Definitional Functoriality for Dependent (Sub)Types

Théo Laurent, Meven Lennon-Bertrand, Kenji Maillard

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 assistant rely crucially on a definitional equality, identifying the types and terms that are automatically undistinguishable for the underlying type theory. This paper extends type theory with definitional *functor laws*, two equations satisfied propositionally by a large class of container-like type constructors $F : \mathrm{Type} \to \mathrm{Type}$, equipped with a $\mathrm{map}_F : (A \to B) \to F\,A \to F\,B$, 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 then used to justify modularly a structural form of coercive subtyping, propagating subtyping through type formers. 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

1 INTRODUCTION

Dependent type theory is the foundation of many proof assistants [The Coq Development Team 2022; Moura and Ullrich 2021; Agda Development Team 2023; Brady 2021; Swamy et al. 2016], and at its heart lies definitional equality, an equational theory that is automatically decided by the implementation of these proof systems. For users, a type theory providing a rich definitional equality means less work to prove that things can be identified. 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 mostly impractical as a basis for a proof assistant [Castellan et al. 2017]. As a result, to design usable proof assistants, a well-behaved equational theory that strikes the right balance between expressivity and decidability needs to be carved out. In this paper, we show that we can extend type theory with definitional equalities for *functor laws*, while maintaining this subtle balance. We prove in particular, that definitional equality and type-checking remain decidable in this extension.

Functors and their laws. The notion of functor is pervasive both in mathematics [MacLane 1971] and functional programming [Lipovača 2010], abstracting the concept of *a parametrized construction that whose elements can be transformed.* In the setting of a rich type theory, a functor can be seen as a type former $F : \mathrm{Type} \to \mathrm{Type}$ equipped with an operation $\mathrm{map}_F : (A \to B) \to F\,A \to F\,B$ propagating any function $f : A \to B$ between types A and B to a function from $F\,A$ to $F\,B$. A functor should also respect the categorical structure exhibited by the category of Types and functions, preserving identities and compositions:

$$\mathrm{map}_F\ \mathrm{id} = \mathrm{id} \tag{id-eq}$$

$$\mathrm{map}_F\ f \circ \mathrm{map}_F\ g = \mathrm{map}_F\,(f \circ g) \tag{comp-eq}$$

These two equations are known as the *functor laws*. For many container-like functors, such as **List** A, lists of elements taken in a type A, a map function can be defined such that these equations can be shown *propositionally*, *e.g.* by induction inside the type theory. 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 on the nose, that is, 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

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

Going from one dataset to the other amounts to map either `glue` or its left inverse:

$$\begin{aligned} \mathrm{map}_{\mathbf{List}}\ \texttt{glue} &: \mathbf{List}\ \{a\colon \mathbf{N}; b\colon \mathbf{B}\} \to \mathbf{List}\ \{x\colon \mathbf{B}; y\colon \mathbf{N}; z\colon \mathbf{N}\}, \\ \mathrm{map}_{\mathbf{List}}\ \texttt{glue\_retr} &: \mathbf{List}\ \{x\colon \mathbf{B}; y\colon \mathbf{N}; z\colon \mathbf{N}\} \to \mathbf{List}\ \{a\colon \mathbf{N}; b\colon \mathbf{B}\}. \end{aligned}$$

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 $\mathrm{map}_{\mathbf{List}}$: the boilerplate of manipulating explicitly the functor laws is handled directly and transparently by the type theory. Moreover, observe that in this example the retraction $\texttt{glue\_retr} \circ \texttt{glue} \cong \mathrm{id}$ is definitional thanks to surjective pairing. Combined with definitional functor laws, the following simplification step is discharged automatically by the type-checker:[1]

$$\mathrm{map}_{\mathbf{List}}\ \texttt{glue\_retr}\ (\mathrm{map}_{\mathbf{List}}\ \texttt{glue}\ l) \cong \mathrm{map}_{\mathbf{List}}\ \mathrm{id}\ l \cong l$$

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 may 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 $\texttt{ZToQ}\ (\texttt{natToZ}\ 0)$.

Now, if we want the same mechanism to apply when we pass the list $[0 :: 1 :: 2]$ to a function expecting a $\mathbf{List}\,\mathbf{Q}$, we need to provide a way to propagate the coercions on lists. We can expect to solve this problem by declaring $\mathrm{map}_{\mathbf{List}}$ as a coercion, too: whenever there is a coercion $f\colon A \to B$, then $\mathrm{map}_{\mathbf{List}}\ f$ should be a coercion from $\mathbf{List}\ A$ to $\mathbf{List}\ B$. However, by doing so, we actually caused more trouble than we solved, as there are now two coercions from $\mathbf{List}\,\mathbf{N}$ to $\mathbf{List}\,\mathbf{Q}$, $\mathrm{map}_{\mathbf{List}}(\texttt{ZToQ} \circ \texttt{natToZ})$ and $(\mathrm{map}_{\mathbf{List}}\ \texttt{ZToQ}) \circ (\mathrm{map}_{\mathbf{List}}\ \texttt{natToZ})$. In the absence of definitional functor laws for $\mathrm{map}_{\mathbf{List}}$, 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 $\mathrm{map}_{\mathbf{List}}(\texttt{ZToQ} \circ \texttt{natToZ})\ l$ while the other is $\mathrm{map}_{\mathbf{List}}\ \texttt{ZToQ}\ (\mathrm{map}_{\mathbf{List}}\ \texttt{natToZ}\ l)$. This makes functorial operations like map virtually unusable with coercions, because they are too dangerous.

Structural subtyping. This last example suggests a connection with *subtyping*. Subtyping equips the collection of type with a *subtyping order* $\preccurlyeq$ that allows to move seamlessly terms from a subtype to a supertype, *i.e.* from A to A' when $A \preccurlyeq A'$. An important aspect of subtyping is *structural subtyping*, *e.g.* how can we propagate an existing subtyping notion structurally through type formers.

