# “Upon This Quote I Will Build My Church Thesis”

Pierre-Marie Pédrot
INRIA
France
pierre-marie.pedrot@inria.fr

## ABSTRACT

The internal Church thesis (CT) is a logical principle stating that one can associate to any function $f : \mathbb{N} \to \mathbb{N}$ a concrete code, in some Turing-complete language, that computes $f$. While the compatibility of CT in simpler systems has been long known, its compatibility with dependent type theory is still an open question.

In this paper, we answer this question positively. We define “MLTT”, a type theory extending MLTT with quote operators in which CT is derivable. We furthermore prove that “MLTT” is consistent, strongly normalizing and enjoys canonicity using a rather standard logical relation model. All the results in this paper have been mechanized in Coq[1].

## KEYWORDS

Martin-Löf type theory, dependent types, Church’s thesis, logical relation

## 1 INTRODUCTION

“Calculemus!” By these words, Leibniz famously enjoined the reader to compute. Contemporary logicians took this motto as a founding principle after the progressive discovery of the proof-as-program correspondence. This major breakthrough, also known as the Curry-Howard equivalence, is the seemingly simple observation that proofs and programs are the same object, in an essential way [11].

Although not primarily associated to the Curry-Howard school of thought, the Russian constructivists led by Markov pushed this tenet to an extreme point, postulating that *all* mathematical objects were indeed algorithms. Their doctrine [25] is materialized by a foundational system that can be summarily described as a neutral Bishop-style intuitionistic logic extended with two additional axioms [36]: Markov’s principle (MP) and Church’s thesis (CT).

Of those two axioms, Markov’s principle is the simplest one, as it requires little to state:

$$\forall f : \mathbb{N} \to \mathbb{N}. \neg\neg(\exists n : \mathbb{N}. f\ n = 0) \to \exists n : \mathbb{N}. f\ n = 0$$

and is equally easy to understand. Alternative formulations include “a Turing machine that does not loop terminates” or “non-zero reals are apart from zero”. Markov’s principle is a sweet spot of semi-classical logic, as it can be given a computational content preserving the witness property [12].

Church’s thesis is somewhat more involved, as it requires the internal definition of a computation model in the logic itself. Assuming a logic rich enough, this is traditionally [20] achieved by defining the decidable Kleene predicate $\mathsf{T}\,(p, n, k)$ and its associated primitive recursive decoding function $\mathsf{U} : \mathbb{N} \to \mathbb{N}$. Here, $p : \mathbb{N}$ is assumed to be the syntactic code of some program in the chosen Turing-complete computation model, $n : \mathbb{N}$ some integer argument, and $k : \mathbb{N}$ some encoding of evaluation traces of the model. Under these assumptions, $\mathsf{T}\,(p, n, k)$ holds whenever $k$ is the trace of the fact that $p$ applied to $n$ evaluates to some integer $v$. Furthermore, if indeed the predicate holds, then $\mathsf{U}\,k$ should return the value $v$.

Assuming we have settled this computational paraphernalia, CT is simply the internal statement

$$\forall f : \mathbb{N} \to \mathbb{N}. \exists p : \mathbb{N}. \forall n : \mathbb{N}. \exists k : \mathbb{N}. \mathsf{T}\,(p, n, k) \land \mathsf{U}\,k = f\ n.$$

Said otherwise, any function $f$ definable in the theory is known to be computable by some concrete algorithm $p$ from within the theory.

Contrarily to MP, which is a consequence of excluded middle, CT is a very anti-classical axiom [36]. Assuming very weak forms of choice, it contradicts excluded middle, or even LPO. Similarly, it is also incompatible with choice-like principles like double negation shift. Finally, it also makes the logic very intensional as it contradicts function extensionality, under the same kind of weak choice assumptions.

The consistency of MP and CT w.r.t. some logical system is typically proved via realizability. Quite remarkably, Kleene’s seminal paper [21] already proves that CT is compatible with Heyting’s arithmetic. For a more expressive theory, the effective topos is a famous example of a realizability topos in which both principles hold [17].

On the other side of the iron curtain, one major offshoot of the Curry-Howard philosophical stance is Martin-Löf’s type theory [26], in short MLTT, a famous foundation for constructive mathematics. It is the theoretical underpinning of several widely used proof assistants such as Agda, Coq or Lean. In these software systems, there is no formal separation between proofs and programs, as they live in the same syntax and obey the same typing and computation rules. This monistic credo turns MLTT into the quintessential intuitionistic foundation of our modern times, blending logic and computation into the very same cast.

In the wake of the Swedish tradition of neutrality, MLTT does not pick a side in the constructivist feud. It neither proves fan principles from the Brouwerian band nor mechanistic axioms from the Markovian clique. Considering the claim above that in MLTT proofs *are* programs, it does seem a bit surprising that it is not biased towards the latter side. Surely it ought to be easy to convert MLTT to the Markovian orthodoxy, for otherwise the Curry-Howard mantra would be but a misleading advertisement. Let us survey the current status of each Russian axiom individually in the canon of dependent type theory.

As we have just explained, it is known that MP is not derivable in MLTT [4]. Since it is a consequence of classical logic, it holds in classical models like the ZF one [38], but it is also possible to add MP to MLTT while retaining the computational properties through

[1]Development browsable at https://github.com/ppedrot/quote-mltt.

a form of delimited exceptions [30]. Note that, as usual for such an expressive system, the exact statement of MP may matter [8].

As for CT, we already stated that it is negated by classical models, and thus is not a consequence of MLTT. By contrast with MP, the compatibility of CT with dependent type theory is a much more contentious subject. To make things simpler, we will therefore focus on this single principle in this paper, and deliberately ignore Markov's principle. The proviso about phrasing of the statement mattering a lot is even more paramount with CT, which is the chief reason why the problem and its answers are a matter of debate. In the remainder we will prove that MLTT is indeed compatible with the strongest form of Church's thesis usually agreed upon, but for this we first need to explain what we actually mean by these sibylline words. We dedicate the next section to a thorough exegesis of this topic.

## 2 A COMPREHENSIVE CT SCAN

Contrarily to more archaic systems, MLTT does not need a realizability interpretation to turn its proofs into programs. In some sense, it is *already* a realizability interpretation, as MLTT terms are literally *bona fide* programs. It should therefore be very natural to add CT to MLTT.

As a matter of fact, as long as the context is empty, the following rule is admissible

$$\frac{\vdash M : \mathbb{N} \to \mathbb{N}}{\vdash \langle M \rangle : \Sigma p : \mathbb{N}.\, \Pi n : \mathbb{N}.\, \Sigma k : \mathbb{N}.\, \mathsf{T}\,(p, n, k) \times \mathsf{U}\, k = M\, n}$$

where $\langle M \rangle$ is some term derived from $M$ in a systematic way. Depending on the pursued goal, this process is variously known in the type theory world as extraction [22] or quotation [34]. Obviously, a rule that is derivable for closed sequents is not necessarily internalizable in the theory, so there is a non-trivial gap to fill there.

An additional issue is that dependent type theories have various notions of existence. Typically, they contrast dependent sum types $\Sigma x : A.\, B$ with existential types $\exists x : A.\, B$. The precise details depend on the exact theory considered, but the general rule is that the former corresponds to actual, constructive evidence, while the latter stands for mere existence, i.e. no computational content can be extracted from this type. Such non-computational types are called *propositions*, an umbrella term for mildly related concepts. The three most common instances of propositions are captured by the realizability-driven Prop universe of CIC [29], the hProp subuniverse inspired by the univalent world [37], and the SProp universe of strict propositions [10]. Regardless of the setting, Σ-types validate choice by construction through large elimination or projections, while existential types may or may not validate choice.

The arithmetic statement of CT mentions two existential quantifiers, hence we have a priori at least 4 possible translations into MLTT. In practice, the second one returns an enumerable proposition, so that for most notions of proposition, namely Prop with singleton elimination or hProp with unique choice, the use of ∃ or Σ results in equivalent statements. We will thus always stick to a Σ-type for this quantifier. More problematic is the first existential quantifier, the nature of which leads to radically different worlds. For conciseness, we will call $\mathsf{CT}_\Sigma$ (resp. $\mathsf{CT}_\exists$) the statement of Church's thesis with a Σ (resp. ∃) type as the first existential quantifier.

As mere existence does not validate choice by default, $\mathsf{CT}_\exists$ is much closer to the traditional first-order setting. When ∃ is taken to live in the Prop universe of CIC, the relative expressivity of $\mathsf{CT}_\exists$ has been studied extensively in the setting of synthetic computability [6, 7]. An important remark is that the lack of choice prevents building an internal quoting function $(\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ that associates to some function its concrete code. As already hinted at before, this means that $\mathsf{CT}_\exists$ does not necessarily contradict function extensionality. Actually, we can even go much further: in the case where propositions are identified with hProps, $\mathsf{CT}_\exists$ turns out to be compatible not only with MLTT but also with full-blown univalence [35]. More generally and quite counterintuitively, univalence is compatible with many principles that would make the hardcore Bishop-style intuitionist raise a suspicious eyebrow, as long as they are squashed enough and thus made computationally harmless [33, 35, 37].

Contrastingly, as Σ-types come with intuitionistic (non)-choice built-in, $\mathsf{CT}_\Sigma$ is the telltale of an extremely weird mathematical realm. For starters, it immediately implies the existence of a quoting function and breaks both function extensionality and classical logic. The consistency of $\mathsf{CT}_\Sigma$ with MLTT is an open problem that has been lingering for a while and seems to be considered a difficult question by experts [23, 24, 35]. The best result thus far [18] is the consistency of $\mathsf{CT}_\Sigma$ with a stripped-down version of MLTT without the so-called $\xi$ rule:

$$\frac{\Gamma, x : A \vdash M \equiv N : B}{\Gamma \vdash \lambda x : A.\, M \equiv \lambda x : A.\, N : \Pi x : A.\, B}$$

Although we understand the difficulties experienced by the authors of this paper and acknowledge their distinct goals, we consider that removing this congruence from MLTT when precisely trying to implement a principle implying the existence of a quoting function is unarguably throwing the baby with the bathwater. The lack of the $\xi$ rule basically prevents any computation under a $\lambda$-abstraction, which means that functions are morally identified with their code. Ubiquitary conversion is a stepping stone of MLTT for program reasoning, so treating functions as blackboxes is a no-go.

