



HAL
open science

Definitional Functoriality for Dependent (Sub)Types

Théo Laurent, Meven Lennon-Bertrand, Kenji Maillard

► **To cite this version:**

Théo Laurent, Meven Lennon-Bertrand, Kenji Maillard. Definitional Functoriality for Dependent (Sub)Types. 2023. hal-04160858v2

HAL Id: hal-04160858

<https://hal.science/hal-04160858v2>

Preprint submitted on 23 Oct 2023 (v2), last revised 8 Apr 2024 (v3)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution 4.0 International License

Definitional Functoriality for Dependent (Sub)Types

THÉO LAURENT, Inria, France

MEVEN LENNON-BERTRAND, University of Cambridge, United Kingdom

KENJI MAILLARD, Gallinette Project Team, Inria, France

Dependently-typed proof assistants rely crucially on definitional equality, which relates types and terms that are automatically identified in the underlying type theory. This paper extends type theory with definitional *functor laws*, equations satisfied propositionally by a large class of container-like type constructors $F: \text{Type} \rightarrow \text{Type}$, equipped with a $\text{map}_F: (A \rightarrow B) \rightarrow F A \rightarrow F B$, such as lists or trees. Promoting these equations to definitional ones strengthen the theory, enabling slicker proofs and more automation for functorial type constructors. This extension is used to modularly justify a structural form of coercive subtyping, propagating subtyping through type formers in a map-like fashion. We show that the resulting notion of coercive subtyping, thanks to the extra definitional equations, is equivalent to a natural and implicit form of subsumptive subtyping. The key result of decidability of type-checking in a dependent type system with functor laws for lists has been entirely mechanized in Coq.

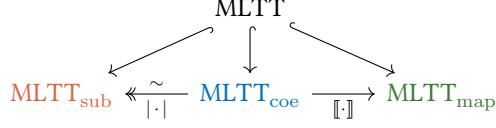
CCS Concepts: • **Theory of computation** → **Type theory**.

Additional Key Words and Phrases: Subtyping, Dependent type system, Bidirectional typing, Logical relation.

1 INTRODUCTION

Dependent type theory is the foundation of many proof assistants: Coq [The Coq Development Team 2022], LEAN [Moura and Ullrich 2021], AGDA [Agda Development Team 2023], IDRIS [Brady 2021], F* [Swamy et al. 2016]. At its heart lies definitional equality, an equational theory that is automatically decided by the implementation of these proof systems. The more expressive definitional equality is, the less work is requested from users to identify objects. However, there is a fundamental tension at play: making the equational theory too rich leads to both practical and theoretical issues, the most prominent one being the undecidability of definitional equality. This default plagues the otherwise appealing Extensional Type Theory (ETT) [Martin-Löf and Sambin 1984], a type theory which makes every provable equality definitional, thus making ETT rather impractical as a basis for a proof assistant [Castellan et al. 2017]. As a result, to design usable proof assistants we need to carve out a well-behaved equational theory, that strikes the right balance between expressivity and decidability. In this paper, we show that we can maintain this subtle balance while extending intensional type theory with map operations explicating the functorial character of type former, and satisfying *definitional functor laws*. We prove in particular that definitional equality and type-checking remain decidable in this extension, that we dub MLTT_{map} .

The map primitives introduced in MLTT_{map} have a computational behaviour reminiscent of *structural subtyping*, which propagates existing subtyping structurally through type-formers, and should satisfy reflexivity and transitivity laws similar to the functor laws. Guided by the design of MLTT_{map} , we devise a second system, MLTT_{coe} , with explicit coercions witnessing structural subtyping. To gauge the expressivity of MLTT_{coe} , we relate it to a third system, MLTT_{sub} , where subtyping is implicit, as users of a type system should expect. A simple translation $|\cdot|$ from MLTT_{coe} to MLTT_{sub} erases coercions. We show that this erasure can be inverted, elaborating coercions back. For this to be type preserving, it is crucial that MLTT_{coe} satisfies our new definitional equalities, which allows us to reflect the equations implicitly satisfied in MLTT_{sub} due to coercions being transparent. Fig. 1 synthesizes the three theories that we introduce and their relationships. They

Fig. 1. Relation between MLTT, MLTT_{map} , MLTT_{coe} and MLTT_{sub}

all extend Martin-Löf Type Theory (MLTT) [Martin-Löf and Sambin 1984].¹ Let us now explore in more details these three systems.

Functors and their laws. The notion of functor is pervasive both in mathematics [MacLane 1971] and functional programming [Lipovača 2010], capturing the concept of *a parametrized construction applying to objects and their transformations*. In the setting of type theory, a type former $F: \text{dom}(F) \rightarrow \text{Type}$ is a functor when it is equipped with an operation $\text{map}_F f: F A \rightarrow F B$ for any morphism $f: \text{hom}_F(A, B)$ between two object A, B in the domain of F . A functor should also respect the specified categorical structure on $\text{dom}(F)$, preserving identities and compositions:

$$\text{map}_F \text{id}^F = \text{id} \quad (\text{id-eq})$$

$$(\text{map}_F f) \circ (\text{map}_F g) = \text{map}_F (f \circ^F g) \quad (\text{comp-eq})$$

These two equations are known as the *functor laws*. For many container-like functors, such as $\text{List } A$, lists of elements taken in a type A , a map function can be defined in vanilla type theory such that these equations can be shown *propositionally*, e.g. by induction. Such propositional equations need however to be used explicitly while being careful that all identifications relying on them are made coherently. This is not acceptable from a user perspective: such structural and naturally occurring identifications should hold definitionally!

Example 1.1 (Representation change). Consider a dataset of pairs of a number and a boolean, represented as a list of numbers. For compatibility purpose, we may need to embed these pairs into a larger dataset using

$$\text{glue } (r : \{a: \mathbf{N}; b: \mathbf{B}\}) : \{x: \mathbf{B}; y: \mathbf{N}; z: \mathbf{N}\} \stackrel{\text{def}}{=} \{x := r.b; y := r.a; z := \text{if } r.b \text{ then } r.a \text{ else } 42\}.$$

Going from one dataset to the other amounts to map either `glue` or its left inverse `glue_retr`, which forgets the extra field:

$$\begin{aligned} \text{map}_{\text{List}} \text{ glue} &: \text{List } \{a: \mathbf{N}; b: \mathbf{B}\} \rightarrow \text{List } \{x: \mathbf{B}; y: \mathbf{N}; z: \mathbf{N}\}, \\ \text{map}_{\text{List}} \text{ glue_retr} &: \text{List } \{x: \mathbf{B}; y: \mathbf{N}; z: \mathbf{N}\} \rightarrow \text{List } \{a: \mathbf{N}; b: \mathbf{B}\}. \end{aligned}$$

If the functor laws only hold propositionally, each consecutive simplification of back and forth changes of representation needs to be explicitly lifted to lists, and applied. The uncontrolled accumulation of repetitive proof steps, even as simple as these, can quickly burden proof development. In presence of definitional functor laws, instead, any sequence of representation changes will reduce to a single map_{List} : the boilerplate of explicitly manipulating the functor laws is handled automatically by the type theory. Moreover, observe that in this example the retraction $\text{glue_retr} \circ \text{glue} \cong \text{id}$ is definitional thanks to surjective pairing. Combined with definitional functor laws, the following simplification step is discharged automatically by the type-checker:²

$$\text{map}_{\text{List}} \text{ glue_retr } (\text{map}_{\text{List}} \text{ glue } l) \cong \text{map}_{\text{List}} \text{ id } l \cong l$$

¹We provide a translation from MLTT_{coe} to MLTT_{map} $[\cdot]$ in Appendix C.3 and conjecture it is type-preserving, but do not establish any formal result on the translation. None of the results we present rely on this translation.

²We formalize this example, showing that this conversion indeed holds in our system, in file `Example_1_1`.

Note that these equations are valid in any context, in particular under binders, whereas for propositional identifications, rewriting under binders is only possible in presence of the additional axiom of function extensionality.

Example 1.2 (Coherence of coercions). Proof assistants may provide the ability for users to declare automatically-inserted functions acting as glue code (coercions in `Coq`, instance arguments in `AGDA`, `has_coe` typeclass in `LEAN`). Working with natural (\mathbf{N}), integer (\mathbf{Z}) and rational (\mathbf{Q}) numbers, we want every \mathbf{N} to be automatically coerced to an integer, and so declare a `natToZ` coercion. Similarly, we can also declare a `ZToQ` coercion. If we write `0` (a \mathbf{N}) where a \mathbf{Q} is expected, this is accepted, and `0` is silently transformed to `ZToQ (natToZ 0)`.

Now, if we want the same mechanism to apply when we pass the list `[0 :: 1 :: 2]` to a function expecting a `List Q`, we need to provide a way to propagate the coercions on lists. We can expect to solve this problem by declaring `map_List` as a coercion, too: whenever there is a coercion $f : A \rightarrow B$, then `map_List f` should be a coercion from `List A` to `List B`. However, by doing so, we would cause more trouble than we solve, as there would be two coercions from `List N` to `List Q`, `map_List (ZToQ ∘ natToZ)` and `(map_List ZToQ) ∘ (map_List natToZ)`. In the absence of definitional functor laws for `map_List`, these two are *not* definitionally equal. To add insult to injury, coercions are by default not printed to the user, yielding puzzling error messages like “*l and l are not convertible*” (!), because one is secretly `map_List (ZToQ ∘ natToZ) l` while the other is `map_List ZToQ (map_List natToZ l)`. This makes `map` virtually unusable with coercions.

Structural subtyping. This last example suggests a connection with *subtyping*. Subtyping equips the collection of type with a *subtyping order* \preceq that allows to seamlessly transport terms from a subtype to a supertype, *i.e.* from A to A' when $A \preceq A'$. An important aspect of subtyping is *structural subtyping*, *i.e.* how an existing subtyping notion extends structurally through other type formers of the type theory. Typically, we want to have `List A` \preceq `List A'` whenever $A \preceq A'$. In the context of the F^* program verification platform that heavily uses refinement subtyping, the inability to propagate subtyping on inductive datatypes such as lists has been a long-standing issue that never got solved properly [Hrițcu 2014]. The absence of structural subtyping also has a history of causing difficulties to `AGDA` [Cockx 2020a; Escot, Poiret, et al. 2023].

Definitional equalities for subtyping. From interactive theorem prover users' perspective, subtyping should be implicit, transparently providing the expected glue to smoothen the writing of complex statements. From a meta-theoretical perspective, on the other hand, it is useful to explicitly represents all the necessary information of a typing derivation, including where subtyping is used. The first approach is known as *subsumptive subtyping*, on the left, whereas the latter is embodied by *coercive subtyping*, on the right:

$$\text{SUB} \frac{\Gamma \vdash_{\text{sub}} t : A \quad \Gamma \vdash_{\text{sub}} A \preceq A'}{\Gamma \vdash_{\text{sub}} t : A'} \qquad \text{COE} \frac{\Gamma \vdash_{\text{coe}} t : A \quad \Gamma \vdash_{\text{coe}} A \preceq A'}{\Gamma \vdash_{\text{coe}} \text{coe}_{A,A'} t : A'}$$

We want to present subsumptive subtyping to users, but ground the system on the better-behaved coercive subtyping. Informally, an application of `SUB` in the subsumptive type theory MLTT_{sub} should correspond to an application of `COE` in the coercive type theory MLTT_{coe} . However, given a derivation \mathcal{D} of $\Gamma \vdash_{\text{sub}} t : A$ we can apply `SUB` together with a reflexivity proof $\Gamma \vdash_{\text{sub}} A \preceq A$ to yield a new derivation \mathcal{D}' with the same conclusion $\Gamma \vdash_{\text{sub}} t : A$. \mathcal{D} and \mathcal{D}' respectively give terms $\Gamma \vdash_{\text{coe}} t' : A$ and $\Gamma \vdash_{\text{coe}} \text{coe}_{A,A} t' : A$ in MLTT_{coe} . Since t' and $\text{coe}_{A,A} t'$ both correspond to the same MLTT_{sub} term t , they need to be equated if we want both type theories to be equivalent. Similarly, transitivity of subtyping implies that coercions should compose definitionally, that is $\Gamma \vdash_{\text{coe}} \text{coe}_{B,C} (\text{coe}_{A,B} t') \cong \text{coe}_{A,C} t' : C$ should always hold in MLTT_{coe} .

Functor laws meet structural subtyping. Z. Luo and Adams [2008] showed that the functorial composition law `comp-eq` is enough to make structural coercive subtyping compose definitionally, because a structural coercion between lists $\text{coe}_{\text{List } A, \text{List } B}$ behaves exactly as the function obtained by mapping $\text{coe}_{A, B}$ on every element of the list. We further investigate this bridge between coercive subtyping and functoriality of type formers, in particular the identity functor law `id-eq` needed to handle reflexivity of subtyping, and extend Z. Luo and Adams’s limited type system to full-blown Martin-Löf Type Theory, with universes and large elimination. This understanding leads to a modular design of subtyping: structural subtyping for a type former relies on a functorial structure, and can be considered orthogonally to other type formers of the theory or to the base subtyping. Moreover, definitional functor laws are sufficient to make structural coercive subtyping for any type former flexible enough to interpret subsumptive subtyping.

Contributions. We make the following contributions:

- we design MLTT_{map} , an extension of Martin-Löf Type Theory (MLTT) exhibiting the functorial nature of standard type formers ($\Pi, \Sigma, \text{List}, \mathbf{W}, \text{Id}, +$) and satisfying definitional functor laws (Section 3);
- we mechanize the metatheory of a substantial fragment of MLTT_{map} in Coq, extending a formalization of MLTT [Adjedj et al. 2023], proving it is normalizing and has decidable type-checking (Section 4);
- we develop bidirectional presentations for MLTT_{sub} and MLTT_{coe} , which extend MLTT respectively with subsumptive and coercive subtyping;
- we leverage these presentations and the extra functorial equations satisfied by `coe` in MLTT_{coe} to give back and forth, type-preserving translations between the two systems (Section 5).

The necessary technical background, notations and definitions for MLTT are introduced in Section 2, while Section 6 details the related and future work. Detailed proofs and complete typing rules are given in appendix.

2 TYPE THEORY AND ITS METATHEORY

We work in the setting of dependent type theories *à la* Martin-Löf (MLTT) [Martin-Löf and Sambin 1984], an ideal abstraction of the type theories underlying existing proof assistants such as AGDA, Coq, F* or LEAN. The (declarative) typing rules describing MLTT use five categories of judgements, characterizing the well-formed context, types and terms (Figure 2), and providing the equational theory on types and terms (Figure 3). Two terms related by this equational theory are said to be *definitionally equal* or *convertible*, to stress on the fact that these terms will be identified by proof assistants implementing this theory, without any need for manual equational reasoning.

Variables and substitution. Throughout the paper, we use named variables ($x, y \dots$) for readability purposes, but we think of them as de Bruijn indices, which is what we use in the Coq formalization. In particular, we do not consider freshness conditions. A substitution σ consists of a list of terms, and we write $t[\sigma]$ for its parallel substitution in the term t . The substitution (id, u) replaces the 0th de Bruijn index by the term u , leaving all other variables intact. We will write $t[(\text{id}, u)]$ simply as $t[u]$, which would be written $t[x := u]$ in more verbose notation, if x correspond to the 0th de Bruijn index in t . Typing in all systems is extended pointwise to substitutions.

Negative types: dependent products and sums. Dependent function types $\Pi x: A. B$ are introduced using a λ -abstraction $\lambda x: A. t$ and eliminated with application $t u$. We use braces to indicate arguments that will be left implicit, e.g. $\lambda\{x: A\}. t$ of type $\Pi\{x: A\}. B$. We also include dependent (strong) sum types $\Sigma x: A. B$, introduced with pairs $(t, u)_{x.B}$ and eliminated through projections $\pi_1 p$ and $\pi_2 p$. Both of these come with an η -law beside their standard β -laws.

$$\boxed{\Gamma \vdash T}$$
 Type T is well-formed under context Γ

$$\boxed{\Gamma \vdash t : T}$$
 Term t has type T under context Γ

$$\begin{array}{c}
 \text{VAR} \frac{\vdash \Gamma \quad (x : A) \in \Gamma}{\Gamma \vdash x : A} \qquad \text{SORT} \frac{\vdash \Gamma}{\Gamma \vdash \text{Type}_i : \text{Type}_{i+1}} \qquad \text{EL} \frac{\Gamma \vdash A : \text{Type}_i}{\Gamma \vdash A} \\
 \\
 \text{FUN} \frac{\Gamma \vdash A : \text{Type}_i \quad \Gamma, x : A \vdash B : \text{Type}_i}{\Gamma \vdash \Pi x : A. B : \text{Type}_i} \qquad \text{ABS} \frac{\Gamma \vdash A \quad \Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x : A. t : \Pi x : A. B} \qquad \text{APP} \frac{\Gamma \vdash t : \Pi x : A. B \quad \Gamma \vdash u : A}{\Gamma \vdash t u : B[u]} \\
 \\
 \text{LIST} \frac{\Gamma \vdash A : \text{Type}_i}{\Gamma \vdash \mathbf{List} A : \text{Type}_i} \qquad \text{NIL} \frac{\Gamma \vdash A}{\Gamma \vdash \varepsilon_A : \mathbf{List} A} \qquad \text{CONS} \frac{\Gamma \vdash A \quad \Gamma \vdash a : A \quad \Gamma \vdash l : \mathbf{List} A}{\Gamma \vdash a ::_A l : \mathbf{List} A} \\
 \\
 \text{LISTIND} \frac{\Gamma, l : \mathbf{List} A \vdash P \quad \Gamma \vdash A \quad \Gamma \vdash s : \mathbf{List} A \quad \Gamma \vdash b_\varepsilon : P[\varepsilon_A] \quad \Gamma, a : A, l : \mathbf{List} A, h : P[l] \vdash b_\bullet : P[a ::_A l]}{\Gamma \vdash \text{ind}_{\mathbf{List} A}(s; l.P; b_\varepsilon, a.l.h.b_\bullet) : P[s]} \\
 \\
 \text{CONV} \frac{\Gamma \vdash t : A \quad \Gamma \vdash A \cong B}{\Gamma \vdash t : B}
 \end{array}$$

Fig. 2. Declarative typing for MLTT (Complete rules: Appendix B.1)

Universes of types. Rule **SORT** introduce a countable hierarchy of universes Type_i , which are types for types. Any inhabitant of a universe is a well-formed type by **EL** and, in order to make the presentation compact, we do not repeat rules applying both for universes and types, implicitly assuming that a rule given for terms of some universe Type_i has a counterpart as a type judgement whenever it makes sense. Appendix B provides the full set of rules for reference. For instance, in addition to **FUN**, we have a type-level equivalent

$$\text{FUNTY} \frac{\Gamma \vdash A \quad \Gamma, x : A \vdash B}{\Gamma \vdash \Pi x : A. B}$$

Positive types: inductive types and families. As we wish to study the functorial status of type formers, *parametrized* inductive types are our main focus. Our running example is the type of lists $\mathbf{List} A$, parametrized by a single type A , and inhabited by the empty list ε_A and the consing $hd ::_A tl$ of a head $hd : A$ onto a tail $tl : \mathbf{List} A$. Lists are eliminated using the dependent eliminator $\text{ind}_{\mathbf{List} A}(s; l.P; b_\varepsilon, a.l.h.b_\bullet)$, which performs recursion on the scrutinee s , returning a value in $P[s]$, using the two branches b_ε and b_\bullet corresponding to the two constructors $::$ and ε . b_\bullet binds three variables corresponding to the head a , tail l and induction hypothesis h on the tail. More generally recursive datatypes are often encoded in MLTT via $\mathbf{W} x : A. B$, the type of well-founded trees with nodes labelled by $a : A$ of arity $B a$. Finally, Martin-Löf identity types $\mathbf{Id} A x y$ represents equalities between two elements $x, y : A$ and is introduced with the reflexivity proof $\text{refl}_{A,a} : \mathbf{Id} A a a$. A general inductive type scheme is outside the scope of this paper, and we choose to present these type formers because, in combination with dependent sums, and the empty $\mathbf{0}$, unit $\mathbf{1}$ and boolean \mathbf{B} types, they are enough to emulate all indexed inductive types [Abbott et al. 2005; Altenkirch,

$$\begin{array}{c}
\boxed{\Gamma \vdash t \cong u : A} \quad \text{Terms } u \text{ and } v \text{ are convertible at type } A \text{ under context } \Gamma \\
\boxed{\Gamma \vdash A \cong B} \quad \text{Types } A \text{ and } B \text{ are convertible under context } \Gamma \\
\beta\text{FUN} \frac{\Gamma \vdash A \quad \Gamma, x : A \vdash t : B \quad \Gamma \vdash u : A}{\Gamma \vdash (\lambda x : A. t) u \cong t[u] : B[u]} \quad \eta\text{FUN} \frac{\Gamma, x : A \vdash f x \cong g x : B}{\Gamma \vdash f \cong g : \Pi x : A. B} \\
\beta\text{NIL} \frac{\Gamma \vdash A \quad \Gamma, l : \mathbf{List} A \vdash P}{\Gamma \vdash b_\varepsilon : P[\varepsilon_A]} \quad \Gamma, a : A, l : \mathbf{List} A, h : P[l] \vdash b_\varepsilon : P[a ::_A l]}{\Gamma \vdash \mathbf{ind}_{\mathbf{List} A}(\varepsilon_A; l.P; b_\varepsilon, a.l.h.b_\varepsilon) \cong b_\varepsilon : P[\varepsilon_A]} \\
\beta\text{CONS} \frac{\Gamma \vdash A \quad \Gamma \vdash a : A \quad \Gamma \vdash l : \mathbf{List} A}{\Gamma, l : \mathbf{List} A \vdash P} \quad \Gamma \vdash b_\varepsilon : P[\varepsilon_A] \quad \Gamma, a : A, l : \mathbf{List} A, h : P[y] \vdash b_\varepsilon : P[a ::_A l]}{\Gamma \vdash \mathbf{ind}_{\mathbf{List} A}(a ::_A l; l.P; b_\varepsilon, a.l.h.b_\varepsilon) \cong b_\varepsilon[\mathbf{id}, a, l, \mathbf{ind}_{\mathbf{List} A}(l; l.P; b_\varepsilon, a.l.h.b_\varepsilon)] : P[a ::_A l]} \\
\text{CONVCONV} \frac{\Gamma \vdash t \cong t' : A \quad \Gamma \vdash A \cong A'}{\Gamma \vdash t \cong t' : A'} \quad \text{ELCONV} \frac{\Gamma \vdash A \cong A' : \mathbf{Type}_i}{\Gamma \vdash A \cong A'} \\
\text{REFL} \frac{\Gamma \vdash t : A}{\Gamma \vdash t \cong t : A} \quad \text{SYM} \frac{\Gamma \vdash t \cong u : A}{\Gamma \vdash u \cong t : A} \quad \text{TRANS} \frac{\Gamma \vdash t \cong u : A \quad \Gamma \vdash u \cong v : A}{\Gamma \vdash t \cong v : A}
\end{array}$$

Fig. 3. Declarative conversion for MLTT (complete rules: Appendix B.1)

[Ghani, et al. 2015; Hugunin 2020]. We will see however in Section 3.1 that this standard encoding does not necessarily yield definitionally functorial maps on the encoded datatype. As a result, we consider positive sum types $A + B$ instead of plain booleans, introduced by the alternative $\mathbf{inj}^l a$ for $a : A$ or $\mathbf{inj}^r b$ for $b : B$ and eliminated by pattern matching. As $\mathbf{0}$ and $\mathbf{1}$ are not parametrized, their presentation in our setting is entirely standard. The specific types we present cover all the difficulties of general inductive types: recursion, branching, parameters, and indices.

Rules in the paper and in the appendix. To avoid cluttering the paper with too many rules, we focus in the text on the most interesting ones, and on two types: dependent functions and lists. Together, they cover the technically interesting points of our development: dependent product types have a binder and come with an η -law; lists are a parametrized datatype, for which definitional functor laws are challenging. Complete rules for reference are given in Appendix B.

2.1 Metatheoretical properties

In order to show that the extension of MLTT from Figure 1 are well-behaved and suitable for implementations, we seek to establish the following meta-theoretical properties.

Consistency and canonicity. In order to be employed as a logic, a type theory should not allow to derive every statement. This is equivalent to showing that there is no closed term of the empty type, *i.e.* that $\vdash t : \mathbf{0}$ is not derivable for any t . This *consistency* property is satisfied by all our type theories and an easy consequence of the stronger *canonicity* results, which characterizes the inhabitants of inductive types in the empty context up to conversion, as those obtained by repeated applications of constructors. Consistency follows from canonicity, because $\mathbf{0}$ has no constructors.

$$\boxed{t \rightsquigarrow^1 t'} \quad \text{Term } t \text{ weak-head reduces in one step to term } t'$$

$$\beta_{\text{RED}} \frac{}{(\lambda x: A.t) u \rightsquigarrow^1 t[u]} \quad \beta_{\text{REDNIL}} \frac{}{\text{ind}_{\text{List } A}(\varepsilon_A; x.P; b_\varepsilon, x.y.z.b_\varepsilon) \rightsquigarrow^1 b_\varepsilon}$$

$$\beta_{\text{REDCONS}} \frac{}{\text{ind}_{\text{List } A}(a ::_A l; x.P; b_\varepsilon, x.y.z.b_\varepsilon) \rightsquigarrow^1 b_\varepsilon[\text{id}, a, l, \text{ind}_{\text{List } A}(l; z.P; b_\varepsilon, x.y.z.b_\varepsilon)]}$$

$$\text{REDAPP} \frac{t \rightsquigarrow^1 t'}{t u \rightsquigarrow^1 t' u} \quad \text{REDIND} \frac{t \rightsquigarrow^1 t'}{\text{ind}_{\text{List } A}(t; z.P; b_\varepsilon, x.y.z.b_\varepsilon) \rightsquigarrow^1 \text{ind}_{\text{List } A}(t'; z.P; b_\varepsilon, x.y.z.b_\varepsilon)}$$

$$\boxed{t \rightsquigarrow^* t'} \quad \text{Term } t \text{ weak-head reduces in multiple steps to term } t'$$

$$\text{REDBASE} \frac{}{t \rightsquigarrow^* t} \quad \text{REDSTEP} \frac{t \rightsquigarrow^1 t' \quad t' \rightsquigarrow^* t''}{t \rightsquigarrow^* t''}$$

$$\boxed{\text{nf } f} \stackrel{\text{def}}{=} n \mid \Pi x: t_1.t_2 \mid \text{Type}_i \mid \mathbf{List} \ t \mid \lambda x: A.t \mid \varepsilon_A \mid t_1 ::_A t_2 \quad \text{weak-head normal forms}$$

$$\boxed{\text{ne } n} \stackrel{\text{def}}{=} x \mid n \ t \mid \text{ind}_{\text{List } A}(n; t, t) \quad \text{weak-head neutrals}$$

 Fig. 4. Weak-head reduction and normal forms (t stands for an arbitrary term) (complete rules: Appendix B.2)

Decidability of type-checking and conversion. A consistent proof assistant must also be able to check whether a proof is valid or not, *i.e.* whether a typing judgement is derivable in the type theory. In a dependent type system where terms encode the essential structure of derivations, the main obstacle to decidability lies in the conversion of types and terms.

Normal forms for terms and derivations. In order to establish both consistency and decidability of type-checking, we exhibit a function computing normal forms of term. Inspecting the possible normal forms in the empty context entails canonicity and thus consistency. Moreover, on these normal forms conversion is easily decidable, and so we can build on normalization to decide conversion. Finally, we can go further, and use normalization to build canonical representatives of typing and conversion derivations, which we rely on to relate our different systems.

Injectivity of type constructors. A more technical, but very important property is injectivity of type constructors, for instance the fact that whenever $\Pi x: A.B \cong \Pi x: A'.B'$, then $A \cong A'$ and $B \cong B'$. This property fails in ETT, where the equational theory is too rich. For dependent type theories like ours, injectivity of type constructors is the main obstruction to subject reduction, *i.e.* that reduction is type-preserving, and included in conversion on typed instances.

2.2 Neutrals, normals, and reduction

Before getting to the techniques we use to establish these properties, we must introduce a last element missing in Figure 3: computation. Indeed, most of the rules in that figure can be oriented, and thus seen not just as equalities but as computations to be performed. This leads to the definition of weak-head reduction \rightsquigarrow^* in Figure 4, an evaluation strategy for open terms which reduces just as much as needed in order to uncover the head constructor of a term. This means reducing not just at top level, as rule **REDAPP** shows: if our term is an application, we might need to reduce the function in order to expose a λ -abstraction and subsequently β -reduce the term with the (call-by-name) rule **β_{RED}** . However, we do *not* allow reduction in the argument of an application, so

that reduction remains deterministic: there is at most one possible reduct for any term. Weak-head reduction is the only reduction that will be used throughout this article.

The normal forms (nf) for weak-head reduction, *i.e.* the terms that cannot reduce, are inductively characterized at the bottom of Figure 4, together with the companion notion of neutral forms (ne). Normal forms can be either a canonical term, starting with a head constructor (for instance, a λ -abstraction or ε), or a neutral term. Neutral terms are stuck computations, blocked by some head variable, *e.g.* xu cannot reduce further, and will be unstuck once x is substituted by a λ -abstraction.

2.3 Proof methods

We can now go through the techniques we use to establish the properties of Section 2.1.

Logical relations. Logical relations are our main tool to obtain normalization and canonicity results. We follow the approach of Abel et al. [2017], who formalize a logical relation for MLTT in AGDA. At a high-level, the logical relation is based on reducibility, a complex predicate on types and terms, which in particular implies having a weak-head normal form. Combining this property with the fundamental lemma stating that every well-typed term is reducible, *i.e.* that the logical relation is a model of MLTT, we obtain weak-head normalization.

PROPERTY 2.1 (WEAK-HEAD NORMALIZATION). *If $\Gamma \vdash t : T$, then there exists a normal form t' (*i.e.* a term t' such that $\text{nf } t'$), such that $t \rightsquigarrow^* t'$.*

Strong normal forms can be obtained through inspection of reducibility derivations for a term, that contain iterated reduction steps to a normal form. These derivations are obtained for any well-typed term by using the fundamental lemma of the logical relation.

We use the logical relation not only to characterize the normal forms of terms but also the conversion between them, showing that a proof of convertibility between two terms can be transformed to a canonical shape interleaving weak-head reduction sequences and congruence steps between weak-head normal forms. We detail in Section 4 the novel challenges we encountered when adapting the approach of Abel et al. [2017] to parameterized inductive types.

Bidirectional typing and algorithmic conversion. Our second tool is a presentation of conversion and typing that, while still inductively defined, is as close as possible to an actual implementation. Typing is bidirectional [Pierce and Turner 2000; Lennon-Bertrand 2021], *i.e.* the declarative typing predicate of Figure 2 is decomposed into type inference and type checking shown in Figure 5.³ We use bidirectional typing for its rigid, canonical derivation structure, rather than for its ability to cut down type annotations on terms. As a result, although we use bidirectional judgements, all of our terms infer a type, in contrast to what is common in the bidirectional literature [Dunfield and Krishnaswami 2021; McBride 2022], where some terms can only be checked.