[1] We formalize this example, showing that this conversion indeed holds in our system, in file Example_1_1.

In the context of the F* program verification platform that heavily uses refinement subtyping, the inability to propagate subtyping on inductive datatypes has been a long-standing issue that never got solved properly [Hriţcu 2014]. Structural subtyping also has a history of causing difficulties to Agda [Cockx 2020; Escot, Poiret, et al. 2023].

Definitional equalities for subtyping. From the perspective of the users of interactive theorem provers, subtyping should be as implicit as possible, 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:

$$\textsc{Sub}\ \frac{\Gamma \vdash_{\text{sub}} t : A \qquad \Gamma \vdash_{\text{sub}} A \preccurlyeq A'}{\Gamma \vdash_{\text{sub}} t : A'} \qquad\qquad \textsc{Coe}\ \frac{\Gamma \vdash_{\text{coe}} t : A \qquad \Gamma \vdash_{\text{coe}} A \preccurlyeq A'}{\Gamma \vdash_{\text{coe}} \mathrm{coe}_{A,A'}\, t : A'}$$

Ideally, we would like to present subsumptive subtyping to users, but ground the meta-theory of the system on the better-behaved coercive subtyping. Informally, an application of Sub in the subsumptive type theory $\mathrm{MLTT}_{\text{sub}}$ should correspond to an application of Coe in the coercive type theory $\mathrm{MLTT}_{\text{coe}}$. Now, given a derivation $\mathcal{D}$ of $\Gamma \vdash_{\text{sub}} t : A$ we can apply Sub together with a reflexivity proof $\Gamma \vdash_{\text{sub}} A \preccurlyeq A$ to yield a new derivation $\mathcal{D}'$ with the same conclusion $\Gamma \vdash_{\text{sub}} t : A$. $\mathcal{D}$ and $\mathcal{D}'$ correspond respectively to terms $\Gamma \vdash_{\text{coe}} t' : A$ and $\Gamma \vdash_{\text{coe}} \mathrm{coe}_{A,A}\, t' : A$ in $\mathrm{MLTT}_{\text{coe}}$. Since t' and $\mathrm{coe}_{A,A}\, t'$ both correspond to the same $\mathrm{MLTT}_{\text{sub}}$ 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 $\Gamma \vdash_{\text{coe}} \mathrm{coe}_{B,C}(\mathrm{coe}_{A,B}\, t') \cong \mathrm{coe}_{A,C}\, t' : C$ should hold in $\mathrm{MLTT}_{\text{coe}}$.

From structural subsumptive subtyping to functorial maps. Z. Luo and Adams [2008] showed that the functorial composition law comp-eq is enough to make structural coercive subtyping compose definitionally. Indeed, a structural coercion between lists $\mathrm{coe}_{\mathbf{List}\ A,\mathbf{List}\ B}$ behaves exactly as the function obtained by mapping $\mathrm{coe}_{A,B}$ on every element of the list. We investigate further this bridge between coercive subtyping and functoriality of type formers, in particular the identity functor law id-eq needed to handle reflexivity of subtyping. This understanding leads to a modular design of subtyping: structural subtyping for a type former exhibits its functorial nature, and can be considered independently of the other type formers of the theory. Moreover, definitional functors laws are sufficient to make structural coercive subtyping for any type former flexible enough to interpret subsumptive subtyping.

Concretely, we define three different type systems, corresponding to the three ideas encountered so far: $\mathrm{MLTT}_{\text{map}}$, with map constructors for each type former, and their definitional functor laws; $\mathrm{MLTT}_{\text{coe}}$, with coercive subtyping; and $\mathrm{MLTT}_{\text{sub}}$, with subsumptive subtyping. All three type theories extend a standard presentation of Martin-Löf Type Theory (MLTT) [Martin-Löf and Sambin 1984] and are fully dependent, with universes and large eliminations—which Z. Luo and Adams [2008] do not cover. We investigate their relationships, as pictured in fig. 1. Erasing coercions induce an equivalence $|{-}|$ from $\mathrm{MLTT}_{\text{coe}}$ to $\mathrm{MLTT}_{\text{sub}}$: erasure is type-preserving, and any well-typed $\mathrm{MLTT}_{\text{sub}}$ term can be annotated with coercions to yield back a well-typed $\mathrm{MLTT}_{\text{coe}}$ term. Moreover, we provide a translation from $\mathrm{MLTT}_{\text{coe}}$ to $\mathrm{MLTT}_{\text{map}}$ making explicit the modular nature of coercions, and conjecture that it is an embedding.

Contributions. We make the following contributions:

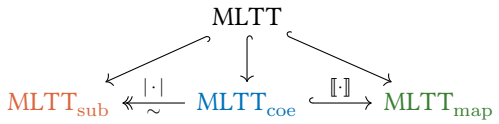

Fig. 1. Relation between MLTT, $\mathrm{MLTT}_{\mathrm{map}}$, $\mathrm{MLTT}_{\mathrm{coe}}$ and $\mathrm{MLTT}_{\mathrm{sub}}$

- we design $\mathrm{MLTT}_{\mathrm{map}}$, an extension of Martin-Löf Type Theory exhibiting the functorial nature of standard type formers (Π, Σ, **List**, **W**, **Id**) and satisfying definitional functor laws (section 3);
- we mechanize the metatheory of a substantial fragment of $\mathrm{MLTT}_{\mathrm{map}}$ in Coq, proving it is normalizing and has decidable type-checking (section 4);
- we develop bidirectional presentations for $\mathrm{MLTT}_{\mathrm{sub}}$ and $\mathrm{MLTT}_{\mathrm{coe}}$, dependent types systems with respectively subsumptive and coercive subtyping, and leverage these to give back and forth translations between the two systems (section 5);
- we show how to compile the coercions of $\mathrm{MLTT}_{\mathrm{coe}}$ down to the functorial maps of $\mathrm{MLTT}_{\mathrm{map}}$ (section 5.5).

