text{tt})$ $\Gamma \vdash \text{if}_{x.C(x)}(\text{ff}; t; f) \equiv f : C(\text{ff})$	

No further rules are required.

7. ACKNOWLEDGMENT

Many of the ideas in this note were developed in joint work with Carlo Angiuli, Daniel Gratzer, and Robert Harper. I am also thankful to Thierry Coquand and Mike Shulman for their contributions to the ideas behind this note.

REFERENCES

- [Acz78] Peter Aczel. *A General Church-Rosser Theorem*. Tech. rep. University of Manchester, 1978 (cit. on p. 1).
- [Car78] John Cartmell. “Generalised Algebraic Theories and Contextual Categories”. PhD thesis. Oxford University, Jan. 1978 (cit. on p. 2).
- [Dyb96] Peter Dybjer. “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. 1).

- [FM10] Marcelo Fiore and Ola Mahmoud. “Second-Order Algebraic Theories”. In: *Mathematical Foundations of Computer Science 2010: 35th International Symposium, MFCS 2010, Brno, Czech Republic, August 23-27, 2010. Proceedings*. Ed. by Petr Hliněný and Antonín Kučera. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 368–380. ISBN: 978-3-642-15155-2 (cit. on p. 1).
- [FPT99] Marcelo Fiore, Gordon Plotkin, and Daniele Turi. “Abstract syntax and variable binding”. In: *Proceedings of the 14th Symposium on Logic in Computer Science*. 1999, pp. 193–202 (cit. on p. 1).
- [Har16] Robert Harper. *Practical Foundations for Programming Languages*. Second. New York, NY, USA: Cambridge University Press, 2016 (cit. on p. 1).
- [HHP93] Robert Harper, Furio Honsell, and Gordon Plotkin. “A Framework for Defining Logics”. In: *J. ACM* 40.1 (Jan. 1993), pp. 143–184. ISSN: 0004-5411. doi: 10.1145/138027.138060. URL: <http://doi.acm.org/10.1145/138027.138060> (cit. on p. 3).
- [LW15] Peter LeFanu Lumsdaine 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. 6).
- [Mar84] Per Martin-Löf. *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. 1).
- [NPS90] Bengt Nordström, Kent Peterson, and Jan M. Smith. *Programming in Martin-Löf’s Type Theory*. Vol. 7. International Series of Monographs on Computer Science. NY: Oxford University Press, 1990 (cit. on p. 2).
- [SAG19] Jonathan Sterling, Carlo Angiuli, and Daniel Gratzer. “Cubical Syntax for Reflection-Free Extensional Equality”. In: *4th International Conference on Formal Structures for Computation and Deduction (FSCD 2019)*. Ed. by Herman Geuvers. Vol. 131. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019, 31:1–31:25. ISBN: 978-3-95977-107-8. doi: 10.4230/LIPIcs.FSCD.2019.31. arXiv: 1904.08562. URL: <http://drops.dagstuhl.de/opus/volltexte/2019/10538> (cit. on p. 6).
- [Shu19] Michael Shulman. *All $(\infty, 1)$ -toposes have strict univalent universes*. Apr. 2019. arXiv: 1904.07004. URL: <https://arxiv.org/abs/1904.07004> (cit. on pp. 5, 6).
- [Ste18] Jonathan Sterling. *Algebraic Type Theory and Universe Hierarchies*. Dec. 2018. arXiv: 1902.08848 [math.LO]. URL: <https://arxiv.org/abs/1902.08848> (cit. on p. 6).
- [Str91] Thomas Streicher. *Semantics of Type Theory: Correctness, Completeness, and Independence Results*. Cambridge, MA, USA: Birkhauser Boston Inc., 1991. ISBN: 0-8176-3594-7 (cit. on p. 2).
- [Uem19] Taichi Uemura. *A General Framework for the Semantics of Type Theory*. 2019. arXiv: 1904.04097. URL: <https://arxiv.org/abs/1904.04097> (cit. on p. 2).