The presentation of algorithmic conversion in Figure 6 combines ideas from both bidirectional typing and the presentation of Abel et al. [2017]. Crucially, it gets rid entirely of the generic transitivity rule TRANS, and instead uses term-directed reduction, intertwined with comparison of the heads of weak-head normal forms. Algorithmic conversion is mutually defined with a second relation, dedicated to comparing weak-head neutral forms, which is called when encountering neutrals at positive types. We think of general conversion as “checking”, *i.e.* as taking a type as input, while neutral comparison is “inferring”, *i.e.* the type is an output.

Using the consequences of the logical relation, we can show that this algorithmic presentation has many desirable properties. For instance, transitivity is admissible, even though there is no

³Following Lennon-Bertrand [2021], to avoid clashing with Coq’s \Rightarrow in the formalization, we pick \triangleright as the symbol for inference, and \triangleleft as the one for checking, instead of the slightly more standard \Rightarrow and \Leftarrow .

$\boxed{\Gamma \vdash t \triangleright T}$ Term t infers type T in context Γ

$$\begin{array}{c}
 \text{SORT} \frac{}{\Gamma \vdash \text{Type}_i \triangleright \text{Type}_{i+1}} \qquad \text{VAR} \frac{(x:T) \in \Gamma}{\Gamma \vdash x \triangleright T} \\
 \\
 \text{PROD} \frac{\Gamma \vdash A \triangleright_h \text{Type}_i \quad \Gamma, x: A \vdash B \triangleleft \text{Type}_i}{\Gamma \vdash \Pi x: A. B \triangleright \text{Type}_i} \qquad \text{LIST} \frac{\Gamma \vdash A \triangleright_h \text{Type}_i}{\Gamma \vdash \mathbf{List} A \triangleright \text{Type}_i} \\
 \\
 \text{ABS} \frac{\Gamma \vdash A \triangleright_h \text{Type}_i \quad \Gamma, x: A \vdash t \triangleright B}{\Gamma \vdash \lambda x: A. t \triangleright \Pi x: A. B} \qquad \text{APP} \frac{\Gamma \vdash t \triangleright_h \Pi x: A. B \quad \Gamma \vdash u \triangleleft A}{\Gamma \vdash t u \triangleright B[u]} \\
 \\
 \text{NIL} \frac{\Gamma \vdash A \triangleright_h \text{Type}_i}{\Gamma \vdash \varepsilon_A \triangleright \mathbf{List} A} \qquad \text{CONS} \frac{\Gamma \vdash A \triangleright_h \text{Type}_i \quad \Gamma \vdash a \triangleleft A \quad \Gamma \vdash l \triangleleft \mathbf{List} A}{\Gamma \vdash a ::_A l \triangleright \mathbf{List} A} \\
 \\
 \text{LISTIND} \frac{\Gamma \vdash A \triangleright_h \text{Type}_i \quad \Gamma \vdash s \triangleleft \mathbf{List} A \quad \Gamma, x: \mathbf{List} A \vdash P \triangleright_h \text{Type}_j \quad \Gamma \vdash b_\varepsilon \triangleleft P[\varepsilon_A] \quad \Gamma, x: A, y: \mathbf{List} A, z: P[y] \vdash b_\varepsilon \triangleleft P[x ::_A y]}{\Gamma \vdash \text{ind}_{\mathbf{List} A}(s; z. P; b_\varepsilon, x.y.z.b_\varepsilon) \triangleright P[s]}
 \end{array}$$

$\boxed{\Gamma \vdash t \triangleleft T}$ Term t checks against type T

$\boxed{\Gamma \vdash t \triangleright_h T}$ Term t infers the reduced type T

$$\begin{array}{c}
 \text{CHECK} \frac{\Gamma \vdash t \triangleright T' \quad \Gamma \vdash T' \cong T \triangleleft}{\Gamma \vdash t \triangleleft T} \qquad \text{INFRED} \frac{\Gamma \vdash t \triangleright T' \quad T' \rightsquigarrow^* T}{\Gamma \vdash t \triangleright_h T}
 \end{array}$$

Fig. 5. Typing rules for algorithmic/bidirectional typing (complete rules: Appendix B.2)

dedicated rule. Collecting the properties derived from the logical relation, we can obtain our second main property: equivalence between the algorithmic and declarative presentations.

PROPERTY 2.2 (EQUIVALENCE OF THE PRESENTATIONS). *If $\Gamma \vdash t : T$, then $\Gamma \vdash t \triangleleft T$. Conversely, if $\vdash \Gamma$, $\Gamma \vdash T$ and $\Gamma \vdash t \triangleleft T$, then $\Gamma \vdash t : T$.*

Note that the implication from the bidirectional judgement to the declarative one is not absolute, it only holds if the context and type are well-formed. In general, our algorithmic presentations are “garbage-in, garbage-out”: they maintain well-formation of types and contexts, but do not enforce them. Thus, most properties of the algorithmic derivations only hold if their inputs are well-formed, in the sense of Figure 7. Note that in checking and inference modes, while the term is certainly an input of the judgement, it is of course not assumed to be well-formed in advance, since this is what the judgement itself asserts. This algorithmic presentation, being syntax directed, is well suited to design implementations and establish relationships between MLTT_{map} , MLTT_{coe} and MLTT_{sub} .

3 A FUNCTORIAL TYPE THEORY

We develop an extension MLTT_{map} of MLTT with primitive map_F operations for each parametrized type former F of MLTT, that is Π , Σ , $+$, \mathbf{List} , \mathbf{W} , and \mathbf{Id} . These map operations internalize the functorial character of the type formers,⁴ and by design *definitionally* satisfy the generic functor

⁴These equations are all propositionally true in MLTT, proven by induction for datatypes.

$\Gamma \vdash t \approx t' \triangleright T$	Neutrals t and t' are comparable, inferring the type T	
$\text{NVAR} \frac{(x:T) \in \Gamma}{\Gamma \vdash x \approx x \triangleright T}$	$\text{NAPP} \frac{\Gamma \vdash n \approx_h n' \triangleright \Pi x: A.B \quad \Gamma \vdash u \cong u' \triangleleft A}{\Gamma \vdash n u \approx n' u' \triangleright B[u]}$	
$\text{NLISTIND} \frac{\Gamma \vdash A \cong A' \triangleleft \quad \Gamma \vdash s \approx s' \triangleright S \quad \Gamma, l: \mathbf{List} A \vdash P \cong P' \triangleleft \quad \Gamma \vdash b_\varepsilon \cong b'_\varepsilon \triangleleft P[\varepsilon_A] \quad \Gamma, a: A, l: \mathbf{List} A, h: P[y] \vdash b_\varepsilon \cong b'_\varepsilon \triangleleft P[a ::_A l]}{\Gamma \vdash \text{ind}_{\mathbf{List} A}(s; l.P; b_\varepsilon, a.l.h.b_\varepsilon) \approx \text{ind}_{\mathbf{List} A'}(s'; l.P'; b'_\varepsilon, a.l.h.b'_\varepsilon) \triangleright P[s]}$		
$\Gamma \vdash T \cong_h T' \triangleleft$	Reduced types T and T' are convertible	
$\Gamma \vdash t \cong_h t' \triangleleft A$	Reduced terms t and t' are convertible at type A	
$\text{CUNI} \frac{}{\Gamma \vdash \text{Type}_i \cong_h \text{Type}_j \triangleleft \text{Type}_k}$	$\text{CLIST} \frac{\Gamma \vdash A \cong A' \triangleleft \text{Type}_i}{\Gamma \vdash \mathbf{List} A \cong_h \mathbf{List} A' \triangleleft \text{Type}_i}$	
$\text{CPROD} \frac{\Gamma \vdash A \cong A' \triangleleft \text{Type}_i \quad \Gamma, x: A' \vdash B \cong B' \triangleleft \text{Type}_i}{\Gamma \vdash \Pi x: A.B \cong_h \Pi x: A'.B' \triangleleft \text{Type}_i}$	$\text{CFUN} \frac{\Gamma, x: A \vdash f x \cong f' x \triangleleft B}{\Gamma \vdash f \cong_h f' \triangleleft \Pi x: A.B}$	
$\text{CNIL} \frac{}{\Gamma \vdash \varepsilon_A \cong_h \varepsilon_{A'} \triangleleft \mathbf{List} A''}$	$\text{CCONS} \frac{\Gamma \vdash a \cong a' \triangleleft A'' \quad \Gamma \vdash l \cong l' \triangleleft \mathbf{List} A''}{\Gamma \vdash a ::_A l \cong_h a' ::_{A'} l' \triangleleft \mathbf{List} A''}$	
$\text{NEUUNI} \frac{\Gamma \vdash n \approx n' \triangleright T}{\Gamma \vdash n \cong_h n' \triangleleft \text{Type}_i}$	$\text{NEULIST} \frac{\Gamma \vdash n \approx_h n' \triangleright S}{\Gamma \vdash n \cong_h n' \triangleleft \mathbf{List} A}$	$\text{NEUNEU} \frac{\text{ne } M \quad \Gamma \vdash n \approx n' \triangleright N}{\Gamma \vdash n \cong_h n' \triangleleft M}$
$\Gamma \vdash T \cong T' \triangleleft$	Types T and T' are convertible	
$\Gamma \vdash t \cong t' \triangleleft T$	Terms t and t' are convertible at type T	
$\Gamma \vdash t \approx_h t' \triangleright T$	Neutrals t and t' are comparable, inferring the reduced type T	
$\text{TMRED} \frac{t \rightsquigarrow^* u \quad t' \rightsquigarrow^* u' \quad T \rightsquigarrow^* U \quad \Gamma \vdash u \cong_h u' \triangleleft U}{\Gamma \vdash t \cong t' \triangleleft T}$	$\text{NRED} \frac{\Gamma \vdash n \approx n' \triangleright T \quad T \rightsquigarrow^* S}{\Gamma \vdash n \approx_h n' \triangleright S}$	

Fig. 6. Algorithmic conversion and comparison of neutral terms (complete rules: Appendix B.2)

Judgement	Input(s)	Inputs are well-formed
$\Gamma \vdash t \triangleright T$	Γ, t	$\vdash \Gamma$
$\Gamma \vdash t \triangleleft T$	Γ, T, t	$\vdash \Gamma$ and $\Gamma \vdash T$
$\Gamma \vdash T \cong T' \triangleleft$	Γ, T and T'	$\vdash \Gamma, \Gamma \vdash T$ and $\Gamma \vdash T'$
$\Gamma \vdash t \cong t' \triangleleft T$	Γ, t, t' and T	$\vdash \Gamma, \Gamma \vdash T, \Gamma \vdash t: T$ and $\Gamma \vdash t': T$
$\Gamma \vdash t \approx t' \triangleright T$	Γ, t and t'	$\vdash \Gamma, \text{ne } t, \text{ne } t',$ and $\exists A, A' \text{ s.t. } \Gamma \vdash t: A, \Gamma \vdash t': A'$

Fig. 7. Well-formed inputs (for $\cong_h, \approx_h, \triangleright_h$, similar to their non-reduced variants)

laws for each type former F :

$$\text{map}_F \text{id} \cong \text{id} \quad (\text{id-eq})$$

$$\text{map}_F f \circ \text{map}_F g \cong \text{map}_F (f \circ g) \quad (\text{comp-eq})$$

Section 3.1 describes the structure needed on type formers to state their functoriality in MLTT_{map} . In Section 3.2 we show how definitionally functorial map_F are definable in vanilla MLTT for type formers with an η -law. Section 3.3 introduces the main content of this paper, the extension of the equational theory on neutral terms required to enforce the functor laws on inductive type formers. We explain the technical design choices needed to define and use the logical relations for MLTT_{map} and obtain as a consequence that the theory enjoys consistency, canonicity, and decidable conversion and type-checking. We implement these design choices in Coq for a simplified but representative version of MLTT_{map} , with one universe and the Π , Σ , **List** and **N** type formers, with their respective map operators. This formalization is detailed in the following Section 4.

3.1 Functorial structure on type formers

In order to state the functor laws for a type former F , such as Π , Σ , **List**, **W**, **Id**, we must specify the categorical structures involved. A type former F is parametrized by a telescope of parameters that we collectively refer to as $\text{dom}(F)$, and produces a type. We will always equip the codomain Type of a type former F with the category structure of functions between types, with the standard identity and composition. Note that composition is associative and unital up to conversion, thanks to η laws on function types.

The domain $\text{dom}(F)$ of a type former must also be equipped with the structure of a category. We introduce the judgement $\Delta \vdash_{\text{map}} X : \text{dom}(F)$ to stand for a substitution in context Δ of the telescope of parameters of F , and given two such instances X_1 and X_2 , the judgement $\Delta \vdash_{\text{map}} \varphi : \text{hom}_F(X_1, X_2)$ classifies morphisms between X_1 and X_2 . We require F to be also equipped with identities and a definitionally associative and unital composition:

$$\frac{\Delta \vdash_{\text{map}} X : \text{dom}(F)}{\Delta \vdash_{\text{map}} \text{id}_X^F : \text{hom}_F(X, X)} \quad \frac{\Delta \vdash_{\text{map}} \varphi : \text{hom}_F(X, Y) \quad \Delta \vdash_{\text{map}} \psi : \text{hom}_F(Y, Z)}{\Delta \vdash_{\text{map}} \psi \circ^F \varphi : \text{hom}_F(X, Z)}$$

For instance, in the case of dependent products, we take $\text{dom}(\Pi)$ and $\text{hom}(\Pi)$ to be

$$\begin{aligned} \Delta \vdash_{\text{map}} (A, B) : \text{dom}(\Pi) &\iff \Delta \vdash_{\text{map}} A \wedge \Delta, a : A \vdash_{\text{map}} B \\ \Delta \vdash_{\text{map}} (f, g) : \text{hom}_{\Pi}((A_1, B_1), (A_2, B_2)) &\iff \Delta \vdash_{\text{map}} f : A_2 \rightarrow A_1 \wedge \\ &\quad \Delta, a : A_2 \vdash_{\text{map}} g : B_1[f a] \rightarrow B_2 \end{aligned}$$

with identity $\text{id}_{(A,B)}^{\Pi} \stackrel{\text{def}}{=} (\text{id}_A, \text{id}_B)$ and composition $(f, g) \circ^{\Pi} (f', g') \stackrel{\text{def}}{=} (f' \circ f, g \circ g')$.

The domain and morphism for each type former are described in Figure 8. Identities and compositions are given by the categorical structure on Type for **List** and **Id**, and are defined componentwise, for Σ and **W**, similarly to Π . Figure 9 presents the conversion rules of MLTT_{map} , extending those of MLTT with general functoriality rules and specific rules for each type former. For each type former F , map_F is introduced using **MAP** and witnesses the functorial nature of F , that is F maps morphisms φ in its domain between two instances of its parameters X, Y (left implicit) to functions between types

$$\Delta \vdash_{\text{map}} \varphi : \text{hom}_F(X, Y) \implies \Delta \vdash_{\text{map}} \text{map}_F \varphi : F X \rightarrow F Y$$

These mapping operations obey the two functor laws, as stated by **MAPID** and **MAPCOMP**.

The computational behaviour of maps, as defined by weak-head reduction, depends on the type former. On Π and Σ , map is defined by its observation, namely application for Π and first and second projections for Σ . On inductive types such as **List**, **W**, **Id** and $+$, map traverses constructors,

Type former F	Domain $\Delta \vdash_{\text{map}} X : \text{dom}(F)$	Morphisms $\Delta \vdash_{\text{map}} \varphi : \text{hom}_F(\cdot_1, \cdot_2)$
List	$\Delta \vdash_{\text{map}} A$	$\Delta \vdash_{\text{map}} f : A_1 \rightarrow A_2$
Π	$\Delta \vdash_{\text{map}} A \wedge$ $\Delta, a : A \vdash_{\text{map}} B$	$\Delta \vdash_{\text{map}} f : A_2 \rightarrow A_1 \wedge$ $\Delta, a : A_2 \vdash_{\text{map}} g : B_1[f a] \rightarrow B_2$
Σ	idem	$\Delta \vdash_{\text{map}} f : A_1 \rightarrow A_2 \wedge$ $\Delta, a : A_1 \vdash_{\text{map}} g : B_1 \rightarrow B_2[f a]$
W	idem	$\Delta \vdash_{\text{map}} f : A_1 \rightarrow A_2 \wedge$ $\Delta, a : A_1 \vdash_{\text{map}} g : B_2[f a] \rightarrow B_1$
Id	$\Delta \vdash_{\text{map}} A \wedge$ $\Delta \vdash_{\text{map}} x : A \wedge$ $\Delta \vdash_{\text{map}} y : A \wedge$	$\Delta \vdash_{\text{map}} f : A_1 \rightarrow A_2 \wedge$ $\Delta \vdash_{\text{map}} f x_1 \cong x_2 : A_2 \wedge$ $\Delta \vdash_{\text{map}} f y_1 \cong y_2 : A_2$
$+$	$\Delta \vdash_{\text{map}} A \wedge \Delta \vdash_{\text{map}} B$	$\Delta \vdash_{\text{map}} f : A_1 \rightarrow A_2 \wedge \Delta \vdash_{\text{map}} g : B_1 \rightarrow B_2$

Fig. 8. Domain and categorical structure on type formers

applying the provided morphism on elements of the parameter type(s), and itself to recursive arguments. This corresponds to the usual notion of map on lists. On **W**-types, the map operation relabels the nodes of the trees using its first component, and reorganizes the subtrees according to its second component. On identity types, the reflexivity proof $\text{refl}_{A,a}$ at a point $a : A$ is mapped to the reflexivity proof at $f a : A'$ for $f : A \rightarrow A'$. On sum types $A + B$, either the first or second component of the morphism (f, g) is employed depending on the constructor inj^l or inj^r . Each reduction rule has a corresponding conversion rule that can be found in Appendix B.3.

Functorial maps and type former encodings. Positive sum types $A + B$ can be simulated in MLTT by the type $\Sigma b : \mathbf{B} . \delta(b, A, B)$, where $\delta(b, A, B) := \text{ind}_{\mathbf{B}}(b; z. \text{Type}_i; A, B)$, which admits the adequate introduction and elimination rules. The encoding induces a mapping from $\text{dom}(+)$ to $\text{dom}(\Sigma)$, that maps a morphism $\Delta \vdash_{\text{map}} (f, g) : \text{hom}_+(\mathbb{A}_1, \mathbb{B}_1), (\mathbb{A}_2, \mathbb{B}_2)$ to $\Delta \vdash_{\text{map}} (\text{id}_{\mathbf{B}}, f \oplus g) : \text{hom}_{\Sigma}((\mathbf{B}, \delta(b, A, B)), (\mathbf{B}, \delta(b, A', B')))$ where $f \oplus g$ is defined as

$$\Delta, b : \mathbf{B} \vdash_{\text{map}} \text{ind}_{\mathbf{B}}(b; z. \delta(z, A, B) \rightarrow \delta(z, A', B'); f, g) : \delta(b, A, B) \rightarrow \delta(b, A', B').$$

We can show by case analysis on \mathbf{B} that this mapping satisfies the propositional functor laws. However, it falls short from satisfying the definitional ones.⁵ It is thus not enough to compose map_{Σ} with this mapping to obtain a functorial action on sum types $A + B$, and explains why we add $+$ primitively.

This obstruction to inductive encodings would motivate a general definition of functorial map for a scheme of indexed inductive types. However, it seems already non-trivial to specify the categorical structure on the domain of an arbitrary inductive type, let alone generate the type and equations for the corresponding map operation. Thus, we rather concentrate on understanding the theory on quintessential examples, leaving out the question of a general treatment to future work.

3.2 Extensional types and map

A type A is extensional when its elements are characterized by their observation, *i.e.* any element is convertible to its η -expansion, an elimination followed by an introduction – an equation usually called η -law. For extensional type former, it is possible to define a map operation satisfying the

⁵This would amount to an instance of the η -law for \mathbf{B} .

For each type former F (Π , Σ , **List**, **W**, **Id**, $+$)

$$\begin{array}{c}
 \text{MAP} \frac{\Gamma \vdash_{\text{map}} X, Y : \text{dom}(F) \quad \Gamma \vdash_{\text{map}} f : \text{hom}_F(X, Y)}{\Gamma \vdash_{\text{map}} \text{map}_F f : F X \rightarrow F Y} \qquad \text{MAPID} \frac{\Gamma \vdash_{\text{map}} X : \text{dom}(F) \quad \Gamma \vdash_{\text{map}} t : F X}{\Gamma \vdash_{\text{map}} \text{map}_F \text{id}_X^F t \cong t : F X} \\
 \\
 \text{MAPCOMP} \frac{\Gamma \vdash_{\text{map}} g : \text{hom}_F(X, Y) \quad \Gamma \vdash_{\text{map}} f : \text{hom}_F(Y, Z) \quad \Gamma \vdash_{\text{map}} t : F X}{\Gamma \vdash_{\text{map}} \text{map}_F f (\text{map}_F g t) \cong \text{map}_F (f \circ^F g) t : F Z}
 \end{array}$$

Specific rules

$$\begin{array}{c}
 \text{map}_{\text{List}} f (hd :: tl) \rightsquigarrow^1 f hd :: \text{map}_{\text{List}} f tl \qquad \text{map}_{\text{List}} f \varepsilon \rightsquigarrow^1 \varepsilon \\
 \\
 \pi_1 (\text{map}_{\Sigma} f p) \rightsquigarrow^1 \pi_1 f (\pi_1 p) \qquad \pi_2 (\text{map}_{\Sigma} f p) \rightsquigarrow^1 \pi_2 f (\pi_2 p) \\
 \\
 \text{map}_{\Pi} f h t \rightsquigarrow^1 (\pi_2 f) (h ((\pi_1 f) t)) \qquad \text{map}_{\text{Id}} f \text{refl}_{A,a} \rightsquigarrow^1 \text{refl}_{B,fa} \\
 \\
 \text{map}_{\mathbf{W}} \{T\} \{T'\} f (\text{sup } a k) \rightsquigarrow^1 \text{sup}_{x.\pi_2 T'} (\pi_1 f a) (\lambda x : (\pi_2 T' (\pi_1 f a)). \text{map}_{\mathbf{W}} f (k (\pi_2 g x))) \\
 \\
 \text{map}_+ (f, g) (\text{inj}^l a) \rightsquigarrow^1 \text{inj}^l (f a) \qquad \text{map}_+ (f, g) (\text{inj}^r b) \rightsquigarrow^1 \text{inj}^r (g b) \\
 \\
 \text{REDMAPCOMP} \frac{\text{ne } n \quad F \in \{\mathbf{List}, \text{Id}, +, \mathbf{W}\}}{\text{map}_F f (\text{map}_F g n) \rightsquigarrow^1 \text{map}_F (f \circ g) n}
 \end{array}$$

 Fig. 9. MLTT_{map} (extends Figures 2 to 4, complete rules: Appendix B.3)

functor laws. In MLTT and MLTT_{map} , both (strong) dependent sums Σ and dependent products Π have such extensionality laws, and so their map operations are definable.

$$\begin{array}{c}
 \text{map}_{\Pi} ((g, f) : \text{hom}_{\Pi}((A, B), (A', B'))) (h : \Pi(x : A) B) \stackrel{\text{def}}{=} \lambda x : A'. f (h (g x)) \\
 \text{map}_{\Sigma} ((g, f) : \text{hom}_{\Sigma}((A, B), (A', B'))) (p : \Sigma(x : A) B) \stackrel{\text{def}}{=} (g (\pi_1 p), f (\pi_2 p))
 \end{array}$$

LEMMA 3.1. map_{Π} and map_{Σ} satisfy the definitional functor laws *MAPID* and *MAPCOMP*.

Appendix C.1 gives a direct proof and the accompanying artifact also shows that the functor laws hold for CoQ 's Π and Σ types.⁶ The specific rule of Figure 9 hold by β -reduction.

3.3 New equations for neutral terms in dependent type theory

Inductive types in MLTT do not satisfy a definitional η -law. For identity types, the η -law is equivalent to the equality reflection principle of extensional MLTT [Jacobs 2001], whose equational theory is undecidable [Castellan et al. 2017]. Extensionality principles for inductive types with recursive occurrences as **List** or **W** are also likely to break the decidability of the equational theory, by adapting an argument for streams [McBride 2009]. The result of the previous section hence does not apply, and it is instructive to look at the actual obstruction. Consider the case of **List**, and the

⁶In file `mapPiSigmaFunctorLaws`.

$\text{nf } f$	$\stackrel{\text{def}}{=} \dots \mid c$	weak-head normal forms
$\text{ne } n$	$\stackrel{\text{def}}{=} \dots \mid \text{ind}_{\mathbf{List } A}(c; t; t)$	weak-head neutrals
$\text{cne } c$	$\stackrel{\text{def}}{=} n \mid \text{map}_{\mathbf{List}} f n$	compacted neutrals

Fig. 10. Weak-head normal and neutrals for MLTT_{map} (extends Figure 4)

equation for preservation of identities:

$$\Gamma \vdash_{\text{map}} \text{map}_{\mathbf{List}} \text{id}_A l \cong l : \mathbf{List } A. \quad (\star)$$

If we were to define $\text{map}_{\mathbf{List}}$ by induction on lists as is standard, we would get

$$\text{map}_{\mathbf{List}}(f: A \rightarrow B)(l: \mathbf{List } A) \stackrel{\text{def}}{=} \text{ind}_{\mathbf{List}}(\mathbf{List } B; l; \varepsilon_B, \text{hd}.tl.ih_{tl}.(f \text{hd}) ::_B ih_{tl})$$

We can observe that Eq. (\star) is validated on closed canonical terms of type \mathbf{List} :

$$\text{map}_{\mathbf{List}} \text{id}_A \varepsilon_A \cong \varepsilon_A \quad \text{map}_{\mathbf{List}} \text{id}_A (\text{hd} ::_A tl) \cong (\text{id}_A \text{hd}) ::_A \text{map}_{\mathbf{List}} \text{id}_A tl \stackrel{\text{ind. hyp.}}{\cong} \text{hd} ::_A tl$$

However, on neutral terms, typically variables, we are stuck as long as we stay within the equational theory of MLTT:

$$A: \text{Type}, x: \mathbf{List } A \not\vdash \text{map}_{\mathbf{List}} \text{id}_A x \cong x : \mathbf{List } A$$

In order to validate Eq. (\star) , MLTT_{map} must thus at the very least extend the equational theory on neutral terms. Allais et al. [2013] show in the simply-typed case that these equations between neutral terms are actually the only obstruction to functor laws, and in the remainder of this section we discuss how to adapt MLTT to this idea.

Map composition and compacted neutrals. The first step in order to validate the functor laws is to get as close as possible to a canonical representation during reduction. In order to deal with composition of maps, we extend reduction with **REDMAPCOMP**, fusing consecutive stuck maps. In order to preserve the deterministic nature of weak-head reduction, map compaction should only apply when no other rule does. To achieve this, the type former F should not be extensional, because map_{Π} is already handled through the η -expansion of **CFUN**, and similarly for map_{Σ} . Moreover, the mapped term should be neither a canonical form where map already has a computational behaviour, nor a map itself that could fire the same rule. To control this, we split between neutrals – which cannot contain a map as their head –, and *compacted neutrals*, which can start with at most one map, as shown in Figure 10 alongside normal forms. Allais et al. [2013] also features a decomposition of normal forms into three different classes akin to this, although their normal forms for lists are more complex than ours as they validate more definitional equations than functor laws.

Maps on identities. For identities, using a similar reduction-based approach is difficult: turning the equation $\Gamma \vdash_{\text{map}} \text{map}_{\mathbf{List}} \text{id}_A l \cong l : \mathbf{List } A$ into a reduction raises issues similar to those encountered with η -laws. Orienting it as an expansion $l \rightsquigarrow^* \text{map}_{\mathbf{List}} \text{id}_A l$ requires knowledge of the type to ensure the expansion only applies to lists, and is potentially non-terminating. Accommodating type-directed reduction would require a deep reworking of our setting.

As a result, just like for η on functions in rule **CFUN**, we implement this rule as part of conversion, rather than as a reduction. We also incorporate it carefully in the notion of reducible conversion in the logical relation, where we do have access to enough properties of the type theories. Since the equation is always validated by canonical forms, we only need to enforce it on compacted neutrals. The logical relation for an inductive type I (\mathbf{List} , \mathbf{W} , \mathbf{Id} , $+$) thus specifies that a neutral n is reducibly convertible to a compacted neutral $\text{map}_I f m$, whenever the neutrals n and m are

convertible and f agree with the identity of $\text{dom}(I)$ on any neutral term. See [MAPNECONVREDL](#) in the next section for the exact rule.

Eliminators: fusion or no fusion? When considering the interaction between `map` and the eliminator ind_{List} , arises a design choice: should we also fuse them, *i.e.* implement the following reduction rule, which pushes the `map` from the scrutinee into the branches?

$$\text{ind}_I(\text{map}_{\text{List}} f n; l.P; b_\varepsilon, a.l.h.b_{\cdot}) \rightsquigarrow^1 \text{ind}_I(n; l.P[\text{map}_{\text{List}} f l]; b_\varepsilon, a.l.h.b_{\cdot}[\text{id}, f a, \text{map}_{\text{List}} f l, h])$$

From the point of view of functorial equations, this is not necessary. In [Figure 10](#), and the rest of this paper, we thus take the most conservative approach, and do not add this reduction rule.

However, from the point of view of a subsumptive bidirectional subtyping, this fusion is necessary if we wish to infer the parameters of the inductive types from the scrutinee (as in [Fus](#) below), rather than store them in the induction node (as in [NoFus](#)).

$$\text{Fus} \frac{\Gamma \vdash_{\text{sub}} s \triangleright_h \text{List } A \quad \Gamma, l: \text{List } A \vdash_{\text{sub}} P \triangleright_h \text{Type} \quad \dots}{\Gamma \vdash_{\text{sub}} \text{ind}_{\text{List}}(s; l.P; \dots) \triangleright P[s]} \quad \text{NoFus} \frac{\Gamma \vdash_{\text{sub}} A \triangleleft \quad \Gamma \vdash_{\text{sub}} s \triangleleft \text{List } A \quad \Gamma, l: \text{List } A \vdash_{\text{sub}} P \triangleright_h \text{Type} \quad \dots}{\Gamma \vdash_{\text{sub}} \text{ind}_{\text{List } A}(s; l.P; \dots) \triangleright P[s]}$$

Rule [Fus](#) is more appealing, as it removes an unnecessary conversion test between the type of s and that stored in the node. Yet, elaborating it to a coercive system requires this target to have the extra fusion law above. Intuitively, this is because rule [Fus](#) does not fix the parameter type at which the eliminator is typed, and so this parameter can change, which in a coercive system corresponds to pushing coercions into the branches, as in the fusion equation above.

Experimenting MLTT_{map} within proof assistants through rewrite rules. Even though we have not attempted a justification of the metatheory of MLTT_{map} with a presentation purely based on rewriting, it is still possible to use oriented version of the functor laws to experiment with this theory: `AGDA` experimentally supports rewrite rules [[Cockx 2020b](#)] while ongoing implementation work exists for `Coq` [[Gilbert, Leray, et al. 2023](#)]. As an illustration, we implemented [Example 1.1](#) in `Agda`.⁷ Concretely, we postulate a new constant map_F and add the following rules:

$$\begin{aligned} \text{map}_F B' C f (\text{map}_F A B g x) &\rightsquigarrow^1 \text{map}_F A C (\lambda z: A. f (g z)) x && \text{(comp-rew)} \\ \text{map}_F A A' (\lambda z: A''. z) x &\rightsquigarrow^1 x && \text{(id-rew)} \end{aligned}$$

together with the usual definition of map_F on the constructors of the type former F . We rely on typing information to enforce that redundant data coincide, for instance that A , A' and A'' are convertible in `id-rew`.