The remainder of the paper introduces the necessary technical background, notations and definitions for MLTT in section 2, while section 6 details the related and future work. The supplementary material contains an appendix with detailed proofs and complete typing rules, and the Coq formalization.

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 (fig. 2), and providing the equational theory on types and terms (fig. 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 the (kernel of) any proof assistant implementing this theory, without any need for manual equational proofs.

Variables and substitution. Throughout the paper, we use named variables ($x, y \ldots$) for readability purposes, but follow the de Bruijn indices discipline employed in the Coq formalization. In particular, we do not bother further with 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. By a slight abuse of notation, we will sometimes write it simply u, so that if x correspond to the 0th de Bruijn index in t, $t[u]$ is what would be written $t[x := u]$ in more verbose notation. Typing in all systems is extended pointwise to substitutions in the standard fashion, see appendix B.1 for the rules.

Negative types: dependent products and sums. Dependent function types $\Pi\, x\colon A.B$ are introduced using a λ-abstraction $\lambda\, x\colon A.t$ and eliminated with application $t\, u$. We use braces to indicate arguments that will be left implicit, *e.g.* $\lambda\{x\colon A\}.t$ of type $\Pi\{x\colon A\}.B$. We also include dependent (strong) sum types $\Sigma\, x\colon A.B$, 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.

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

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

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

$$\textsc{Var}\ \frac{\vdash \Gamma \qquad (x : A \in \Gamma)}{\Gamma \vdash x : A} \qquad \textsc{Sort}\ \frac{\vdash \Gamma}{\Gamma \vdash \mathrm{Type}_i : \mathrm{Type}_{i+1}} \qquad \textsc{El}\ \frac{\Gamma \vdash A : \mathrm{Type}_i}{\Gamma \vdash A}$$

$$\textsc{Fun}\ \frac{\Gamma \vdash A : \mathrm{Type}_i \qquad \Gamma, x : A \vdash B : \mathrm{Type}_i}{\Gamma \vdash \Pi\, x : A.B : \mathrm{Type}_i} \qquad \textsc{List}\ \frac{\Gamma \vdash A : \mathrm{Type}_i}{\Gamma \vdash \mathbf{List}\, A : \mathrm{Type}_i}$$

$$\textsc{Abs}\ \frac{\Gamma \vdash A \qquad \Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda\, x : A.t : \Pi\, x : A.B} \qquad \textsc{App}\ \frac{\Gamma \vdash t : \Pi\, x : A.B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\ u : B[u]}$$

$$\textsc{Nil}\ \frac{\Gamma \vdash A}{\Gamma \vdash \varepsilon_A : \mathbf{List}\, A} \qquad \textsc{Cons}\ \frac{\Gamma \vdash A \qquad \Gamma \vdash a : A \qquad \Gamma \vdash l : \mathbf{List}\, A}{\Gamma \vdash a ::_A l : \mathbf{List}\, A}$$

$$\textsc{ListInd}\ \frac{\begin{array}{c}\Gamma \vdash A \qquad \Gamma \vdash s : \mathbf{List}\, A \\ \Gamma, l : \mathbf{List}\, A \vdash P \qquad \Gamma \vdash b_\varepsilon : P[\varepsilon_A] \qquad \Gamma, a : A, l : \mathbf{List}\, A, h : P[l] \vdash b_{::} : P[a ::_A l]\end{array}}{\Gamma \vdash \mathrm{ind}_{\mathbf{List}\, A}(s; l.P; b_\varepsilon, a.l.h.b_{::}) : P[s]}$$

$$\textsc{Conv}\ \frac{\Gamma \vdash t : A \qquad \Gamma \vdash A \cong B}{\Gamma \vdash t : B}$$

Fig. 2. Declarative typing for MLTT

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

$$\textsc{FunTy}\ \frac{\Gamma \vdash A \qquad \Gamma, x : A \vdash B}{\Gamma \vdash \Pi\, x : A.B}$$

Positive types: inductive types and families. As we wish to study the functorial status of type formers, *parametrized* inductive types will be 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 $tl : \mathbf{List}\, A$. List are eliminated using the dependent eliminator $\mathrm{ind}_{\mathbf{List}\, A}(s; l.P; b_\varepsilon, a.l.h.b_{::})$, which performs recursion on the scrutinee s, returning a value in $P[s]$, using the two branches b_ε and $b_{::}$ corresponding to the two constructors ε and $::$. $b_{::}$ binds three variables corresponding to the head a, tail l and the 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 $\mathrm{refl}_{A,a} : \mathbf{Id}\ A\, a\, a$. A general indexed-inductive type scheme is outside the scope of this paper, however $\mathbf{W}$ and $\mathbf{Id}$ are enough to emulate the indexed-inductive types present in various proof assistants together with dependent sums, and the empty $\mathbf{0}$, unit $\mathbf{1}$ and boolean $\mathbf{B}$ types [Abbott et al. 2005; Altenkirch, Ghani, et al. 2015; Hugunin 2020]. As the latter three are not parametrized, their presentation in our setting is entirely standard.