Given the strong relationship between quoting functions and metaprogramming, we should also mention some attempts at making the latter a first-class citizen of dependent type theory. These systems are built with practical programming in mind rather than constructive mathematics, but the endeavours are similar enough that they are worth highlighting. There is in particular a wealth of literature on contextual types [27], a special kind of modal types [13] capturing well-typed terms in an object context. Although contextual types can coexist with dependent types [16], the ability to pattern-match on code in these settings is still a difficult problem that is only satisfyingly handled in the absence of dependent types [3, 19]. The closest thing to what we achieve in this paper is an unpublished line of work for dependently-typed quoting [14].

## 3 HIGH-LEVEL DESCRIPTION

We now give the high-level intuitions that we develop technically later on. In this paper, we define "MLTT", read "MLTT with quotes", a minute extension of MLTT with quoting operators that implement $\mathsf{CT}_\Sigma$ in a straightforward way. As already explained, $\mathsf{CT}_\Sigma$ holds externally in MLTT. If we want it as an internal rule, there are two problems to solve: first, handling terms in an arbitrary context, and second, showing that our hypothetical internalization preserves conversion.

Despite the aura of complexity surrounding this question, our solution is disappointingly simple. The first problem will be handled in the most trivial way one can think of. Namely, the primitives computing the internal version of $\mathsf{CT}_\Sigma$ will simply block as long as their arguments are not closed. Since the return type of these primitives is first-order, this will not be a problem as it will not endanger canonicity.

The second problem is solved in a similarly obvious manner. Given two terms $M \equiv N : \mathbb{N} \to \mathbb{N}$ one needs to ensure that the quotation of these terms agree. In particular, the integer code returned by these operations must be the same. This sounds complicated, as in general two equivalent functions may have different codes. In Turing-complete languages, this is actually impossible to achieve in a computable way, due to Rice's theorem. But in MLTT, there is a very simple way to find a canonical representative of an equivalence class of functions: just pick the normal form! Conversion in MLTT is decidable, as it virtually amounts to comparing the normal forms of the considered terms for syntactic equality. This requires that all well-typed terms have a normal form, but this property is usually taken for granted in dependent type theories and will for sure hold true in "MLTT".

As the astute reader may complain about, this is not enough in presence of $\eta$-rules, which are included in our system. But even in this case, our normalization trick can be adapted by simply maximally $\eta$-reducing and stripping all annotations from the normal form. As a result, it is possible to associate a canonical code to equivalence classes of convertible terms even up to $\eta$-conversion, and importantly, the resulting program has the same extensional behaviour as the source term.

Despite the intuitive simplicity of the above guiding ideas, proofs about dependent type theory are very tedious and error-prone, let alone when they contain bits of computability. To keep the naysayer at bay, all proofs were mechanized in the Coq proof assistant. For easy reference, we will add hyperlinks to the Coq development signalled by the icon. Note that for readability, we use named variables in the paper, but the actual formalization relies on De Bruijn indices. This will add some impedance to the matching between the paper statements and the actual Coq code, but it should be straightforward to go back and forth.

## 4 BASIC TYPE THEORY

Let us fix some conventions. Since we will be juggling quite a bit between levels, we will use a different font style to refer to objects from the metatheory, with types in **bold** and type ascription written in set-theoretic style $x \in \mathbf{X}$. Some metatheoretical types of interest are $\mathbf{X} \Rightarrow \mathbf{Y}$, the metafunctions from $\mathbf{X}$ to $\mathbf{Y}$, and $\mathbf{N}$ the metaintegers. We will write **term** for the type of "MLTT" terms defined later on.

Our base theory will be an instance of MLTT featuring one Russell-style universe, negative $\Pi$ and $\Sigma$ types with definitional $\eta$-rules, together with a natural number type, an empty type and an identity type. We recall the syntax of this theory in Figure 1. The typing rules are standard and feature five kinds of judgments: context well-formedness, type well-formedness, term well-typedness, type conversion and typed term conversion. To pin down the conventions we expose a representative excerpt of the rules in Figure 2.

We use the usual notations $A \to B$ and $A \times B$ for non-dependent product and sum respectively. We will write $M = N$ for $\mathsf{Id}\,A\,M\,N$ when $A$ is clear from the context. In practice, we will almost always use it with $A := \mathbb{N}$. Similarly, we will sometime drop the annotations of $\lambda$-abstrations and pairs. If $n \in \mathbf{N}$, we write $[n]_\mathbb{N} \in \mathbf{term}$ for the unary numeral associated to $n$.

We will also use some notational devices to discriminate between intended meanings. We will write $\Lambda := \mathbb{N}$ for numbers coding for programs.

Partial functions will play an important role. In type theory, there is a standard encoding [32] going through the partiality monad $\wp(A) := \mathbb{N} \to \texttt{option}\,A$. A term $p : \wp(A)$ is undefined if for all $n : \mathbb{N}$, $p\;n = \texttt{None}$. Otherwise its associated value is the first $v$ s.t. $p\;n = \texttt{Some}\;v$.

Although we could encode it, we do not have a built-in option type in MLTT. Since we will only ever consider partial integers in this paper, we will rely on a simpler encoding.

*Definition 4.1 (Partial integers).* We define the type of partial integers $\mathbb{N}_\wp := \mathbb{N} \to \mathbb{N}$. The intuitive meaning of a partial integer $P : \mathbb{N}_\wp$ is the following.

- If for all $n : \mathbb{N}$, $P\;n = 0$, then $P$ is undefined.
- Otherwise, let $n_0$ the smallest integer such that $P\;n_0 = \mathsf{S}\,v$ for some $v$. Then $P$ evaluates to $v$.

**Notation 4.2.** Given $P : \mathbb{N}_\wp$ we define the shift of $P$ as

$$P^+ : \mathbb{N}_\wp := \lambda k : \mathbb{N}.\, P\;(\mathsf{S}\,k).$$

The intuitive meaning of evaluation from Definition 4.1 can be internalized in MLTT through the step evaluation predicate.

*Definition 4.3 (Step evaluation).* Given $M : \mathbb{N}_\wp$, $V : \mathbb{N}$ and $K : \mathbb{N}$, we define the step-evaluation predicate, written $M \rightsquigarrow V \parallel K$ and read "$M$ evaluates to $V$ in $K$ steps" as

$$\begin{aligned} &M \rightsquigarrow V \parallel K : \square := \\ &\quad \mathsf{rec}_\mathbb{N}\,(\_.\,\mathbb{N}_\wp \to \square)\;(\lambda p : \mathbb{N}_\wp.\, p\;0 = \mathsf{S}\,V) \\ &\qquad (n, r.\, \lambda p : \mathbb{N}_\wp.\, (p\;0 = 0) \times (r\;p^+))\;K\;M. \end{aligned}$$

*Remark* 4.4. If $n \in \mathbf{N}$, then assuming well-typed enough arguments we have

$$\begin{aligned} &M \rightsquigarrow V \parallel [n]_\mathbb{N} \equiv \\ &\quad (M\;0 = 0) \times \ldots \times (M\;[n-1]_\mathbb{N} = 0) \times (M\;[n]_\mathbb{N} = \mathsf{S}\,V). \end{aligned}$$

Given the algorithmically-friendly nature of MLTT, we will pick a slightly nicer, but equivalent, phrasing of $\mathsf{CT}_\Sigma$. Following [7], we will merge the Kleene predicate $\mathsf{T}$ and its associated decoding function $\mathsf{U}$ into a single function $\texttt{run} : \Lambda \to \mathbb{N} \to \mathbb{N}_\wp$. Computation traces will simply be the number of steps needed to reach a value, which will be accounted for by the step-evaluation predicate.

*Definition 4.5.* Henceforth, we will define CT in MLTT as

$$\Pi f : \mathbb{N} \to \mathbb{N}.\, \Sigma p : \Lambda.\, \Pi n : \mathbb{N}.\, \Sigma k : \mathbb{N}.\, \texttt{run}\;p\;n \rightsquigarrow f\;n \parallel k$$

for some model $\texttt{run} : \Lambda \to \mathbb{N} \to \mathbb{N}_\wp$.

$$\begin{array}{rl} M, N, A, B ::= & x \mid M\,N \mid \lambda x : A.\,M \mid \square \mid \Pi x : A.\,B \mid \Sigma x : A.\,B \mid \langle M, N :: x : A, B\rangle \mid M.\pi_1 \mid M.\pi_2 \\ & \mid \mathbb{N} \mid 0 \mid \mathsf{S}\,M \mid \mathsf{rec}_{\mathbb{N}}\,(n.\,P)\,R_0\,(m, r.\,R_S)\,V \mid \bot \mid \mathsf{rec}_{\bot}\,(e.\,P)\,V \mid \mathsf{Id}\,A\,M\,N \mid \mathsf{refl}\,A\,M \mid \mathsf{rec}_{\mathsf{Id}}\,A\,M\,(y, e.\,P)\,R\,N\,V \end{array}$$