4 FORMALIZING NEW EQUATIONS FOR NEUTRAL LISTS

In this section we expose the main components of the accompanying `Coq` formalization, which covers normalization, equivalence of declarative and algorithmic typing, decidability of type-checking, and canonicity for a subset of MLTT_{map} with $\mathbf{0}$, \mathbf{N} , $\mathbf{\Pi}$, $\mathbf{\Sigma}$, \mathbf{List} and a single universe. The formalization extends a port to `Coq` [[Adjedj et al. 2023](#)] of a previous `AGDA` formalization [[Abel et al. 2017](#)], which has already been extended multiple times [[Gilbert, Cockx, et al. 2019](#); [Pujet and Tabareau 2022, 2023](#)]. We focus on the challenges to establish the functor laws on lists, and direct the reader either to the `Coq` code, or to [Abel et al.](#) and [Adjedj et al.](#) for other details. The formalization spans ~26k lines of code, approximately 9k of which are specific to our extension with lists and definitionally functorial maps and are new compared to [Adjedj et al.](#) Text in [blue](#) refer to files in the companion artifact.

⁷See file `map.agda` in the companion artifact.

4.1 A logical relation with functor laws on list

The Coq development defines both declarative and algorithmic presentations of MLTT_{map} and proves their equivalence through a logical relation parametrized by a generic typing interface⁸ instantiated by both presentations. Beyond generic variants of the typing and conversion judgement, the interface uses two extra judgements: $\Gamma \vdash_{\text{map}} t \rightsquigarrow^* t' : A$ stating that t reduces to t' and that they are both well typed at type A in context Γ ; and $\Gamma \vdash_{\text{map}} n \approx n' : A$ stating that n and n' are convertible neutral terms.

Definition of the logical relation. Because we are dealing with dependent types, the usual strategy of reducibility proofs to define reducibility of terms by induction on their types fails. Rather, we need to define reducibility of types and of terms mutually, the latter defined out of a witness of the former, and the former reusing the latter for the universe. Following [Abel et al. \[2017\]](#), we thus first define for each type former F what it means to be a type reducible as F , and then what it means to be a reducible term and reducibly convertible terms at such a type reducible as F . A type is then reducible if it is reducible as F for some type former F . As we extend the logical relation to handle **List** and map_{List} , we focus on a high level description of the reducibility of types as lists and the reducible convertibility of terms of type **List**, the most challenging elements in the definition.⁹ Two points required specific attention with respect to prior work. First, to handle the fact that constructors contain their parameters, we need to impose reducible conversions between these and the parameters coming from the type. Second, in order to validate composition of map on neutrals that may contain a map , we need to equip neutrals with additional reducibility data, rather than pure typing information.

A type X is reducible as a list in context Γ , written $\Gamma \Vdash_{\text{List}} X$, if it weak-head reduces to **List** A for some parameter type A reducible in any context Δ extending Γ via a weakening $\rho : \text{Wk}(\Delta, \Gamma)$. If $\mathfrak{R} : \Gamma \Vdash_{\text{List}} X$ is a witness that X is reducible as a list, then $\mathbb{P}(\mathfrak{R})$ is for the parameter type A of this witness, and $\mathbb{P}_{\perp}(\mathfrak{R}) : \Pi\{\rho : \text{Wk}(\Delta, \Gamma)\}. \Delta \Vdash \mathbb{P}(\mathfrak{R})[\rho]$ is its witness of reducibility.

Reducible convertibility of terms as lists $\Gamma \Vdash t \cong t' : A \mid \mathfrak{R}$ is defined in [Figure 11](#). Two terms t and t' are reducibly convertible as lists according to $\mathfrak{R} : \Gamma \Vdash_{\text{List}} X$ if they reduce to normal forms v, v' that are reducibly convertible as normal forms of type list $\Gamma \Vdash_{\text{nf}} v \cong v' : A \mid \mathfrak{R}$ (**LISTRED**). Straightforwardly, two canonical forms are convertible if they are both ε (**NILRED**) or both $- :: -$ (**CONSRED**) with reducibly convertible heads and tails.

For compacted neutral forms, we need to consider four cases according to whether each of the left or the right hand-side term is a map_{List} . **NERED** provides the easy case where both terms are actually neutral, with a single premise requiring that these are convertible as neutrals for the generic typing interface. **MAPMAPCONVRED** gives the congruence rule for stuck map_{List} , relating $\text{map}_{\text{List}} f n$ and $\text{map}_{\text{List}} f' n'$ when the mapped lists n and n' are convertible as neutrals and the bodies $f x$ and $f' x$ of the functions are reducibly convertible. Note that at this point of the logical relation, we do not know that the domain of the functions f and f' is reducible, only that their codomain is, as provided by $\mathbb{P}_{\perp}(\mathfrak{R})$. This constraint motivates both the η -expansion of the functions on the fly before comparing them, and the necessity of a Kripke-style quantification on larger contexts for the reducibility of the parameter type $\mathbb{P}_{\perp}(\mathfrak{R})$, together ensuring that the recursive reducible conversion happens at a reducible type, namely an adequate instance of $\mathbb{P}(\mathfrak{R})$. Finally, the symmetric rules **NEMAPCONVREDR** and **MAPNECONVREDL** deal with the comparison of a map_{List} against a neutral n , that can be morally thought as $\text{map}_{\text{List}} \text{id } n$, and indeed the

⁸Defined in [GenericTyping](#)

⁹Available in file [LogicalRelation](#).

$$\begin{array}{c}
 \text{LISTRED} \frac{\Gamma \vdash_{\text{map}} t \rightsquigarrow^* v : \mathbf{List} \mathbb{P}(\mathfrak{A}) \quad \Gamma \vdash_{\text{map}} t' \rightsquigarrow^* v' : \mathbf{List} \mathbb{P}(\mathfrak{A}) \quad \Gamma \Vdash_{\text{nf}} v \cong v' : X \mid \mathfrak{A}}{\Gamma \Vdash t \cong t' : X \mid \mathfrak{A}} \\
 \\
 \text{NERED} \frac{\Gamma \vdash_{\text{map}} n \approx n' : \mathbf{List} \mathbb{P}(\mathfrak{A})}{\Gamma \Vdash_{\text{nf}} n \cong n' : X \mid \mathfrak{A}} \qquad \text{NILRED} \frac{\Gamma \Vdash \mathbb{P}(\mathfrak{A}) \cong P \mid \mathbb{P}_{\perp}(\mathfrak{A}) \quad \Gamma \Vdash \mathbb{P}(\mathfrak{A}) \cong P' \mid \mathbb{P}_{\perp}(\mathfrak{A})}{\Gamma \Vdash_{\text{nf}} \varepsilon_P \cong \varepsilon_{P'} : X \mid \mathfrak{A}} \\
 \\
 \text{CONSERED} \frac{\Gamma \Vdash \mathbb{P}(\mathfrak{A}) \cong P \mid \mathbb{P}_{\perp}(\mathfrak{A}) \quad \Gamma \Vdash \mathbb{P}(\mathfrak{A}) \cong P' \mid \mathbb{P}_{\perp}(\mathfrak{A}) \quad \Gamma \Vdash hd \cong hd' : \mathbb{P}(\mathfrak{A}) \mid \mathbb{P}_{\perp}(\mathfrak{A}) \quad \Gamma \Vdash tl \cong tl' : X \mid \mathbb{P}_{\perp}(\mathfrak{A})}{\Gamma \Vdash_{\text{nf}} hd ::_P tl \cong hd' ::_{P'} tl' : X \mid \mathfrak{A}} \\
 \\
 \text{MAPNECONVREDL} \frac{\Gamma \vdash_{\text{map}} n \approx n' : \mathbf{List} \mathbb{P}(\mathfrak{A}) \quad \Gamma, x : \mathbb{P}(\mathfrak{A}) \Vdash f x \cong x : \mathbb{P}(\mathfrak{A}) \mid \mathbb{P}_{\perp}(\mathfrak{A})}{\Gamma \Vdash_{\text{nf}} \text{map}_{\mathbf{List}} f n \cong n' : X \mid \mathfrak{A}} \qquad \text{NEMAPCONVREDR} \dots \\
 \\
 \text{MAPMAPCONVRED} \frac{\Gamma \vdash_{\text{map}} n \approx n' : \mathbf{List} A \quad \Gamma, x : A \Vdash f x \cong f' x : \mathbb{P}(\mathfrak{A}) \mid \mathbb{P}_{\perp}(\mathfrak{A})}{\Gamma \Vdash_{\text{nf}} \text{map}_{\mathbf{List}} f n \cong \text{map}_{\mathbf{List}} f' n' : X \mid \mathfrak{A}}
 \end{array}$$

 Fig. 11. Reducible convertibility of lists (where \mathfrak{A} is a proof of $\Gamma \Vdash_{\mathbf{List}} X$)

premises correspond to what one would obtain with **MAPMAPCONVRED** in that case, up to an inlined β -reduction step.

Validity of the functor laws. All the expected properties extend to this new logical relation: reflexivity, symmetry, transitivity, irrelevance with respect to reducible conversion, stability by weakening and anti-reduction.¹⁰ These properties are essential in order to show that the logical relation validates the functor laws on any reducible term. The proof proceeds through an usual argument for logical relations: on canonical forms, the functor laws hold as observed already in Section 3.3; on compacted neutrals and neutral forms, we need to show that any compositions of $\text{map}_{\mathbf{List}}$ reduce to a single map of a function with a reducible body, which amounts to show that composing reducible functions produces reducible outputs on reducible inputs. This last step in the proof reflect our assumption that the categorical structure equipping domains of type formers, here $\text{dom}(\mathbf{List})$, should be definitionally associative and unital.

4.2 Deciding conversion and typechecking for **MLTT**_{map}

Equivalence between declarative and algorithmic typing. Instantiating the generic typing interface of the logical relation with declarative typing provides metatheoretic consequences of the existence of normal forms, among which normalization, injectivity of type constructors and subject reduction. Using those, we can show that algorithmic typing is sound directly by induction, and also that it fits the generic typing interface of the logical relation, which lets us derive that it is complete with respect to declarative typing.

This part of the proof is close to [Abel et al. \[2017\]](#) and [Adjedj et al. \[2023\]](#). The main change is that we adapt algorithmic conversion to reflect the addition of compacted neutrals in our definition of normal forms, by introducing a third mutually defined relation to compare these compacted

¹⁰Available in the directory [LogicalRelation](#).

neutrals. The main idea is summed up in rules **LISTNECONV** and **NEMAPLISTL** below: when comparing compacted neutrals, we use the new relation \approx_{map} , which simulates the behaviour of the logical relation from Figure 11 on compacted neutrals.

$$\text{LISTNECONV} \frac{\Gamma \vdash_{\text{map}} c \approx_{\text{map}} c' \triangleleft \mathbf{List} A}{\Gamma \vdash_{\text{map}} c \cong_h c' \triangleleft \mathbf{List} A} \quad \text{NEMAPLISTL} \frac{\Gamma \vdash_{\text{map}} n \approx_h n' \triangleright \mathbf{List} A \quad \Gamma, x: A \vdash_{\text{map}} f x \cong x \triangleleft B}{\Gamma \vdash_{\text{map}} \text{map}_{\mathbf{List}} f n \approx_{\text{map}} n' \triangleleft \mathbf{List} B}$$

Implementation of the decidability algorithms. Our final result is decidability of conversion and type checking.¹¹ The main difficulty compared to [Adjedj et al. \[2023\]](#) is the addition of a compaction phase to weak-head evaluation. Concretely, evaluation is implemented using a stack machine, on which elimination forms are pushed as they are encountered. When the machine hits a variable, in the original formalisation it means the whole term – the variable against the stack of eliminations, which is now stuck – is a neutral. However, this is not the case for us: we want to compute a compacted neutral. Thus, while in the original formalization the stack is simply traversed and accumulated on the variable, we implement compaction on the fly, by merging successive map operations from the stack as we unpile them. In turn, to reason about this operation we have to explicit the formerly implicit invariant that the stack is always “well-typed” – in a suitable sense we had to define.

5 SUBTYPING, COERCIVE AND SUBSUMPTIVE

The main application we develop for our definitional functor laws is structural subtyping. More precisely, we describe two extensions of MLTT. The first, MLTT_{sub} , has subsumptive subtyping: whenever $\vdash_{\text{sub}} t : A \preceq A'$, then also $\vdash_{\text{sub}} t : A'$, leaving subtyping implicit. The second, MLTT_{coe} , features coercive subtyping, witnessed by an operator $\text{coe}_{A,A'} t$ explicitly marking where subtyping is used and well-typed whenever $\vdash_{\text{coe}} t : A \preceq A'$. The computational behaviour of coe on type formers coincides with that of map in MLTT_{map} .

In Section 5.1, we give both an algorithmic and declarative presentation of MLTT_{coe} , but only an algorithmic presentation of MLTT_{sub} . The idea is that the declarative version of MLTT_{coe} serves as a specification for MLTT_{sub} , with the algorithmic MLTT_{coe} as an intermediate step to relate the two. In the context of a proof assistant or dependently typed programming language, MLTT_{sub} would be the flexible, user-facing system, and MLTT_{coe} its well-behaved foundation.

We explain in Section 5.2 how to adapt the metatheoretic work on MLTT_{map} to MLTT_{coe} , and in particular the logical relation. Section 5.3 relates MLTT_{coe} and MLTT_{sub} : there is a simple erasure $|\cdot|$ from the former to the latter which removes coercions, and we show it is type-preserving; conversely, we show that any well-typed MLTT_{sub} term can be elaborated to a well-typed MLTT_{coe} term. The extra definitional functor laws are essential at this stage, to ensure that all equalities valid in MLTT_{sub} still hold in MLTT_{coe} . Since we are in a dependently-typed system, if equations valid in MLTT_{sub} failed to hold in MLTT_{coe} , elaboration could not be type-preserving. Finally, Section 5.4 discuss the implications of this equivalence for coherence.

5.1 The type systems MLTT_{sub} and MLTT_{coe}

5.1.1 Algorithmic MLTT_{sub} . MLTT_{sub} replaces **CHECK** in MLTT from Figure 5 with the following rule, which uses subtyping \preceq instead of conversion:

$$\text{CHECKSUB} \frac{\Gamma \vdash_{\text{sub}} t \triangleright T' \quad \Gamma \vdash_{\text{sub}} T' \preceq T \triangleleft}{\Gamma \vdash_{\text{sub}} t \triangleleft T}$$

¹¹See file [Decidability](#) for the high-level function/theorem.

$$\boxed{\Gamma \vdash_{\text{sub}} T \lesssim_{\text{h}} T' \triangleleft} \quad \text{Reduced type } T \text{ is a subtype of reduced type } T'$$

$$\begin{array}{c}
 \text{UNISUB} \frac{}{\Gamma \vdash_{\text{sub}} \text{Type}_i \lesssim_{\text{h}} \text{Type}_i \triangleleft} \quad \text{PRODSUB} \frac{\Gamma \vdash_{\text{sub}} A' \lesssim A \triangleleft \quad \Gamma, x: A' \vdash_{\text{sub}} B \lesssim B' \triangleleft}{\Gamma \vdash_{\text{sub}} \Pi x: A.B \lesssim_{\text{h}} \Pi x: A'.B' \triangleleft} \\
 \text{LISTSUB} \frac{\Gamma \vdash_{\text{sub}} A \lesssim A' \triangleleft}{\Gamma \vdash_{\text{sub}} \mathbf{List} A \lesssim_{\text{h}} \mathbf{List} A' \triangleleft} \quad \text{NEUSUB} \frac{\Gamma \vdash_{\text{sub}} n \approx_{\text{h}} n' \triangleright T}{\Gamma \vdash_{\text{sub}} n \lesssim_{\text{h}} n' \triangleleft}
 \end{array}$$

Fig. 12. Algorithmic subtyping between reduced types (extends Figure 6, complete rules: Appendix B.7)

$$\begin{array}{c}
 \text{RECY} \frac{\mathcal{L} \in \mathcal{P}_f(\text{Lbl}) \quad \forall l \in \mathcal{L}. \quad \Gamma \vdash_{\text{sub}} A_l \triangleleft}{\Gamma \vdash_{\text{sub}} \{l: A_l\}_{l \in \mathcal{L}} \triangleleft} \quad \text{RECSUB} \frac{\mathcal{K} \subseteq \mathcal{L} \quad \forall k \in \mathcal{K}. \quad \Gamma \vdash_{\text{sub}} A_k \lesssim B_k \triangleleft}{\Gamma \vdash_{\text{sub}} \{l: A_l\}_{l \in \mathcal{L}} \lesssim_{\text{h}} \{k: A_k\}_{k \in \mathcal{K}} \triangleleft}
 \end{array}$$

Fig. 13. Records, typing and subtyping (extends Figures 5, 6 and 12, complete rules: Appendices B.5 and B.6)

Subtyping, defined in Figure 12, orients type-level conversion from Figure 6, taking into account co- and contravariance. It relies on neutral comparison and term-level conversion, both of which are *not* altered with respect to Figure 6: subtyping is a type-level concept only.

5.1.2 A type of records for a non-trivial instance of subtyping. While the rule of Figure 12 let us propagate subtyping structurally through type formers, for the resulting system to be any different from MLTT, we need some base non-trivial subtyping. Its exact choice is largely orthogonal to the focus of this paper on the structural aspect of subtyping, and indeed the development of this section is relatively independent of it. Still, for our subtyping not to be degenerate, we must fix something.

We thus choose a simple example, presented in Figure 13. We fix a countable set of labels Lbl , and for each finite set $\mathcal{L} \subseteq \text{Lbl}$ of it and \mathcal{L} -indexed family of types A_l we introduce a (non-dependent) record type $\{l: A_l\}_{l \in \mathcal{L}}$.¹² To each record type corresponds a record constructor $\{l := a_l\}_{l \in \mathcal{L}}$, as well as projections $t.l$. Subtyping between record types is defined as inclusion of the set of labels, and pairwise subtyping of types at the same label, *i.e.* both depth and width subtyping.

5.1.3 Algorithmic MLTT_{coe} . In contrast with MLTT_{sub} , the rule **CHECK** in MLTT_{coe} is crucially *not* altered. Instead, subtyping is *only* allowed when *explicitly* marked by *coe*, as follows:

$$\text{COE} \frac{\Gamma \vdash_{\text{coe}} A \triangleleft \quad \Gamma \vdash_{\text{coe}} A' \triangleleft \quad \Gamma \vdash_{\text{coe}} t \triangleleft A \quad \Gamma \vdash_{\text{coe}} A \lesssim A' \triangleleft}{\Gamma \vdash_{\text{coe}} \text{coe}_{A,A'} t \triangleright A'}$$

Inference rules for all other type and term formers are similar to those of Figures 5 and 13. Reduction must of course be extended to give an operational behaviour to *coe*, and is given in Figure 14, together with normal forms. Operationally, $\text{coe}_{A,A'} t$ reduces the types A and A' to head normal forms, then behaves like the relevant map, propagating *coe* recursively. Since $\text{coe}_{A,A'} t$ is well-typed only when A is a subtype of A' , the type formers of their head normal forms have to agree, ensuring that we can always rely on this behaviour to enact structural subtyping. Just like *map* on Σ -types (see Fig. 9), coercions between records are stuck, even on constructors, and only reduce once they are projected. Finally, again as for *map*, rule **COECO** lets us compact a succession of

¹²We choose to avoid dependency mainly for simplicity purposes, but see no difficulty to have dependent records instead.

$$\boxed{t \rightsquigarrow^1 t'}$$

$$\text{nf } f \left\{ \begin{array}{l} (\text{coe}_{\Pi x:A'.B', \Pi x:A.B} f) a \rightsquigarrow^1 \text{coe}_{B'[\text{coe}_{A,A'} a], B[a]}(f(\text{coe}_{A,A'} a)) \\ (\text{coe}_{\{l_i:A_i\}_{i \in [n]}, \{k_j:B_j\}_{j \in [m]}} f).l \rightsquigarrow^1 \text{coe}_{A_i, B_j} f.l \quad l = l_i = k_j \end{array} \right.$$

$$\text{coe}_{\text{Type}_i, \text{Type}_i} t \rightsquigarrow^1 t \qquad \text{coe}_{\text{List } A, \text{List } A'} \varepsilon \rightsquigarrow^1 \varepsilon$$

$$\text{coe}_{\text{List } A, \text{List } A'}(h :: t) \rightsquigarrow^1 \text{coe}_{A, A'} h :: \text{coe}_{\text{List } A, \text{List } A'} t \qquad \text{CoEL} \frac{A \rightsquigarrow^1 A'}{\text{coe}_{A, B} t \rightsquigarrow^1 \text{coe}_{A', B} t}$$

$$\text{CoER} \frac{\text{nf}^\oplus \text{ or ne } A \quad B \rightsquigarrow^1 B'}{\text{coe}_{A, B} t \rightsquigarrow^1 \text{coe}_{A, B'} t} \qquad \text{CoETM} \frac{\text{nf}^\oplus \text{ or ne } A, B \quad t \rightsquigarrow^1 t'}{\text{coe}_{A, B} t \rightsquigarrow^1 \text{coe}_{A, B} t'}$$

$$\text{CoECO} \frac{\text{nf}^\oplus \text{ or ne } U, U', T, T' \quad \text{ne } n}{\text{coe}_{U, U'} \text{coe}_{T, T'} n \rightsquigarrow^1 \text{coe}_{T, U'} n}$$

$\text{nf } f$	$\stackrel{\text{def}}{=} n \mid P \mid N \mid \lambda x:t.t \mid \{l := t_l\}_{l \in \mathcal{L}} \mid$ weak-head normal forms
$\text{nf}^\ominus N$	$\stackrel{\text{def}}{=} \Pi x:t.t \mid \{l : t_l\}_{l \in \mathcal{L}} \mid \dots$ negative whnf types
$\text{nf}^\oplus P$	$\stackrel{\text{def}}{=} \text{Type}_i \mid \text{List } t \mid \dots$ positive whnf types
$\text{ne } n$	$\stackrel{\text{def}}{=} x \mid n t \mid n.l \mid \text{ind}_P(n; t; t) \mid \dots$ weak-head neutrals
$\text{cne } c$	$\stackrel{\text{def}}{=} n \mid \text{coe}_{P, P} n \mid \text{coe}_{n, n} n$ compacted neutrals

Fig. 14. Weak-head reduction rules for coercion (extends Figure 4, complete rules: Appendix B.8)

$$\boxed{\Gamma \vdash_{\text{coe}} t \approx_{\text{coe}} t' \triangleleft T} \quad \text{Compacted neutrals } t \text{ and } t' \text{ are comparable at type } T$$

$$\text{NCOE} \frac{\Gamma \vdash_{\text{coe}} n \approx n' \triangleright S''}{\Gamma \vdash_{\text{coe}} \text{coe}_{S, T} n \approx_{\text{coe}} \text{coe}_{S', T'} n' \triangleleft T''} \qquad \text{NCOEL} \frac{\Gamma \vdash_{\text{coe}} n \approx n' \triangleright S''}{\Gamma \vdash_{\text{coe}} \text{coe}_{S, T} n \approx_{\text{coe}} n' \triangleleft T''}$$

$$\text{NCOER} \frac{\Gamma \vdash_{\text{coe}} n \approx n' \triangleright S''}{\Gamma \vdash_{\text{coe}} n \approx_{\text{coe}} \text{coe}_{S', T'} n' \triangleleft T''} \qquad \text{NNOCOE} \frac{\Gamma \vdash_{\text{coe}} n \approx n' \triangleright S''}{\Gamma \vdash_{\text{coe}} n \approx_{\text{coe}} n' \triangleleft T''}$$

$$\boxed{\Gamma \vdash_{\text{coe}} t \cong_h t' \triangleleft T}$$

$$\text{NEULIST} \frac{\Gamma \vdash_{\text{coe}} n \approx_{\text{coe}} n' \triangleleft \text{List } A}{\Gamma \vdash_{\text{coe}} n \cong_h n' \triangleleft \text{List } A} \qquad \text{NEU} \frac{\Gamma \vdash_{\text{coe}} n \approx_{\text{coe}} n' \triangleleft M \quad \text{ne } M}{\Gamma \vdash_{\text{coe}} n \cong_h n' \triangleleft M}$$

Fig. 15. Algorithmic comparison of neutrals in MLTT_{coe} (extends Figure 6, complete rules: Appendix B.9)

stuck coe . This only applies to positive types (characterized by nf^\oplus): for negative/extensional types we do not compact coercions, waiting for the term to be observed to trigger further reduction.

Neutral conversion is described at the top of Figure 15 and features an additional comparison between compacted neutrals similar to MLTT_{map} (**LISTNECONV**). Rule **NCoe** is a congruence for coercions, where the source and target types necessarily agree by typing invariants, and are thus not compared. Rules **NCoeL** and **NCoeR** handle identity coercions. Accordingly, \approx_{coe} is carefully used whenever normal forms can be compacted neutrals, e.g. at neutral and positive types, as shown at the bottom of Figure 15. Apart from this change, conversion at the term and type level and subtyping are similar to those of MLTT_{sub} .

5.1.4 Declarative MLTT_{coe} . The declarative presentation of MLTT_{coe} , noted \vdash_{coe} , straightforwardly extends MLTT (Figures 2 and 3) with typing and conversion rules for records and coe similar to the ones of the algorithmic presentation. Most importantly, it contains the following two rules for definitional identity and composition of coercions.

$$\text{COEID} \frac{\Gamma \vdash_{\text{coe}} t : A}{\Gamma \vdash_{\text{coe}} \text{coe}_{A,A} t \cong t : A} \quad \text{COETRANS} \frac{\Gamma \vdash_{\text{coe}} t : A \quad \Gamma \vdash_{\text{coe}} A \preccurlyeq A' \quad \Gamma \vdash_{\text{coe}} A' \preccurlyeq A''}{\Gamma \vdash_{\text{coe}} \text{coe}_{A',A''} \text{coe}_{A,A'} t \cong \text{coe}_{A,A''} t : A''}$$

5.2 Equivalence of algorithmic and declarative typing

All the metatheoretic properties of MLTT_{map} mentioned in Section 4 carry over to MLTT_{coe} . We highlight the following two. Note that the second in particular implies that inferred types in the algorithmic system are principal.

THEOREM 5.1 (WEAK-HEAD NORMALIZATION). *If $\Gamma \vdash_{\text{coe}} t : T$, then there exists a weak-head normal form t' such that $t \rightsquigarrow^* t'$.*

THEOREM 5.2 (SOUNDNESS AND COMPLETENESS OF ALGORITHMIC TYPING). *If $\vdash_{\text{coe}} \Gamma$ and $\Gamma \vdash_{\text{coe}} t \triangleright T$ then $\Gamma \vdash_{\text{coe}} t : T$, and similarly for the other judgements. Conversely, if $\Gamma \vdash_{\text{coe}} t : T$, then $\Gamma \vdash_{\text{coe}} t \triangleleft T$, and similarly for the other judgements.*

Proof ideas from MLTT_{map} carry over to MLTT_{coe} , and we did not mechanize this part of the paper, focusing our formalization effort on the most challenging aspect of the theory. We sketch how to extend the logical relation for MLTT_{map} to MLTT_{coe} – the proofs of equivalence between the declarative and algorithmic systems from the logical relation then remain mostly unchanged.

PROOF SKETCH (EXTENDING THE LOGICAL RELATION TO MLTT_{coe}). MLTT_{coe} has three main differences compared to MLTT_{map} : record types, subtyping and coercions.

First, we need to define the logical relation at record types and show the validity of introduction and elimination forms. Since, records behave as iterated Cartesian products, the reducibility proof carries over. Thus, reducibility at record types is defined as reducibility of each projection.

Second, we need to extend reducible type-level conversion to handle subtyping. As the structure of the two judgements is exactly the same, apart from the base subtyping case, we can parameterize reducible conversion by a *conversion problem*,¹³ a three-valued variant indicating conversion, subtyping, or supertyping, the latter being needed to handle contravariance and the left bias of reducible conversion, which is defined on a proof of reducibility of its left type.

Finally, we need to show that $\text{coe}_{A,A'} t$ is reducible whenever A is a reducible subtype of A' , and t is reducible at A . Because of the former, both must have normal forms which are either constructed with the same type former F , both record types, or both neutrals. In the first case, $\text{coe}_{A,A'} t$ behaves like map_F , and so the reducibility proofs from Section 4 carry over. In case A and A' are both neutral, $\text{coe}_{A,A'} t$ might compact if t is a coercion itself, but this is also similar to the case of a neutral map for lists in MLTT_{map} , and so the proof from Section 4 carries over again.

¹³This technique is borrowed from the way cumulativity is handled in METACoQ [Sozeau et al. 2023].

We are left with the case of record types. We need to show that if t is reducible at $\{l : A_l\}_{l \in \mathcal{L}}$ which is a reducible subtype of $\{k : B_k\}_{k \in \mathcal{K}}$ then $\text{coe}_{\{l : A_l\}_{l \in \mathcal{L}}, \{k : B_k\}_{k \in \mathcal{K}}} t$ is reducible. By definition of reducibility at record types as reducibility of all projections, and closure of reducibility by anti-reduction, it is enough to show that each $\text{coe}_{A_k, B_k} t.k$ is reducible at B_k for $k \in \mathcal{K}$. Combining the reducibility of $t.k$ obtained from that of t , together with the induction hypothesis on the reducible subtyping $A_k \preceq B_k$ completes this step. \square

5.3 Elaboration and erasure

We can now turn to the correspondence between MLTT_{sub} and MLTT_{coe} . The translation in the forward direction, *erasure* $|\cdot|$, simply removes coercions $|\text{coe}_{A, A'} t| = t$ and is otherwise a congruence. It is lifted pointwise to contexts. We first show that erasure is sound, meaning that it preserves typing and conversion, and then that it is also invertible, *i.e.* that any well-typed MLTT_{sub} term t' elaborates to a well-typed MLTT_{coe} term t whose erasure is $t' = |t|$.