$\boxed{\Gamma \vdash t \cong u : A}$ Terms u and v are convertible at type A under context Γ

$\boxed{\Gamma \vdash A \cong B}$ Types A and B are convertible under context Γ

$$\beta\textsc{Fun}\ \frac{\Gamma \vdash A \qquad \Gamma, x : A \vdash t : B \qquad \Gamma \vdash u : A}{\Gamma \vdash (\lambda\, x : A.t)\ u \cong t[u] : B[u]} \qquad \eta\textsc{Fun}\ \frac{\Gamma, x : A \vdash f\ x \cong g\ x : B}{\Gamma \vdash f \cong g : \Pi\, x : A.B}$$

$$\iota\textsc{Nil}\ \frac{\begin{array}{c}\Gamma \vdash A \qquad \Gamma, l : \mathbf{List}\ A \vdash P \\ \Gamma \vdash b_\varepsilon : P[\varepsilon_A] \qquad \Gamma, a : A, l : \mathbf{List}\ A, h : P[l] \vdash b_{::} : P[a ::_A l]\end{array}}{\Gamma \vdash \mathrm{ind}_{\mathbf{List}\ A}(\varepsilon_A; l.P; b_\varepsilon, a.l.h.b_{::}) \cong b_\varepsilon : P[\varepsilon_A]}$$

$$\iota\textsc{Cons}\ \frac{\begin{array}{c}\Gamma \vdash A \qquad \Gamma \vdash a : A \qquad \Gamma \vdash l : \mathbf{List}\ A \\ \Gamma, l : \mathbf{List}\ A \vdash P \qquad \Gamma \vdash b_\varepsilon : P[\varepsilon_A] \qquad \Gamma, a : A, l : \mathbf{List}\ A, h : P[y] \vdash b_{::} : P[a ::_A l]\end{array}}{\Gamma \vdash \mathrm{ind}_{\mathbf{List}\ A}(a ::_A l; l.P; b_\varepsilon, a.l.h.b_{::}) \cong b_{::}[\mathrm{id}, a, l, \mathrm{ind}_{\mathbf{List}\ A}(l; l.P; b_\varepsilon, a.l.h.b_{::})] : P[a ::_A l]}$$

$$\textsc{FunCong}\ \frac{\Gamma \vdash A \cong A' : \mathrm{Type}_i \qquad \Gamma, x : A \vdash B \cong B' : \mathrm{Type}_i}{\Gamma \vdash \Pi\, x : A.B \cong \Pi\, x : A'.B' : \mathrm{Type}_i} \qquad \text{other congruences omitted}$$

$$\textsc{ConvConv}\ \frac{\Gamma \vdash t \cong t' : A \qquad \Gamma \vdash A \cong A'}{\Gamma \vdash t \cong t' : A'} \qquad \textsc{ElConv}\ \frac{\Gamma \vdash A \cong A' : \mathrm{Type}_i}{\Gamma \vdash A \cong A'}$$

$$\textsc{Refl}\ \frac{\Gamma \vdash t : A}{\Gamma \vdash t \cong t : A} \qquad \textsc{Sym}\ \frac{\Gamma \vdash t \cong u : A}{\Gamma \vdash u \cong t : A} \qquad \textsc{Trans}\ \frac{\Gamma \vdash t \cong u : A \qquad \Gamma \vdash u \cong v : A}{\Gamma \vdash t \cong v : A}$$

Fig. 3. Declarative conversion for MLTT

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

This section presents the meta-theoretical properties we seek to establish in this paper in order to show that the extension of MLTT from fig. 1 are well-behaved and suitable for implementations.

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 term 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 positive types in the empty context as those obtained by repeated applications of constructors, up to conversion. Consistency follows from canonicity, because $\mathbf{0}$ has no constructors, and so it cannot have any inhabitant in the empty context.

Decidability of type-checking and conversion. A consistent proof assistant must also be able to check whether a proof is valid or not. This corresponds to the problem of checking whether a typing judgement is derivable in our type theory. In a dependent type system where terms encode

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

$$\beta\textsc{Red}\ \frac{}{(\lambda\, x{:}\, A.t)\ u \rightsquigarrow^1 t[u]} \qquad \iota\textsc{RedNil}\ \frac{}{\mathrm{ind}_{\mathbf{List}\ A}(\varepsilon_A; x.P; b_\varepsilon, x.y.z.b_{::}) \rightsquigarrow^1 b_\varepsilon}$$

$$\iota\textsc{RedCons}\ \frac{}{\mathrm{ind}_{\mathbf{List}\ A}(a ::_A l; x.P; b_\varepsilon, x.y.z.b_{::}) \rightsquigarrow^1 b_{::}[\mathrm{id}, a, l, \mathrm{ind}_{\mathbf{List}\ A}(l; z.P; b_\varepsilon, x.y.z.b_{::})]}$$

$$\textsc{RedApp}\ \frac{t \rightsquigarrow^1 t'}{t\ u \rightsquigarrow^1 t'\ u} \qquad \textsc{RedInd}\ \frac{t \rightsquigarrow^1 t'}{\mathrm{ind}_{\mathbf{List}\ A}(t; z.P; b_\varepsilon, x.y.z.b_{::}) \rightsquigarrow^1 \mathrm{ind}_{\mathbf{List}\ A}(t'; z.P; b_\varepsilon, x.y.z.b_{::})}$$

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

$$\textsc{RedBase}\ \frac{}{t \rightsquigarrow^\star t} \qquad \textsc{RedStep}\ \frac{t \rightsquigarrow^\star t' \qquad t' \rightsquigarrow^1 t''}{t \rightsquigarrow^\star t''}$$

$$\boxed{\mathrm{nf}\ f} \stackrel{\text{def}}{=} n \mid \Pi\, x{:}\, t_1.t_2 \mid \mathrm{Type}_i \mid \mathbf{List}\, t \mid \lambda\, x{:}\, A.t \mid \varepsilon_A \mid t_1 ::_A t_2 \quad \text{weak-head normal forms}$$
$$\boxed{\mathrm{ne}\ n} \stackrel{\text{def}}{=} x \mid n\ t \mid \mathrm{ind}_{\mathbf{List}\ A}(t; n; t) \quad \text{weak-head neutrals}$$

Fig. 4. Weak-head reduction and normal forms (t stands for an arbitrary term)

the essential structure of derivations, the main obstacle to decidability lie 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 for term. Inspecting the possible normal forms in the empty context entails canonicity and 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 typically fails in ETT where the equational theory is too rich. For expressive 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 that is missing from fig. 3: computations. 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 $\rightsquigarrow^\star$ in fig. 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 βRed. 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 fig. 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.* $x\ u$ cannot reduce further, and will be unstuck once x is substituted by a λ-abstraction.