**Figure 1: Syntax of** MLTT **(Coq definition )**

$$\frac{}{\vdash \cdot} \qquad \frac{\Gamma \vdash A}{\vdash \Gamma, x : A} \qquad \frac{\vdash \Gamma \qquad (x : A) \in \Gamma}{\Gamma \vdash x : A} \qquad \frac{\vdash \Gamma}{\Gamma \vdash \square} \qquad \frac{\Gamma \vdash A : \square}{\Gamma \vdash A} \qquad \frac{\Gamma \vdash A \qquad \Gamma, x : A \vdash B}{\Gamma \vdash \Pi x : A.\,B}$$

$$\frac{\Gamma \vdash A \qquad \Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x : A.\,M : \Pi x : A.\,B} \qquad \frac{\Gamma \vdash A \qquad \Gamma, x : A \vdash B \qquad \Gamma \vdash M : A \qquad \Gamma \vdash N : B\{x := M\}}{\Gamma \vdash \langle M, N :: x : A, B\rangle : \Sigma x : A.\,B}$$

$$\frac{\Gamma, n : \mathbb{N} \vdash P \qquad \Gamma \vdash R_0 : P\{n := 0\} \qquad \Gamma, m : \mathbb{N}, r : P\{n := m\} \vdash R_S : P\{n := \mathsf{S}\,m\} \qquad \Gamma \vdash V : \mathbb{N}}{\Gamma \vdash \mathsf{rec}_{\mathbb{N}}\,(n.\,P)\,R_0\,(m, r.\,R_S)\,V : P\{n := V\}}$$

$$\frac{\Gamma \vdash M : \Pi x : A.\,B}{\Gamma \vdash \lambda x : A.\,M\,x \equiv M : \Pi x : A.\,B} \qquad \frac{\Gamma \vdash A \qquad \Gamma \vdash A \equiv A_1 \qquad \Gamma \vdash A \equiv A_2 \qquad \Gamma, x : A \vdash M \equiv N : B}{\Gamma \vdash \lambda x : A_1.\,M \equiv \lambda x : A_2.\,N : \Pi x : A.\,B}$$

**Figure 2: Curated excerpt of** MLTT **typing rules (Coq definition )**

$$M, N, A, B ::= \ldots \mid \text{⚲}\,M \mid \text{ϛ}\,M\,N \mid \varrho\,M\,N$$

**Figure 3: Additional syntax of** “MLTT”

## 5 “MLTT” EXTENSIONS

We now turn to the definition of the extensions that define “MLTT” proper.

*Definition 5.1.* The new term constructors of “MLTT” are defined in Figure 3, and will be collectively referred to as the *quoting primitives.*

We give names exposing the intuition of the meaning of those terms. The term $\text{⚲}\,M$ is the *quote* of $M$ which is intended to return the code of the function $M$. The term $\text{ϛ}\,M\,N$ is the *step-count* of $M$ applied to $N$, i.e. it will return the number of steps needed to evaluate the quote of $M$ on argument $N$. Finally, $\varrho\,M\,N$ is the *reflection* of $M$ applied to $N$, which produces a proof that indeed the quote of $M$ fullfils the expected runtime behaviour.

*Definition 5.2 (Computation model).* A computation model for “MLTT” is described by the following:

- a metafunction $\ulcorner\cdot\urcorner \in \mathbf{term} \Rightarrow \mathbf{N}$;
- an “MLTT” term $\mathsf{run} \in \mathbf{term}$.

To define the typing system alone we do not need any additional requirements on those objects, but let us give some intuition before proceeding any further.

The $\ulcorner\cdot\urcorner$ function is just some arbitrary Gödel numbering of the syntax. As already explained, the run function is going to serve as a Kleene predicate expressed in a functional form. In particular, we expect $\mathsf{run} : \Lambda \to \mathbb{N} \to \mathbb{N}_{\wp}$ and $\mathsf{run}\ p\ n$ to compute the result of the application of the program $p$ to the argument $n$ if it exists. In practice, run is expected to be defined in MLTT alone, or for that matter, in a much weaker fragment corresponding to PRA. In the remainder of this paper, we assume a fixed computation model.

**Notation 5.3.** We write

$$M \,\text{⚲}\, N := \mathsf{run}\ (\text{⚲}\,M)\ N \rightsquigarrow M\,N \parallel \text{ϛ}\,M\,N$$

for the specification of the quoting operators on $M$ and $N$. Furthermore, given any $p, n, k \in \mathbf{N}$, let

$$\mathbf{run}\ p\ n\ k := \mathsf{run}\ [p]_{\mathbb{N}}\ [n]_{\mathbb{N}}\ [k]_{\mathbb{N}}.$$

We now have enough to define the typing rules of the additional “MLTT” primitives in Figure 4. We delay the description of the conversion rules of these primitives to a later point, because we still need more infrastructure.

$$\frac{\Gamma \vdash M : \mathbb{N} \to \mathbb{N}}{\Gamma \vdash \text{⚲}\,M : \Lambda} \qquad \frac{\Gamma \vdash M : \mathbb{N} \to \mathbb{N} \qquad \Gamma \vdash N : \mathbb{N}}{\Gamma \vdash \text{ϛ}\,M\,N : \mathbb{N}}$$

$$\frac{\Gamma \vdash M : \mathbb{N} \to \mathbb{N} \qquad \Gamma \vdash N : \mathbb{N}}{\Gamma \vdash \varrho\,M\,N : M \,\text{⚲}\, N}$$

**Figure 4: Typing rules of** “MLTT”

Nonetheless, these rules are already sufficient to state the one internal property of “MLTT” we care about.

Theorem 5.4. “MLTT” *proves* CT.

Proof. CT is trivially implemented by

$$\lambda f : \mathbb{N} \to \mathbb{N}.\ \langle \text{⚲}\,f, \lambda n : \mathbb{N}.\ \langle \text{ϛ}\,f\,n, \varrho\,f\,n\rangle\rangle.$$

□

To finish the specification of “MLTT”, we now need to define the conversion rules of the theory. This requires some heavy definitions.

*Definition 5.5 (Quasi-closedness).* Given $M \in \mathbf{term}$ and $\Gamma \in \mathbf{ctx}$, we say that $M$ is $\Gamma$-quasi-closed, written $\mathbf{clos}_{\Gamma}\,M$ if, ignoring $\lambda$ and pair annotations, all free variables of $M$ are in $\Gamma$. Some representative cases are defined at Figure 5. In the particular case where $\Gamma$ is empty, we will write $\mathbf{clos}\,M$.

*Definition 5.6 (Deep normal and neutral forms).* We recall the inductive definition of deep normal and neutral forms of MLTT terms at Figure 6 and define the additional rules for “MLTT” at Figure 7.

$$\frac{x \in \Gamma}{\mathbf{clos}_\Gamma\, x} \qquad \frac{\mathbf{clos}_\Gamma\, M \qquad \mathbf{clos}_\Gamma\, N}{\mathbf{clos}_\Gamma\, M\, N}$$

$$\frac{\mathbf{clos}_{\Gamma,x:A}\, M}{\mathbf{clos}_\Gamma\, (\lambda x : A.\, M)} \qquad \frac{\mathbf{clos}_\Gamma\, A \qquad \mathbf{clos}_{\Gamma,x:A}\, B}{\mathbf{clos}_\Gamma\, (\Pi x : A.\, B)}$$

$$\frac{\mathbf{clos}_\Gamma\, A \qquad \mathbf{clos}_{\Gamma,x:A}\, B}{\mathbf{clos}_\Gamma\, (\Sigma x : A.\, B)} \qquad \frac{\mathbf{clos}_\Gamma\, M \qquad \mathbf{clos}_\Gamma\, N}{\mathbf{clos}_\Gamma\, \langle M, N :: x : A, B\rangle}$$

**Figure 5: Quasi-closedness (excerpt)**

$$\frac{}{\mathbf{dne}\, x} \qquad \frac{\mathbf{dne}\, M}{\mathbf{dnf}\, M} \qquad \frac{\mathbf{dne}\, M \qquad \mathbf{dnf}\, N}{\mathbf{dne}\, M\, N}$$

$$\frac{\mathbf{dnf}\, M}{\mathbf{dnf}\, (\lambda x : A.\, M)} \qquad \frac{\mathbf{dnf}\, A \qquad \mathbf{dnf}\, B}{\mathbf{dnf}\, (\Pi x : A.\, B)}$$

$$\frac{\mathbf{dnf}\, P \qquad \mathbf{dne}\, V}{\mathbf{dne}\, \mathsf{rec}_\bot\, (e.\, P)\, V} \qquad \frac{\mathbf{dnf}\, M \qquad \mathbf{dnf}\, N}{\mathbf{dnf}\, \langle M, N :: x : A, B\rangle}$$

**Figure 6: Deep normal / neutral MLTT terms (excerpt)**

Normal forms are, as usual, terms which cannot be simplified further. Neutral forms are a subcase of the former, intuitively capturing the notion of a term whose evaluation is blocked due to a missing variable. In particular, they cannot trigger new conversions when they are substituted for a variable. Our definition is standard for the MLTT fragment, except maybe that we we ignore $\lambda$ and pair type annotations just like for the **clos** predicate. Only worth discussing are the newly introduced terms of “MLTT”.

The quote primitives do not generate any new non-neutral normal forms. Indeed, their expected types are concrete datatypes, so if we want canonicity we just cannot create new constructors for those. They do generate new neutrals, though. The intuition is that these primitives only compute on closed normal forms, so if one of their argument is not closed, they will block and thus be neutral.

**Notation 5.7.** We write **clnf** $M$ if both **clos** $M$ and **dnf** $M$.

The last non-trivial ingredient needed is erasure of “MLTT” terms. We rely on it to quotient normal forms w.r.t. the various $\eta$-rules of our system.

*Definition 5.8 (Erasure).* Given $M \in$ **term**, we define its erasure $\|M\| \in$ **term** by induction on $M$. This operation can be understood as the composition of two finer-grained primitives: first, replace $\lambda$ and pair type annotations with a dummy term, and second, perform maximal $\eta$-reduction of $\lambda$ and pair nodes. We choose $\Box$ for the dummy term, but any closed normal term would do. We give the relevant operations in Figure 8, all other cases are term homomorphisms.