5.3.1 Soundness of erasure. Erasure translates from a constrained system to a more liberal one. Establishing its soundness is relatively easy as long as the reduction rules of Figure 14 is designed so that the lemmas below hold. The key point is that reduction rules for coe do *not* change the structure of the erased term, and erase to exactly zero steps of reduction. For instance, the rule below is inadequate, as it would η -expand terms at function types more in MLTT_{coe} than in MLTT_{sub} . It remains nonetheless true, but only as a conversion.

$$\text{coe}_{\Pi x:A'.B', \Pi x:A.B} f \rightsquigarrow^1 \lambda x:A. \text{coe}_{B'[\text{coe}_{A, A'} x], B} (f \text{coe}_{A, A'} x)$$

LEMMA 5.3 (ERASURE OF REDUCTION). *If $t \rightsquigarrow^* u$, then also $|t| \rightsquigarrow^* |u|$.*

PROOF. By induction on the number of steps, and then on the derivation of one-step reduction. Coercion reduction in MLTT_{coe} map to zero steps on the erased terms, while other reduction steps map to their counterpart after erasure, using that erasure commutes with substitution. \square

THEOREM 5.4 (ERASURE PRESERVES SUBTYPING). *The following implications hold whenever the inputs of the first hypothesis are well-formed:*

- (1) *if $\Gamma \vdash_{\text{coe}} t \approx_{\text{coe}} u \triangleleft T$ then there exists T' such that $\Gamma \vdash_{\text{coe}} T' \preceq T \triangleleft$ and $|\Gamma| \vdash_{\text{sub}} |t| \approx |u| \triangleright |T'|$;*
- (2) *if $\Gamma \vdash_{\text{coe}} T \preceq_{\text{h}} U \triangleleft$, then $|\Gamma| \vdash_{\text{sub}} |T| \preceq_{\text{h}} |U| \triangleleft$;*
- (3) *and similarly for the other subtyping and conversion judgements.*

PROOF. By mutual induction, each rule being mapped to their counterpart. Rules for $\Gamma \vdash_{\text{coe}} n \approx_{\text{coe}} n' \triangleleft T$ are simply dropped, as that judgement is replaced by $\Gamma \vdash_{\text{sub}} n \approx n' \triangleright S$ in MLTT_{sub} . Lemma 5.3 is employed whenever terms and types are reduced to normal forms. \square

THEOREM 5.5 (SOUNDNESS OF ERASURE – INDUCTION). *The following implications hold, whenever the inputs of the premise are well-formed:*

- *if $\Gamma \vdash_{\text{coe}} t \triangleright T$, then there exists T' such that $\Gamma \vdash_{\text{coe}} T' \preceq T \triangleleft$ and $|\Gamma| \vdash_{\text{sub}} |t| \triangleright |T'|$;*
- *if $\Gamma \vdash_{\text{coe}} t \triangleright_{\text{h}} T$, then there exists T' such that $\Gamma \vdash_{\text{coe}} T' \preceq_{\text{h}} T \triangleleft$ and $|\Gamma| \vdash_{\text{sub}} |t| \triangleright_{\text{h}} |T'|$;*
- *if $\Gamma \vdash_{\text{coe}} t \triangleleft T$, then there exists T' such that $\Gamma \vdash_{\text{coe}} T' \preceq T \triangleleft$ and $|\Gamma| \vdash_{\text{sub}} |t| \triangleright |T'|$.*

PROOF. By mutual induction. Checking needs transitivity of \preceq . Reduced inference relies on Lemma 5.3 to handle reduction. Finally, each rule for inference can be mapped to its counterpart, noting that $\Gamma \vdash_{\text{coe}} T' \preceq T \triangleleft$ and $|\Gamma| \vdash_{\text{sub}} |t| \triangleright |T'|$ together imply, by Theorem 5.4, $|\Gamma| \vdash_{\text{sub}} |t| \triangleleft |T|$, so that induction hypothesis on checking premises in MLTT_{coe} can be turned into checking premises in MLTT_{sub} . Finally, for the introduction of $\text{coe}_{A, B} t$, subtyping between A and B is combined with the subtyping derivation obtained by induction hypothesis on t . \square

In the end, combining with completeness of algorithmic MLTT_{coe} , we obtain the following high-level corollary that erasure preserves typing.

COROLLARY 5.6 (SOUNDNESS OF ERASURE). *If $\Gamma \vdash_{\text{coe}} t : T$, then $|\Gamma| \vdash_{\text{sub}} |t| \triangleleft |T|$.*

5.3.2 Elaboration. This direction is more challenging: as we add annotations, we must show that these do not hinder conversion checking. We follow the proof strategy of a similar proof of elaboration soundness in [Lennon-Bertrand et al. \[2022\]](#). The core of the argument are so-called “catch-up lemmas”, which ensure that annotations never block redexes.

LEMMA 5.7 (CATCH UP, FUNCTION TYPE). *If $\Gamma \vdash_{\text{coe}} f a : B$ and $|f| = \lambda x : A'. t'$, then there exists t such that $|t| = t'$ and $f a \rightsquigarrow^* t[a]$.*

LEMMA 5.8 (CATCH UP, RECORD TYPE). *If $\Gamma \vdash_{\text{coe}} r.l : A$ and $|r| = \{l := u_l\}_{l \in \mathcal{L}}$, then there exists t such that $|t| = u_l$ and $r.l \rightsquigarrow^* t$.*

LEMMA 5.9 (CATCH UP, POSITIVE TYPES). *If T is a positive type (i.e. it is $\text{Type}_i, \mathbf{N}, \mathbf{List}, \mathbf{W}, +$) and $\Gamma \vdash_{\text{coe}} t : T$ is such that $|t|$ is a canonical form, then t reduces to a term with the same head constructor, and arguments which erase to those of $|t|$.*

PROOF. The idea is always the same: because t erases to a canonical form, it must be that same canonical form, surrounded by coercions. Because all types in these coercions are related by subtyping to the type of t , which is canonical as t is, all these coercions must reduce away. A detailed proof for the most challenging case, that of functions, is given in [Appendix C.2](#). \square

From these catch-up lemmas it follows that erasure is a backward simulation, therefore that it preserves subtyping, and finally that it is type-preserving. Proofs are all by induction, and given in [Appendix C.2](#).

LEMMA 5.10 (ERASURE IS A BACKWARD SIMULATION). *Assume that $\Gamma \vdash_{\text{coe}} t : T$. If $|t| \rightsquigarrow^* u'$, with u' a weak-head normal form, then $t \rightsquigarrow^* u$, with u a weak-head normal form such that $|u| = u'$.*

LEMMA 5.11 (ELABORATION PRESERVES SUBTYPING). *The following implications hold whenever the inputs of the conclusions are well-formed:*

- (1) *if $|\Gamma| \vdash_{\text{sub}} |T| \triangleleft |U| \triangleleft$, then $\Gamma \vdash_{\text{coe}} T \triangleleft U \triangleleft$;*
- (2) *if $|\Gamma| \vdash_{\text{sub}} |t| \cong |u| \triangleleft |T|$, then $\Gamma \vdash_{\text{coe}} t \cong u \triangleleft T$;*
- (3) *if $|\Gamma| \vdash_{\text{sub}} |t| \approx |u| \triangleright T$, then $\Gamma \vdash_{\text{coe}} t \approx u \triangleright T$;*
- (4) *and similarly for the other judgements.*

Finally, the main theorem states that we can elaborate terms using implicit subtyping to explicit coercions, in a type-preserving way.

THEOREM 5.12 (ELABORATION – INDUCTION). *The following implications hold, whenever inputs to the conclusion are well-formed:*

- (1) *if $|\Gamma| \vdash_{\text{sub}} t' \triangleright T'$, then there exists t and T such that $t' = |t|$, $T' = |T|$, and $\Gamma \vdash_{\text{coe}} t \triangleright T$;*
- (2) *if $|\Gamma| \vdash_{\text{sub}} t' \triangleright_h T'$, then there exists t and T such that $t' = |t|$, $T' = |T|$, and $\Gamma \vdash_{\text{coe}} t \triangleright_h T$;*
- (3) *if $|\Gamma| \vdash_{\text{sub}} t' \triangleleft |T|$, then there exists t such that $t' = |t|$ and $\Gamma \vdash_{\text{coe}} t \triangleleft T$.*

PROOF. Once again, by mutual induction. Each rule is mapped to its counterpart, but for **CHECK-SUB**, where we need to insert a coercion in the elaborated term. This coercion is well-typed by [Lemma 5.11](#). \square

We can unfold the assumption of input well-formation, to get the following high-level corollary.

COROLLARY 5.13 (ELABORATION). *If $\Gamma \vdash_{\text{coe}} T$ and $|\Gamma| \vdash_{\text{sub}} t' \triangleleft |T|$, then there exists t such that $\Gamma \vdash_{\text{coe}} t : T$, and $|t| = t'$.*

Note that, to establish this equivalence we did *not* need to develop any meta-theory for MLTT_{sub} : having the meta-theory of MLTT_{coe} was enough! Nonetheless, now that the equivalence between the two systems has been established, we can use it to transport meta-theoretic properties, such as normalization, from MLTT_{coe} to MLTT_{sub} .

5.4 Coherence

An important property of elaboration is *coherence*, stating that the elaboration of a well-typed term does not depend on its typing derivation. In our algorithmic setting, a term has at most one typing derivation and so at most one elaboration. However, multiple well-typed terms in MLTT_{coe} can still erase to the same MLTT_{sub} term. While only one of them is the result of elaboration as defined in Corollary 5.13, all these distinct terms should still behave similarly. The following is a direct consequence of Lemma 5.11, and shows that the equations imposed on *coe* are enough to give us a very strong form of coherence: it holds up to definitional equality, rather than in a weaker, semantic way. Another way to look at this is that the scenario of Example 1.2 cannot happen, thanks to our new equations: if two terms erase to the same coercion-free one in MLTT_{sub} , then they *must* be convertible in MLTT_{coe} . Hidden coercions cannot be responsible for failures of conversion.

THEOREM 5.14 (COHERENCE). *If t, u are such that $\Gamma \vdash_{\text{coe}} t \triangleleft T$ and $\Gamma \vdash_{\text{coe}} u \triangleleft T$, and moreover $|t| = |u|$ (i.e. t and u are both “elaborations” of the same MLTT_{sub} term), then $\Gamma \vdash_{\text{coe}} t \cong u \triangleleft T$.*

PROOF. By reflexivity (obtained through the equivalence with the declarative system), $\Gamma \vdash_{\text{coe}} t \cong t \triangleleft T$. Using Theorem 5.5 (soundness of erasure), we get $|\Gamma| \vdash_{\text{sub}} |t| \cong |t| \triangleleft |T|$, and so also $|\Gamma| \vdash_{\text{sub}} |t| \cong |u| \triangleleft |T|$. But then by Lemma 5.11 (elaboration preserving conversion), we can come back, and obtain $\Gamma \vdash_{\text{coe}} t \cong u \triangleleft T$. \square

6 RELATED AND FUTURE WORK

Adding definitional equations to dependent type theory. Strub [2010] endows a dependent type theory with additional equations from first order decidable theories, with further extensions to a universe hierarchy and large eliminations in Jouannaud and Strub [2017] and Barras et al. [2011]. Equational theories can sometimes be presented by a confluent set of rewrite rules, a case advocated by Cockx et al. [2021]. They show through counter-examples that ensuring type preservation in dependent type theory is a subtle matter and do not ensure normalization of the resulting theory. On the theoretical side, categorical tools are being developed to prove general conservativity and strictification results for type theories [Bocquet 2021, 2023] extending the seminal work of Hofmann [1997] on conservativity of extensional type theory with respect to intensional type theory [Winterhalter et al. 2019].

Formalized metatheory with logical relations. Allais et al. [2013] propose to add a variety of fusion laws for lists, including our functor laws, to a simply typed λ -calculus, only sketching an extension to dependent types. While we depart from their normalization by evaluation approach, we retain some traces of it, typically in the presence of three classes of normal forms (see Figures 10 and 14) instead of the usual normal/neutral. Formalizing logical relations for MLTT is a difficult exercise, pioneered by Abel et al. [2017] in AGDA using inductive-recursive definitions, and Wiczorek and Biernacki [2018] in Coq using impredicativity. We build upon and extend a Coq reimplementations of the former [Adjedj et al. 2023].

Cast and coercion operators. Pujet and Tabareau [2022, 2023] extend Abel et al. [2017] to establish the metatheory of observational type theory [Altenkirch, McBride, et al. 2007]. Their work features a cast operator behaving similarly to `coe`, but guarded by an internal proof of equality instead of an external subtyping derivation. Their cast does not satisfy definitional transitivity, and we give evidence in Appendix A that such an extension would break metatheoretical properties. Another cast primitive with a similar operational behaviour appears in cast calculi for gradual typing [Siek et al. 2015], and indeed our proof that elaboration is type preserving in Section 5.3 is inspired by a similar one for GCIC, which combines gradual and dependent types [Lennon-Bertrand et al. 2022]. In this case, casting is allowed between any two types, but the absence of guard is compensated by the possibility of runtime errors, making the type theory inconsistent.

Functorial maps for inductive type schemes. Z. Luo and Adams [2008] describe the construction of `map` for a class of strictly positive operators on paper, but do not implement it. Deriving `map`-like construction is a typical example of metaprogramming frameworks for proof assistants, e.g. `Coq-ELPI` [Tassi 2018; Dunchev et al. 2015] in `Coq`, and the generics `AGDA` library [Escot and Cockx 2022] derives a fold operation, from which `map` can be easily obtained. In a simply typed setting, Barral and Soloviev [2006] employ rewriting techniques, in particular rewriting postponement, to show that an oriented variant of the functor laws are confluent and normalizing. These techniques rely on normalization, and could not be easily adapted to the dependent setting, however the idea of postponing the reduction step for identity appear in our logical relation as well. In a short abstract, McBride and Nordvall Forsberg [2021] investigate a notion of functorial adapters that generalizes and unifies both the `CHECK` rule from bidirectional typing and the `COE` rule from `MLTTcoe`.

Subtyping, dependent types and algorithmic derivations. Coherence of coercions in presence of structural subtyping is a challenging problem. To address the issue, Z. Luo and Y. Luo [2005] introduce a notion of weak transitivity, weakening the coherence of the transitivity up to propositional equality. This solution does not interact well with dependency, forcing them to restrict structural subtyping to a class of non-dependent inductives, e.g. excluding (positive) Σ . Z. Luo and Adams [2008] show that the transitivity of coercions is admissible in presence of definitional compositions – called χ -rules there – for inductive schemata. They rely on a conjecture that strong normalization and subject reduction hold in presence of these χ -rules, explicitly mentioning that the metatheory with those additional equality rules is “largely unknown”. We provide such results, and have formalized them for `List`. We use a completely different proof technique, that scales to a theory with universes and large elimination. Both aforementioned papers employ a strict order for subtyping and do not consider the functor law for the identity, nor tackle decidability of type-checking.

Aspinall and Compagnoni [2001] investigate the relationship between subtyping and dependent types using algorithmic derivations to control the subtyping derivations for a variant of λP , a type theory logically much weaker than `MLTT`. Lungu and Z. Luo [2018] study an elaboration of a subsumptive presentation into coercive one in presence of a coherent signature of subtyping relations between base types. Assuming normalization, they show that subtyping extends to Π types, setting aside other parametrized types. While they work over an abstract signature of coercions, the functor laws we study are needed to instantiate this signature with meaningful datatypes while respecting their assumptions. We explain the relation of these algorithmic system with bidirectional systems, notably the one of Abel et al. [2017], contributing to a sharper picture.

Integration with other forms of subtyping. As we mentioned in Section 5, our design of base subtyping was guided by simplicity. Our work on structural subtyping should integrate mostly seamlessly with other, more ambitious forms of subtyping. Coercions between dependent records form the foundation of hierarchical organizations of mathematical structures [Cohen et al. 2020;

Affeldt et al. 2020; Wieser 2023], and should be a simple extension of our framework. This could lead to vast simplification of the complex apparatus currently needed to deal with these hierarchies.

Refinement subtyping is heavily used in F^* but also in COQ’s PROGRAM [Sozeau 2007] to specify the behaviour of programs. Relativizing any result of decidability of type-checking to that of the chosen fragment of refinements, an implementation of refinement subtyping using definitionally irrelevant propositions [Gilbert, Cockx, et al. 2019] to preserve coherence¹⁴ should be within reach.

Our techniques for structural subtyping should also apply well in the context of algebraic approaches to cumulativity between universes [Sterling 2019; Kovács 2022]. Cumulativity goes beyond mere subtyping, as it also involves definitional isomorphisms between two copies of the same type at different universe levels. Our definitional functor laws already allow these to interact well with map operations, but it would be interesting to investigate which extra definitional equations are needed – and can be realized – to make structural cumulativity work seamlessly, hopefully obtaining a translation from Russel-style to Tarski-style universes similar to our elaboration from $MLTT_{sub}$ to $MLTT_{coe}$.

REFERENCES

- Michael Gordon Abbott, Thorsten Altenkirch, and Neil Ghani. 2005. “Containers: Constructing strictly positive types.” *Theor. Comput. Sci.*, 342, 1, 3–27. doi: [10.1016/j.tcs.2005.06.002](https://doi.org/10.1016/j.tcs.2005.06.002).
- Andreas Abel, Joakim Öhman, and Andrea Vezzosi. Dec. 2017. “Decidability of Conversion for Type Theory in Type Theory.” *Proc. ACM Program. Lang.*, 2, POPL, Article 23, (Dec. 2017), 29 pages. doi: [10.1145/3158111](https://doi.org/10.1145/3158111).
- Arthur Adjedj, Meven Lennon-Bertrand, Kenji Maillard, Pierre-Marie Pédrot, and Loïc Pujet. Sept. 2023. “Martin-Löf à la Coq.” working paper or preprint. (Sept. 2023). <https://inria.hal.science/hal-04214008>.
- Reynald Affeldt, Cyril Cohen, Marie Kerjean, Assia Mahboubi, Damien Rouhling, and Kazuhiko Sakaguchi. June 2020. “Competing inheritance paths in dependent type theory: a case study in functional analysis.” In: *IJCAR 2020 - International Joint Conference on Automated Reasoning*. Paris, France, (June 2020), 1–19. <https://inria.hal.science/hal-02463336>.
- Agda Development Team. 2023. *Agda 2.6.3 documentation*. <https://agda.readthedocs.io/en/v2.6.3/>.
- Guillaume Allais, Conor McBride, and Pierre Boutillier. 2013. “New Equations for Neutral Terms: A Sound and Complete Decision Procedure, Formalized.” In: *Proceedings of the 2013 ACM SIGPLAN Workshop on Dependently-Typed Programming (DTP ’13)*. Association for Computing Machinery, Boston, Massachusetts, USA, 13–24. ISBN: 9781450323840. doi: [10.1145/2502409.2502411](https://doi.org/10.1145/2502409.2502411).
- Thorsten Altenkirch, Neil Ghani, Peter G. Hancock, Conor McBride, and Peter Morris. 2015. “Indexed containers.” *J. Funct. Program.*, 25. doi: [10.1017/S095679681500009X](https://doi.org/10.1017/S095679681500009X).
- Thorsten Altenkirch, Conor McBride, and Wouter Swierstra. 2007. “Observational Equality, Now!” In: *Proceedings of the 2007 Workshop on Programming Languages Meets Program Verification (PLPV ’07)*. Association for Computing Machinery, Freiburg, Germany, 57–68. ISBN: 9781595936776. doi: [10.1145/1292597.1292608](https://doi.org/10.1145/1292597.1292608).
- David Aspinall and Adriana Compagnoni. 2001. “Subtyping dependent types.” *Theoretical Computer Science*, 266, 1, 273–309. doi: [https://doi.org/10.1016/S0304-3975\(00\)00175-4](https://doi.org/10.1016/S0304-3975(00)00175-4).
- Freirc Barral and Sergei Soloviev. 2006. “Inductive Type Schemas as Functors.” In: *Computer Science - Theory and Applications, First International Symposium on Computer Science in Russia, CSR 2006, St. Petersburg, Russia, June 8-12, 2006, Proceedings (Lecture Notes in Computer Science)*. Ed. by Dima Grigoriev, John Harrison, and Edward A. Hirsch. Vol. 3967. Springer, 35–45. ISBN: 3-540-34166-8. doi: [10.1007/11753728_7](https://doi.org/10.1007/11753728_7).
- Bruno Barras, Jean-Pierre Jouannaud, Pierre-Yves Strub, and Qian Wang. 2011. “CoQMTU: A Higher-Order Type Theory with a Predicative Hierarchy of Universes Parametrized by a Decidable First-Order Theory.” In: *Proceedings of the 26th Annual IEEE Symposium on Logic in Computer Science, LICS 2011, June 21-24, 2011, Toronto, Ontario, Canada*. IEEE Computer Society, 143–151. ISBN: 978-0-7695-4412-0. doi: [10.1109/LICS.2011.37](https://doi.org/10.1109/LICS.2011.37).
- Rafaël Bocquet. 2021. “Strictification of Weakly Stable Type-Theoretic Structures Using Generic Contexts.” In: *27th International Conference on Types for Proofs and Programs, TYPES 2021, June 14-18, 2021, Leiden, The Netherlands (Virtual Conference) (LIPICs)*. Ed. by Henning Basold, Jesper Cockx, and Silvia Ghilezan. Vol. 239. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 3:1–3:23. ISBN: 978-3-95977-254-9. doi: [10.4230/LIPICs.TYPES.2021.3](https://doi.org/10.4230/LIPICs.TYPES.2021.3).
- Rafaël Bocquet. 2023. “Towards coherence theorems for equational extensions of type theories.” *CoRR*, abs/2304.10343. arXiv: [2304.10343](https://arxiv.org/abs/2304.10343). doi: [10.48550/arXiv.2304.10343](https://doi.org/10.48550/arXiv.2304.10343).

¹⁴With proof-relevant propositions, different proofs of $p \Rightarrow q$ induce different coercions between $\{x \mid p\}$ and $\{x \mid q\}$, breaking coherence.

- Edwin C. Brady. 2021. “Idris 2: Quantitative Type Theory in Practice (Artifact).” *Dagstuhl Artifacts Ser.*, 7, 2, 10:1–10:7. doi: [10.4230/DARTS.7.2.10](https://doi.org/10.4230/DARTS.7.2.10).
- Simon Castellan, Pierre Clairambault, and Peter Dybjer. 2017. “Undecidability of Equality in the Free Locally Cartesian Closed Category (Extended version).” *Log. Methods Comput. Sci.*, 13, 4. doi: [10.23638/LMCS-13\(4:22\)2017](https://doi.org/10.23638/LMCS-13(4:22)2017).
- Jesper Cockx. 2020a. *Disable all subtyping by default?* <https://github.com/agda/agda/issues/4474>. Accessed: 2023-07-10. (2020).
- Jesper Cockx. 2020b. “Type Theory Unchained: Extending Agda with User-Defined Rewrite Rules.” In: *25th International Conference on Types for Proofs and Programs (TYPES 2019)* (Leibniz International Proceedings in Informatics (LIPIcs)). Ed. by Marc Bezem and Assia Mahboubi. Vol. 175. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2:1–2:27. ISBN: 978-3-95977-158-0. doi: [10.4230/LIPIcs.TYPES.2019.2](https://doi.org/10.4230/LIPIcs.TYPES.2019.2).
- Jesper Cockx, Nicolas Tabareau, and Théo Winterhalter. 2021. “The Taming of the Rew: A Type Theory with Computational Assumptions.” *Proceedings of the ACM on Programming Languages*. POPL 2021. <https://hal.archives-ouvertes.fr/hal-02901011>.
- Cyril Cohen, Kazuhiko Sakaguchi, and Enrico Tassi. June 2020. “Hierarchy Builder: algebraic hierarchies made easy in Coq with Elpi.” In: *FSCD 2020 - 5th International Conference on Formal Structures for Computation and Deduction* 167. Paris, France, (June 2020), 34:1–34:21. doi: [10.4230/LIPIcs.FSCD.2020.34](https://doi.org/10.4230/LIPIcs.FSCD.2020.34).
- Cvetan Dunchev, Ferruccio Guidi, Claudio Sacerdoti Coen, and Enrico Tassi. Nov. 2015. “ELPI: fast, Embeddable, λ Prolog Interpreter.” In: *Proceedings of LPAR*. Suva, Fiji, (Nov. 2015). <https://inria.hal.science/hal-01176856>.
- Jana Dunfield and Neel Krishnaswami. May 2021. “Bidirectional Typing.” *ACM Computing Surveys*, 54, 5, Article 98, (May 2021), 38 pages. doi: [10.1145/3450952](https://doi.org/10.1145/3450952).
- Lucas Escot and Jesper Cockx. 2022. “Practical Generic Programming over a Universe of Native Datatypes.” *Proc. ACM Program. Lang.*, 6, ICFP, Article 113, 25 pages. doi: [10.1145/3547644](https://doi.org/10.1145/3547644).
- Lucas Escot, Josselin Poirer, Joris Ceulemans, Andreas Nuyts, and Malin Altenmüller. 2023. “Read the mode and stay positive.” In: *29th International Conference on Types for Proofs and Programs*.
- Gaëtan Gilbert, Jesper Cockx, Matthieu Sozeau, and Nicolas Tabareau. Jan. 2019. “Definitional Proof-Irrelevance without K.” *Proceedings of the ACM on Programming Languages*. POPL’19 3, POPL, (Jan. 2019), 1–28. doi: [10.1145/3290316.11453290316](https://doi.org/10.1145/3290316.11453290316).
- Gaëtan Gilbert, Yann Leray, Nicolas Tabareau, and Théo Winterhalter. June 2023. *The Rewster: The Coq Proof Assistant with Rewrite Rules*. Presentation at TYPES’23. (June 2023). <https://media.upv.es/#/portal/video/bf2591f0-34a6-11ee-8485-f133f82f8945>.
- Martin Hofmann. 1997. *Extensional constructs in intensional type theory*. CPHC/BCS distinguished dissertations. Springer. ISBN: 978-3-540-76121-1.
- Cătălin Hrițcu. 2014. *Polarities: subtyping for datatypes*. <https://github.com/FStarLang/FStar/issues/65>. Accessed: 2023-07-04. (2014).
- Jasper Hugunin. 2020. “Why Not W?” In: *26th International Conference on Types for Proofs and Programs, TYPES 2020, March 2-5, 2020, University of Turin, Italy* (LIPIcs). Ed. by Ugo de’Liguoro, Stefano Berardi, and Thorsten Altenkirch. Vol. 188. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 8:1–8:9. ISBN: 978-3-95977-182-5. doi: [10.4230/LIPIcs.TYPES.2020.8](https://doi.org/10.4230/LIPIcs.TYPES.2020.8).
- Bart P. F. Jacobs. 2001. *Categorical Logic and Type Theory*. Studies in logic and the foundations of mathematics. Vol. 141. North-Holland. ISBN: 978-0-444-50853-9. <http://www.elsevierdirect.com/product.jsp?isbn=9780444508539>.
- Jean-Pierre Jouannaud and Pierre-Yves Strub. 2017. “Coq without Type Casts: A Complete Proof of Coq Modulo Theory.” In: *LPAR-21, 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Maun, Botswana, May 7-12, 2017* (EPIc Series in Computing). Ed. by Thomas Eiter and David Sands. Vol. 46. EasyChair, 474–489. doi: [10.29007/bjpg](https://doi.org/10.29007/bjpg).
- András Kovács. 2022. “Generalized Universe Hierarchies and First-Class Universe Levels.” In: *30th EACSL Annual Conference on Computer Science Logic (CSL 2022)* (Leibniz International Proceedings in Informatics (LIPIcs)). Ed. by Florin Manea and Alex Simpson. Vol. 216. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 28:1–28:17. ISBN: 978-3-95977-218-1. doi: [10.4230/LIPIcs.CSL.2022.28](https://doi.org/10.4230/LIPIcs.CSL.2022.28).
- Meven Lennon-Bertrand. 2021. “Complete Bidirectional Typing for the Calculus of Inductive Constructions.” In: *12th International Conference on Interactive Theorem Proving (ITP 2021)* (Leibniz International Proceedings in Informatics (LIPIcs)). Ed. by Liron Cohen and Cezary Kaliszyk. Vol. 193. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN: 978-3-95977-188-7. doi: [10.4230/LIPIcs.ITP.2021.24](https://doi.org/10.4230/LIPIcs.ITP.2021.24).
- Meven Lennon-Bertrand, Kenji Maillard, Nicolas Tabareau, and Éric Tanter. Apr. 2022. “Gradualizing the Calculus of Inductive Constructions.” *ACM Transactions on Programming Languages and Systems*, 44, 2, Article 7, (Apr. 2022), 82 pages. doi: [10.1145/3495528](https://doi.org/10.1145/3495528).
- Miran Lipovača. 2010. *Learn You a Haskell for Great Good!* <http://learnyouahaskell.com/>. (2010). <http://learnyouahaskell.com/>.

- Georgiana Elena Lungu and Zhaohui Luo. 2018. “On Subtyping in Type Theories with Canonical Objects.” In: *22nd International Conference on Types for Proofs and Programs (TYPES 2016)* (Leibniz International Proceedings in Informatics (LIPIcs)). Ed. by Silvia Ghilezan, Herman Geuvers, and Jelena Ivetić. Vol. 97. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 13:1–13:31. ISBN: 978-3-95977-065-1. DOI: [10.4230/LIPIcs.TYPES.2016.13](https://doi.org/10.4230/LIPIcs.TYPES.2016.13).
- Zhaohui Luo and Robin Adams. 2008. “Structural subtyping for inductive types with functorial equality rules.” *Mathematical Structures in Computer Science*, 18, 5, 931–972. DOI: [10.1017/S0960129508006956](https://doi.org/10.1017/S0960129508006956).
- Zhaohui Luo and Yong Luo. 2005. “Transitivity in coercive subtyping.” *Inf. Comput.*, 197, 1-2, 122–144. DOI: [10.1016/j.ic.2004.10.008](https://doi.org/10.1016/j.ic.2004.10.008).
- Saunders MacLane. 1971. *Categories for the Working Mathematician*. Graduate Texts in Mathematics, Vol. 5. Springer-Verlag, New York.
- Per Martin-Löf and Giovanni Sambin. 1984. *Intuitionistic Type Theory*. Studies in Proof Theory 1. Napoli: Bibliopolis.
- Conor McBride. 2009. “Grins from my Ripley Cupboard.” (2009).
- Conor McBride. 2022. “Types Who Say Ni.” (2022).
- Conor McBride and Frederik Nordvall Forsberg. June 2021. *Functorial Adapters*. 27th International Conference on Types for Proofs and Programs. (June 2021).
- Leonardo de Moura and Sebastian Ullrich. 2021. “The Lean 4 Theorem Prover and Programming Language.” In: *Automated Deduction - CADE 28 - 28th International Conference on Automated Deduction, Virtual Event, July 12-15, 2021, Proceedings* (Lecture Notes in Computer Science). Ed. by André Platzer and Geoff Sutcliffe. Vol. 12699. Springer, 625–635. ISBN: 978-3-030-79875-8. DOI: [10.1007/978-3-030-79876-5_37](https://doi.org/10.1007/978-3-030-79876-5_37).
- Benjamin C. Pierce and David N. Turner. Jan. 2000. “Local Type Inference.” *ACM Transactions on Programming Languages and Systems*, 22, 1, (Jan. 2000), 1–44. DOI: [10.1145/345099.345100](https://doi.org/10.1145/345099.345100).
- Loïc Pujet and Nicolas Tabareau. Jan. 2023. “Impredicative Observational Equality.” *Proc. ACM Program. Lang.*, 7, POPL, Article 74, (Jan. 2023), 26 pages. DOI: [10.1145/3571739](https://doi.org/10.1145/3571739).
- Loïc Pujet and Nicolas Tabareau. 2022. “Observational Equality: Now for Good.” *Proc. ACM Program. Lang.*, 6, POPL, Article 32, 27 pages. DOI: [10.1145/3498693](https://doi.org/10.1145/3498693).
- Jeremy G. Siek, Michael M. Vitousek, Matteo Cimini, and John Tang Boyland. 2015. “Refined Criteria for Gradual Typing.” In: *1st Summit on Advances in Programming Languages (SNAPL 2015)* (Leibniz International Proceedings in Informatics (LIPIcs)). Ed. by Thomas Ball, Rastislav Bodik, Shriram Krishnamurthi, Benjamin S. Lerner, and Greg Morrisett. Vol. 32. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 274–293. DOI: [10.4230/LIPIcs.SNAPL.2015.274](https://doi.org/10.4230/LIPIcs.SNAPL.2015.274).
- Matthieu Sozeau. 2007. “Subset Coercions in Coq.” In: *Types for Proofs and Programs*. Ed. by Thorsten Altenkirch and Conor McBride. Springer Berlin Heidelberg, Berlin, Heidelberg, 237–252. ISBN: 978-3-540-74464-1.
- Matthieu Sozeau, Yannick Forster, Meven Lennon-Bertrand, Jakob Botsch Nielsen, Nicolas Tabareau, and Théo Winterhalter. Apr. 2023. “Correct and Complete Type Checking and Certified Erasure for Coq, in Coq.” Preprint. (Apr. 2023). <https://inria.hal.science/hal-04077552>.
- Jonathan Sterling. 2019. “Algebraic Type Theory and Universe Hierarchies.” *CoRR*, abs/1902.08848. arXiv: [1902.08848](https://arxiv.org/abs/1902.08848).
- Pierre-Yves Strub. 2010. “Coq Modulo Theory.” In: *Computer Science Logic, 24th International Workshop, CSL 2010, 19th Annual Conference of the EACSL, Brno, Czech Republic, August 23-27, 2010. Proceedings* (Lecture Notes in Computer Science). Ed. by Anuj Dawar and Helmut Veith. Vol. 6247. Springer, 529–543. ISBN: 978-3-642-15204-7. DOI: [10.1007/978-3-642-15205-4_40](https://doi.org/10.1007/978-3-642-15205-4_40).
- Nikhil Swamy et al. 2016. “Dependent types and multi-monadic effects in F.” 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*. Ed. by Rastislav Bodik and Rupak Majumdar. ACM, 256–270. ISBN: 978-1-4503-3549-2. DOI: [10.1145/2837614.2837655](https://doi.org/10.1145/2837614.2837655).
- Enrico Tassi. Jan. 2018. “Elpi: an extension language for Coq (Metaprogramming Coq in the Elpi λ Prolog dialect).” working paper or preprint. (Jan. 2018). <https://inria.hal.science/hal-01637063>.
- [SW] The Coq Development Team, *The Coq Proof Assistant* version 8.16, Sept. 2022. DOI: [10.5281/zenodo.7313584](https://doi.org/10.5281/zenodo.7313584), URL: <https://doi.org/10.5281/zenodo.7313584>.
- Paweł Wieczorek and Dariusz Biernacki. 2018. “A Coq Formalization of Normalization by Evaluation for Martin-Löf Type Theory.” In: *Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP 2018)*. Association for Computing Machinery, Los Angeles, CA, USA, 266–279. ISBN: 9781450355865. DOI: [10.1145/3167091](https://doi.org/10.1145/3167091).
- Eric Wieser. 2023. *Multiple inheritance hazards in algebraic typeclass hierarchies*. (2023). arXiv: [2306.00617](https://arxiv.org/abs/2306.00617) [cs.LG].
- Théo Winterhalter, Matthieu Sozeau, and Nicolas Tabareau. 2019. “Eliminating reflection from type theory.” In: *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2019, Cascais, Portugal, January 14-15, 2019*. Ed. by Assia Mahboubi and Magnus O. Myreen. ACM, 91–103. ISBN: 978-1-4503-6222-1. DOI: [10.1145/3293880.3294095](https://doi.org/10.1145/3293880.3294095).