2.3 Proof methods

With these definitions in hand, we can now go through the main techniques that we will use to establish the meta-theoretical properties we mentioned in section 2.1.

Logical relations. Logical relations are our main tool to establish normalization and canonicity results. We follow the approach of Abel et al. [2017], who formalize a logical relation for MLTT with a single universe in Agda. At a high-level, the logical relation is a model of MLTT ensuring that every denoted term is reducible: it has a weak-head normal form with reducible subterms. Combining this property with the fundamental lemma stating that every well-typed term is reducible, we obtain a first technical property, 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* $\mathrm{nf}\, t'$*), such that* $t \rightsquigarrow^\star t'$*.*

Moreover, since the subterms of a weak-head normal form are reducible as well, we can iterate this process, obtaining full normal forms for any well-typed term. We use the logical relation not only to characterize the normal forms of the terms but also the conversion between them, showing that a proof of convertibility between two terms can be reduced to a standard canonical shape interleaving weak-head reduction sequences and congruence steps between weak-head normal forms. While the technique is rather standard, it is also fairly involved, and adapting it to handle our new type theory, in particular parametrized inductive types and their new definitional functor laws, requires significant modifications.

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 fig. 2 is decomposed into type inference and type checking shown in fig. 5.[2] 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 conversion in fig. 6 combines ideas from both bidirectional typing and the presentation in Abel et al. [2017]. In particular, this presentation 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, while it does not have a dedicated rule for transitivity of conversion, such a rule is nonetheless admissible. Collecting the properties derived from the

[2]Following Lennon-Bertrand [2021], to avoid clashing with Coq's => 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

$$\textsc{Sort}\ \frac{}{\Gamma \vdash \mathrm{Type}_i \triangleright \mathrm{Type}_{i+1}} \qquad \textsc{Var}\ \frac{(x\colon T) \in \Gamma}{\Gamma \vdash x \triangleright T}$$

$$\textsc{Prod}\ \frac{\Gamma \vdash A \triangleright_{\mathrm{h}} \mathrm{Type}_i \qquad \Gamma, x\colon A \vdash B \triangleright_{\mathrm{h}} \mathrm{Type}_i}{\Gamma \vdash \Pi\, x\colon A.B \triangleright \mathrm{Type}_i} \qquad \textsc{List}\ \frac{\Gamma \vdash A \triangleright_{\mathrm{h}} \mathrm{Type}_i}{\Gamma \vdash \mathbf{List}\ A \triangleright \mathrm{Type}_i}$$

$$\textsc{Abs}\ \frac{\Gamma \vdash A \triangleright_{\mathrm{h}} \mathrm{Type}_i \qquad \Gamma, x\colon A \vdash t \triangleright B}{\Gamma \vdash \lambda\, x\colon A.t \triangleright \Pi\, x\colon A.B} \qquad \textsc{App}\ \frac{\Gamma \vdash t \triangleright_{\mathrm{h}} \Pi\, x\colon A.B \qquad \Gamma \vdash u \triangleleft A}{\Gamma \vdash t\ u \triangleright B[u]}$$

$$\textsc{Nil}\ \frac{\Gamma \vdash A \triangleright_{\mathrm{h}} \mathrm{Type}_i}{\Gamma \vdash \varepsilon_A \triangleright \mathbf{List}\ A} \qquad \textsc{Cons}\ \frac{\Gamma \vdash A \triangleright_{\mathrm{h}} \mathrm{Type}_i \qquad \Gamma \vdash a \triangleleft A \qquad \Gamma \vdash l \triangleleft \mathbf{List}\ A}{\Gamma \vdash a ::_A l \triangleright \mathbf{List}\ A}$$

$$\textsc{ListInd}\ \frac{\begin{array}{c}\Gamma \vdash A \triangleright_{\mathrm{h}} \mathrm{Type}_i \qquad \Gamma \vdash s \triangleleft \mathbf{List}\ A \qquad \Gamma, x\colon \mathbf{List}\ A \vdash P \triangleright_{\mathrm{h}} \mathrm{Type}_j \\ \Gamma \vdash b_\varepsilon \triangleleft P[\varepsilon_A] \qquad \Gamma, x\colon A, y\colon \mathbf{List}\ A, z\colon P[y] \vdash b_{::} \triangleleft P[x ::_A y]\end{array}}{\Gamma \vdash \mathrm{ind}_{\mathbf{List}\ A}(s; z.P; b_\varepsilon, x.y.z.b_{::}) \triangleright P[s]}$$

$\boxed{\Gamma \vdash t \triangleleft T}$ Term t checks against type T $\qquad$ $\boxed{\Gamma \vdash t \triangleright_{\mathrm{h}} T}$ Term t infers the reduced type T

$$\textsc{Check}\ \frac{\Gamma \vdash t \triangleright T' \qquad \Gamma \vdash T' \cong T \triangleleft}{\Gamma \vdash t \triangleleft T} \qquad \textsc{InfRed}\ \frac{\Gamma \vdash t \triangleright T' \qquad T' \rightsquigarrow^\star T}{\Gamma \vdash t \triangleright_{\mathrm{h}} T}$$

Fig. 5. Typing rules for algorithmic/bidirectional typing

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 fig. 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, since said judgement fill that role. This algorithmic presentation, being syntax directed, is well suited to design implementations and establish relationships between $\mathrm{MLTT}_{\mathrm{map}}$, $\mathrm{MLTT}_{\mathrm{coe}}$ and $\mathrm{MLTT}_{\mathrm{sub}}$.