**Notation 5.9.** Given $M \in$ **term** we write $⟦M⟧ \in \mathbf{N} := \ulcorner \|M\| \urcorner$, the Gödel number of the erasure of $M$.

*Definition 5.10.* Given $k, v \in \mathbf{N}$ we define $[k, v]_{\text{♀}} \in$ **term** by induction on $k$ as

$$\begin{array}{lcl} [0, v]_{\text{♀}} & := & \mathsf{refl}\, \mathbb{N}\, (\mathsf{S}\, [v]_{\mathbb{N}}) \\ [k+1, v]_{\text{♀}} & := & (\mathsf{refl}\, \mathbb{N}\, 0) \times [k, v]_{\text{♀}}. \end{array}$$

We now have all the necessary definitions to define the new conversion rules of “MLTT” at Figure 9. We give some intuitions about these conversion rules by paraphrasing the rules. The congruence rules are self-evident. The computation rule for $\text{♀}\, M$ is simply stating that quoting a closed normal term produces the Gödel number of its erasure. The two other rules reflect the behaviour of the run operator in the theory itself. They start by assuming that $M$ is a closed normal term, so its quote is a concrete code. Assuming canonicity, for any $n, k \in \mathbf{N}$, **run** $⟦M⟧\ n\ k$ must be convertible to a numeral. Similarly, $M\ [n]_{\mathbb{N}}$ must be convertible to some numeral $v$.

Since we expect run to model the computation of “MLTT”, there must be some $k_0 \in \mathbf{N}$ s.t. **run** $⟦M⟧\ n\ k_0$ is convertible to $\mathsf{S}\, v$. If $k_0$ is the smallest such bound, then $\text{⍑}\, M\ [n]_{\mathbb{N}}$ returns $k_0$, and $\varrho\, M\ [n]_{\mathbb{N}}$ must provide a closed proof of $M\ \text{♀}\ [n]_{\mathbb{N}}$. But given the previous assumptions, $M\ \text{♀}\ [n]_{\mathbb{N}}$ is convertible to a finite sequence of products of equalities $0 = 0$ with a trailing equality $\mathsf{S}\, v = \mathsf{S}\, v$. This type is trivially inhabited by the term $[k_0, v]_{\text{♀}}$.

## 6 COMPUTATIONAL ADEQUACY

There is still one missing piece for “MLTT” to make sense. Indeed, in the intuitive explanation of the conversion rules for the quoting primitives we gave above, we argued that run should model the runtime behaviour of “MLTT”. In spite of this, we have made no additional assumption on Definition 5.2 so far. We make this requirement formal as *computational adequacy* in this section. This first forces us to endow “MLTT” with a notion of evaluation.

*Definition 6.1 (Weak-head normal and neutral forms).* We define weak-head normal **whnf** and neutral **whne** forms similarly to their deep counterparts from Figure 6, the only difference being that we do not require non-neutral subterms to be in normal form. Weak neutral forms for the quoting operators are the same as deep neutral forms from Figure 7.

*Definition 6.2 (Evaluation).* We mutually define two step-indexed evaluation relations $\downarrow$ and $\Downarrow$ in the metatheory, respectively computing the weak-head and the deep normal form. An excerpt of the MLTT fragment is presented in Figure 10. Weak evaluation for “MLTT” extensions is defined in Figure 11, the rules for deep evaluation being the same.

Although both relations are given as inference rules, they really are step-indexed recursive functions in the metatheory, of type **term** $\Rightarrow$ **N** $\Rightarrow$ **option term**. We write $M \uparrow_k$ and $M \Uparrow_k$ when no derivation is possible, implicitly meaning that the corresponding function returns **None**. We will use the same notations without the $k$ index to existentially quantify over this index, e.g. $M \downarrow N$ means that there exists some $k \in \mathbf{N}$ s.t. $M \downarrow_k N$.

The evaluation rules for the MLTT fragment are, once again, standard. The only interesting rules are the ones for the quoting primitives, as they make weak-head evaluation depend on deep evaluation. All evaluation paths for the quoting primitives start by deeply evaluating their arguments and checking whether they are quasi-closed. If not, they immediately return. Otherwise, they perform a macroscopic evaluation step. For $\text{♀}\, M$, that just means quoting the closed normal form into a number. For $\text{⍑}\, M\, N$ and

$$\frac{\neg\mathbf{clos}\,M \qquad \mathbf{dnf}\,M}{\mathbf{dne}\,(\text{ϙ}\,M)} \qquad \frac{\neg(\mathbf{clos}\,M \wedge \mathbf{clos}\,N) \qquad \mathbf{dnf}\,M \qquad \mathbf{dnf}\,N}{\mathbf{dne}\,(\text{ϛ}\,M\,N)} \qquad \frac{\neg(\mathbf{clos}\,M \wedge \mathbf{clos}\,N) \qquad \mathbf{dnf}\,M \qquad \mathbf{dnf}\,N}{\mathbf{dne}\,(\varrho\,M\,N)}$$

**Figure 7: Deep normal and neutral forms of** “MLTT” **extensions**

$$\begin{array}{lcl}
\|\lambda x : A.\,M\| & := & \begin{cases} N & \text{if } \|M\| = N\ x \text{ and } x \notin N \\ \lambda x : \square.\,\|M\| & \text{otherwise} \end{cases} \\
\|\langle M, N :: x : A, B\rangle\| & := & \begin{cases} P & \text{if } \|M\| = P.\pi_1 \text{ and } \|N\| = P.\pi_2 \\ \langle \|M\|, \|N\| :: x : \square, \square\rangle & \text{otherwise} \end{cases}
\end{array}$$

**Figure 8: Term erasure**

$$\frac{\Gamma \vdash M \equiv M' : \mathbb{N} \to \mathbb{N}}{\Gamma \vdash \text{ϙ}\,M \equiv \text{ϙ}\,M' : \Lambda} \qquad \frac{\Gamma \vdash M : \mathbb{N} \to \mathbb{N} \qquad \mathbf{clnf}\,M}{\Gamma \vdash \text{ϙ}\,M \equiv [\llbracket M \rrbracket]_{\mathbb{N}} : \Lambda}$$

$$\frac{\Gamma \vdash M \equiv M' : \mathbb{N} \to \mathbb{N} \qquad \Gamma \vdash N \equiv N' : \mathbb{N}}{\Gamma \vdash \text{ϛ}\,M\,N \equiv \text{ϛ}\,M'\,N' : \mathbb{N}} \qquad \frac{\Gamma \vdash M \equiv M' : \mathbb{N} \to \mathbb{N} \qquad \Gamma \vdash N \equiv N' : \mathbb{N}}{\Gamma \vdash \varrho\,M\,N \equiv \varrho\,M'\,N' : M\,\text{Ϙ}\,N}$$

$$\frac{\Gamma \vdash M : \mathbb{N} \to \mathbb{N} \qquad \mathbf{clnf}\,M \qquad \Gamma \vdash \mathbf{run}\,\llbracket M \rrbracket\,n\,k_0 \equiv \mathsf{S}\,[v]_{\mathbb{N}} : \mathbb{N} \qquad \forall k < k_0.\quad \Gamma \vdash \mathbf{run}\,\llbracket M \rrbracket\,n\,k \equiv 0 : \mathbb{N}}{\Gamma \vdash \text{ϛ}\,M\,[n]_{\mathbb{N}} \equiv [k_0]_{\mathbb{N}} : \mathbb{N}}$$

$$\frac{\Gamma \vdash M : \mathbb{N} \to \mathbb{N} \qquad \mathbf{clnf}\,M \qquad \Gamma \vdash \mathbf{run}\,\llbracket M \rrbracket\,n\,k_0 \equiv \mathsf{S}\,[v]_{\mathbb{N}} : \mathbb{N} \qquad \forall k < k_0.\quad \Gamma \vdash \mathbf{run}\,\llbracket M \rrbracket\,n\,k \equiv 0 : \mathbb{N}}{\Gamma \vdash \varrho\,M\,[n]_{\mathbb{N}} \equiv [k_0, v]_{\text{Ϙ}} : M\,\text{Ϙ}\,[n]_{\mathbb{N}}}$$

**Figure 9: New conversion rules of** “MLTT”

$$\frac{}{x \downarrow_k x} \qquad \frac{}{\lambda x : A.\,M \downarrow_k \lambda x : A.\,M}$$

$$\frac{M \downarrow_k M_0 \qquad \mathbf{whne}\,M_0}{M\ N \downarrow_{k+1} M_0\ N} \qquad \frac{M \Downarrow_k M_0}{\lambda x : A.\,M \Downarrow_{k+1} \lambda x : A.\,M_0}$$

$$\frac{M \downarrow_k \lambda x : A.\,M_0 \qquad M_0\{x := N\} \downarrow_k R}{M\ N \downarrow_{k+1} R}$$

$$\frac{}{x \Downarrow_k x} \qquad \frac{M \downarrow_k \lambda x : A.\,M_0 \qquad M_0\{x := N\} \Downarrow_k R}{M\ N \Downarrow_{k+1} R}$$

$$\frac{M \downarrow_k M' \qquad \mathbf{whne}\,M' \qquad M' \Downarrow_k M_0 \qquad N \Downarrow_k N_0}{M\ N \Downarrow_{k+1} M_0\ N_0}$$

**Figure 10: Step-indexed evaluation for** MLTT **(excerpt)**

$\varrho\,M\,N$, it corresponds to performing a guarded $\mu$-recursion to find the smallest index such that the erasure of $M\ N$ evaluates to a numeral.

*Definition 6.3 (Computational adequacy).* We say that the computation model is adequate if it satisfies the following properties.