A INTERNAL SUBTYPING AND UNDECIDABILITY OF CONVERSION

The goal of the coercive approach is to reflect all the potential ambiguities present in a subtyping derivation. As such, wouldn't it be easier to just internalize the notion of subtype and let type theory deal with it? The following observation shows that there exists a big obstruction to any decidability result for conversion as long as we want to stay equivalent to the subsumptive presentation of subtyping.

OBSERVATION A.1 (NO-GO OF INTERNAL SUBTYPING). *Suppose that \mathcal{T} is a type theory with a family $\text{sub } A B$ for any two types A and B , equipped with reflexivity witnesses $\text{refl}_A : \text{sub } A A$ and transitivity witnesses $\text{trans } w w' : \text{sub } A C$ for $w : \text{sub } A B$ and $w' : \text{sub } B C$, as well as a coercion function $\text{coe}_{A,B} : \text{sub } A B \rightarrow A \rightarrow B$, such that $\text{coe}_{A,A} \text{refl}_A \cong \text{id}_A$ and $\text{coe}_{B,C} w \circ \text{coe}_{A,B} w' \cong \text{coe}_{A,C}(\text{trans } w_{A,B} w_{B,C})$. Then \mathcal{T} embeds definitional models of the untyped λ -calculus, and in particular divergent terms.*

Indeed, whenever a context provides inhabitants of both $\text{sub } A B$ and $\text{sub } B A$, $\text{coe}_{A,B}$ and $\text{coe}_{B,A}$ provide a definitional isomorphism $A \cong B$. In particular any context inhabiting $\text{sub } A A \rightarrow A$ and $\text{sub } A \rightarrow A A$, for instance an inconsistent one, provides a definitional retraction of $A \rightarrow A$ onto A , hence a non-trivial model of the untyped λ -calculus with a divergent element $\Omega_A : A$. This observation motivates our external approach to subtyping with a specific judgement of subtyping that cannot be abstracted upon.

B COMPLETE TYPING RULES

B.1 Declarative MLTT

$\boxed{\vdash \Gamma}$ Context Γ is well-formed

$$\frac{}{\vdash \cdot} \qquad \frac{\vdash \Gamma \quad \Gamma \vdash A : \text{Type}_i}{\vdash \Gamma, x : A}$$

$\boxed{\Gamma \vdash \sigma : \Delta}$ σ is a well-typed substitution between contexts Γ and Δ

$$\frac{}{\Gamma \vdash \cdot : \cdot} \qquad \frac{\Gamma \vdash \sigma : \Delta \quad \Gamma \vdash t : A[\sigma]}{\Gamma \vdash (\sigma, t) : \Delta, x : A}$$

$\boxed{\Gamma \vdash T}$ Type T is well-formed in context Γ

$$\begin{array}{c} \text{EL} \frac{\Gamma \vdash A : \text{Type}_i}{\Gamma \vdash A} \qquad \text{FUNTY} \frac{\Gamma \vdash A \quad \Gamma, x : A \vdash B}{\Gamma \vdash \Pi x : A. B} \qquad \text{LISTTY} \frac{\Gamma \vdash A}{\Gamma \vdash \mathbf{List } A} \\ \\ \text{SIGTY} \frac{\Gamma \vdash A \quad \Gamma, x : A \vdash B}{\Gamma \vdash \Sigma x : A. B} \qquad \text{TREETY} \frac{\Gamma \vdash A \quad \Gamma, x : A \vdash B}{\Gamma \vdash \mathbf{W } x : A. B} \\ \\ \text{IDTY} \frac{\Gamma \vdash A \quad \Gamma \vdash a : A \quad \Gamma \vdash a' : A}{\Gamma \vdash \mathbf{Id}_A a a'} \qquad \text{SUMTY} \frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A + B} \end{array}$$

$\boxed{\Gamma \vdash t : T}$ Term t has type T under context Γ

$$\begin{array}{c}
\text{CONV} \frac{\Gamma \vdash t : A \quad \Gamma \vdash A \cong B}{\Gamma \vdash t : B} \quad \text{VAR} \frac{\vdash \Gamma \quad (x : A) \in \Gamma}{\Gamma \vdash x : A} \quad \text{SORT} \frac{\vdash \Gamma}{\Gamma \vdash \mathbf{Type}_i : \mathbf{Type}_{i+1}} \\
\\
\text{FUNUNI} \frac{\Gamma \vdash A : \mathbf{Type}_i \quad \Gamma, x : A \vdash B : \mathbf{Type}_i}{\Gamma \vdash \Pi x : A. B : \mathbf{Type}_i} \quad \text{ABS} \frac{\Gamma \vdash A \quad \Gamma, x : A \vdash B}{\Gamma \vdash \lambda x : A. t : \Pi x : A. B} \quad \text{APP} \frac{\Gamma \vdash t : \Pi x : A. B \quad \Gamma \vdash u : A}{\Gamma \vdash t u : B[u]} \\
\\
\text{LISTUNI} \frac{\Gamma \vdash A : \mathbf{Type}_i}{\Gamma \vdash \mathbf{List} A : \mathbf{Type}_i} \quad \text{NIL} \frac{\Gamma \vdash A}{\Gamma \vdash \varepsilon_A : \mathbf{List} A} \quad \text{CONS} \frac{\Gamma \vdash A \quad \Gamma \vdash a : A \quad \Gamma \vdash l : \mathbf{List} A}{\Gamma \vdash a ::_A l : \mathbf{List} A} \\
\\
\text{LISTIND} \frac{\Gamma, x : \mathbf{List} A \vdash P \quad \Gamma \vdash A \quad \Gamma \vdash s : \mathbf{List} A \quad \Gamma \vdash b_\varepsilon : P[\varepsilon_A] \quad \Gamma, x : A, y : \mathbf{List} A, z : P[y] \vdash b_\varepsilon : P[x ::_A y]}{\Gamma \vdash \mathbf{ind}_{\mathbf{List} A}(s; z. P; b_\varepsilon, x.y.z.b_\varepsilon) : P[s]} \\
\\
\text{EMPTYUNI} \frac{}{\Gamma \vdash \mathbf{0} : \mathbf{Type}_0} \quad \text{UNITUNI} \frac{}{\Gamma \vdash \mathbf{1} : \mathbf{Type}_0} \quad \text{UNITTM} \frac{}{\Gamma \vdash () : \mathbf{1}} \\
\\
\text{EMPTYIND} \frac{\Gamma \vdash s : \mathbf{0} \quad \Gamma \vdash P}{\Gamma \vdash \mathbf{ind}_0(s; P) : P} \quad \text{UNITIND} \frac{\Gamma \vdash s : \mathbf{1} \quad \Gamma, z : \mathbf{1} \vdash P \quad \Gamma \vdash b_0 : P[()]}{\Gamma \vdash \mathbf{ind}_1(s; z. P; b_0) : P[s]} \\
\\
\text{SIGUNI} \frac{\Gamma \vdash A : \mathbf{Type}_i \quad \Gamma, x : A \vdash B : \mathbf{Type}_i}{\Gamma \vdash \Sigma x : A. B : \mathbf{Type}_i} \quad \text{PAIR} \frac{\Gamma \vdash t : A \quad \Gamma \vdash u : B[t]}{\Gamma \vdash (t, u)_{x.B} : \Sigma x : A. B} \\
\\
\text{PROJ}_1 \frac{\Gamma \vdash p : \Sigma x : A. B}{\Gamma \vdash \pi_1 p : A} \quad \text{PROJ}_2 \frac{\Gamma \vdash p : \Sigma x : A. B}{\Gamma \vdash \pi_2 p : B[u]} \\
\\
\text{TREEUNI} \frac{\Gamma \vdash A : \mathbf{Type}_i \quad \Gamma, x : A \vdash B : \mathbf{Type}_i}{\Gamma \vdash \mathbf{W} x : A. B : \mathbf{Type}_i} \quad \text{SUP} \frac{\Gamma, x : A \vdash B \quad \Gamma \vdash a : A \quad \Gamma \vdash k : B[a] \rightarrow \mathbf{W} x : A. B}{\Gamma \vdash \mathbf{sup}_{x.B} a k : \mathbf{W} x : A. B} \\
\\
\text{TREEIND} \frac{\Gamma \vdash A \quad \Gamma, x : A \vdash B \quad \Gamma \vdash s : \mathbf{W} x : A. B \quad \Gamma, z : \mathbf{W} x : A. B \vdash P \quad \Gamma, x : A, y : B[x] \rightarrow \mathbf{W} x : A. B, h : \Pi z : B[x]. P[y z] \vdash b : P[\mathbf{sup}_{x.B} x y]}{\Gamma \vdash \mathbf{ind}_{\mathbf{W} x : A. B}(s; z. P; x.y.z.b) : P[s]} \\
\\
\text{BOOLUNI} \frac{}{\Gamma \vdash \mathbf{B} : \mathbf{Type}_0} \quad \text{TRUE} \frac{}{\Gamma \vdash \mathbf{tt} : \mathbf{B}} \quad \text{FALSE} \frac{}{\Gamma \vdash \mathbf{ff} : \mathbf{B}} \\
\\
\text{BOOLIND} \frac{\Gamma \vdash s : \mathbf{B} \quad \Gamma, z : \mathbf{B} \vdash P \quad \Gamma \vdash b_{\mathbf{tt}} : P[\mathbf{tt}] \quad \Gamma \vdash b_{\mathbf{ff}} : P[\mathbf{ff}]}{\Gamma \vdash \mathbf{ind}_{\mathbf{B}}(s; z. P; b_{\mathbf{tt}}, b_{\mathbf{ff}}) : P[s]} \\
\\
\text{SUMUNI} \frac{\Gamma \vdash A : \mathbf{Type}_i \quad \Gamma \vdash B : \mathbf{Type}_i}{\Gamma \vdash A + B : \mathbf{Type}_i} \quad \text{SUMINJLEFT} \frac{\Gamma \vdash B \quad \Gamma \vdash a : A}{\Gamma \vdash \mathbf{inj}_B^l a : A + B}
\end{array}$$

$$\begin{array}{c}
 \text{SUMINJRIGHT} \frac{\Gamma \vdash A \quad \Gamma \vdash b : B}{\Gamma \vdash \text{inj}_A^r b : A + B} \\
 \\
 \text{SUMIND} \frac{\Gamma \vdash s : A + B \quad \Gamma, z : A + B \vdash P}{\Gamma, x : A \vdash b_l : P[\text{inj}_B^l x] \quad \Gamma, x : B \vdash b_r : P[\text{inj}_A^r x]}{\Gamma \vdash \text{ind}_+(s; z.P; x.b_l, x.b_r) : P[s]} \\
 \\
 \text{IDTY} \frac{\Gamma \vdash A \quad \Gamma \vdash a : A \quad \Gamma \vdash a' : A}{\Gamma \vdash \mathbf{Id}_A a a'} \qquad \text{REFLTM} \frac{\Gamma \vdash A \quad \Gamma \vdash a : A}{\Gamma \vdash \text{refl}_{A,a} : \mathbf{Id}_A a a} \\
 \\
 \text{IDIND} \frac{\Gamma \vdash s : \mathbf{Id}_A a a' \quad \Gamma \vdash A \quad \Gamma \vdash a : A \quad \Gamma \vdash a' : A}{\Gamma, x : A, y : A, z : \mathbf{Id}_A x y \vdash P \quad \Gamma, x : A \vdash b : P[\text{id}, x, x, \text{refl}_{A,x}]}{\Gamma \vdash \text{ind}_{\mathbf{Id}_A}(s; x.y.z.P; x.b) : P[\text{id}, a, a', s]}
 \end{array}$$

$\boxed{\Gamma \vdash T \cong T'}$ Types T and T' are convertible in context Γ

$$\begin{array}{c}
 \text{REFLTY} \frac{\Gamma \vdash A}{\Gamma \vdash A \cong A} \quad \text{TRANS TY} \frac{\Gamma \vdash A \cong B \quad \Gamma \vdash B \cong C}{\Gamma \vdash A \cong C} \quad \text{ELC} \frac{\Gamma \vdash A \cong A' : \text{Type}_i}{\Gamma \vdash A \cong A'} \\
 \\
 \text{FUNTYC} \frac{\Gamma \vdash A \cong A' \quad \Gamma, x : A \vdash B \cong B'}{\Gamma \vdash \Pi x : A.B \cong \Pi x : A'.B'} \quad \text{LISTTYC} \frac{\Gamma \vdash A \cong A'}{\Gamma \vdash \mathbf{List} A \cong \mathbf{List} A'} \\
 \\
 \text{SIGTYC} \frac{\Gamma \vdash A \cong A' \quad \Gamma, x : A \vdash B \cong B'}{\Gamma \vdash \Sigma x : A.B \cong \Sigma x : A'.B'} \quad \text{TREETYC} \frac{\Gamma \vdash A \cong A' \quad \Gamma, x : A \vdash B \cong B'}{\Gamma \vdash \mathbf{W} x : A.B \cong \mathbf{W} x : A'.B'} \\
 \\
 \text{IDTYC} \frac{\Gamma \vdash A \cong A' \quad \Gamma \vdash t \cong t' : A}{\Gamma \vdash u \cong u' : A} \quad \text{SUMTYC} \frac{\Gamma \vdash A \cong A' \quad \Gamma \vdash B \cong B'}{\Gamma \vdash A + B \cong A' + B'}
 \end{array}$$

$\boxed{\Gamma \vdash t \cong t' : T}$ Terms t and t' are convertible at type T in context Γ

$$\begin{array}{c}
 \text{REFL} \frac{\Gamma \vdash t : A}{\Gamma \vdash t \cong t : A} \quad \text{TRANS} \frac{\Gamma \vdash t \cong u : A \quad \Gamma \vdash u \cong v : A}{\Gamma \vdash t \cong v : A} \quad \text{CONV} \frac{\Gamma \vdash t \cong t' : A \quad \Gamma \vdash A \cong B}{\Gamma \vdash t \cong t' : B} \\
 \\
 \beta_{\text{FUN}} \frac{\Gamma \vdash A \quad \Gamma, x : A \vdash B}{\Gamma, x : A \vdash t : B \quad \Gamma \vdash u : A}{\Gamma \vdash (\lambda x : A.t) u \cong t[u] : B[u]} \quad \eta_{\text{FUN}} \frac{\Gamma, x : A \vdash f x \cong g x : B}{\Gamma \vdash f \cong g : \Pi x : A.B} \\
 \\
 \beta_{\text{SIG}_1} \frac{\Gamma \vdash A \quad \Gamma, x : A \vdash B}{\Gamma \vdash t : A \quad \Gamma \vdash u : B[t]}{\Gamma \vdash \pi_1(t, u)_{x.B} \cong t : A} \quad \beta_{\text{SIG}_2} \frac{\Gamma \vdash A \quad \Gamma, x : A \vdash B}{\Gamma \vdash t : A \quad \Gamma \vdash u : B[t]}{\Gamma \vdash \pi_2(t, u)_{x.B} \cong u : B[t]} \\
 \\
 \eta_{\text{SIG}} \frac{\Gamma \vdash A \quad \Gamma, x : A \vdash B \quad \Gamma \vdash p : \Sigma x : A.B}{\Gamma \vdash p \cong (\pi_1 p, \pi_2 p)_{x.B} : \Sigma x : A.B}
 \end{array}$$

$$\begin{array}{c}
\beta_{\text{NIL}} \frac{\Gamma \vdash A \quad \Gamma, x: \mathbf{List} A \vdash P \quad \Gamma \vdash b_\varepsilon : P[\varepsilon_A] \quad \Gamma, x: A, y: \mathbf{List} A, z: P[y] \vdash b_\varepsilon : P[x ::_A y]}{\Gamma \vdash \text{ind}_{\mathbf{List} A}(\varepsilon_A; z.P; b_\varepsilon, x.y.z.b_\varepsilon) \cong b_\varepsilon : P[\varepsilon_A]} \\
\beta_{\text{CONS}} \frac{\Gamma \vdash A \quad \Gamma \vdash a: A \quad \Gamma \vdash l: \mathbf{List} A \quad \Gamma, x: \mathbf{List} A \vdash P \quad \Gamma \vdash b_\varepsilon : P[\varepsilon_A] \quad \Gamma, x: A, y: \mathbf{List} A, z: P[y] \vdash b_\varepsilon : P[x ::_A y]}{\Gamma \vdash \text{ind}_{\mathbf{List} A}(a ::_A l; z.P; b_\varepsilon, x.y.z.b_\varepsilon) \cong b_\varepsilon[\text{id}, a, l, \text{ind}_{\mathbf{List} A}(l; z.P; b_\varepsilon, x.y.z.b_\varepsilon)] : P[a ::_A l]} \\
\beta_{\text{TRUE}} \frac{\Gamma, z: \mathbf{B} \vdash P \quad \Gamma \vdash b_{\text{tt}} : P[\text{tt}] \quad \Gamma \vdash b_{\text{ff}} : P[\text{ff}]}{\Gamma \vdash \text{ind}_{\mathbf{B}}(\text{tt}; z.P; b_{\text{tt}}, b_{\text{ff}}) \cong b_{\text{tt}} : P[\text{tt}]} \quad \beta_{\text{FALSE}} \frac{\Gamma, z: \mathbf{B} \vdash P \quad \Gamma \vdash b_{\text{tt}} : P[\text{tt}] \quad \Gamma \vdash b_{\text{ff}} : P[\text{ff}]}{\Gamma \vdash \text{ind}_{\mathbf{B}}(\text{ff}; z.P; b_{\text{tt}}, b_{\text{ff}}) \cong b_{\text{ff}} : P[\text{ff}]} \\
\beta_{\text{INJLEFT}} \frac{\Gamma \vdash A + B \quad \Gamma \vdash a: A \quad \Gamma, z: A + B \vdash P \quad \Gamma, x: A \vdash b_l : P[\text{inj}_B^l x] \quad \Gamma, x: B \vdash b_r : P[\text{inj}_A^r x]}{\Gamma \vdash \text{ind}_+(\text{inj}_B^l a; z.P; x.b_l, x.b_r) \cong b_l[a] : P[\text{inj}_B^l a]} \\
\beta_{\text{INJRIGHT}} \frac{\Gamma \vdash A + B \quad \Gamma \vdash b: B \quad \Gamma, z: A + B \vdash P \quad \Gamma, x: A \vdash b_l : P[\text{inj}_B^l x] \quad \Gamma, x: B \vdash b_r : P[\text{inj}_A^r x]}{\Gamma \vdash \text{ind}_+(\text{inj}_A^l b; z.P; x.b_l, x.b_r) \cong b_r[b] : P[\text{inj}_A^r b]} \\
\beta_{\text{TREE}} \frac{\Gamma, x: A \vdash B \quad \Gamma \vdash a: A \quad \Gamma \vdash k: B[a] \rightarrow \mathbf{W} x: A.B \quad \Gamma, z: \mathbf{W} x: A.B \vdash P \quad \Gamma, x: A, y: B[x] \rightarrow W x: A.B, h: \Pi z: B[x]. P[y z] \vdash b: P[\text{sup}_{x.B} x y]}{\Gamma \vdash \text{ind}_{\mathbf{W} x: A.B}(\text{sup}_{x.B} a k; z.P; x.y.z.b) \cong b[\text{id}, a, k, (\lambda z: B[x]. \text{ind}_{\mathbf{W} x: A.B}(k z; z.P; x.y.z.b))] : P[\text{sup}_{x.B} a k]} \\
\beta_{\text{REFL}} \frac{\Gamma \vdash A \quad \Gamma \vdash a: A \quad \Gamma, x: A, y: A, z: \mathbf{Id}_A x y \vdash P \quad \Gamma, x: A \vdash b: P[\text{id}, x, \text{refl}_{A,x}]}{\Gamma \vdash \text{ind}_{\mathbf{Id}_A}(\text{refl}_{A,a}; x.y.z.P; x.b) \cong b[a] : P[\text{id}, a, a, \text{refl}_{A,a}]} \\
\text{FUNCONG} \frac{\Gamma \vdash A \cong A' : \text{Type}_i \quad \Gamma, x: A \vdash B \cong B' : \text{Type}_i}{\Gamma \vdash \Pi x: A.B \cong \Pi x: A'.B' : \text{Type}_i} \quad \text{other congruences omitted}
\end{array}$$

B.2 Algorithmic MLTT

$$\boxed{t \rightsquigarrow^1 t'} \quad \text{Term } t \text{ weak-head reduces in one step to term } t'$$

$$\begin{array}{c}
\beta_{\text{FUN}} \frac{}{(\lambda x: A.t) u \rightsquigarrow^1 t[u]} \quad \beta_{\text{SIG}_1} \frac{}{\pi_1(t, u)_{x.B} \rightsquigarrow^1 t} \quad \beta_{\text{SIG}_2} \frac{}{\pi_2(t, u)_{x.B} \rightsquigarrow^1 u} \\
\beta_{\text{REDNIL}} \frac{}{\text{ind}_{\mathbf{List} A}(\varepsilon_A; x.P; b_\varepsilon, x.y.z.b_\varepsilon) \rightsquigarrow^1 b_\varepsilon} \\
\beta_{\text{REDCONS}} \frac{}{\text{ind}_{\mathbf{List} A}(a ::_A l; x.P; b_\varepsilon, x.y.z.b_\varepsilon) \rightsquigarrow^1 b_\varepsilon[\text{id}, a, l, \text{ind}_{\mathbf{List} A}(l; z.P; b_\varepsilon, x.y.z.b_\varepsilon)]} \\
\beta_{\text{TREE}} \frac{}{\text{ind}_{\mathbf{W} x: A.B}(\text{sup}_{x.B} a k; z.P; x.y.z.b) \rightsquigarrow^1 b[\text{id}, a, k, (\lambda z: B[x]. \text{ind}_{\mathbf{W} x: A.B}(k z; z.P; x.y.z.b))]}
\end{array}$$

Definitional Functoriality for Dependent (Sub)Types

$$\begin{array}{c}
\beta_{\text{TRUE}} \frac{}{\text{ind}_{\mathbf{B}}(\text{tt}; z.P; b_{\text{tt}}, b_{\text{ff}}) \rightsquigarrow^1 b_{\text{tt}}} \qquad \beta_{\text{FALSE}} \frac{}{\text{ind}_{\mathbf{B}}(\text{ff}; z.P; b_{\text{tt}}, b_{\text{ff}}) \rightsquigarrow^1 b_{\text{ff}}} \\
\beta_{\text{INJLEFT}} \frac{}{\text{ind}_{+}(\text{inj}^l a; z.P; x.b_l, x.b_r) \rightsquigarrow^1 b_l[a]} \\
\beta_{\text{INJRIGHT}} \frac{}{\text{ind}_{+}(\text{inj}^r b; z.P; x.b_l, x.b_r) \rightsquigarrow^1 b_r[b]} \qquad \beta_{\text{REFL}} \frac{}{\text{ind}_{\text{Id}_A}(\text{refl}_{A,a}; x.z.P; x.b) \rightsquigarrow^1 b[a]} \\
\text{REDAPP} \frac{t \rightsquigarrow^1 t'}{t u \rightsquigarrow^1 t' u} \qquad \text{REDSIG}_1 \frac{t \rightsquigarrow^1 t'}{\pi_1 t \rightsquigarrow^1 \pi_1 t'} \qquad \text{REDSIG}_2 \frac{t \rightsquigarrow^1 t'}{\pi_2 t \rightsquigarrow^1 \pi_2 t'} \\
\text{REDIND} \frac{t \rightsquigarrow^1 t'}{\text{ind}_T(t; P; \vec{b}) \rightsquigarrow^1 \text{ind}_T(t'; P; \vec{b})}
\end{array}$$

$\boxed{t \rightsquigarrow^* t'}$ Term t weak-head reduces in multiple steps to term t'

$$\text{REDBASE} \frac{}{t \rightsquigarrow^* t} \qquad \text{REDSTEP} \frac{t \rightsquigarrow^1 t' \quad t' \rightsquigarrow^* t''}{t \rightsquigarrow^* t''}$$

$\boxed{\text{nf } f}$ $\stackrel{\text{def}}{=} n \mid \text{Type}_i \mid \Pi x:t.t \mid \lambda x:t.t \mid \mathbf{List } t \mid \varepsilon_t \mid t ::_i t \mid \text{weak-head normal forms}$
 $\Sigma x:t.t \mid (t, t) \mid \mathbf{W } x:t.t \mid \text{sup}_t t t \mid \mathbf{0} \mid \mathbf{1} \mid () \mid$
 $\mathbf{B} \mid \text{tt} \mid \text{ff} \mid \mathbf{Id}_t t t' \mid \text{refl}_{t,t} \mid t + t \mid \text{inj}_t^l t \mid \text{inj}_t^r t$

$\boxed{\text{ne } n}$ $\stackrel{\text{def}}{=} x \mid n t \mid \text{ind}_T(n; t; t) \mid \pi_1 n \mid \pi_2 n$ weak-head neutrals

$\boxed{\Gamma \vdash T \triangleleft}$ T is a type in Γ

$$\begin{array}{c}
\text{FUNTY} \frac{\Gamma \vdash A \triangleleft \quad \Gamma, x: A \vdash B \triangleleft}{\Gamma \vdash \Pi x: A. B \triangleleft} \qquad \text{LISTTY} \frac{\Gamma \vdash A \triangleleft}{\Gamma \vdash \mathbf{List } A \triangleleft} \\
\text{SIGTY} \frac{\Gamma \vdash A \triangleleft \quad \Gamma, x: A \vdash B \triangleleft}{\Gamma \vdash \Sigma x: A. B \triangleleft} \qquad \text{TREETY} \frac{\Gamma \vdash A \triangleleft \quad \Gamma, x: A \vdash B \triangleleft}{\Gamma \vdash \mathbf{W } x: A. B \triangleleft} \qquad \text{EMPTYTY} \frac{}{\Gamma \vdash \mathbf{0} \triangleleft} \\
\text{UNITTY} \frac{}{\Gamma \vdash \mathbf{1} \triangleleft} \qquad \text{BOOLTY} \frac{}{\Gamma \vdash \mathbf{B} \triangleleft} \qquad \text{IDTY} \frac{\Gamma \vdash A \triangleleft \quad \Gamma \vdash a \triangleleft A \quad \Gamma \vdash a' \triangleleft A}{\Gamma \vdash \mathbf{Id}_A a a' \triangleleft} \\
\text{SUMTY} \frac{\Gamma \vdash A \triangleleft \quad \Gamma \vdash B \triangleleft}{\Gamma \vdash A + B \triangleleft} \qquad \text{EL} \frac{\Gamma \vdash A \triangleright_h \text{Type}_i \quad A \text{ is not a canonical form}}{\Gamma \vdash A \triangleleft}
\end{array}$$