3 A FUNCTORIAL TYPE THEORY

The first extension of MLTT we develop is the type theory $\mathrm{MLTT}_{\mathrm{map}}$, which extends the former with primitive map_F operations for each parametrized type former F of MLTT, that is Π, Σ, **List**, **W**, and **Id**. These operations internalize the functorial character of the type formers. Crucially,

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

$$\textsc{NVar}\ \frac{(x : T \in \Gamma)}{\Gamma \vdash x \approx x \triangleright T} \qquad \textsc{NApp}\ \frac{\Gamma \vdash n \approx_{\mathrm{h}} n' \triangleright \Pi\, x : A.B \qquad \Gamma \vdash u \cong u' \triangleleft A}{\Gamma \vdash n\ u \approx n'\ u' \triangleright B[u]}$$

$$\textsc{NListInd}\ \frac{\begin{array}{c}\Gamma \vdash A \cong A' \triangleleft \qquad \Gamma \vdash s \approx s' \triangleright S \qquad \Gamma, l : \mathbf{List}\ A \vdash P \cong P' \triangleleft \\ \Gamma \vdash b_\varepsilon \cong b'_\varepsilon \triangleleft P[\varepsilon_A] \qquad \Gamma, a : A, l : \mathbf{List}\ A, h : P[y] \vdash b_{::} \cong b'_{::} \triangleleft P[a ::_{\mathrm{A}} l]\end{array}}{\Gamma \vdash \mathrm{ind}_{\mathbf{List}\ A}(s; l.P; b_\varepsilon, a.l.h.b_{::}) \approx \mathrm{ind}_{\mathbf{List}\ A'}(s'; l.P'; b'_\varepsilon, a.l.h.b'_{::}) \triangleright P[s]}$$

$\boxed{\Gamma \vdash T \cong_{\mathrm{h}} T' \triangleleft}$ Reduced types T and T' are convertible

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

$$\textsc{CUni}\ \frac{}{\Gamma \vdash \mathrm{Type}_i \cong_{\mathrm{h}} \mathrm{Type}_j \triangleleft \mathrm{Type}_k} \qquad \textsc{CList}\ \frac{\Gamma \vdash A \cong A' \triangleleft \mathrm{Type}_i}{\Gamma \vdash \mathbf{List}\ A \cong_{\mathrm{h}} \mathbf{List}\ A' \triangleleft \mathrm{Type}_i}$$

$$\textsc{CProd}\ \frac{\Gamma \vdash A \cong A' \triangleleft \mathrm{Type}_i \qquad \Gamma, x : A' \vdash B \cong B' \triangleleft \mathrm{Type}_i}{\Gamma \vdash \Pi\, x : A.B \cong_{\mathrm{h}} \Pi\, x : A'.B' \triangleleft \mathrm{Type}_i} \qquad \textsc{CFun}\ \frac{\Gamma, x : A \vdash f\ x \cong f'\ x \triangleleft B}{\Gamma \vdash f \cong_{\mathrm{h}} f' \triangleleft \Pi\, x : A.B}$$

$$\textsc{CNil}\ \frac{}{\Gamma \vdash \varepsilon_A \cong_{\mathrm{h}} \varepsilon_{A'} \triangleleft \mathbf{List}\ A''} \qquad \textsc{CCons}\ \frac{\Gamma \vdash a \cong a' \triangleleft A'' \qquad \Gamma \vdash l \cong l' \triangleleft \mathbf{List}\ A''}{\Gamma \vdash a ::_{\mathrm{A}} l \cong_{\mathrm{h}} a' ::_{\mathrm{A}'} l' \triangleleft \mathbf{List}\ A''}$$

$$\textsc{NeuUni}\ \frac{\Gamma \vdash n \approx n' \triangleright T}{\Gamma \vdash n \cong_{\mathrm{h}} n' \triangleleft \mathrm{Type}_i} \qquad \textsc{NeuList}\ \frac{\Gamma \vdash n \approx_{\mathrm{h}} n' \triangleright S}{\Gamma \vdash n \cong_{\mathrm{h}} n' \triangleleft \mathbf{List}\ A} \qquad \textsc{NeuNeu}\ \frac{\mathrm{ne}\ M \qquad \Gamma \vdash n \approx n' \triangleright N}{\Gamma \vdash n \cong_{\mathrm{h}} n' \triangleleft M}$$

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

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

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

$$\textsc{TmRed}\ \frac{\begin{array}{cc}t \rightsquigarrow^\star u & t' \rightsquigarrow^\star u' \\ T \rightsquigarrow^\star U & \Gamma \vdash u \cong_{\mathrm{h}} u' \triangleleft U\end{array}}{\Gamma \vdash t \cong t' \triangleleft T} \qquad \textsc{NRed}\ \frac{\Gamma \vdash n \approx n' \triangleright T \qquad T \rightsquigarrow^\star S}{\Gamma \vdash n \approx_{\mathrm{h}} n' \triangleright S}$$

Fig. 6. Algorithmic conversion and comparison of neutral terms

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'	$\vdash \Gamma$, ne t, ne t', and $\exists A, A'$ s.t. $\Gamma \vdash t : A$, $\Gamma \vdash t' : A'$

Fig. 7. Well-formed inputs (for $\cong_{\mathrm{h}}, \approx_{\mathrm{h}}, \triangleright_{\mathrm{h}}$, similar to their non-reduced variants)