- We have a derivation $\vdash \mathsf{run} : \Lambda \to \mathbb{N} \to \mathbb{N}_{\wp}$.
- For all $M \in$ **term** and $n, k \in \mathbf{N}$ whenever $M\ [n]_{\mathbb{N}} \Uparrow_k$ then $\mathbf{run}\,\llbracket M \rrbracket\ n\ k \Downarrow 0$.
- For all $M \in$ **term** and $n, k, v \in \mathbf{N}$ whenever $M\ [n]_{\mathbb{N}} \Downarrow_k [v]_{\mathbb{N}}$ then $\mathbf{run}\,\llbracket M \rrbracket\ n\ k \Downarrow \mathsf{S}\,[v]_{\mathbb{N}}$.

The last two points are essentially stating that the metatheoretical deep evaluation function coincides with the object `run` evaluation function. This leaves very little leeway in the potential implementation of `run`.

## 7 SOUNDNESS THEOREMS

In the remainder of this paper, we assume that our globally fixed computational model is adequate. Under this implicit assumption, we get the following results.

Theorem 7.1 (Consistency). *“MLTT” is consistent.*

Theorem 7.2 (Strong normalization). *Well-typed terms in “MLTT” are strongly normalizing.*

Theorem 7.3 (Canonicity). *Any “MLTT” term $M$ such that $\vdash M : \mathbb{N}$ normalizes to a numeral.*

The rest of this paper is dedicated to the sketch of the proof of the three above theorems, which all derive from the same property.

## 8 BASICS OF MLTT LOGICAL RELATIONS

We will prove our metatheoretical results about “MLTT” using a logical relation model. The base model is basically the now famous inductive-recursive one from Abel et al. [1], with a handful of surprisingly minor tweaks. In this section we briefly recall the principles of Abel-style logical relations.

From a high-level point of view, Abel-style logical relations for MLTT are just a glorified kind of realizability with *a lot* of bells and whistles. Notably, in the semantic world, well-formed types $\Gamma \Vdash A$

$$\frac{M \Downarrow_k M_0 \qquad \mathbf{clos}\, M_0}{\text{♀}\, M \downarrow_{k+1} [\llbracket M_0 \rrbracket]_{\mathbb{N}}} \qquad \frac{M \Downarrow_k M_0 \qquad \neg\mathbf{clos}\, M_0}{\text{♀}\, M \downarrow_{k+1} \text{♀}\, M_0}$$

$$\frac{M \Downarrow_k M_0 \qquad N \Downarrow_k N_0 \qquad \neg(\mathbf{clos}\, M_0 \wedge \mathbf{clos}\, N_0)}{\text{ꞇ}\, M\, N \downarrow_{k+1} \text{ꞇ}\, M_0\, N_0} \qquad \frac{M \Downarrow_k M_0 \qquad N \Downarrow_k N_0 \qquad \neg(\mathbf{clos}\, M_0 \wedge \mathbf{clos}\, N_0)}{\varrho\, M\, N \downarrow_{k+1} \varrho\, M_0\, N_0}$$

$$\frac{M \Downarrow_k M_0 \qquad N \Downarrow_k [n]_{\mathbb{N}} \qquad \mathbf{clos}\, M_0 \qquad k_0 \leq k \qquad \|M_0\|\ [n]_{\mathbb{N}} \Downarrow_{k_0} [v]_{\mathbb{N}} \qquad \forall k' < k_0.\ \|M_0\|\ [n]_{\mathbb{N}} \Uparrow_{k'}}{\text{ꞇ}\, M\, N \downarrow_{k+1} [k_0]_{\mathbb{N}}}$$

$$\frac{M \Downarrow_k M_0 \qquad N \Downarrow_k [n]_{\mathbb{N}} \qquad \mathbf{clos}\, M_0 \qquad k_0 \leq k \qquad \|M_0\|\ [n]_{\mathbb{N}} \Downarrow_{k_0} [v]_{\mathbb{N}} \qquad \forall k' < k_0.\ \|M_0\|\ [n]_{\mathbb{N}} \Uparrow_{k'}}{\varrho\, M\, N \downarrow_{k+1} [k_0, v]_{\text{♀}}}$$

**Figure 11: Step-indexed weak-head evaluation of** “MLTT” **extensions**

are defined inductively, while well-typed terms $\mathbf{p}_A \mid \Gamma \Vdash M : A$ are defined recursively over a proof $\mathbf{p}_A \in \Gamma \Vdash A$. In turn, well-typedness is extremely similar to, say, Kleene realizability [36], i.e. $\mathbf{p}_A \mid \Gamma \Vdash M : A$ essentially means that $M$ weak-head normalizes to some value $V$, further satisfying some conditions depending on $A$. Due to conversion being an integral part of typing, one also needs to associate to a given semantic type additional predicates for type and term convertibility, but they are the expected ones.

The major departure from usual realizability in this kind of model is the proper handling of variables. All semantic predicates are actually presheaves over contexts equipped with weakenings. Furthermore neutral terms enjoy a very specific status, see e.g. Lemma 8.6. This will turn out to be important for our goals.

The relation itself is parameterized by syntactic predicates for the various notions of typing and reduction. These predicates must satisfy a list of closure properties, which are typically weaker than the ones enjoyed by the standard typing rules of MLTT. Another way to understand this is that logical relations turn a liberal type system, called *declarative*, with many closure rules that is easy to work with, into a lower-level type system that is hardly usable but which is similar in spirit to the steps performed by a typing algorithm. Proofs in this generic system are in some sense *normalized* compared to the declarative system. As a matter of fact, this is a way to prove decidability of type-checking, since an algorithmic presentation of type-checking will satisfy the low-level closure properties. A notable difference with the declarative system is that generic typing requires making explicit the notion of neutral terms, through the introduction of neutral convertibility which can be understood as the usual convertibility restricted to neutral terms.

In the remainder of this section we assume fixed some instance of syntactic typing and reduction rules for MLTT. Due to their sheer size and the fact they are neither surprising nor new, we will not state the closure properties here, but rather link the corresponding Coq code directly.

*Definition 8.1 (Generic Typing).* We define the notion of generic typing as a list of closure properties that our typing rules must satisfy.

Given generic notions of typing and reduction, one can define reducibility in the abstract. Our base logical relation is unchanged w.r.t. Abel et al. [1], so we will not give the full definition in this paper. To begin with, writing everything in full on paper would be both unreadable and uninformative, and probably ridden with typos. Rather, we recall below some representative type formers to build intuition and point to the Coq development for more details. We will also consciously ignore universe level issues. They are technical although important bookkeeping details, but we consider that they clutter the simple principles behind logical relations, and since our results are backed up by a mechanized proof, we should not have to care about such minutiae in the human-readable paper.

*Definition 8.2 (Reducibility).* We give a partial but evocative definition of reducibility at Figure 12.

As already explained, we gloss over the details and instead concentrate on the high-level view. We abuse implicit arguments in Figure 12 to keep things readable, and we also omit additional well-formedness conditions. At the risk of repeating ourselves, this is really just the run-of-the-mill complete presheaf model for MLTT, where presheaves have been manually encoded by means of quantifications over all weakenings of the current context. Note the lack of naturality conditions thanks to Lemma 8.3. In our setting, typed reduction is simply untyped reduction annotated with proofs that both sides are well-typed and convertible. Similarly, well-typed neutrals are untyped neutrals together with a proof of well-typedness and self-convertibility. Without further consideration for the low-level details, the logical relation satisfies some salient properties that we are going to list below.

Lemma 8.3 (Reducibility irrelevance). *For all proofs of type reducibility $\mathbf{p}_A, \mathbf{q}_A \in \Gamma \Vdash A$, if $\mathbf{p}_A \mid \Gamma \Vdash M : A$ then $\mathbf{q}_A \mid \Gamma \Vdash M : A$ and similarly for the other statements.*

This allows us to silently drop the proof of type formation for reducibility statements as a notational device. As long as there is one, it does not matter which one we pick.

Lemma 8.4 (Reducibility escape). *If $\Gamma \Vdash M : A$ then $\Gamma \vdash M : A$ and similarly for the other statements.*

Lemma 8.5 (Anti-reduction). *Reducibility is closed by typed anti-weak-reduction.*

Lemma 8.6 (Neutral reflection). *Given a weakly neutral term syntactically self-convertible at some type $A$, then it is reducible at that type.*

Just like for standard realizability, we need to close reducibility by substitution to state the fundamental lemma. The relation resulting

Reducibly well-formed types

$$\Gamma \Vdash^{\mathbb{N}} A := \left\{ \; \mathsf{red} \in \Gamma \vdash A \twoheadrightarrow^{*} \mathbb{N} \; \right\} \quad \Gamma \Vdash^{\Pi} A := \left\{ \begin{array}{ll} \mathsf{dom} & \in \mathbf{term} \\ \mathsf{cod} & \in \mathbf{term} \\ \mathsf{red} & \in \Gamma \vdash A \twoheadrightarrow^{*} \Pi x : \mathsf{dom}.\, \mathsf{cod} \\ \mathbf{dom} & \in \Pi(\rho : \Delta \leq \Gamma).\, \Delta \Vdash \mathsf{dom}\{\rho\} \\ \mathbf{cod} & \in \Pi(\rho : \Delta \leq \Gamma)\,(\boldsymbol{x} : \mathbf{dom} \mid \Delta \Vdash x : \mathsf{dom}\{\rho\}).\, \Delta \Vdash \mathsf{cod}\{\rho, x\} \\ \ldots & \end{array} \right\}$$

Reducibly well-typed terms at type $\mathbb{N}$ for $\mathbf{p} \in \Gamma \Vdash^{\mathbb{N}} A$

$$\frac{\Gamma \vdash M \twoheadrightarrow^{*} 0 : \mathbb{N}}{\mathbf{p} \mid \Gamma \Vdash^{\mathbb{N}} M : A} \qquad \frac{\Gamma \vdash M \twoheadrightarrow^{*} \mathsf{S}\, N : \mathbb{N} \qquad \mathbf{p} \mid \Gamma \Vdash^{\mathbb{N}} N : A}{\mathbf{p} \mid \Gamma \Vdash^{\mathbb{N}} M : A} \qquad \frac{\Gamma \vdash M \twoheadrightarrow^{*} N : \mathbb{N} \qquad \mathbf{whne}\, N \qquad \Gamma \vdash_{\mathbf{ne}} N : \mathbb{N}}{\mathbf{p} \mid \Gamma \Vdash^{\mathbb{N}} M : A}$$