$\boxed{\Gamma \vdash t \triangleright T}$ Term t infers type T in context Γ

$$\begin{array}{c}
\text{SORT} \frac{}{\Gamma \vdash \text{Type}_i \triangleright \text{Type}_{i+1}} \qquad \text{VAR} \frac{(x:T) \in \Gamma}{\Gamma \vdash x \triangleright T} \qquad \text{FUN} \frac{\Gamma \vdash A \triangleright_h \text{Type}_i}{\Gamma \vdash \Pi x: A. B \triangleright \text{Type}_i}
\end{array}$$

$$\begin{array}{c}
\text{ABS} \frac{\Gamma \vdash A \triangleleft \quad \Gamma, x: A \vdash t \triangleright B}{\Gamma \vdash \lambda x: A. t \triangleright \Pi x: A. B} \quad \text{APP} \frac{\Gamma \vdash t \triangleright_{\text{h}} \Pi x: A. B \quad \Gamma \vdash u \triangleleft A}{\Gamma \vdash t u \triangleright B[u]} \\
\\
\text{LIST} \frac{\Gamma \vdash A \triangleright_{\text{h}} \text{Type}_i}{\Gamma \vdash \mathbf{List} A \triangleright \text{Type}_i} \quad \text{NIL} \frac{\Gamma \vdash A \triangleleft}{\Gamma \vdash \varepsilon_A \triangleright \mathbf{List} A} \quad \text{CONS} \frac{\Gamma \vdash A \triangleleft \quad \Gamma \vdash a \triangleleft A \quad \Gamma \vdash l \triangleleft \mathbf{List} A}{\Gamma \vdash a ::_A l \triangleright \mathbf{List} A} \\
\\
\text{LISTIND} \frac{\Gamma, x: \mathbf{List} A \vdash P \triangleright \quad \Gamma \vdash A \triangleleft \quad \Gamma \vdash s \triangleleft \mathbf{List} A \quad \Gamma \vdash b_\varepsilon \triangleleft P[\varepsilon_A] \quad \Gamma, x: A, y: \mathbf{List} A, z: P[y] \vdash b_{::} \triangleleft P[x ::_A y]}{\Gamma \vdash \text{ind}_{\mathbf{List} A}(s; z.P; b_\varepsilon, x.y.z.b_{::}) \triangleright P[s]} \\
\\
\text{EMPTY} \frac{}{\Gamma \vdash \mathbf{0} \triangleright \text{Type}_0} \quad \text{EMPTYIND} \frac{\Gamma \vdash s \triangleleft \mathbf{0} \quad \Gamma \vdash P \triangleleft}{\Gamma \vdash \text{ind}_0(s; P) \triangleright P} \quad \text{UNITUNI} \frac{}{\Gamma \vdash \mathbf{1} \triangleright \text{Type}_0} \\
\\
\text{UNITTM} \frac{}{\Gamma \vdash () \triangleright \mathbf{1}} \quad \text{UNITIND} \frac{\Gamma \vdash s \triangleleft \mathbf{1} \quad \Gamma, z: \mathbf{1} \vdash P \triangleleft \quad \Gamma \vdash b_() \triangleleft P[()]}{\Gamma \vdash \text{ind}_1(s; z.P; b_()) \triangleright P[s]} \\
\\
\text{SIG} \frac{\Gamma \vdash A \triangleright_{\text{h}} \text{Type}_i \quad \Gamma, x: A \vdash B \triangleleft \text{Type}_i}{\Gamma \vdash \Sigma x: A. B \triangleright \text{Type}_i} \quad \text{PAIR} \frac{\Gamma \vdash t \triangleright A \quad \Gamma, x: A \vdash B \triangleleft}{\Gamma \vdash u \triangleleft B[t]} \\
\frac{}{\Gamma \vdash (t, u)_{x.B} \triangleright \Sigma x: A. B} \\
\\
\text{PROJ}_1 \frac{\Gamma \vdash p \triangleright_{\text{h}} \Sigma x: A. B}{\Gamma \vdash \pi_1 p \triangleright A} \quad \text{PROJ}_2 \frac{\Gamma \vdash p \triangleright \Sigma x: A. B}{\Gamma \vdash \pi_2 p \triangleright B[u]} \\
\\
\text{TREE} \frac{\Gamma \vdash A \triangleright_{\text{h}} \text{Type}_i \quad \Gamma, x: A \vdash B \triangleleft \text{Type}_i}{\Gamma \vdash \mathbf{W} x: A. B \triangleleft \text{Type}_i} \\
\\
\text{SUP} \frac{\Gamma \vdash a \triangleright A \quad \Gamma, x: A \vdash B \triangleleft \quad \Gamma \vdash k \triangleleft B[a] \rightarrow \mathbf{W} x: A. B}{\Gamma \vdash \text{sup}_{x.B} a k \triangleright \mathbf{W} x: A. B} \\
\\
\text{TREEIND} \frac{\Gamma \vdash A \triangleleft \quad \Gamma, x: A \vdash B \triangleleft \quad \Gamma \vdash s \triangleleft \mathbf{W} x: A. B \quad \Gamma, z: \mathbf{W} x: A. B \vdash P \triangleleft}{\Gamma, x: A, y: B[x] \rightarrow W x: A. B, h: \Pi z: B[x]. P[y z] \vdash b \triangleleft P[\text{sup}_{x.B} x y]}{\Gamma \vdash \text{ind}_{\mathbf{W} x: A. B}(s; z.P; x.y.z.b) \triangleright P[s]} \\
\\
\text{BOOLUNI} \frac{}{\Gamma \vdash \mathbf{B} \triangleright \text{Type}_0} \quad \text{TRUE} \frac{}{\Gamma \vdash \text{tt} \triangleright \mathbf{B}} \quad \text{FALSE} \frac{}{\Gamma \vdash \text{ff} \triangleright \mathbf{B}} \\
\\
\text{BOOLIND} \frac{\Gamma \vdash s \triangleleft \mathbf{B} \quad \Gamma, z: \mathbf{B} \vdash P \triangleleft \quad \Gamma \vdash b_{\text{tt}} \triangleleft P[\text{tt}] \quad \Gamma \vdash b_{\text{ff}} \triangleleft P[\text{ff}]}{\Gamma \vdash \text{ind}_{\mathbf{B}}(s; z.P; b_{\text{tt}}, b_{\text{ff}}) \triangleright P[s]} \\
\\
\text{SUM} \frac{\Gamma \vdash A \triangleright_{\text{h}} \text{Type}_i \quad \Gamma \vdash B \triangleleft \text{Type}_i}{\Gamma \vdash A + B \triangleright \text{Type}_i} \quad \text{SUMINJLEFT} \frac{\Gamma \vdash B \triangleleft \quad \Gamma \vdash a \triangleright A}{\Gamma \vdash \text{inj}_B^l a \triangleright A + B} \\
\\
\text{SUMINJRIGHT} \frac{\Gamma \vdash A \triangleleft \quad \Gamma \vdash b \triangleright B}{\Gamma \vdash \text{inj}_A^r b \triangleright A + B}
\end{array}$$

Definitional Functoriality for Dependent (Sub)Types

$$\text{SUMIND} \frac{\Gamma \vdash A \triangleleft \quad \Gamma \vdash B \triangleleft \quad \Gamma \vdash s \triangleleft A + B \quad \Gamma, z: A + B \vdash P \triangleleft \quad \Gamma, x: A \vdash b_l \triangleleft P[\text{inj}_B^l x] \quad \Gamma, x: B \vdash b_r \triangleleft P[\text{inj}_A^r x]}{\Gamma \vdash \text{ind}_{A+B}(s; z.P; x.b_l, x.b_r) \triangleright P[s]}$$

$$\text{ID} \frac{\Gamma \vdash A \triangleright_h \text{Type}_i \quad \Gamma \vdash a \triangleleft A \quad \Gamma \vdash a' \triangleleft A}{\Gamma \vdash \mathbf{Id}_A a a' \triangleright \text{Type}_i} \quad \text{REFLTM} \frac{\Gamma \vdash A \triangleleft \quad \Gamma \vdash a \triangleleft A}{\Gamma \vdash \text{refl}_{A,a} \triangleright \mathbf{Id}_A a a}$$

$$\text{IDIND} \frac{\Gamma \vdash s \triangleright_h \mathbf{Id}_{A'} a a' \quad \Gamma, x, y: A, z: \mathbf{Id}_A x y \vdash P \triangleleft \quad \Gamma \vdash A \triangleleft \quad \Gamma, x: A \vdash b \triangleleft P[\text{id}, x, x, \text{refl}_{A,x}]}{\Gamma \vdash \text{ind}_{\mathbf{Id}_A}(s; x.y.z.P; x.b) \triangleright P[\text{id}, a, a', s]}$$

$\boxed{\Gamma \vdash t \triangleleft T}$ Term t checks against type T

$$\text{CHECK} \frac{\Gamma \vdash t \triangleright T' \quad \Gamma \vdash T' \cong T \triangleleft}{\Gamma \vdash t \triangleleft T}$$

$\boxed{\Gamma \vdash t \triangleright_h T}$ Term t infers the reduced type T

$$\text{INFRED} \frac{\Gamma \vdash t \triangleright T \quad \Gamma \vdash T \rightsquigarrow^* T'}{\Gamma \vdash t \triangleright_h T'}$$

$\boxed{\Gamma \vdash T \cong T' \triangleleft}$ Types T and T' are convertible

$$\text{TYRED} \frac{T \rightsquigarrow^* U \quad T' \rightsquigarrow^* U' \quad \Gamma \vdash U \cong_h U' \triangleleft}{\Gamma \vdash T \cong T' \triangleleft}$$

$\boxed{\Gamma \vdash t \cong t' \triangleleft A}$ Terms t and t' are convertible at type T

$$\text{TMRED} \frac{t \rightsquigarrow^* u \quad t' \rightsquigarrow^* u' \quad T \rightsquigarrow^* U \quad \Gamma \vdash u \cong_h u' \triangleleft U}{\Gamma \vdash t \cong t' \triangleleft T}$$

$\boxed{\Gamma \vdash T \cong_h T' \triangleleft}$ Reduced types T and T' are convertible

$$\text{CUNITY} \frac{}{\Gamma \vdash \text{Type}_i \cong_h \text{Type}_i \triangleleft}$$

$$\text{CPRODTY} \frac{\Gamma \vdash A \cong A' \triangleleft \quad \Gamma, x: A' \vdash B \cong B' \triangleleft}{\Gamma \vdash \Pi x: A.B \cong_h \Pi x: A'.B' \triangleleft}$$

$$\text{CLISTTY} \frac{\Gamma \vdash A \cong A' \triangleleft}{\Gamma \vdash \mathbf{List} A \cong_h \mathbf{List} A' \triangleleft}$$

$$\text{CSIGTY} \frac{\Gamma \vdash A \cong A' \triangleleft \quad \Gamma, x: A \vdash B \cong B' \triangleleft}{\Gamma \vdash \Sigma x: A.B \cong_h \Sigma x: A'.B' \triangleleft}$$

$$\text{CTREETY} \frac{\Gamma \vdash A \cong A' \triangleleft \quad \Gamma, x: A \vdash B \cong B' \triangleleft}{\Gamma \vdash \mathbf{W} x: A.B \cong_h \mathbf{W} x: A'.B' \triangleleft}$$

$$\text{CIdTY} \frac{\Gamma \vdash A \cong A' \triangleleft \quad \Gamma \vdash t \cong t' \triangleleft A \quad \Gamma \vdash u \cong u' \triangleleft A}{\Gamma \vdash \mathbf{Id}_A t u \cong_h \mathbf{Id}_{A'} t' u' \triangleleft}$$

$$\text{CSUMTY} \frac{\Gamma \vdash A \cong A' \triangleleft \quad \Gamma \vdash B \cong B' \triangleleft}{\Gamma \vdash A + B \cong A' + B' \triangleleft} \quad \text{CREFLTY} \frac{T \text{ is } \mathbf{0}, \mathbf{1} \text{ or } \mathbf{B}}{\Gamma \vdash T \cong_h T \triangleleft}$$

$$\text{NEUTY} \frac{\Gamma \vdash n \approx n' \triangleright T}{\Gamma \vdash n \cong_h n' \triangleleft}$$

$\boxed{\Gamma \vdash t \cong_h t' \triangleleft A}$ Reduced terms t and t' are convertible at type A

$$\text{CUNI} \frac{}{\Gamma \vdash \text{Type}_i \cong_h \text{Type}_j \triangleleft \text{Type}_k} \quad \text{CFUN} \frac{\Gamma \vdash A \cong A' \triangleleft \text{Type}_i \quad \Gamma, x: A' \vdash B \cong B' \triangleleft \text{Type}_i}{\Gamma \vdash \Pi x: A.B \cong_h \Pi x: A'.B' \triangleleft \text{Type}_i}$$

$$\text{CFUNETA} \frac{\Gamma, x: A \vdash f x \cong f' x \triangleleft B}{\Gamma \vdash f \cong_h f' \triangleleft \Pi x: A.B} \quad \text{CSIG} \frac{\Gamma \vdash A \cong A' \triangleleft \text{Type}_i \quad \Gamma, x: A' \vdash B \cong B' \triangleleft \text{Type}_i}{\Gamma \vdash \Sigma x: A.B \cong_h \Sigma x: A'.B' \triangleleft \text{Type}_i}$$

$$\text{CSIGETA} \frac{\Gamma \vdash \pi_1 p \cong \pi_1 p' \triangleleft A \quad \Gamma \vdash \pi_2 p \cong \pi_2 p' \triangleleft B[\pi_1 p]}{\Gamma \vdash p \cong_h p' \triangleleft \Sigma x: A.B} \quad \text{CLIST} \frac{\Gamma \vdash A \cong A' \triangleleft \text{Type}_i}{\Gamma \vdash \mathbf{List} A \cong_h \mathbf{List} A' \triangleleft \text{Type}_i}$$

$$\text{CNIL} \frac{}{\Gamma \vdash \varepsilon_A \cong_h \varepsilon_{A'} \triangleleft \mathbf{List} A''} \quad \text{CCONS} \frac{\Gamma \vdash a \cong a' \triangleleft A'' \quad \Gamma \vdash l \cong l' \triangleleft \mathbf{List} A''}{\Gamma \vdash a ::_A l \cong_h a' ::_{A'} l' \triangleleft \mathbf{List} A''}$$

$$\text{CREFLUNI} \frac{T \text{ is } \mathbf{0}, \mathbf{1} \text{ or } \mathbf{B}}{\Gamma \vdash T \cong_h T \triangleleft \text{Type}_0} \quad \text{CUNITK} \frac{}{\Gamma \vdash () \cong_h () \triangleleft \mathbf{1}} \quad \text{CREFLBOOL} \frac{t \text{ is tt or ff}}{\Gamma \vdash t \cong_h t \triangleleft \mathbf{B}}$$

$$\text{CTREE} \frac{\Gamma \vdash A \cong A' \triangleleft \text{Type}_i \quad \Gamma, x: A' \vdash B \cong B' \triangleleft \text{Type}_i}{\Gamma \vdash \mathbf{W} x: A.B \cong_h \mathbf{W} x: A'.B' \triangleleft \text{Type}_i}$$

$$\text{CSUP} \frac{\Gamma \vdash a \cong a' \triangleleft A'' \quad \Gamma \vdash k \cong k' \triangleleft B''[a] \rightarrow \mathbf{W} x: A''.B''}{\Gamma \vdash \sup_{x.B} a k \cong_h \sup_{x.B'} a' k' \triangleleft \mathbf{W} x: A''.B''}$$

$$\text{CSUM} \frac{\Gamma \vdash A \cong A' \triangleleft \text{Type}_i \quad \Gamma \vdash B \cong B' \triangleleft \text{Type}_i}{\Gamma \vdash A + B \cong A' + B' \triangleleft \text{Type}_i}$$

$$\text{CInjLEFT} \frac{\Gamma \vdash B \cong B' \triangleleft \quad \Gamma \vdash a \cong a' \triangleleft A}{\Gamma \vdash \text{inj}_B^l a \cong \text{inj}_{B'}^l a' \triangleleft A + B} \quad \text{CInjRIGHT} \frac{\Gamma \vdash A \cong A' \triangleleft \quad \Gamma \vdash b \cong b' \triangleleft B}{\Gamma \vdash \text{inj}_A^r b \cong \text{inj}_{A'}^r b' \triangleleft A + B}$$

$$\text{REFLREFL} \frac{}{\Gamma \vdash \text{refl}_{A,a} \cong \text{refl}_{A',a'} \triangleleft \mathbf{Id}_{A'} tu} \quad \text{NEU NEU} \frac{\Gamma \vdash n \approx n' \triangleright S \quad \text{ne } M}{\Gamma \vdash n \cong_h n' \triangleleft M}$$

$$\text{NEUPOS} \frac{\Gamma \vdash n \approx n' \triangleright S \quad T \text{ is } \text{Type}_i, \mathbf{0}, \mathbf{1}, \mathbf{B}, \mathbf{List} A, \mathbf{W} x: A.B \text{ or } \mathbf{Id}_A a a'}{\Gamma \vdash n \cong_h n' \triangleleft T}$$

$\boxed{\Gamma \vdash t \approx_h t' \triangleright T}$ Neutrals t and t' are comparable, inferring the reduced type T

$$\text{NRED} \frac{\Gamma \vdash n \approx n' \triangleright T \quad T \rightsquigarrow^* S}{\Gamma \vdash n \approx_h n' \triangleright S}$$

$\boxed{\Gamma \vdash t \approx t' \triangleright T}$ Neutrals t and t' are comparable, inferring the type T

$$\begin{array}{c} \text{NVAR} \frac{(x:T) \in \Gamma}{\Gamma \vdash x \approx x \triangleright T} \quad \text{NAPP} \frac{\Gamma \vdash n \approx_h n' \triangleright \Pi x: A. B \quad \Gamma \vdash u \cong u' \triangleleft A}{\Gamma \vdash n u \approx n' u' \triangleright B[u]} \\ \\ \text{NLISTIND} \frac{\Gamma \vdash A \cong A' \triangleleft \quad \Gamma \vdash s \approx s' \triangleright S \quad \Gamma, z: \mathbf{List} A \vdash P \cong P' \triangleleft \quad \Gamma \vdash b_\varepsilon \cong b'_\varepsilon \triangleleft P[\varepsilon_A] \quad \Gamma, x: A, y: \mathbf{List} A, z: P[y] \vdash b_{\varepsilon'} \cong b'_{\varepsilon'} \triangleleft P[x ::_A y]}{\Gamma \vdash \text{ind}_{\mathbf{List} A}(s; z.P; b_\varepsilon, x.y.z.b_{\varepsilon'}) \approx \text{ind}_{\mathbf{List} A'}(s'; z.P'; b'_\varepsilon, x.y.z.b'_{\varepsilon'}) \triangleright P[s]} \\ \\ \text{NEMPTYIND} \frac{\Gamma \vdash s \approx_h s' \triangleright \mathbf{0} \quad \Gamma \vdash P \cong P' \triangleleft}{\Gamma \vdash \text{ind}_{\mathbf{0}}(s; P) \approx \text{ind}_{\mathbf{0}}(s'; P') \triangleright P} \\ \\ \text{NUNITIND} \frac{\Gamma \vdash s \approx_h s' \triangleright \mathbf{1} \quad \Gamma, z: \mathbf{1} \vdash P \cong P' \triangleleft \quad \Gamma \vdash b \cong b' \triangleleft P[()]}{\Gamma \vdash \text{ind}_{\mathbf{1}}(s; z.P; b) \approx \text{ind}_{\mathbf{0}}(s'; z.P'; b') \triangleright P[s]} \\ \\ \text{NSIG}_1 \frac{\Gamma \vdash n \approx_h n' \triangleright \Sigma x: A. B}{\Gamma \vdash \pi_1 n \approx \pi_1 n' \triangleright A} \quad \text{NSIG}_2 \frac{\Gamma \vdash n \approx_h n' \triangleright \Sigma x: A. B}{\Gamma \vdash \pi_2 n \approx \pi_2 n' \triangleright B[\pi_1 n]} \\ \\ \text{NTREEIND} \frac{\Gamma \vdash A \cong A' \triangleleft \quad \Gamma, x: A \vdash B \cong B' \triangleleft \quad \Gamma \vdash s \approx s' \triangleright S \quad \Gamma, z: \mathbf{W} x: A. B \vdash P \cong P' \triangleleft \quad \Gamma, x: A, y: B[x] \rightarrow W x: A. B, h: \Pi z: B[x]. P[yz] \vdash b \cong b' \triangleleft P[\text{sup}_{x.B} x y]}{\Gamma \vdash \text{ind}_{\mathbf{W} x: A. B}(s; z.P; x.y.z.b) \approx \text{ind}_{\mathbf{W} x: A'. B'}(s'; z.P'; x.y.z.b') \triangleright P[s]} \\ \\ \text{NBOOLIND} \frac{\Gamma \vdash s \approx_h s' \triangleright S \quad \Gamma, z: \mathbf{B} \vdash P \cong P' \triangleleft \quad \Gamma \vdash b_{\text{tt}} \cong b'_{\text{tt}} \triangleleft P[\text{tt}] \quad \Gamma \vdash b_{\text{ff}} \cong b'_{\text{ff}} \triangleleft P[\text{ff}]}{\Gamma \vdash \text{ind}_{\mathbf{B}}(s; z.P; b_{\text{tt}}, b_{\text{ff}}) \approx \text{ind}_{\mathbf{B}}(s'; z.P'; b'_{\text{tt}}, b'_{\text{ff}}) \triangleright P[s]} \\ \\ \text{NSUMIND} \frac{\Gamma \vdash A \cong A' \triangleleft \quad \Gamma \vdash B \cong B' \triangleleft \quad \Gamma \vdash s \approx_h s' \triangleright S \quad \Gamma, z: A + B \vdash P \cong P' \triangleleft \quad \Gamma, x: A \vdash b_l \cong b'_l \triangleleft P[\text{inj}^l x] \quad \Gamma, x: B \vdash b_r \cong b'_r \triangleleft P[\text{inj}^r x]}{\Gamma \vdash \text{ind}_{A+B}(s; z.P; x.b_l, x.b_r) \approx \text{ind}_{A'+B'}(s'; z.P'; x.b'_l, x.b'_r) \triangleright P[s]} \\ \\ \text{NIDIND} \frac{\Gamma \vdash A \cong A' \triangleleft \quad \Gamma \vdash s \approx_h s' \triangleright \mathbf{Id}_{A'} a a' \quad \Gamma, x: A, y: A, z: \mathbf{Id}_A x y \vdash P \cong P' \triangleleft \quad \Gamma, x: A \vdash b \cong b' \triangleleft P[\text{id}, x, x, \text{refl}_{A,x}]}{\Gamma \vdash \text{ind}_{\mathbf{Id}_A}(s; x.y.z.P; x.b) \approx \text{ind}_{\mathbf{Id}_{A'}}(s'; x.y.z.P'; x.b') \triangleright P[\text{id}, a, a', s]}$$

B.3 Declarative MLTT_{map}

Extend the rules of Appendix B.1. In rule **MAPCOMP**, we rely on conversion at domain and morphism types for a type former F . These are obtained from the judgements of Fig. 8 by replacing every typing judgment by a conversion judgement, and forgetting conversions. For instance, for Π types, we have

$$\begin{aligned} \Delta \vdash_{\text{map}} (A, B) \cong (A', B') : \text{dom}(\Pi) &\iff \Delta \vdash_{\text{map}} A \cong A' \wedge \Delta, x: A \vdash_{\text{map}} B \cong B' \\ \Delta \vdash_{\text{map}} (f, g) \cong (f', g') : \text{hom}_{\Pi}((A_1, B_1), (A_2, B_2)) &\iff \Delta \vdash_{\text{map}} f \cong f' : A_2 \rightarrow A_1 \wedge \\ &\quad \Delta, x: A_2 \vdash_{\text{map}} g \cong g' : B_1[f x] \rightarrow B_2 \end{aligned}$$

For each type former F ($\Pi, \Sigma, \text{List}, \mathbf{W}, \text{Id}, +$)

$$\begin{array}{c} \text{MAP} \frac{\Gamma \vdash_{\text{map}} X, Y : \text{dom}(F) \quad \Gamma \vdash_{\text{map}} f : \text{hom}_F(X, Y)}{\Gamma \vdash_{\text{map}} \text{map}_F f : F X \rightarrow F Y} \qquad \text{MAPID} \frac{\Gamma \vdash_{\text{map}} X : \text{dom}(F) \quad \Gamma \vdash_{\text{map}} t : F X}{\Gamma \vdash_{\text{map}} \text{map}_F \text{id}_X^F t \cong t : F X} \end{array}$$

$$\text{MAPCOMP} \frac{\Gamma \vdash_{\text{map}} X, Y, Z : \text{dom}(F) \quad \Gamma \vdash_{\text{map}} g : \text{hom}_F(X, Y) \quad \Gamma \vdash_{\text{map}} f : \text{hom}_F(Y, Z) \quad \Gamma \vdash_{\text{map}} t : F X}{\Gamma \vdash_{\text{map}} \text{map}_F f(\text{map}_F g t) \cong \text{map}_F(f \circ g) t : F Z}$$

$$\text{MAPCONG} \frac{\Gamma \vdash_{\text{map}} X \cong X' : \text{dom}(F) \quad \Gamma \vdash_{\text{map}} Y \cong Y' : \text{dom}(F) \quad \Gamma \vdash_{\text{map}} f : \text{hom}_F(X, Y) \quad \Gamma \vdash_{\text{map}} f' : \text{hom}_F(X', Y') \quad \Gamma \vdash_{\text{map}} f \cong f' : \text{hom}_F(X, Y)}{\Gamma \vdash_{\text{map}} \text{map}_F f \cong \text{map}_F f' : F X \rightarrow F Y}$$

$\Gamma \vdash_{\text{map}} t \cong u : A$

$$\text{MAPFUN} \frac{\Gamma \vdash_{\text{map}} (f, g) : \text{hom}_{\Pi}((A, B), (A', B')) \quad \Gamma \vdash_{\text{map}} h : \Pi x: A. B \quad \Gamma \vdash_{\text{map}} a' : A'}{\Gamma \vdash_{\text{map}} \text{map}_{\Pi}(f, g) h a' \cong g(h(f a')) : B'[a']}$$

$$\text{MAPSIG}_1 \frac{\Gamma \vdash_{\text{map}} (f, g) : \text{hom}_{\Sigma}((A, B), (A', B')) \quad \Gamma \vdash_{\text{map}} p : \Sigma x: A. B}{\Gamma \vdash_{\text{map}} \pi_1(\text{map}_{\Sigma}(f, g) p) \cong f(\pi_1 p) : A'}$$

$$\text{MAPSIG}_2 \frac{\Gamma \vdash_{\text{map}} (f, g) : \text{hom}_{\Sigma}((A, B), (A', B')) \quad \Gamma \vdash_{\text{map}} p : \Sigma x: A. B}{\Gamma \vdash_{\text{map}} \pi_2(\text{map}_{\Sigma}(f, g) p) \cong g(\pi_2 p) : B'[f(\pi_1 p)]}$$

$$\text{MAPLISTNIL} \frac{\Gamma \vdash_{\text{map}} f : \text{hom}_{\text{List}}(A, A')}{\Gamma \vdash_{\text{map}} \text{map}_{\text{List}} f \varepsilon_A \cong \varepsilon_{A'} : \text{List } A'}$$

$$\text{MAPLISTCONS} \frac{\Gamma \vdash_{\text{map}} f : \text{hom}_{\text{List}}(A, A') \quad \Gamma \vdash_{\text{map}} hd : A \quad \Gamma \vdash_{\text{map}} tl : \text{List } A}{\Gamma \vdash_{\text{map}} \text{map}_{\text{List}} f(hd ::_A tl) \cong (f hd) ::_{A'}(\text{map}_{\text{List}} f tl) : \text{List } A'}$$

$$\text{MAPW} \frac{\Gamma \vdash_{\text{map}} (f, g) : \text{hom}_{\mathbf{W}}((A, B), (A', B')) \quad \Gamma \vdash_{\text{map}} a : A \quad \Gamma \vdash_{\text{map}} k : B a \rightarrow \mathbf{W} x: A. B}{\Gamma \vdash_{\text{map}} \text{map}_{\mathbf{W}}(f, g)(\text{sup}_{x.B} a k) \cong \text{sup}_{x.B'}(f a)(\lambda x: B'. [f a]. \text{map}_{\mathbf{W}}(f, g)(k(g x))) : \mathbf{W} x: A'. B'}$$

Definitional Functoriality for Dependent (Sub)Types

$$\begin{array}{c} \text{MAPID} \frac{\Gamma \vdash_{\text{map}} f : \text{hom}_{\text{Id}}(A, A') \quad \Gamma \vdash_{\text{map}} a : A}{\Gamma \vdash_{\text{map}} \text{map}_{\text{Id}} f \text{ refl}_{A,a} \cong \text{refl}_{A',fa} : \text{Id } A' (fa) (fa)} \\ \text{MAPSUMLEFT} \frac{\Gamma \vdash_{\text{map}} (f, g) : \text{hom}_+((A, B), (A', B')) \quad \Gamma \vdash_{\text{map}} a : A}{\Gamma \vdash_{\text{map}} \text{map}_+(f, g) (\text{inj}^l a) \cong \text{inj}^l (fa) : A' + B'} \\ \text{MAPSUMRIGHT} \frac{\Gamma \vdash_{\text{map}} (f, g) : \text{hom}_+((A, B), (A', B')) \quad \Gamma \vdash_{\text{map}} b : B}{\Gamma \vdash_{\text{map}} \text{map}_+(f, g) (\text{inj}^r b) \cong \text{inj}^r (gb) : A' + B'} \end{array}$$

B.4 Algorithmic MLTT_{map}

Extends Appendix B.2. Replaces the rules already named with the same name in Appendix B.2.