Type former F	Domain $\mathrm{dom}(F)$	Morphism type $\mathrm{Hom}_F(\cdot_1, \cdot_2)$
List	A: Type	$A_1 \to A_2$
Π	(A, B): $\Sigma(A\text{: Type})\,(A \to \text{Type})$	$\Sigma(f\text{: } A_2 \to A_1)(\Pi\{a\text{: } A_2\}, B_1\,(f\,a) \to B_2\,a)$
Σ	idem	$\Sigma(f\text{: } A_1 \to A_2)(\Pi\{a\text{: } A_1\}, B_1\,a \to B_2\,(f\,a))$
W	idem	$\Sigma(f\text{: } A_1 \to A_2)(\Pi\{a\text{: } A_1\}, B_2\,(f\,a) \to B_1\,a)$
Id	A: Type	$A_1 \to A_2$

Fig. 8. Domain and categorical structure on type formers

map will by design satisfy *definitionally* the generic functor laws for each type former F:

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

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

Section 3.1 describes the structure needed on type formers to state their functoriality in $\mathrm{MLTT}_{\mathrm{map}}$. Section 3.2 shows that definitionally functorial map_F is definable in vanilla MLTT for type formers with an η-law. Section 3.3 then 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 $\mathrm{MLTT}_{\mathrm{map}}$ and obtain as a consequence that the theory is consistent, enjoys canonicity, decidable conversion and type-checking. We implement these design choices in Coq for a simplified but representative version of $\mathrm{MLTT}_{\mathrm{map}}$, with one universe and the Π, Σ, **List** and $\mathbf{N}$ 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 various type formers, such as $\Pi, \Sigma, \mathbf{List}, \mathbf{W}, \mathbf{Id}$, we equip these with the required categorical structure. To simplify the exposition we directly use our type theory to present these structures. We thus consider type formers as functions $F\text{: } \mathrm{dom}(F) \to \mathrm{cod}(F)$, with $\mathrm{cod}(F) \stackrel{\text{def}}{=} \text{Type}$ in most cases. In the case of Π, Σ or $\mathbf{W}$, this means currying the parameters of the type former as dependent sums.[3]

The domain $\mathrm{dom}\,(F)$ of a type former is then equipped with the structure of a category, that is with a type of morphisms $\mathrm{Hom}_F(g_1, g_2)$ for any two instance g_1, g_2 of $\mathrm{dom}\,(F)$, together with associated notions of identity $\mathrm{id}\text{: } \mathrm{Hom}_F(g, g)$ and composition $\circ\text{: } \mathrm{Hom}_F(g_2, g_3) \to \mathrm{Hom}_F(g_1, g_2) \to \mathrm{Hom}_F(g_1, g_3)$, associative and unital definitionally. For instance, dependent products have $\mathrm{dom}(\Pi) = \Sigma(A\text{: Type})\,(A \to \text{Type})$ and

$$\mathrm{Hom}_\Pi((A_1, B_1), (A_2, B_2)) \stackrel{\text{def}}{=} \Sigma(f\text{: } A_2 \to A_1)(\Pi\{a\text{: } A_2\}, B_1\,(f\,a) \to B_2\,a)$$

together with componentwise identity and composition:

$$\mathrm{id}_\Pi : \mathrm{Hom}_\Pi((A, B), (A, B)) \stackrel{\text{def}}{=} (\mathrm{id}_A, \lambda\{x\text{: } A\}.\,\mathrm{id}_{B\,x})$$

$$(f, g) \circ_\Pi (f', g') : \mathrm{Hom}_\Pi((A_1, B_1), (A_3, B_3)) \stackrel{\text{def}}{=} (f \circ f', \lambda\{x\text{: } A_3\}.g \circ g')$$

The domain and type of morphism of the type formers of interest are described in fig. 8. Identities and compositions are given by the standard categorical structure on Type for **List** and **Id**, and are defined similarly to Π for Σ and $\mathbf{W}$. For all type formers but **Id**, the codomain of the functor is the category Type, again with its standard categorical structure. For identity types, the codomain

[3] An alternative presentation would use external notions of categorical structures type formers, expressed using a logical framework.

$A \to A \to \mathrm{Type}$ depends on the given type $A : \mathrm{Type}$ in the domain. The adequate structure for the codomain is not that of a plain category, but of a (split) opfibration [Benabou 1985; Jacobs 2001] or displayed category [Ahrens and Lumsdaine 2019] that provide a notion of morphism in the codomain dependent upon a morphism in the domain.

The operation map_F associated to a type former F

$$\mathrm{map}_F \quad : \quad \Pi\{g_1\, g_2 : \Gamma\},\ \mathrm{Hom}_\Gamma(g_1, g_2) \to F\, g_1 \to F\, g_2$$

witnesses the fact that F is not just a type former, but also has a functorial action with respect to the given category structures on its domain and codomain. Figure 9 presents the generic conversion rules of $\mathrm{MLTT}_{\mathrm{map}}$, extending those of MLTT, and the rules specific to each type formers. For each type former F, map_F is introduced using Map and represents the functorial action through F of a morphism $f : \mathrm{Hom}_F(X, Y)$ between two parameters X, Y of F left implicit. These mapping operations obey the functor laws as stated by MapId and MapComp, and are congruent with respect to both the morphism and mapped element (Map-Cong).

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** and **Id**, map is defined on constructors, propagating by applying itself to recursive arguments and using the provided morphism on the elements of the parameter types. As a result, we recover the usual notion of map on lists. On trees described as **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 $\mathrm{refl}_{A,a}$ at a point $a : A$ is mapped to the reflexivity proof at $f\, a : A'$ for $f : A \to A'$. Each reduction rule has a corresponding conversion rule that can be found in appendix B.3.