Reducibly well-typed terms at product type for $\mathbf{p} \in \Gamma \Vdash^{\Pi} A$

$$\mathbf{p} \mid \Gamma \Vdash^{\Pi} M : A := \left\{ \begin{array}{ll} \mathsf{nf} & \in \mathbf{term} \\ \mathsf{red} & \in \Gamma \vdash M \twoheadrightarrow^{*} \mathsf{nf} : \Pi x : \mathsf{dom}.\, \mathsf{cod} \\ \mathbf{app} & \in \Pi(\rho : \Delta \leq \Gamma)\,(\boldsymbol{x} : \mathbf{dom} \mid \Delta \Vdash x : \mathsf{dom}\{\rho\}).\, \mathbf{cod}\;\boldsymbol{x} \mid \Delta \Vdash \mathsf{nf}\{\rho\}\; x : \mathsf{cod}\{\rho, x\} \\ \ldots & \end{array} \right\}$$

**Figure 12: Reducibility (excerpt)**

from this closure is known in the literature as *validity*. We link to the various notions of validity from the development for reference, but refrain from writing them in full in the paper.

Theorem 8.7 (Fundamental lemma). *Well-typed* MLTT *terms are valid.*

## 9 THE LOGICAL RELATION FOR “MLTT”

For technical reasons, we will work with a slightly tweaked version of the additional typing rules of “MLTT”. The rules from Figure 9 are presented for readability purposes, and will be derivable rules of the actually considered system. The differences are the following.

First, instead of a global typing axiom for run, we add it as a premise to the “MLTT” rules that require it, i.e. those for ϛ and $\varrho$. This is just a cosmetic change that strengthens induction over “MLTT” derivations.

Second, for reasons that will become clear soon, we turn the well-typedness premises of those rules into self-convertibility. For instance, the introduction rule of ϙ becomes:

$$\frac{\Gamma \vdash M \equiv M : \mathbb{N} \to \mathbb{N}}{\Gamma \vdash ϙ\, M : \Lambda}$$

The exact rules are defined as an inductive type in the following file. By reflexivity of term conversion, it is clear that these rules imply the ones from Figure 9. Once we have proved Theorem 9.13, we will also be able to derive that self-convertibility implies well-typedness, which actually shows that the two versions are equivalent.

The model itself is defined in terms of a small-step reduction relation, so we need to define it properly for MLTT, in addition to the big-step variant from Section 6.

*Definition 9.1 (Reduction).* We mutually define weak-head and deep reductions, respectively written $M \twoheadrightarrow N$ and $M \Rrightarrow N$ in Figure 13 for an MLTT excerpt and in Figure 14 for the quoting primitives. Once again, deep reduction for the latter is the same as weak-head reduction.

$$\frac{}{(\lambda x : A.\, M)\; N \twoheadrightarrow M\{x := N\}} \qquad \frac{M \twoheadrightarrow M'}{M\; N \twoheadrightarrow M'\; N}$$

$$\frac{}{(\lambda x : A.\, M)\; N \Rrightarrow M\{x := N\}} \qquad \frac{M \twoheadrightarrow M'}{M\; N \Rrightarrow M'\; N}$$

$$\frac{M \Rrightarrow M' \qquad \mathbf{whne}\, M}{M\; N \Rrightarrow M'\; N} \qquad \frac{N \Rrightarrow N' \qquad \mathbf{dne}\, M}{M\; N \Rrightarrow M\; N'}$$

$$\frac{M \Rrightarrow M'}{\lambda x : A.\, M \Rrightarrow \lambda x : A.\, M'}$$

**Figure 13: MLTT reduction (excerpt)**

Note that deep reduction is simply iterated weak-head reduction on the subterms of weak-head normal forms. These reductions are a specific sequentialization of the corresponding evaluation function, and their reflexive-transitive closure compute the same normal forms. Importantly, these relations are deterministic.

One major remark for our proof to go through is that in Abel-style logical relations, the closure properties of type and term conversion are compatible with the existence of a deep normal form. Said otherwise, we never perform conversion on terms potentially introducing non-termination. This can be leveraged by the following definition.

*Definition 9.2 (Deep Conversion).* We define deep term conversion $\Gamma \vdash_{\mathsf{nf}} M \equiv N : A$ as the predicate $\Gamma \vdash M \equiv N : A$ extended with the following side-conditions.

- There is some $M_0$ s.t. $M \Downarrow M_0$ and $\Gamma \vdash M \equiv M_0 : A$.
- There is some $N_0$ s.t. $N \Downarrow N_0$ and $\Gamma \vdash N \equiv N_0 : A$.

$$\frac{M \Rrightarrow M'}{\text{⚲}\, M \twoheadrightarrow \text{⚲}\, M'} \qquad \frac{\mathbf{clnf}\ M}{\text{⚲}\, M \twoheadrightarrow [\ulcorner M \urcorner]_{\mathbb{N}}}$$

$$\frac{M \Rrightarrow M'}{\text{ʈ}\, M\, N \twoheadrightarrow \text{ʈ}\, M'\, N} \qquad \frac{\mathbf{dnf}\ M \qquad N \Rrightarrow N'}{\text{ʈ}\, M\, N \twoheadrightarrow \text{ʈ}\, M\, N'} \qquad \frac{\mathbf{clnf}\ M \qquad \|M\|\ [n]_{\mathbb{N}} \Downarrow_{k_0} [v]_{\mathbb{N}} \qquad \forall k < k_0.\ \|M\|\ [n]_{\mathbb{N}} \Uparrow_k}{\text{ʈ}\, M\, [n]_{\mathbb{N}} \twoheadrightarrow [k_0]_{\mathbb{N}}}$$

$$\frac{M \Rrightarrow M'}{\varrho\, M\, N \twoheadrightarrow \varrho\, M'\, N} \qquad \frac{\mathbf{dnf}\ M \qquad N \Rrightarrow N'}{\varrho\, M\, N \twoheadrightarrow \varrho\, M\, N'} \qquad \frac{\mathbf{clnf}\ M \qquad \|M\|\ [n]_{\mathbb{N}} \Downarrow_{k_0} [v]_{\mathbb{N}} \qquad \forall k < k_0.\ \|M\|\ [n]_{\mathbb{N}} \Uparrow_k}{\varrho\, M\, [n]_{\mathbb{N}} \twoheadrightarrow [k_0, v]_{\text{⚲}}}$$

**Figure 14: Weak-head reduction of "MLTT" extensions**

- $M_0$ and $N_0$ have the same erasure, i.e. $\|M_0\| = \|N_0\|$.

We define similarly deep type conversion $\Gamma \vdash_{\mathsf{nf}} A \equiv B$ and deep neutral conversion.

When instantiating the logical relation with deep conversions, one gets access to the fact that both sides of the conversion deeply normalize, and furthermore they have the same erasure, i.e. they are equal up to $\eta$-expansions and $\lambda$ and pair annotations. This is the reason for the alternative "MLTT" presentation where typing premises of quoting primitives are turned into self-convertibility. Before proving the fundamental lemma, syntactic deep self-convertibility gives more information than just syntactic typability.

Lemma 9.3 (Deep Typing). *The typing rules where the various conversion predicates are replaced by their deep equivalent satisfy the generic typing interface.*

It is somewhat insightful to remark that the requirement that we erase the normal forms before comparing them is critical for the above lemma. Indeed, the generic conversion rules for the negative $\Pi$ and $\Sigma$ types are given directly as $\eta$-expansions. Therefore, two convertible normal forms may differ on their annotations or up to some $\eta$-expansion. Erasing all annotations and maximally $\eta$-reducing ensures thus that the result is unique for each equivalence class.

As a result, the logical relation instantiated with deep typing is a model of MLTT. We only have to check that the additional "MLTT" rules are also interpreted. Due to the fact that semantic self-convertibility is equivalent to semantic well-typedness, it is enough to check the rules from Figure 9, the ones from Figure 4 are a consequence of the former.

Lemma 9.4 (Quote Reducibility). *The conversion rules for the* ⚲ *primitive hold in the reducibility model.*

Proof. We focus on the congruence rule, since the reduction one is a subcase. Let us assume $\Gamma \Vdash M \equiv M' : \mathbb{N} \to \mathbb{N}$. In particular, we know by escape that $\Gamma \vdash_{\mathsf{nf}} M \equiv M' : \mathbb{N} \to \mathbb{N}$. We thus have two terms $M_0$ and $M'_0$ s.t. $M \Downarrow M_0$, $M' \Downarrow M'_0$ and $\|M_0\| = \|M'_0\|$. From this equality, either both $M_0$ and $M'_0$ are quasi-closed or both are not.

In the latter case, both $\text{⚲}\, M_0$ and $\text{⚲}\, M'_0$ are neutral and convertible, so by reflection they are reducibly convertible. We conclude by anti-reduction.

Otherwise, they reduce to numerals $[\ulcorner M_0 \urcorner]_{\mathbb{N}}$ and $[\ulcorner M'_0 \urcorner]_{\mathbb{N}}$, which are equal by erased equality, and thus reducibly convertible. We conclude again by anti-reduction. □

In order to prove reducibility of the two remaining quoting primitives, we need some fair amount of rewriting theory about our quote-extended $\lambda$-calculus. The computation model is completely untyped, and we will only ever care about erased terms, so we are forced to consider variants of reductions that ignore $\lambda$ and pair annotations. The following results are standard [5], although maybe for the interleaving of weak and deep reduction when reducing the quoting primitives.