$$\boxed{\Gamma \vdash_{\text{map}} t \cong_h t' \triangleleft T}$$

$$\text{NEUPOSMAP} \frac{\Gamma \vdash_{\text{map}} n \approx_{\text{map}} n' \triangleleft T \quad T \text{ is Type}_i, \mathbf{List } A, \mathbf{W } x : A.B, A + B \text{ or } \text{Id}_A a a'}{\Gamma \vdash_{\text{map}} n \cong_h n' \triangleleft T}$$

$$\boxed{\Gamma \vdash_{\text{map}} n \approx n' \triangleright T}$$

$$\text{NLISTIND} \frac{\Gamma \vdash_{\text{map}} A \cong A' \triangleleft \quad \Gamma \vdash_{\text{map}} s \approx_{\text{map}} s' \triangleleft \mathbf{List } A \quad \Gamma, z : \mathbf{List } A \vdash_{\text{map}} P \cong P' \triangleleft \quad \Gamma \vdash_{\text{map}} b_\varepsilon \cong b'_\varepsilon \triangleleft P[\varepsilon_A] \quad \Gamma, x : A, y : \mathbf{List } A, z : P[y] \vdash_{\text{map}} b_{\cdot} \cong b'_{\cdot} \triangleleft P[x \cdot_A y]}{\Gamma \vdash_{\text{map}} \text{ind}_{\mathbf{List } A}(s; z.P; b_\varepsilon, x.y.z.b_{\cdot}) \approx \text{ind}_{\mathbf{List } A'}(s'; z.P'; b'_\varepsilon, x.y.z.b'_{\cdot}) \triangleright P[s]}$$

$$\text{NTREEIND} \frac{\Gamma \vdash_{\text{map}} A \cong A' \triangleleft \quad \Gamma, x : A \vdash_{\text{map}} B \cong B' \triangleleft \quad \Gamma \vdash_{\text{map}} s \approx_{\text{map}} s' \triangleleft \mathbf{W } x : A.B \quad \Gamma, z : \mathbf{W } x : A.B \vdash_{\text{map}} P \cong P' \triangleleft \quad \Gamma, x : A, y : B[x] \rightarrow \mathbf{W } x : A.B, h : \Pi z : B[x].P[y z] \vdash_{\text{map}} b \cong b' \triangleleft P[\text{sup}_{x.B} x y]}{\Gamma \vdash_{\text{map}} \text{ind}_{\mathbf{W } x : A.B}(s; z.P; x.y.z.b) \approx \text{ind}_{\mathbf{W } x : A'.B'}(s'; z.P'; x.y.z.b') \triangleright P[s]}$$

$$\text{NSUMIND} \frac{\Gamma \vdash_{\text{map}} A \cong A' \triangleleft \quad \Gamma \vdash_{\text{map}} B \cong B' \triangleleft \quad \Gamma \vdash_{\text{map}} s \approx_{\text{map}} s' \triangleleft A + B \quad \Gamma, z : A + B \vdash_{\text{map}} P \cong P' \triangleleft \quad \Gamma, x : A \vdash_{\text{map}} b_l \cong b'_l \triangleleft P[\text{inj}^l x] \quad \Gamma, x : B \vdash_{\text{map}} b_r \cong b'_r \triangleleft P[\text{inj}^r x]}{\Gamma \vdash_{\text{map}} \text{ind}_{A+B}(s; z.P; x.b_l, x.b_r) \approx \text{ind}_{A'+B'}(s'; z.P'; x.b'_l, x.b'_r) \triangleright P[s]}$$

$$\text{NIDIND} \frac{\Gamma \vdash_{\text{map}} A \cong A' \triangleleft \quad \Gamma \vdash_{\text{map}} s \approx_{\text{map}} s' \triangleleft \text{Id}_A \triangleright a, a' \quad \Gamma, x, y : A, z : \text{Id}_A x y \vdash_{\text{map}} P \cong P' \triangleleft \quad \Gamma, x : A \vdash_{\text{map}} b \cong b' \triangleleft P[\text{id}, x, \text{refl}_{A,x}]}{\Gamma \vdash_{\text{map}} \text{ind}_{\text{Id}_A}(s; x.z.P; b) \approx \text{ind}_{\text{Id}_{A'}}(s'; x.z.P'; b') \triangleright P[\text{id}, a, a', s]}$$

$$\boxed{\text{unmap, unmapfun, unmapfun}_1, \text{unmapfun}_2}$$

$$\begin{array}{llll} \text{unmap}(\text{map}_F f t) & \stackrel{\text{def}}{=} & t & \text{unmap}(t) & \stackrel{\text{def}}{=} & t & \text{otherwise} \\ \text{unmapfun}(\text{map}_F f t, x) & \stackrel{\text{def}}{=} & f x & \text{unmapfun}(t, x) & \stackrel{\text{def}}{=} & x & \text{otherwise} \\ \text{unmapfun}_1(\text{map}_F f t, x) & \stackrel{\text{def}}{=} & \pi_1 f x & \text{unmapfun}_1(t, x) & \stackrel{\text{def}}{=} & x & \text{otherwise} \\ \text{unmapfun}_2(\text{map}_F f t, y) & \stackrel{\text{def}}{=} & \pi_2 f y & \text{unmapfun}_2(t, y) & \stackrel{\text{def}}{=} & y & \text{otherwise} \end{array}$$

$$\boxed{\Gamma \vdash_{\text{map}} n \approx_{\text{map}} n' \triangleleft T}$$

$$\text{UNMAPLIST} \frac{\Gamma \vdash_{\text{map}} \text{unmap}(n) \approx_{\text{h}} \text{unmap}(n') \triangleright \mathbf{List} A \quad \Gamma, x: A \vdash_{\text{map}} \text{unmapfun}(n, x) \cong \text{unmapfun}(n', x) \triangleleft B}{\Gamma \vdash_{\text{map}} n \approx_{\text{map}} n' \triangleleft \mathbf{List} B}$$

$$\text{UNMAPTREE} \frac{\Gamma \vdash_{\text{map}} \text{unmap}(n) \approx_{\text{h}} \text{unmap}(n') \triangleright \mathbf{W} x: A.B \quad \Gamma, x: A \vdash_{\text{map}} \text{unmapfun}_1(n, x) \cong \text{unmapfun}_1(n', x) \triangleleft A' \quad \Gamma, x: A, y: B' [\text{unmapfun}_1(n, x)] \vdash_{\text{map}} \text{unmapfun}_2(n, y) \cong \text{unmapfun}_2(n', y) \triangleleft Bx}{\Gamma \vdash_{\text{map}} n \approx_{\text{map}} n' \triangleleft \mathbf{W} x: A'.B'}$$

$$\text{UNMAPSUM} \frac{\Gamma \vdash_{\text{map}} \text{unmap}(n) \approx_{\text{h}} \text{unmap}(n') \triangleright A + B \quad \Gamma, x: A \vdash_{\text{map}} \text{unmapfun}_1(n, x) \cong \text{unmapfun}_1(n', x) \triangleleft A' \quad \Gamma, x: B \vdash_{\text{map}} \text{unmapfun}_2(n, x) \cong \text{unmapfun}_2(n', x) \triangleleft B'}{\Gamma \vdash_{\text{map}} n \approx_{\text{map}} n' \triangleleft A' + B'}$$

$$\text{UNMAPID} \frac{\Gamma \vdash_{\text{map}} \text{unmap}(n) \approx_{\text{h}} \text{unmap}(n') \triangleright \mathbf{Id}_A a' \quad \Gamma, x: A \vdash_{\text{map}} \text{unmapfun}(n, x) \cong \text{unmapfun}(n', x) \triangleleft A'}{\Gamma \vdash_{\text{map}} n \approx_{\text{map}} n' \triangleleft \mathbf{Id}_{A'} \triangleright \text{unmapfun}(n, a), \text{unmapfun}(n, a')}$$

$$\boxed{t \rightsquigarrow^1 t'}$$

$$\pi_1(\text{map}_{\Sigma} f p) \rightsquigarrow^1 \pi_1 f(\pi_1 p) \quad \pi_2(\text{map}_{\Sigma} f p) \rightsquigarrow^1 \pi_2 f(\pi_2 p)$$

$$\text{map}_{\Pi} f h t \rightsquigarrow^1 (\pi_2 f)(h((\pi_1 f) t)) \quad \text{map}_{\mathbf{Id}} f \text{refl}_{A,a} \rightsquigarrow^1 \text{refl}_{B,fa}$$

$$\text{map}_{\mathbf{List}} f \varepsilon \rightsquigarrow^1 \varepsilon \quad \text{map}_{\mathbf{List}} f (hd :: tl) \rightsquigarrow^1 f hd :: \text{map}_{\mathbf{List}} f tl$$

$$\text{map}_{\mathbf{W}} \{T\} \{T'\} f (\text{sup } a k) \rightsquigarrow^1 \text{sup}_{x.\pi_2 T'} (\pi_1 f a) (\lambda x: (\pi_2 T' (\pi_1 f a)). \text{map}_{\mathbf{W}} f (k (\pi_2 g x)))$$

$$\text{map}_+ (f, g) (\text{inj}^l a) \rightsquigarrow^1 \text{inj}^l (f a) \quad \text{map}_+ (f, g) (\text{inj}^r b) \rightsquigarrow^1 \text{inj}^r (g b)$$

$$\text{REDMAPCOMP} \frac{\text{ne } n \quad F \in \{\mathbf{List}, \mathbf{Id}, +, \mathbf{W}\}}{\text{map}_F f (\text{map}_F g n) \rightsquigarrow^1 \text{map}_F (f \circ g) n}$$

B.5 Declarative record types

Extends Appendix B.1.

$$\text{RECTY} \frac{\mathcal{L} \in \mathcal{P}_f(\text{Lbl}) \quad \forall l \in \mathcal{L}. \Gamma \vdash A_l}{\Gamma \vdash \{l: A_l\}_{l \in \mathcal{L}}}$$

$$\text{RECUNI} \frac{\mathcal{L} \in \mathcal{P}_f(\text{Lbl}) \quad \forall l \in \mathcal{L}. \Gamma \vdash A_l : \text{Type}_i}{\Gamma \vdash \{l: A_l\}_{l \in \mathcal{L}}^i : \text{Type}_i}$$

Definitional Functoriality for Dependent (Sub)Types

$$\begin{array}{c}
\text{RECTM} \frac{\mathcal{L} \in \mathcal{P}_f(\text{Lbl}) \quad \forall l \in \mathcal{L}. \Gamma \vdash u_l : A_l}{\Gamma \vdash \{l := u_l\}_{l \in \mathcal{L}} : \{l : A_l\}_{l \in \mathcal{L}}} \\
\beta_{\text{REC}} \frac{\mathcal{L} \in \mathcal{P}_f(\text{Lbl}) \quad \forall l \in \mathcal{L}. \Gamma \vdash u_l : A_l}{\Gamma \vdash \{l := u_l\}_{l \in \mathcal{L}}.l \cong u_l : A_l} \\
\text{RECPRJ} \frac{\Gamma \vdash r : \{l : A_l\}_{l \in \mathcal{L}}}{\Gamma \vdash r.l : A_l} \\
\eta_{\text{REC}} \frac{\Gamma \vdash r : \{l : A_l\}_{l \in \mathcal{L}}}{\Gamma \vdash r \cong \{l := r.l\}_{l \in \mathcal{L}} : \{l : A_l\}_{l \in \mathcal{L}}}
\end{array}$$

B.6 Algorithmic record types

Extends Appendix B.2. Record construction terms $\{l := u_l\}_{l \in \mathcal{L}}$ are normal forms, and $r.l$ is neutral whenever r is.

$$\begin{array}{c}
\beta_{\text{REC}} \frac{}{\{l := u_l\}_{l \in \mathcal{L}}.l \rightsquigarrow^1 u_l} \quad \text{RECUNI} \frac{\mathcal{L} \in \mathcal{P}_f(\text{Lbl}) \quad \forall l \in \mathcal{L}. \Gamma \vdash A_l \triangleright_h \text{Type}_i}{\Gamma \vdash \{l : A_l\}_{l \in \mathcal{L}}^i \triangleright \text{Type}_i} \\
\text{RECTM} \frac{\mathcal{L} \in \mathcal{P}_f(\text{Lbl}) \quad \forall l \in \mathcal{L}. \Gamma \vdash u_l \triangleright A_l}{\Gamma \vdash \{l := u_l\}_{l \in \mathcal{L}} \triangleright \{l : A_l\}_{l \in \mathcal{L}}} \quad \text{RECETA} \frac{\forall l \in \mathcal{L}. \Gamma \vdash r.l \cong r'.l \triangleleft A_l}{\Gamma \vdash r \cong_h r' \triangleleft \{l : A_l\}_{l \in \mathcal{L}}} \\
\text{NRECPRJ} \frac{\Gamma \vdash n \approx_h n' \triangleright \{l : A_l\}_{l \in \mathcal{L}}}{\Gamma \vdash n.l \approx n'.l \triangleright A_l}
\end{array}$$

B.7 Algorithmic MLTT_{sub}

Extends Appendices B.2 and B.6, with rule CHECKSUB replacing CHECK .

$$\boxed{\Gamma \vdash_{\text{sub}} t \triangleleft T}$$

$$\text{CHECKSUB} \frac{\Gamma \vdash_{\text{sub}} t \triangleright T' \quad \Gamma \vdash_{\text{sub}} T' \preccurlyeq T \triangleleft}{\Gamma \vdash_{\text{sub}} t \triangleleft T}$$

$$\boxed{\Gamma \vdash_{\text{sub}} T \preccurlyeq T' \triangleleft} \quad \text{Type } T \text{ is a subtype of type } T'$$

$$\text{TYRED} \frac{T \rightsquigarrow^* U \quad T' \rightsquigarrow^* U' \quad \Gamma \vdash_{\text{sub}} U \preccurlyeq_h U' \triangleleft}{\Gamma \vdash_{\text{sub}} T \preccurlyeq T' \triangleleft}$$

$$\boxed{\Gamma \vdash_{\text{sub}} T \preccurlyeq_h T' \triangleleft} \quad \text{Reduced type } T \text{ is a subtype of reduced type } T'$$

$$\text{RECSUB} \frac{\mathcal{K} \subseteq \mathcal{L} \quad \forall k \in \mathcal{K}. \Gamma \vdash_{\text{sub}} A_k \preccurlyeq B_k \triangleleft}{\Gamma \vdash_{\text{sub}} \{l : A_l\}_{l \in \mathcal{L}} \preccurlyeq_h \{k : A_k\}_{k \in \mathcal{K}} \triangleleft}$$

$$\text{PRODSUB} \frac{\Gamma \vdash_{\text{sub}} A' \preccurlyeq A \triangleleft \quad \Gamma, x : A' \vdash_{\text{sub}} B \preccurlyeq B' \triangleleft}{\Gamma \vdash_{\text{sub}} \Pi x : A.B \preccurlyeq_h \Pi x : A'.B' \triangleleft}$$

$$\text{LISTSUB} \frac{\Gamma \vdash_{\text{sub}} A \preccurlyeq A' \triangleleft}{\Gamma \vdash_{\text{sub}} \text{List } A \preccurlyeq_h \text{List } A' \triangleleft}$$

$$\text{SIGSUB} \frac{\Gamma \vdash_{\text{sub}} A \preccurlyeq A' \triangleleft \quad \Gamma, x : A \vdash_{\text{sub}} B \preccurlyeq B' \triangleleft}{\Gamma \vdash_{\text{sub}} \Sigma x : A.B \preccurlyeq_h \Sigma x : A'.B' \triangleleft}$$

$$\text{TREESUB} \frac{\Gamma \vdash_{\text{sub}} A \preccurlyeq A' \triangleleft \quad \Gamma, x : A \vdash_{\text{sub}} B' \preccurlyeq B \triangleleft}{\Gamma \vdash_{\text{sub}} \mathbf{W} x : A.B \preccurlyeq_h \mathbf{W} x : A'.B' \triangleleft}$$

$$\begin{array}{c}
\text{IdSUB} \frac{\Gamma \vdash_{\text{sub}} A \approx A' \triangleleft \quad \Gamma \vdash_{\text{sub}} t \cong t' \triangleleft A'}{\Gamma \vdash_{\text{sub}} \text{Id}_A t u \approx_h \text{Id}_{A'} t' u' \triangleleft} \\
\text{SUMSUB} \frac{\Gamma \vdash_{\text{sub}} A \approx A' \triangleleft \quad \Gamma \vdash_{\text{sub}} B \approx B' \triangleleft}{\Gamma \vdash_{\text{sub}} A + B \approx_h A' + B' \triangleleft} \\
\text{SUBREFL} \frac{T \text{ is Type}_i, \mathbf{0}, \mathbf{1} \text{ or } \mathbf{B}}{\Gamma \vdash_{\text{sub}} T \approx_h T \triangleleft} \\
\text{NEUSUB} \frac{\Gamma \vdash_{\text{sub}} n \approx_h n' \triangleright T}{\Gamma \vdash_{\text{sub}} n \approx_h n' \triangleleft}
\end{array}$$

Admissible rules

$$\begin{array}{c}
\text{CONVSUB} \frac{\Gamma \vdash_{\text{sub}} A \cong A' \triangleleft}{\Gamma \vdash_{\text{sub}} A \approx A' \triangleleft} \quad \text{SUBANTISYM} \frac{\Gamma \vdash_{\text{sub}} A \approx A' \triangleleft \quad \Gamma \vdash_{\text{sub}} A' \approx A \triangleleft}{\Gamma \vdash_{\text{sub}} A \cong A' \triangleleft} \\
\text{SUBTRANS} \frac{\Gamma \vdash_{\text{sub}} A \approx A' \triangleleft \quad \Gamma \vdash_{\text{sub}} A' \approx A'' \triangleleft}{\Gamma \vdash_{\text{sub}} A \approx A'' \triangleleft}
\end{array}$$

B.8 Reduction rules and normal forms for MLTT_{coe}

$t \rightsquigarrow^1 t'$

$$\begin{array}{c}
\text{REDCOEFUN} \frac{\text{nf } f}{(\text{coe}_{\Pi x:A.B, \Pi x:A'.B'} f) a \rightsquigarrow^1 \text{coe}_{B[\text{coe}_{A',A} a], B'[a]}(f(\text{coe}_{A',A} a))} \\
\text{REDCOESIG1} \frac{\text{nf } p}{\pi_1(\text{coe}_{\Sigma x:A.B, \Sigma x:A'.B'} p) \rightsquigarrow^1 \text{coe}_{A,A'}(\pi_1 p)} \\
\text{REDCOESIG2} \frac{\text{nf } p}{\pi_2(\text{coe}_{\Sigma x:A.B, \Sigma x:A'.B'} p) \rightsquigarrow^1 \text{coe}_{B[\pi_1 p], B'[\text{coe}_{A,A'}(\pi_1 p)]}(\pi_2 p)} \\
\text{REDCOEREC} \frac{\text{nf } r}{(\text{coe}_{\{l:A_l\}_{l \in \mathcal{L}}, \{k:A_k\}_{k \in \mathcal{K}}} r).l \rightsquigarrow^1 \text{coe}_{A_l, B_l} r.l} \quad \text{COEREDID} \frac{T \text{ is Type}_i, \mathbf{0}, \mathbf{1} \text{ or } \mathbf{B}}{\text{coe}_{T,T} t \rightsquigarrow^1 t} \\
\text{coe}_{A+B, A'+B'}(\text{inj}_B^l a) \rightsquigarrow^1 \text{inj}_{B'}^l(\text{coe}_{A,A'} a) \quad \text{coe}_{A+B, A'+B'}(\text{inj}_A^r b) \rightsquigarrow^1 \text{inj}_{A'}^r(\text{coe}_{B,B'} b) \\
\text{coe}_{\text{List } A, \text{List } A'} \varepsilon \rightsquigarrow^1 \varepsilon_{A'} \quad \text{coe}_{\text{List } A, \text{List } A'}(h :: t) \rightsquigarrow^1 \text{coe}_{A,A'} h ::_{A'} \text{coe}_{\text{List } A, \text{List } A'} t \\
\text{coe}_{\mathbf{W} x:A.B, \mathbf{W} x:A'.B'}(\text{sup } a l) \rightsquigarrow^1 \\
\text{sup}_{x,B'}(\text{coe}_{A,A'} a) (\lambda x: B'[\text{coe}_{A,A'} a]. \text{coe}_{\mathbf{W} x:A.B, \mathbf{W} x:A'.B'}(k(\text{coe}_{B'[\text{coe}_{A,A'} a], B[a]} x))) \\
\text{coe}_{\text{Id}_A a b, \text{Id}_{A'} a' b'} \text{refl}_{A,a} \rightsquigarrow^1 \text{refl}_{A',(\text{coe}_{A,A'} a)} \\
\text{COEL} \frac{A \rightsquigarrow^1 A'}{\text{coe}_{A,B} t \rightsquigarrow^1 \text{coe}_{A',B} t} \quad \text{COER} \frac{\text{nf}^\oplus \text{ or ne } A \quad B \rightsquigarrow^1 B'}{\text{coe}_{A,B} t \rightsquigarrow^1 \text{coe}_{A,B'} t}
\end{array}$$

$$\text{CoETM} \frac{\text{nf}^\oplus \text{ or ne } A, B \quad t \rightsquigarrow^1 t'}{\text{coe}_{A,B} t \rightsquigarrow^1 \text{coe}_{A,B} t'} \quad \text{CoECOE} \frac{\text{nf}^\oplus \text{ or ne } U, U', T, T' \quad \text{ne } n}{\text{coe}_{U,U'} \text{coe}_{T,T'} n \rightsquigarrow^1 \text{coe}_{T,U'} n}$$

$\text{nf } f$	$\stackrel{\text{def}}{=} n \mid P \mid N \mid \lambda x: t.t \mid (t, t) \mid \{l := t_l\}_{l \in \mathcal{L}} \mid \varepsilon_t \mid t ::_t t \mid \text{sup}_t t t \mid () \mid \text{tt} \mid \text{ff} \mid \text{refl}_{t,t} \mid \text{inj}_t^l t \mid \text{inj}_t^r t \mid \text{coe}_{N,N} f$	weak-head normal forms
$\text{nf}^\ominus N$	$\stackrel{\text{def}}{=} \Pi x: t.t \mid \Sigma x: t.t \mid \{l: t_l\}_{l \in \mathcal{L}}$	negative whnf types
$\text{nf}^\oplus P$	$\stackrel{\text{def}}{=} \text{Type}_i \mid \mathbf{0} \mid \mathbf{1} \mid \mathbf{B} \mid \mathbf{List } t \mid \mathbf{W } x: t.t \mid t + t \mid \mathbf{Id}_t t t'$	other whnf types
$\text{ne } n$	$\stackrel{\text{def}}{=} x \mid n t \mid \pi_1 n \mid \pi_2 n \mid n.l \mid \text{ind}_P(n; t; t)$	weak-head neutrals
$\text{cne } c$	$\stackrel{\text{def}}{=} n \mid \text{coe}_{P,P} n \mid \text{coe}_{n,n} n$	compacted neutrals

B.9 Algorithmic MLTT_{coe}

Extends Appendix B.2 and Appendix B.6.

$$\boxed{\Gamma \vdash_{\text{coe}} t \triangleright T}$$

$$\text{COE} \frac{\Gamma \vdash_{\text{coe}} A \triangleleft \quad \Gamma \vdash_{\text{coe}} A' \triangleleft \quad \Gamma \vdash_{\text{coe}} t \triangleleft A \quad \Gamma \vdash_{\text{coe}} A \preccurlyeq A' \triangleleft}{\Gamma \vdash_{\text{coe}} \text{coe}_{A,A'} t \triangleright A'}$$

$$\boxed{\Gamma \vdash_{\text{coe}} t \approx_{\text{coe}} t' \triangleleft T} \quad \text{Compacted neutrals } t \text{ and } t' \text{ are comparable at type } T$$

$$\text{NCOE} \frac{\Gamma \vdash_{\text{coe}} n \approx n' \triangleright S''}{\Gamma \vdash_{\text{coe}} \text{coe}_{S,T} n \approx_{\text{coe}} \text{coe}_{S',T'} n' \triangleleft T''}$$

$$\text{NCOEL} \frac{\Gamma \vdash_{\text{coe}} n \approx n' \triangleright S''}{\Gamma \vdash_{\text{coe}} \text{coe}_{S,T} n \approx_{\text{coe}} n' \triangleleft T''}$$

$$\text{NCOER} \frac{\Gamma \vdash_{\text{coe}} n \approx n' \triangleright S''}{\Gamma \vdash_{\text{coe}} n \approx_{\text{coe}} \text{coe}_{S',T'} n' \triangleleft T''}$$

$$\text{NNOCOE} \frac{\Gamma \vdash_{\text{coe}} n \approx n' \triangleright S''}{\Gamma \vdash_{\text{coe}} n \approx_{\text{coe}} n' \triangleleft T''}$$

$$\boxed{\Gamma \vdash_{\text{coe}} t \cong_h t' \triangleleft T}$$

$$\text{NEULIST} \frac{\Gamma \vdash_{\text{coe}} n \approx_{\text{coe}} n' \triangleleft \mathbf{List } A}{\Gamma \vdash_{\text{coe}} n \cong_h n' \triangleleft \mathbf{List } A}$$

$$\text{NEUTREE} \frac{\Gamma \vdash_{\text{coe}} n \approx_{\text{coe}} n' \triangleleft \mathbf{W } x: A.B}{\Gamma \vdash_{\text{coe}} n \cong_h n' \triangleleft \mathbf{W } x: A.B}$$

$$\text{NEUID} \frac{\Gamma \vdash_{\text{coe}} n \approx_{\text{coe}} n' \triangleleft \mathbf{Id}_A a a'}{\Gamma \vdash_{\text{coe}} n \cong_h n' \triangleleft \mathbf{Id}_A a a'}$$

$$\text{NEUSUM} \frac{\Gamma \vdash_{\text{coe}} n \approx_{\text{coe}} n' \triangleleft A + B}{\Gamma \vdash_{\text{coe}} n \cong_h n' \triangleleft A + B}$$

$$\text{NEUNEU} \frac{\Gamma \vdash_{\text{coe}} n \approx_{\text{coe}} n' \triangleleft M \quad \text{ne } M}{\Gamma \vdash_{\text{coe}} n \cong_h n' \triangleleft M}$$

$$\boxed{\Gamma \vdash_{\text{coe}} T \preccurlyeq T' \triangleleft}$$

$$\text{TYRED} \frac{T \rightsquigarrow^* U \quad T' \rightsquigarrow^* U' \quad \Gamma \vdash_{\text{coe}} U \cong_h U' \triangleleft}{\Gamma \vdash_{\text{coe}} T \preccurlyeq T' \triangleleft}$$

$$\boxed{\Gamma \vdash_{\text{coe}} T \lesssim_{\text{h}} T' \triangleleft}$$

$$\begin{array}{c} \text{RECSUB} \frac{\mathcal{K} \subseteq \mathcal{L} \quad \forall k \in \mathcal{K}. \Gamma \vdash_{\text{coe}} A_k \lesssim B_k \triangleleft}{\Gamma \vdash_{\text{coe}} \{l : A_l\}_{l \in \mathcal{L}} \lesssim_{\text{h}} \{k : A_k\}_{k \in \mathcal{K}} \triangleleft} \\ \\ \text{PRODSUB} \frac{\Gamma \vdash_{\text{coe}} A' \lesssim A \triangleleft \quad \Gamma, x : A' \vdash_{\text{coe}} B[\text{coe}_{A',A} x] \lesssim B' \triangleleft}{\Gamma \vdash_{\text{coe}} \Pi x : A.B \lesssim_{\text{h}} \Pi x : A'.B' \triangleleft} \\ \\ \text{LISTSUB} \frac{\Gamma \vdash_{\text{coe}} A \lesssim A' \triangleleft}{\Gamma \vdash_{\text{coe}} \text{List } A \lesssim_{\text{h}} \text{List } A' \triangleleft} \quad \text{SIGSUB} \frac{\Gamma \vdash_{\text{coe}} A \lesssim A' \triangleleft \quad \Gamma, x : A \vdash_{\text{coe}} B \lesssim B'[\text{coe}_{A,A'} x] \triangleleft}{\Gamma \vdash_{\text{coe}} \Sigma x : A.B \lesssim_{\text{h}} \Sigma x : A'.B' \triangleleft} \\ \\ \text{TREESUB} \frac{\Gamma \vdash_{\text{coe}} A \lesssim A' \triangleleft \quad \Gamma, x : A \vdash_{\text{coe}} B'[\text{coe}_{A,A'} x] \lesssim B \triangleleft}{\Gamma \vdash_{\text{coe}} \mathbf{W} x : A.B \lesssim_{\text{h}} \mathbf{W} x : A'.B' \triangleleft} \\ \\ \text{IDSUB} \frac{\Gamma \vdash_{\text{coe}} A \lesssim A' \triangleleft \quad \Gamma \vdash_{\text{coe}} \text{coe}_{A,A'} t \cong t' \triangleleft A' \quad \Gamma \vdash_{\text{coe}} \text{coe}_{A,A'} u \cong u' \triangleleft A'}{\Gamma \vdash_{\text{coe}} \text{Id}_A t u \lesssim_{\text{h}} \text{Id}_{A'} t' u' \triangleleft} \\ \\ \text{LISTSUB} \frac{\Gamma \vdash_{\text{coe}} A \lesssim A' \triangleleft \quad \Gamma \vdash_{\text{coe}} B \lesssim B' \triangleleft}{\Gamma \vdash_{\text{coe}} A + B \lesssim_{\text{h}} A' + B' \triangleleft} \quad \text{SUBREFL} \frac{T \text{ is Type}_i, \mathbf{0}, \mathbf{1} \text{ or } \mathbf{B}}{\Gamma \vdash_{\text{coe}} T \lesssim T \triangleleft} \end{array}$$

B.10 Declarative MLTT_{coe}

Extends Appendix B.1 and Appendix B.5.

$$\boxed{\Gamma \vdash_{\text{coe}} t : T}$$

$$\text{COE} \frac{\Gamma \vdash_{\text{coe}} A \quad \Gamma \vdash_{\text{coe}} A' \quad \Gamma \vdash_{\text{coe}} t : A \quad \Gamma \vdash_{\text{coe}} A \lesssim A'}{\Gamma \vdash_{\text{coe}} \text{coe}_{A,A'} t : A'}$$

$$\boxed{\Gamma \vdash_{\text{coe}} t \cong t' : T}$$

$$\begin{array}{c} \text{COEID} \frac{\Gamma \vdash_{\text{coe}} t : A}{\Gamma \vdash_{\text{coe}} \text{coe}_{A,A} t \cong t : A} \quad \text{COETRANS} \frac{\Gamma \vdash_{\text{coe}} t : A \quad \Gamma \vdash_{\text{coe}} A \lesssim A' \quad \Gamma \vdash_{\text{coe}} A' \lesssim A''}{\Gamma \vdash_{\text{coe}} \text{coe}_{A',A''} \text{coe}_{A,A'} t \cong \text{coe}_{A,A''} t : A''} \\ \\ \text{COECONG} \frac{\Gamma \vdash_{\text{coe}} t \cong t' : A \quad \Gamma \vdash_{\text{coe}} A \cong A' \quad \Gamma \vdash_{\text{coe}} B \cong B'}{\Gamma \vdash_{\text{coe}} \text{coe}_{A,B} t \cong \text{coe}_{A',B'} t' : B} \\ \\ \text{COEFUN} \frac{\Gamma \vdash_{\text{coe}} A' \lesssim A \quad \Gamma, x : A' \vdash_{\text{coe}} B[\text{coe}_{A',A} x] \lesssim B' \quad \Gamma \vdash_{\text{coe}} f : \Pi x : A.B \quad \Gamma \vdash_{\text{coe}} a : A'}{\Gamma \vdash_{\text{coe}} (\text{coe}_{\Pi x : A.B, \Pi x : A'.B'} f) a \cong \text{coe}_{B[\text{coe}_{A',A} a], B'[x]} (f(\text{coe}_{A',A} a)) : \Pi x : A'.B'} \\ \\ \text{COESIG1} \frac{\Gamma \vdash_{\text{coe}} A \lesssim A' \quad \Gamma, x : A \vdash_{\text{coe}} B \lesssim B'[\text{coe}_{A,A'} x] \quad \Gamma \vdash_{\text{coe}} p : \Sigma x : A.B}{\Gamma \vdash_{\text{coe}} \pi_1(\text{coe}_{\Sigma x : A.B, \Sigma x : A'.B'} p) \cong \text{coe}_{A,A'}(\pi_1 p) : A'}$$

$$\begin{array}{c}
 \text{COESIG2} \frac{\Gamma \vdash_{\text{coe}} A \preccurlyeq A' \quad \Gamma, x: A \vdash_{\text{coe}} B \preccurlyeq B' [\text{coe}_{A,A'} x] \quad \Gamma \vdash_{\text{coe}} p: \Sigma x: A. B}{\Gamma \vdash_{\text{coe}} \pi_2 (\text{coe}_{\Sigma x: A. B, \Sigma x: A'. B'} p) \cong \text{coe}_{B[\pi_1 p], B' [\text{coe}_{A,A'} (\pi_1 p)]} (\pi_2 p): B' [\text{coe}_{A,A'} (\pi_1 p)]} \\
 \\
 \text{COEREC} \frac{\mathcal{K} \subseteq \mathcal{L} \quad \forall k \in \mathcal{K}. \Gamma \vdash_{\text{coe}} A_k \preccurlyeq B_k \quad \Gamma \vdash r \triangleright \{l: A_l\}_{l \in \mathcal{L}}}{\Gamma \vdash_{\text{coe}} (\text{coe}_{\{l: A_l\}_{l \in \mathcal{L}}, \{k: A_k\}_{k \in \mathcal{K}}} r). k \cong \text{coe}_{A_k, B_k} r. k: B_k} \\
 \\
 \text{COENIL} \frac{\Gamma \vdash_{\text{coe}} A \preccurlyeq A'}{\Gamma \vdash_{\text{coe}} \text{coe}_{\text{List } A, \text{List } A'} \varepsilon_A \cong \varepsilon_{A'}: \text{List } A'} \\
 \\
 \text{COECONS} \frac{\Gamma \vdash_{\text{coe}} A \preccurlyeq A' \quad \Gamma \vdash_{\text{coe}} a: A \quad \Gamma \vdash_{\text{coe}} l: \text{List } A}{\Gamma \vdash_{\text{coe}} \text{coe}_{\text{List } A, \text{List } A'} (a ::_A l) \cong (\text{coe}_{A,A'} a) ::_{A'} (\text{coe}_{\text{List } A, \text{List } A'} l): \text{List } A'} \\
 \\
 \text{COETREE} \frac{\Gamma, x: A \vdash_{\text{coe}} B' [\text{coe}_{A,A'} x] \preccurlyeq B \quad \Gamma \vdash_{\text{coe}} A \preccurlyeq A' \quad \Gamma \vdash_{\text{coe}} a: A \quad \Gamma \vdash_{\text{coe}} k: B a \rightarrow \mathbf{W} x: A. B}{\Gamma \vdash_{\text{coe}} \text{coe}_{\mathbf{W} x: A. B, \mathbf{W} x: A. B'} (\text{sup}_{x. B} a l) \cong \text{sup}_{x. B'} (\text{coe}_{A,A'} a) (\lambda x: B' [\text{coe}_{A,A'} a]. \text{coe}_{\mathbf{W} x: A. B, \mathbf{W} x: A. B'} (k (\text{coe}_{B'} [\text{coe}_{A,A'} a], B[a] x))) : \mathbf{W} x: A'. B'}} \\
 \\
 \text{COEID} \frac{\Gamma \vdash_{\text{coe}} A \preccurlyeq A' \quad \Gamma \vdash_{\text{coe}} a: A}{\Gamma \vdash_{\text{coe}} \text{coe}_{\text{Id}_A a a, \text{Id}_{A'} (\text{coe}_{A,A'} a) (\text{coe}_{A,A'} a)} \text{refl}_{A,a} \cong \text{refl}_{A', (\text{coe}_{A,A'} a)} : \text{Id}_A (\text{coe}_{A,A'} a) (\text{coe}_{A,A'} a)} \\
 \\
 \text{COESUMLEFT} \frac{\Gamma \vdash_{\text{coe}} A \preccurlyeq A' \quad \Gamma \vdash_{\text{coe}} B \preccurlyeq B' \quad \Gamma \vdash_{\text{coe}} a: A}{\Gamma \vdash_{\text{coe}} \text{coe}_{A+B, A'+B'} (\text{inj}^l a) \cong \text{inj}^l (\text{coe}_{A,A'} a): A' + B'} \\
 \\
 \text{COESUMRIGHT} \frac{\Gamma \vdash_{\text{coe}} A \preccurlyeq A' \quad \Gamma \vdash_{\text{coe}} B \preccurlyeq B' \quad \Gamma \vdash_{\text{coe}} b: B}{\Gamma \vdash_{\text{coe}} \text{coe}_{A+B, A'+B'} (\text{inj}^r b) \cong \text{inj}^r (\text{coe}_{B,B'} b): A' + B'}
 \end{array}$$