*Definition 9.5 (Parallel Reduction).* We define parallel reduction up to $\lambda$ and pair annotations for "MLTT" in the usual way. Since the reduction rules for the quoting primitives are macroscopic, their parallel version cause no trouble.

Lemma 9.6 (Inclusion). *Parallel reduction contains deep reduction.*

Lemma 9.7 (Confluence). *Parallel reduction is confluent.*

*Definition 9.8 (Standard Reduction).* We define standard reduction up to $\lambda$ and pair annotations for "MLTT" in the usual way. Standard reduction for quoting primitives is defined through standard reduction of the subterms rather than closure by weak-head reduction.

Lemma 9.9 (Parallel inclusion). *Standard reduction contains parallel reduction.*

Lemma 9.10 (Standardisation). *Standard reduction to a deep normal form implies deep evaluation up to $\lambda$ and pair annotations.*

Lemma 9.11 (Erasure Irrelevance). *Standard reduction to a deep normal form is preserved by erasure.*

The equivalences linking these various reductions and evaluations make it easy to switch to one view or another, depending on the kind of property one wants to show. With these tools in hand, we can tackle the remaining reducibility proofs. Note also that computational adequacy is not required for any of the above properties, it is only used in the following lemma.

Lemma 9.12 (Reflection Reducibility). *The conversion rules for the* ʈ *and* $\varrho$ *primitives hold in the reducibility model.*

Proof. Once again we focus on the congruence rule. Since the two primitives behave computationally the same, we only treat the $\varrho$ case which is more involved typingwise. Let us assume $\Gamma \Vdash M \equiv M' : \mathbb{N} \to \mathbb{N}$ and $\Gamma \Vdash N \equiv N' : \mathbb{N}$. By the same argument as before, we have deeply normal terms $M_0$, $M'_0$ (resp. $N_0$ and $N'_0$) convertible to the original terms and with the same erasure, and thus with the same closedness.

If not all these terms are closed, then $\varrho\, M_0\, N_0$ and $\varrho\, M_0'\, N_0'$ are convertible neutral terms, we conclude by the same argument as above.

Otherwise, they are all closed. In this case, by reducibility, $N_0 = N_0' = [n]_{\mathbb{N}}$ for some $n \in \mathbf{N}$. Since reducibility is closed by application, we get $\Gamma \Vdash M\, N \equiv M'\, N' : \mathbb{N}$. This implies that both applications deeply evaluate to the same semantic integer, i.e. a series of successors ending either with 0 or a neutral. By confluence, we rule out the second case as $M_0\, N_0$ is closed, so these terms reduce to some numeral $[v]_{\mathbb{N}}$. By confluence and standardisation, $M_0\ [n]_{\mathbb{N}}$ also evaluates to $[v]_{\mathbb{N}}$. By irrelevance of erasure, $\|M_0\|\ [n]_{\mathbb{N}}$ also evaluates to $[v]_{\mathbb{N}}$. By computational adequacy, there exists some $k_0$ s.t. $\mathbf{run}\ [\![M_0]\!]\ n\ k_0 \Downarrow \mathsf{S}\ [v]_{\mathbb{N}}$ and $\mathbf{run}\ [\![M_0]\!]\ n\ k' \Downarrow 0$ for all $k' < k_0$. Given that $M_0$ and $M_0'$ have the same erasure, this also holds for $M_0'$. Hence, we can fire the $\rotatebox[origin=c]{180}{$\varrho$}$ and $\varrho$ reduction rules, so that $\rotatebox[origin=c]{180}{$\varrho$}\, M\, N$ and $\rotatebox[origin=c]{180}{$\varrho$}\, M'\, N'$ evaluate to $[k_0]_{\mathbb{N}}$ and $\varrho\, M\, N$ and $\varrho\, M'\, N'$ evaluate to $[k_0, v]_{\text{♀}}$. We are almost done, the only remaining problem is to show that this value is indeed of semantic type $M\ \text{♀}\ [n]_{\mathbb{N}}$. But this is a trivial, albeit technically annoying consequence of the previous normalizations properties of the various terms implied. □

Finally, one also has to prove validity for these conversion rules, that is to say that they are stable by substitution. This turns out to be a trivial fact. Congruence rules are stable by substitution by construction. As for the other conversion rules, it is enough to observe that both quasi-closedness and erasure of quasi-closed terms are stable by substitution. We get the generalization below as an immediate result.

Theorem 9.13 (Fundamental Lemma for "MLTT"). *Well-typed* "MLTT" *terms are valid.*

The metatheoretical facts we claimed in Section 7 are direct consequences of the basic properties of the logical relation. Hence we are done.

## 10 MECHANIZATION

While the Abel et al. proof has been implemented in Agda [1], our mechanization is written in Coq and is based upon a recent work by Adjedj et al. [2] that encodes away induction-recursion into standard inductive types through a technique known as small induction recursion [15]. Apart from this, the two formalizations are globally the same. The only advantage of the Coq version, which we believe to be decisive, is the availability of tactics. The largish corpus of proofs relating the numerous untyped reductions are rendered tractable by the reliance on handcrafted automatization, when such proofs would have been a nightmare to write explicitly in Agda as terms.

All theorems stated in the paper have been formalized. The only thing that we did not formally prove was the actual existence of adequate models of computation. We globally axiomatized one in the development instead. The precise list of axioms can be retrieved by applying the `Print Assumptions` command to the toplevel theorems.

The reader may reasonably complain about the use of axioms, as they may very well be inconsistent. Yet, the existence of adequate models is at the same time an utterly trivial fact, and a technically extremely challenging task. Indeed, at an intuitive level `run` has *already* been implemented in MLTT, because we defined evaluation as a step-indexed function in our metatheory, which turns out to be a variant of MLTT. As explained before, we do not even need that much, the fragment we use to implement basically fits into PRA. In any paper proof, such a computability result would be immediately brushed off as obvious, barely requiring any explanation. Now, in the land of formal proofs, given the non-trivial size of our untyped language, implementing evaluation and proving anything about it in Coq was already cumbersome with transparent conversion rules, tactics and handy notations. By contrast, doing the same directly in the object theory is monstruous enough to drive any sane person into sheer madness, let alone considering that we have to work up to a Gödel encoding. Out of common decency we will not attempt to close this unquestionable gap.

## 11 FUTURE WORK

Although we did not formally prove it out of unrepented sloth, it is quite clear that "MLTT" enjoys decidability of type-checking. Since type interpretation is unchanged in the logical relation, the global shape of the algorithm remains the same as the one for MLTT. The only potential trouble comes from computational behaviour of the quoting primitives. We briefly discuss why they cause no issue. At heart, the usual algorithm applies rules eagerly for typing, and switches to another behaviour when facing conversion, in which case it recursively weak-head reduces the compared terms and compares subterms modulo $\eta$-expansion. In order to handle "MLTT", we thus need to ensure that weak-head evaluation to a normal form is computable on well-typed terms. But this is a consequence of the fundamental lemma, and actually, if we squint at our reducibility proofs long enough, they already contain a conversion algorithm. Also, given that our Coq proof is based on a development that proves decidability of type-checking for MLTT [2], it should be quite simple to prove it for "MLTT".

Another topic that may be of interest is the compatibility of MLTT with the full Russian axiomatic. We know that MLTT is compatible with MP, and after this paper we know that MLTT is compatible with CT. Hence a natural question is whether MLTT can accommodate both axioms at the same time. We are convinced that not only this is the case, but also that the model described here can be easily extended to handle MP. The reason for this confidence lies in the uncanny similarity between those two principles. Just like CT, it is true that Markov's rule, the external version of MP, is derivable in MLTT. This can be showed using a variant of Friedman's trick [9, 28, 31]. Again, like CT, the return type of this rule is a concrete data type, i.e. a $\Sigma^0_1$ formula. It should therefore be easy to add a binary term former $\mu$ for MP

$$\frac{\Gamma \vdash M : \mathbb{N} \to \mathbb{N} \qquad \Gamma \vdash N : \neg\neg(\Sigma n : \mathbb{N}.\, M\ n = 0)}{\Gamma \vdash \mu\, M\, N : \Sigma n : \mathbb{N}.\, M\ n = 0}$$

that blocks until both arguments are closed and returns the witness extracted by Friedman's trick when this is the case. If we do not care about decidability, we could even opt for an unbounded search. Note that one must be careful to return the same integer for convertible arguments, i.e. $\mu$ must be congruent, but this is easily obtained as the return type can be turned into an `hProp` by considering instead the *smallest* index where $M$ evaluates to 0.

More generally, we trust that many such axioms can be added to MLTT this way. The constraints are that these axioms should be admissible on closed terms and return $\Sigma^0_1$ formulae. Furthermore, the external process turning closed proofs into proofs of the conclusion should produce a unique term for each equivalence class of convertible arguments. We leave the exploration of this subject for a later time.

## 12 CONCLUSION

In this paper, we proved that it was possible to add $\mathsf{CT}_\Sigma$ to MLTT without disrupting any of the desirable properties of the resulting system such as consistency, strong normalization and canonicity. Against all odds, the model used for this is the most straightforward one can conceive, as it boils down to *the* standard logical relation for MLTT. The computational content of our variant of Church's thesis is too borderline stupid to even be considered unsurprising: it computes on closed terms, period. The only mildly non-trivial phenomenon is the need for erasure of annotations and maximal $\eta$-reduction, but that trick is hardly worthy of attention. As a result, we are still bewildered about the reason for which this problem was believed to be hard.

The theorems have been formalized in Coq, greatly reducing the risk of an accidental error in the proof. For simplicity, the development still axiomatizes the computation model in the object theory. In all likelihood, these axioms would be deemed self-evident in a computability theory paper proof.

Finally, we believe that the generic recipe we followed for this model can be generalized to many other admissible principles in dependent type theory.