$\boxed{\Gamma \vdash_{\text{coe}} T \preccurlyeq T'}$ T is a subtype of T' in context Γ

$$\begin{array}{c}
 \text{RECSUB} \frac{\mathcal{K} \subseteq \mathcal{L} \quad \forall k \in \mathcal{K}. \Gamma \vdash_{\text{coe}} A_k \preccurlyeq B_k}{\Gamma \vdash_{\text{coe}} \{l: A_l\}_{l \in \mathcal{L}} \preccurlyeq \{k: A_k\}_{k \in \mathcal{K}}} \\
 \\
 \text{PRODSUB} \frac{\Gamma \vdash_{\text{coe}} A' \preccurlyeq A \quad \Gamma, x: A' \vdash_{\text{coe}} B [\text{coe}_{A', A} x] \preccurlyeq B'}{\Gamma \vdash_{\text{coe}} \Pi x: A. B \preccurlyeq \Pi x: A'. B'} \\
 \\
 \text{LISTSUB} \frac{\Gamma \vdash_{\text{coe}} A \preccurlyeq A'}{\Gamma \vdash_{\text{coe}} \text{List } A \preccurlyeq \text{List } A'} \quad \text{SIGSUB} \frac{\Gamma \vdash_{\text{coe}} A \preccurlyeq A' \quad \Gamma, x: A \vdash_{\text{coe}} B \preccurlyeq B' [\text{coe}_{A,A'} x]}{\Gamma \vdash_{\text{coe}} \Sigma x: A. B \preccurlyeq \Sigma x: A'. B'} \\
 \\
 \text{TREESUB} \frac{\Gamma \vdash_{\text{coe}} A \preccurlyeq A' \quad \Gamma, x: A \vdash_{\text{coe}} B' [\text{coe}_{A,A'} x] \preccurlyeq B}{\Gamma \vdash_{\text{coe}} \mathbf{W} x: A. B \preccurlyeq \mathbf{W} x: A'. B'}
 \end{array}$$

$$\begin{array}{c}
\text{IDSUB} \frac{\Gamma \vdash_{\text{coe}} A \preccurlyeq A' \quad \Gamma \vdash_{\text{coe}} \text{coe}_{A,A'} t \cong t' : A' \quad \Gamma \vdash_{\text{coe}} \text{coe}_{A,A'} u \cong u' : A'}{\Gamma \vdash_{\text{coe}} \mathbf{Id}_A t u \preccurlyeq \mathbf{Id}_{A'} t' u'} \\
\\
\text{SUMSUB} \frac{\Gamma \vdash_{\text{coe}} A \preccurlyeq A' \quad \Gamma \vdash_{\text{coe}} B \preccurlyeq B'}{\Gamma \vdash_{\text{coe}} A + B \preccurlyeq A' + B'} \\
\\
\text{SUBREFL} \frac{\Gamma \vdash_{\text{coe}} A \cong A'}{\Gamma \vdash_{\text{coe}} A \preccurlyeq A'} \quad \text{SUBTRANS} \frac{\Gamma \vdash_{\text{coe}} A \preccurlyeq A' \quad \Gamma \vdash_{\text{coe}} A' \preccurlyeq A''}{\Gamma \vdash_{\text{coe}} A \preccurlyeq A''}
\end{array}$$

C PROOFS OF LEMMAS

This section contains additional lemmas and proofs omitted from the body of the paper.

C.1 From Section 3.2

$$\begin{array}{l}
\text{map}_{\Pi} ((g, f) : \text{hom}_{\Pi}((A, B), (A', B'))) (h : \Pi(x : A)B) \stackrel{\text{def}}{=} \lambda x : A'. f (h (g x)) \\
\text{map}_{\Sigma} ((g, f) : \text{hom}_{\Pi}((A, B), (A', B'))) (p : \Sigma(x : A)B) \stackrel{\text{def}}{=} (g (\pi_1 p), f (\pi_2 p))
\end{array}$$

LEMMA C.1. map_{Π} and map_{Σ} satisfy the functor laws *MAPID* and *MAPCOMP*.

PROOF. For the preservation of identities, we have:

$$\begin{array}{l}
\text{map}_{\Pi} (\text{id}_A, \lambda\{x : A\}. \text{id}_{Bx}) h \cong \lambda x : A. \text{id}_{Bx} (h (\text{id}_A x)) \cong \lambda x : A. g x \cong g \\
\text{map}_{\Sigma} (\text{id}_A, \lambda\{x : A\}. \text{id}_{Bx}) p \cong (\text{id}_A (\pi_1 p), \text{id}_{B(\pi_1 p)} (\pi_2 p)) \cong (\pi_1 p, \pi_2 p) \cong p
\end{array}$$

For $(g, f) : \text{hom}_{\Pi}((A_2, B_2), (A_3, B_3))$, $(g', f') : \text{hom}_{\Pi}((A_1, B_1), (A_2, B_2))$ and $h : \Pi x : A_1. B_1 x$ we compute

$$\begin{aligned}
\text{map}_{\Pi} (g, f) (\text{map}_{\Pi} (g', f') h) &\cong \lambda x : A. f ((\lambda x' : A'. f' (h (g' x')))) (g x)) \\
&\cong \lambda x : A. f (f' (h (g' (g x)))) \\
&\cong \lambda x : A. (f \circ f') (h ((g' \circ g) x)) \cong \text{map}_{\Pi} ((g, f) \circ (g', f')) h
\end{aligned}$$

Similarly, for $(g, f) : \text{hom}_{\Sigma}((A_2, B_2), (A_3, B_3))$, $(g', f') : \text{hom}_{\Sigma}((A_1, B_1), (A_2, B_2))$ and $p : \Sigma x : A_1. B_1 x$:

$$\begin{aligned}
\text{map}_{\Sigma} (g, f) (\text{map}_{\Sigma} (g', f') p) &\cong (g (\pi_1 \text{map}_{\Sigma} (g', f') p), f (\pi_2 \text{map}_{\Sigma} (g', f') p)) \\
&\cong (g (g' (\pi_1 p)), f (f' (\pi_2 p))) \\
&\cong ((g \circ g') (\pi_1 p), (f \circ f') (\pi_2 p)) \\
&\cong \text{map}_{\Sigma} ((g, f) \circ (g', f')) p
\end{aligned}$$

□

C.2 From Section 5.3

LEMMA C.2 (CATCH UP, FUNCTION TYPE (LEMMA 5.7)). *If $\Gamma \vdash_{\text{coe}} f a : B$ and $|f| = \lambda x : A'. t'$, then there exists t such that $|t| = t'$ and $f a \rightsquigarrow^* t[a]$.*

PROOF. We must have that $f = (\text{coe}_{T_1, \dots, T_n} (\lambda x : A. t_0))^{15}$ for some A, t_0 such that $|A| = A'$ and $|t_0| = t'$. Moreover, by well-typing we know that there exists some B_0 such that $\Gamma, x : A \vdash_{\text{coe}}$

¹⁵That is, a string of coercions $\text{coe}_{T_{n-1}, T_n} (\dots \text{coe}_{T_1, T_2} (\lambda x : A. t_0))$.

$t_0 \triangleright B_0$, $\Gamma \vdash_{\text{coe}} \Pi x: A. B_0 \cong T_1 \preceq T_2 \cong \dots \preceq T_n \triangleleft$. By inversions, we must have $T_i \rightsquigarrow^* \Pi x: A_i. B_i$, with the A_i and B_i again related. But now we can use the reduction rule of coe on product types, and get

$$f a \rightsquigarrow^* \text{coe}_{B'_0, B'_1, \dots, B'_n} ((\lambda x: A. t_0) (\text{coe}_{A_n, \dots, A_1, A} a))$$

where the B'_i are obtained by adequately substituting coercions in the B_i . Now all the B'_i are well-typed by subject reduction, so they must have weak-head normal forms B''_i , and once all of them have been reduced to weak-head normal form by a combination of **CoEL**, **CoER** and **CoETM**, we can finally reduce the inner β -redex, obtaining

$$f a \rightsquigarrow^* \text{coe}_{B''_0, B''_1, \dots, B''_n} (t_0 [\text{coe}_{A_n, \dots, A_1, A} a])$$

Now we can conclude, as indeed

$$\begin{aligned} |\text{coe}_{B''_0, B''_1, \dots, B''_n} (t_0 [\text{coe}_{A_n, \dots, A_1, A} a])| &= |t_0 [\text{coe}_{A_n, \dots, A_1, A} a]| \\ &= |t_0| [|\text{coe}_{A_n, \dots, A_1, A} a|] \\ &= |t_0| [|x|] = |t_0| = t' \end{aligned}$$

□

LEMMA C.3 (ERASURE IS A BACKWARD SIMULATION (LEMMA 5.10)). *Assume that $\Gamma \vdash_{\text{coe}} t : T$. If $|t| \rightsquigarrow^* u'$, with u' a weak-head normal form, then $t \rightsquigarrow^* u$, with u a weak-head normal form such that $|u| = u'$.*

PROOF. First, if $|t| \rightsquigarrow^* u'$, then there exists u such that $t \rightsquigarrow^* u$ and $|u| = u'$. Indeed, the previous catch-up lemmas ensure that redexes never get blocked by coercions. On function types, the lemma exactly says that a term erasing to a β -redex is able to simulate the β -reduction. On positive types, by the catch-up lemma again, coercions on a constructor reduce away until the constructor is exposed directly to the destructor, and so the reduction can kick in.

Second, if $|u|$ is a weak-head normal form, then there exists a weak-head normal form v such that $u \rightsquigarrow^* v$ and $|v| = |u|$. Indeed, if $|u|$ is a weak-head normal form but u is not, it must be because either $|u|$ is a constructor of a positive type, or a neutral. In the first case, the catch-up lemmas let us conclude. In the second, we can iterate **CoECoE** to fuse coercions until u reduces to a compacted neutral, which is a weak-head normal form. □

LEMMA C.4 (ELABORATION PRESERVES SUBTYPING (LEMMA 5.11)). *The following implications hold whenever the inputs of the conclusions are well-formed:*

- (1) if $|\Gamma| \vdash_{\text{sub}} |T| \preceq_h^m |U| \triangleleft$, then $\Gamma \vdash_{\text{coe}} T \preceq_h^m U \triangleleft$;
- (2) if $|\Gamma| \vdash_{\text{sub}} |T| \preceq_h^m |U| \triangleleft$, then $\Gamma \vdash_{\text{coe}} T \preceq U \triangleleft$;
- (3) if $|\Gamma| \vdash_{\text{sub}} |t| \cong_h |u| \triangleleft |T|$, then $\Gamma \vdash_{\text{coe}} t \cong_h u \triangleleft T$;
- (4) if $|\Gamma| \vdash_{\text{sub}} |t| \cong |u| \triangleleft |T|$, then $\Gamma \vdash_{\text{coe}} t \cong u \triangleleft T$;
- (5) if $|\Gamma| \vdash_{\text{sub}} |t| \approx |u| \triangleright T$, then $\Gamma \vdash_{\text{coe}} t \approx u \triangleright T$;
- (6) if $|\Gamma| \vdash_{\text{sub}} |t| \approx_h |u| \triangleright T$, then $\Gamma \vdash_{\text{coe}} t \approx_h u \triangleright T$.

PROOF. Lemma 5.10 ensures we can always match reductions to weak-head normal forms in MLTT_{sub} with reductions to weak-head normal forms in MLTT_{coe} . As for conversion itself, the key cases are those where the term in MLTT_{coe} is a coercion, that gets erased in MLTT_{sub} . Given the structure of normal forms from Figure 14, this can happen in three situations. If the coercions are between function types or record types, we do not inspect the terms, and instead eagerly η -expand in a type-directed fashion (which triggers further reduction of the now applied coercions). The third case is compacted neutrals. They can appear exactly in the places where MLTT_{coe} uses

the comparison of the compacted neutrals, which strips away the possibly present coercions, as expected. \square

Finally, the main theorem states that we can elaborate terms using implicit subtyping to explicit coercions, in a type-preserving way.

THEOREM C.5 (ELABORATION – INDUCTION). *The following implications hold, whenever inputs to the conclusion are well-formed:*

- (1) if $|\Gamma| \vdash_{\text{sub}} t' \triangleright T'$, then there exists t and T such that $t' = |t|$, $T' = |T|$, and $\Gamma \vdash_{\text{coe}} t \triangleright T$;
- (2) if $|\Gamma| \vdash_{\text{sub}} t' \triangleright_{\text{h}} T'$, then there exists t and T such that $t' = |t|$, $T' = |T|$, and $\Gamma \vdash_{\text{coe}} t \triangleright_{\text{h}} T$;
- (3) if $|\Gamma| \vdash_{\text{sub}} t' \triangleleft |T|$, then there exists t such that $t' = |t|$ and $\Gamma \vdash_{\text{coe}} t \triangleleft T$.

PROOF. Once again, by mutual induction. Each rule is mapped to its counterpart, but for **CHECK-SUB**, where we need to insert a coercion in the elaborated term. This coercion is well-typed by Lemma 5.11. \square

C.3 Translation from MLTT_{coe} to MLTT_{map}

MLTT_{coe} terms contain enough information to entirely capture the subtyping derivations. We exploit this information to define a relation $\llbracket t \rrbracket \simeq t'$ between a MLTT_{coe} term t and a MLTT_{map} term t' , that makes explicit the functorial nature of coercions. The definition of $\llbracket t \rrbracket \simeq t'$ employs an auxiliary relation $\llbracket A \rightsquigarrow B \rrbracket \simeq x$ to translate coercions from A to B , where x is either the special value \star or a MLTT_{map} term f . The value \star arises in the case of an identity coercion that should be erased by the translation. In order to translate records, we assume that we have access to an (effective, decidable) total order on the countable set Lbl of labels, so that we can order in a canonical fashion every finite subsets $\mathcal{L} \subseteq \text{Lbl}$ as $\mathcal{L} = \{l_1 < \dots < l_n\}$.

$$\begin{array}{c}
\text{TSLTY} \frac{}{\llbracket \text{Type}_i \rrbracket \simeq \text{Type}_i} \quad \text{TSLLIST} \frac{\llbracket A \rrbracket \simeq A'}{\llbracket \text{List } A \rrbracket \simeq \text{List } A'} \quad \text{TSLPI} \frac{\llbracket A \rrbracket \simeq A' \quad \llbracket B \rrbracket \simeq B'}{\llbracket \Pi x: A. B \rrbracket \simeq \Pi x: A'. B'} \\
\\
\text{TSLSIG} \frac{\llbracket A \rrbracket \simeq A' \quad \llbracket B \rrbracket \simeq B'}{\llbracket \Sigma x: A. B \rrbracket \simeq \Sigma x: A'. B'} \quad \text{TSLREC} \frac{\forall l \in \mathcal{L}. \llbracket A_l \rrbracket \simeq A'_l \quad \mathcal{L} = \{l_1 < \dots < l_n\}}{\llbracket \{l: A_l\}_{l \in \mathcal{L}} \rrbracket \simeq \Sigma x_{l_1}: A'_{l_1}. \dots A'_{l_n}} \\
\\
\text{TSLVAR} \frac{}{\llbracket x \rrbracket \simeq x} \quad \text{TSLLAM} \frac{\llbracket A \rrbracket \simeq A' \quad \llbracket t \rrbracket \simeq t'}{\llbracket \lambda x: A. t \rrbracket \simeq \lambda x: A'. t'} \quad \text{TSLAPP} \frac{\llbracket u \rrbracket \simeq u' \quad \llbracket v \rrbracket \simeq v'}{\llbracket u v \rrbracket \simeq u' v'} \\
\\
\text{TSLPAIR} \frac{\llbracket u \rrbracket \simeq u' \quad \llbracket v \rrbracket \simeq v'}{\llbracket (u, v) \rrbracket \simeq (u', v')} \quad \text{TSLFST} \frac{\llbracket p \rrbracket \simeq p'}{\llbracket \pi_1 p \rrbracket \simeq \pi_1 p'} \quad \text{TSLSND} \frac{\llbracket p \rrbracket \simeq p'}{\llbracket \pi_2 p \rrbracket \simeq \pi_2 p'} \\
\\
\text{TSLRECTM} \frac{\forall l \in \mathcal{L}. \llbracket u_l \rrbracket \simeq u'_l \quad \mathcal{L} = \{l_1 < \dots < l_n\}}{\llbracket \{l := u_l\} \rrbracket \simeq (u'_{l_1}, \dots, u'_{l_n})} \\
\\
\text{TSLPROJ} \frac{\llbracket p \rrbracket \simeq p' \quad \Gamma \vdash_{\text{coe}} p \{l: A_l\}_{l \in \mathcal{L}} \quad \mathcal{L} = \{l_1 < \dots < l_n\}}{\llbracket p.l_i \rrbracket \simeq \pi_1 \circ \pi_2^{i-1}(p')} \\
\\
\text{TSLCOEID} \frac{\llbracket A \rightsquigarrow B \rrbracket \simeq \star \quad \llbracket t \rrbracket \simeq t'}{\llbracket \text{coe}_{A,B} t \rrbracket \simeq t'} \quad \text{TSLCOE} \frac{\llbracket A \rightsquigarrow B \rrbracket \simeq f \quad \llbracket t \rrbracket \simeq t'}{\llbracket \text{coe}_{A,B} t \rrbracket \simeq f t'}
\end{array}$$

$$\begin{array}{c}
 \text{TSLCOENf} \frac{A \rightsquigarrow^* A' \text{ nf} \quad B \rightsquigarrow^* B' \text{ nf} \quad \llbracket A' \rightsquigarrow B' \rrbracket \simeq x \quad A \neq A' \text{ or } B \neq B'}{\llbracket A \rightsquigarrow B \rrbracket \simeq x} \\
 \\
 \text{TSLCOELISTID} \frac{\llbracket A \rightsquigarrow B \rrbracket \simeq \star}{\llbracket \mathbf{List} A \rightsquigarrow \mathbf{List} B \rrbracket \simeq \star} \quad \text{TSLCOELIST} \frac{\llbracket A \rightsquigarrow B \rrbracket \simeq f}{\llbracket \mathbf{List} A \rightsquigarrow \mathbf{List} B \rrbracket \simeq \text{map}_{\mathbf{List}} f} \\
 \\
 \text{TSLCOEPIIDBOTH} \frac{\llbracket A_2 \rightsquigarrow A_1 \rrbracket \simeq \star \quad \llbracket B_1 \rightsquigarrow B_2 \rrbracket \simeq \star}{\llbracket \Pi x: A_1. B_1 \rightsquigarrow \Pi x: A_2. B_2 \rrbracket \simeq \star} \\
 \\
 \text{TSLCOEPIIDDOM} \frac{\llbracket A_2 \rightsquigarrow A_1 \rrbracket \simeq \star \quad \llbracket B_1 \rightsquigarrow B_2 \rrbracket \simeq g \quad \llbracket A_1 \rrbracket \simeq A'_1}{\llbracket \Pi x: A_1. B_1 \rightsquigarrow \Pi x: A_2. B_2 \rrbracket \simeq \text{map}_{\Pi}(\text{id}_{A'_1}, g)} \\
 \\
 \text{TSLCOEPIIDCOD} \frac{\llbracket A_2 \rightsquigarrow A_1 \rrbracket \simeq f \quad \llbracket B_1 [\text{coe}_{A_2, A_1} x] \rightsquigarrow B_2 \rrbracket \simeq \star \quad \llbracket B_2 \rrbracket \simeq B'_2}{\llbracket \Pi x: A_1. B_1 \rightsquigarrow \Pi x: A_2. B_2 \rrbracket \simeq \text{map}_{\Pi}(f, \text{id}_{B'_2})} \\
 \\
 \text{TSLCOEPI} \frac{\llbracket A_2 \rightsquigarrow A_1 \rrbracket \simeq f \quad \llbracket B_1 [\text{coe}_{A_2, A_1} x] \rightsquigarrow B_2 \rrbracket \simeq g}{\llbracket \Pi x: A_1. B_1 \rightsquigarrow \Pi x: A_2. B_2 \rrbracket \simeq \text{map}_{\Pi}(f, g)} \quad \text{and similarly for } \Sigma \\
 \\
 \text{TSLCOERECID} \frac{\mathcal{K} = \mathcal{L} \quad \forall k \in \mathcal{K}. \llbracket A_k \rightsquigarrow B_k \rrbracket \simeq \star}{\llbracket \{l: A_l\}_{l \in \mathcal{L}} \rightsquigarrow \{k: B_k\}_{k \in \mathcal{K}} \rrbracket \simeq \star} \\
 \\
 \text{TSLCOEREC} \frac{\mathcal{K} \subseteq \mathcal{L} \quad \forall k \in \mathcal{K}. \llbracket A_k \rightsquigarrow B_k \rrbracket \simeq f_k \quad \mathcal{K} = \{k_1 < \dots < k_n\}}{\llbracket \{l: A_l\}_{l \in \mathcal{L}} \rightsquigarrow \{k: B_k\}_{k \in \mathcal{K}} \rrbracket \simeq \lambda p. (f_{k_1} (\pi_1 p), \dots, f_{k_n} (\pi_2^{n-1} p))} \\
 \\
 \text{TSLCOETy} \llbracket \text{Type}_i \rightsquigarrow \text{Type}_i \rrbracket \simeq \star \quad \text{TSLCOENE} \frac{\text{ne } N \quad \text{ne } M}{\llbracket N \rightsquigarrow M \rrbracket \simeq \star}
 \end{array}$$

The translation is extended to contexts pointwise.

$$\frac{}{\llbracket \cdot \rrbracket \simeq \cdot} \quad \frac{\llbracket \Gamma \rrbracket \simeq \Gamma' \quad \llbracket A \rrbracket \simeq A'}{\llbracket \Gamma, x: A \rrbracket \simeq \Gamma', x: A'}$$

We note $\llbracket t \rrbracket \downarrow$ when t is in the domain of the relation and $\llbracket t \rrbracket$ for the image of t when it is defined.

LEMMA C.6 (DETERMINISM OF TRANSLATION). *The translation relation $\llbracket t \rrbracket \simeq t'$ is a partial function, i.e. it is deterministic: for any $t \ t'_1 \ t'_2$, if $\llbracket t \rrbracket \simeq t'_1$ and $\llbracket t \rrbracket \simeq t'_2$ then $t'_1 = t'_2$.*

PROOF. We show by mutual induction on a derivation that $\llbracket A \rightsquigarrow B \rrbracket \simeq x$ is a partial function as well from pairs of MLTT_{coe} types to either \star or a MLTT_{map} term. In the key case TSLCOENf , note that the reduction relation \rightsquigarrow^* is deterministic as well, so we can conclude by induction hypothesis. All other cases are immediate or simple applications of the inductive hypothesis, using the fact that at each step, at most one rule apply. \square

LEMMA C.7 (STABILITY OF TRANSLATION BY WEAKENING). *If ρ is a substitution that maps variables to variables then $\llbracket t \rrbracket[\rho] = \llbracket t[\rho] \rrbracket$.*

PROOF. Immediate by induction on t , the only case interesting case being the translation of variables, with a similar lemma for $\llbracket A \rightsquigarrow B \rrbracket \simeq x$ using that neutrals are preserved. \square

LEMMA C.8 (WELL-TYPED TERMS TRANSLATE). *If $\Gamma \vdash_{\text{coe}} t : A$ then $\llbracket \Gamma \rrbracket \downarrow$, $\llbracket A \rrbracket \downarrow$ and $\llbracket t \rrbracket \downarrow$.*

PROOF. We prove by a straightforward mutual induction on an algorithmic typing derivation that:

- If $\vdash_{\text{coe}} \Gamma$ then $\llbracket \Gamma \rrbracket \downarrow$;
- If $\Gamma \vdash_{\text{coe}} A \triangleleft$ and $\llbracket \Gamma \rrbracket \downarrow$ then $\llbracket A \rrbracket \downarrow$;
- If $\Gamma \vdash_{\text{coe}} t \triangleleft A$ and $\llbracket \Gamma \rrbracket \downarrow$ then $\llbracket t \rrbracket \downarrow$;
- If $\Gamma \vdash_{\text{coe}} t \triangleright A$ and $\llbracket \Gamma \rrbracket \downarrow$ then $\llbracket t \rrbracket \downarrow$;
- If $\Gamma \vdash_{\text{coe}} A \preceq B \triangleleft$ or $\Gamma \vdash_{\text{coe}} A \preceq_h B \triangleleft$ then there exists x such that $\llbracket A \rightsquigarrow B \rrbracket \simeq x$.

□

LEMMA C.9 (IDENTITY COERCIONS). *If $\Gamma \vdash_{\text{coe}} A \cong B \triangleleft$ or $\Gamma \vdash_{\text{coe}} A \cong_h B \triangleleft$ then $\llbracket A \rightsquigarrow B \rrbracket \simeq \star$.*

PROOF. Straightforward mutual induction on the bidirectional conversion derivation. □

LEMMA C.10 (STABILITY OF TRANSLATION BY SUBSTITUTION). *If $\Gamma \vdash_{\text{coe}} t : A$ and $\Delta \vdash_{\text{coe}} \sigma : \Gamma$ then $\llbracket t \rrbracket \llbracket \llbracket \sigma \rrbracket \rrbracket = \llbracket t[\sigma] \rrbracket$ and similarly for typing.*

If $\Gamma \vdash_{\text{coe}} A \preceq B \triangleleft$, $\Delta \vdash_{\text{coe}} \sigma : \Gamma$ and

- $\llbracket A \rightsquigarrow B \rrbracket \simeq \star$ then $\llbracket A[\sigma] \rightsquigarrow B[\sigma] \rrbracket \simeq \star$;
- $\llbracket A \rightsquigarrow B \rrbracket \simeq f$ then $\llbracket A[\sigma] \rightsquigarrow B[\sigma] \rrbracket \simeq f[\sigma]$.

PROOF. Straightforward mutual induction on the bidirectional derivation. □

Forward simulation. Following the proof strategy employed for the equivalence between subsumptive and coercive subtyping, the next step would require to prove that the translation is a forward simulate, i.e. if $\Gamma \vdash_{\text{coe}} t : A$ and $t \rightsquigarrow^1 t'$ then $\llbracket t \rrbracket \rightsquigarrow^* \llbracket t' \rrbracket$. As stated, this lemma does not hold. Indeed, the rule **COECO** leads to reductions of coercions with type annotations which may be convertible but not reduce correctly. We conjecture that a weaker version of the simulation with respect to conversion in MLTT_{map} should hold, that is if $\Gamma \vdash_{\text{coe}} t : A$ and $t \rightsquigarrow^1 t'$ then $\llbracket \Gamma \rrbracket \vdash_{\text{map}} \llbracket t \rrbracket \cong \llbracket t' \rrbracket : \llbracket A \rrbracket$. Such statement should be proved mutually with other properties stating that the translation preserves typing, as follows.

CONJECTURE C.11 (TRANSLATION PRESERVES TYPING).

- (1) *If $\vdash_{\text{coe}} \Gamma$ then $\vdash_{\text{map}} \llbracket \Gamma \rrbracket$*
- (2) *If $\Gamma \vdash_{\text{coe}} A$ then $\llbracket \Gamma \rrbracket \vdash_{\text{map}} \llbracket A \rrbracket$*
- (3) *If $\Gamma \vdash_{\text{coe}} t : A$ then $\llbracket \Gamma \rrbracket \vdash_{\text{map}} \llbracket t \rrbracket : \llbracket A \rrbracket$*
- (4) *If $\Gamma \vdash_{\text{coe}} A \cong B$ then $\llbracket \Gamma \rrbracket \vdash_{\text{map}} \llbracket A \rrbracket \cong \llbracket B \rrbracket$*
- (5) *If $\Gamma \vdash_{\text{coe}} t \cong u : A$ then $\llbracket \Gamma \rrbracket \vdash_{\text{map}} \llbracket t \rrbracket \cong \llbracket u \rrbracket : \llbracket A \rrbracket$*
- (6) *If $\Gamma \vdash_{\text{coe}} A \preceq B$ then either*
 - (a) $\llbracket A \rightsquigarrow B \rrbracket \simeq \star$ and $\llbracket \Gamma \rrbracket \vdash_{\text{map}} \llbracket A \rrbracket \cong \llbracket B \rrbracket$
 - (b) $\llbracket A \rightsquigarrow B \rrbracket \simeq f$ and $\llbracket \Gamma \rrbracket \vdash_{\text{map}} f : \llbracket A \rrbracket \rightarrow \llbracket B \rrbracket$

Preservation of typing, together with catch up lemmas, and a backward simulation lemma, would then allow to lift bidirectional conversion derivations in MLTT_{map} between the translation of terms from MLTT_{coe} . The use of bidirectional conversion is essential here to remain at each step within the translation of MLTT_{coe} terms.

CONJECTURE C.12 (EMBEDDING). $\llbracket - \rrbracket$ embeds MLTT_{coe} into MLTT_{map} : well-typed MLTT_{coe} terms translate to well-typed MLTT_{map} terms, preserving and reflecting conversion.