## REFERENCES

[1] Andreas Abel, Joakim Öhman, and Andrea Vezzosi. 2017. Decidability of Conversion for Type Theory in Type Theory. *Proc. ACM Program. Lang.* 2, POPL, Article 23 (Dec. 2017), 29 pages. https://doi.org/10.1145/3158111

[2] Arthur Adjedj, Meven Lennon-Bertrand, Kenji Maillard, Pierre-Marie Pédrot, and Loïc Pujet. 2024. Martin-Löf à la Coq. In *Proceedings of the 13th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2024, London, UK, January 15-16, 2024*, Sandrine Blazy, Brigitte Pientka, Amin Timany, and Dmitriy Traytel (Eds.). ACM. https://doi.org/10.1145/3636501.3636951

[3] Matt Brown and Jens Palsberg. 2016. Breaking through the normalization barrier: a self-interpreter for f-omega. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, Rastislav Bodík and Rupak Majumdar (Eds.). ACM, 5–17. https://doi.org/10.1145/2837614.2837623

[4] Thierry Coquand and Bassel Mannaa. 2017. The Independence of Markov's Principle in Type Theory. *Log. Methods Comput. Sci.* 13, 3 (2017). https://doi.org/10.23638/LMCS-13(3:10)2017

[5] René David. 1995. Une preuve simple de résultats classiques en lambda-calcul. *Comptes rendus de l'Académie des sciences. Série I, Mathématique* 320 (Jan. 1995), p 1401–1406. https://hal.science/hal-00384995

[6] Yannick Forster. 2021. Church's Thesis and Related Axioms in Coq's Type Theory. In *29th EACSL Annual Conference on Computer Science Logic, CSL 2021, January 25-28, 2021, Ljubljana, Slovenia (Virtual Conference) (LIPIcs, Vol. 183)*, Christel Baier and Jean Goubault-Larrecq (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 21:1–21:19. https://doi.org/10.4230/LIPICS.CSL.2021.21

[7] Yannick Forster. 2021. *Computability in constructive type theory.* Ph. D. Dissertation. Saarland University, Germany. https://d-nb.info/1255182792

[8] Yannick Forster, Dominik Kirst, Bruno da Rocha Paiva, and Vincent Rahli. 2023. Markov's Principles in Constructive Type Theory. In *29th International Conference on Types for Proofs and Programs, TYPES 2023, June 12-15, 2023, Universitat Politècnica de València, Spain.*

[9] Harvey Friedman. 2007. *Classically and intuitionistically provably recursive functions.* Vol. 669. 21–27. https://doi.org/10.1007/BFb0103100

[10] Gaëtan Gilbert. 2019. *A type theory with definitional proof-irrelevance. (Une théorie des types avec insignifiance des preuves définitionnelle).* Ph. D. Dissertation. Mines ParisTech, France. https://tel.archives-ouvertes.fr/tel-03236271

[11] Jean-Yves Girard, Paul Taylor, and Yves Lafont. 1989. *Proofs and Types.* Cambridge University Press.

[12] Kurt Gödel. 1958. Über eine bisher noch nicht benützte Erweiterung des finiten Standpunktes. *Dialectica* 12, 3-4 (1958), 280–287. https://doi.org/10.1111/j.1746-8361.1958.tb01464.x

[13] Daniel Gratzer, G. A. Kavvos, Andreas Nuyts, and Lars Birkedal. 2021. Multimodal Dependent Type Theory. *Log. Methods Comput. Sci.* 17, 3 (2021). https://doi.org/10.46298/LMCS-17(3:11)2021

[14] Håkon Robbestad Gylterud. 2019. Quoting operations as extensions of $\lambda$-calculus and type theory. (2019). https://hakon.gylterud.net/research/quote/cas-2019-02.pdf

[15] Peter Hancock, Conor McBride, Neil Ghani, Lorenzo Malatesta, and Thorsten Altenkirch. [n. d.]. Small Induction Recursion. In *Typed Lambda Calculi and Applications* (Berlin, Heidelberg, 2013), Masahito Hasegawa (Ed.). Springer Berlin Heidelberg, 156–172. https://doi.org/10.1007/978-3-642-38946-7_13

[16] Jason Z. S. Hu, Brigitte Pientka, and Ulrich Schöpp. 2022. A Category Theoretic View of Contextual Types: From Simple Types to Dependent Types. *ACM Trans. Comput. Log.* 23, 4 (2022), 25:1–25:36. https://doi.org/10.1145/3545115

[17] John Hyland. 1982. The Effective Topos. *Studies in logic and the foundations of mathematics* 110 (1982), 165–216.

[18] Hajime Ishihara, Maria Emilia Maietti, Samuele Maschio, and Thomas Streicher. 2018. Consistency of the intensional level of the Minimalist Foundation with Church's thesis and axiom of choice. *Arch. Math. Log.* 57, 7-8 (2018), 873–888. https://doi.org/10.1007/S00153-018-0612-9

[19] Junyoung Jang, Samuel Gélineau, Stefan Monnier, and Brigitte Pientka. 2022. Mœbius: metaprogramming using contextual types: the stage where System F can pattern match on itself. *Proc. ACM Program. Lang.* 6, POPL (2022), 1–27. https://doi.org/10.1145/3498700

[20] Stephen Cole Kleene. 1943. Recursive Predicates and Quantifiers. *Trans. Amer. Math. Soc.* 53 (1943), 41–73. https://doi.org/10.2307/2267986

[21] Stephen Cole Kleene. 1945. On the interpretation of intuitionistic number theory. *Journal of Symbolic Logic* 10 (1945), 109 – 124.

[22] Pierre Letouzey. 2004. *Programmation fonctionnelle certifiée : L'extraction de programmes dans l'assistant Coq. (Certified functional programming : Program extraction within Coq proof assistant).* Ph. D. Dissertation. University of Paris-Sud, Orsay, France. https://tel.archives-ouvertes.fr/tel-00150912

[23] Maria Emilia Maietti. 2009. A minimalist two-level foundation for constructive mathematics. *Ann. Pure Appl. Log.* 160, 3 (2009), 319–354. https://doi.org/10.1016/J.APAL.2009.01.006

[24] Maria Emilia Maietti and Giovanni Sambin. 2005. Toward a minimalistic foundation for constructive mathematics. In *From sets and types to topology and analysis - Towards practicable foundations for constructive mathematics*, Laura Crosilla and Peter M. Schuster (Eds.). Oxford logic guides, Vol. 48. Oxford University Press.

[25] Maurice Margenstern. 1995. L'école constructive de Markov. *Revue d'histoire des mathématiques* (1995).

[26] Per Martin-Löf. 1984. *Intuitionistic type theory.* Studies in proof theory, Vol. 1. Bibliopolis.

[27] Aleksandar Nanevski, Frank Pfenning, and Brigitte Pientka. 2008. Contextual modal type theory. *ACM Trans. Comput. Log.* 9, 3 (2008), 23:1–23:49. https://doi.org/10.1145/1352582.1352591

[28] Erik Palmgren. 1995. The Friedman-Translation for Martin-Löf's Type Theory. *Math. Log. Q.* 41 (1995), 314–326. https://doi.org/10.1002/MALQ.19950410304

[29] Christine Paulin-Mohring. 1989. *Extraction de programmes dans le Calcul des Constructions. (Program Extraction in the Calculus of Constructions).* Ph. D. Dissertation. Paris Diderot University, France. https://tel.archives-ouvertes.fr/tel-00431825

[30] Pierre-Marie Pédrot. 2020. Russian Constructivism in a Prefascist Theory. In *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*, Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller (Eds.). ACM, 782–794. https://doi.org/10.1145/3373718.3394740

[31] Pierre-Marie Pédrot and Nicolas Tabareau. 2018. Failure is Not an Option - An Exceptional Type Theory. In *Programming Languages and Systems - 27th European Symposium on Programming, ESOP 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings (Lecture Notes in Computer Science, Vol. 10801)*, Amal Ahmed (Ed.). Springer, 245–271. https://doi.org/10.1007/978-3-319-89884-1_9

[32] Fred Richman. 1983. Church's Thesis Without Tears. *J. Symb. Log.* 48, 3 (1983), 797–803. https://doi.org/10.2307/2273473

[33] Egbert Rijke, Michael Shulman, and Bas Spitters. 2020. Modalities in homotopy type theory. *Log. Methods Comput. Sci.* 16, 1 (2020). https://doi.org/10.23638/LMCS-16(1:2)2020

[34] Matthieu Sozeau, Abhishek Anand, Simon Boulier, Cyril Cohen, Yannick Forster, Fabian Kunze, Gregory Malecha, Nicolas Tabareau, and Théo Winterhalter. 2020. The MetaCoq Project. *J. Autom. Reason.* 64, 5 (2020), 947–999. https://doi.org/10.1007/S10817-019-09540-0

[35] Andrew W. Swan and Taichi Uemura. 2021. On Church's thesis in cubical assemblies. *Math. Struct. Comput. Sci.* 31, 10 (2021), 1185–1204. https://doi.org/10.1017/S0960129522000068

[36] A.S. Troelstra and D. Dalen. 1988. *Constructivism in Mathematics: An Introduction.* Number vol. 1 in Constructivism in Mathematics. North-Holland.

[37] The Univalent Foundations Program. 2013. *Homotopy Type Theory: Univalent Foundations of Mathematics.* https://homotopytypetheory.org/book, Institute for Advanced Study.

[38] Benjamin Werner. 1997. Sets in Types, Types in Sets. In *Theoretical Aspects of Computer Software, Third International Symposium, TACS '97, Sendai, Japan, September 23-26, 1997, Proceedings (Lecture Notes in Computer Science, Vol. 1281)*, Martín Abadi and Takayasu Ito (Eds.). Springer, 530–346. https://doi.org/10.1007/BFB0014566