3.2 Extensional types and map

A type A has an η-law when every element $a : A$ can be characterized by an η-expansion, *i.e.* as an elimination followed by an introduction in an arbitrary context. For type formers with definitional η-laws, it is possible to define a map operation satisfying the functor laws. In MLTT and $\mathrm{MLTT}_{\mathrm{map}}$, both (strong) dependent sums Σ and dependent products Π have such extensionality laws, and so their map operations are definable.

$$\mathrm{map}_\Pi\ ((g, f) : \mathrm{Hom}_\Pi((A, B), (A', B')))\ (h : \Pi(x : A)B) \stackrel{\mathrm{def}}{=} \lambda\, x : A'.f\ (h\ (g\ x))$$
$$\mathrm{map}_\Sigma\ ((g, f) : \mathrm{Hom}_\Pi((A, B), (A', B')))\ (p : \Sigma(x : A)B) \stackrel{\mathrm{def}}{=} (g\ (\pi_1\, p), f\ (\pi_2\, p))$$

Lemma 3.1. *map_Π and map_Σ satisfy the functor laws MapId and MapComp.*

Appendix C.1 gives a direct proof and the accompanying artifact also contains [4] a proof that the functor laws hold for Coq's Π and Σ types. The specific rule of fig. 9 hold by β-reduction.

3.3 New equations for neutral terms in dependent type theory

Recursive inductive types such as **List**,**W**, or **Id** cannot be endowed with a definitional extensionality principle while retaining decidability of the equational theory [Castellan et al. 2017; McBride 2009]. The result of the previous section hence do not apply, and it is instructive to look at the actual obstruction. Consider the case of **List**, and the equation for preservation of identities:

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

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

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

[4] In file mapPiSigmaFunctorLaws.

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

$$\textsc{Map}\ \frac{\Gamma \vdash_{\mathrm{map}} X, Y : \mathrm{dom}(F) \quad \Gamma \vdash_{\mathrm{map}} f : \mathrm{Hom}_F(X,Y) \quad \Gamma \vdash_{\mathrm{map}} t : F\,X}{\Gamma \vdash_{\mathrm{map}} \mathrm{map}_F\, f\, t : F\,Y} \qquad \textsc{MapId}\ \frac{\Gamma \vdash_{\mathrm{map}} X : \mathrm{dom}(F) \quad \Gamma \vdash_{\mathrm{map}} t : F\,X}{\Gamma \vdash_{\mathrm{map}} \mathrm{map}_F\, \mathrm{id}_{F\,X}\, t \cong t : F\,X}$$

$$\textsc{MapComp}\ \frac{\Gamma \vdash_{\mathrm{map}} X, Y, Z : \mathrm{dom}(F) \quad \Gamma \vdash_{\mathrm{map}} g : \mathrm{Hom}_F(X,Y) \quad \Gamma \vdash_{\mathrm{map}} f : \mathrm{Hom}_F(Y,Z) \quad \Gamma \vdash_{\mathrm{map}} t : F\,X}{\Gamma \vdash_{\mathrm{map}} \mathrm{map}_F\ f\,(\mathrm{map}_F\, g\, t) \cong \mathrm{map}_F (f \circ g)\, t : F\,Z}$$

$$\textsc{Map-Cong}\ \frac{\Gamma \vdash_{\mathrm{map}} X, Y : \mathrm{dom}(F) \quad \Gamma \vdash_{\mathrm{map}} f \cong f' : \mathrm{Hom}_F(X,Y) \quad \Gamma \vdash_{\mathrm{map}} t \cong t' : F\,X}{\Gamma \vdash_{\mathrm{map}} \mathrm{map}_F\ f\, t \cong \mathrm{map}_F\ f'\, t' : F\,Y}$$

Specific rules

$$\mathrm{map}_{\mathbf{List}}\, f\,(hd :: tl) \rightsquigarrow^1 f\, hd :: \mathrm{map}_{\mathbf{List}}\, f\, tl \qquad \mathrm{map}_{\mathbf{List}}\, f\, \varepsilon \rightsquigarrow^1 \varepsilon$$

$$\pi_1\,(\mathrm{map}_\Sigma\ f\, p) \rightsquigarrow^1 \pi_1\, f\,(\pi_1\, p) \qquad \pi_2\,(\mathrm{map}_\Sigma\ f\, p) \rightsquigarrow^1 \pi_2\, f\,(\pi_2\, p)$$

$$\mathrm{map}_\Pi\ f\, h\, t \rightsquigarrow^1 (\pi_2\, f)\,(h\,((\pi_1\, f)\, t))$$

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

$$\mathrm{map}_{\mathbf{Id}}\ f\ \mathrm{refl}_{A,a} \rightsquigarrow^1 \mathrm{refl}_{B,f\,a} \qquad \textsc{RedMapComp}\ \frac{\mathrm{ne}\ n}{\mathrm{map}_{\mathbf{List}}\, f\,(\mathrm{map}_{\mathbf{List}}\, g\, n) \rightsquigarrow^1 \mathrm{map}_{\mathbf{List}}\,(f \circ g)\, n}$$

Fig. 9. $\mathrm{MLTT}_{\mathrm{map}}$, conversion rules and reduction rules for each type former (extends figs. 2 to 4)

We can observe that eq. ($\star$) is validated on closed canonical terms of type **List**:

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

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

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

In order to validate eq. ($\star$), $\mathrm{MLTT}_{\mathrm{map}}$ 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 design choices needed for this extension in the dependently-typed world.

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 for each term. In order to deal with composition of maps, we extend weak head reduction with RedMapComp to fuse consecutive stuck maps. This rule only applies to consecutive map_F for the same type former, which is the only possible case for well-typed terms. Moreover, 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 be an inductive type, because map_Π is already handled through

$$
\begin{array}{lcll}
\boxed{\text{nf } f} & \stackrel{\text{def}}{=} & \cdots \mid c & \text{weak-head normal forms} \\
\boxed{\text{ne } n} & \stackrel{\text{def}}{=} & \cdots \mid \mathrm{ind}_{\mathbf{List}\ A}(t; c; t) & \text{weak-head neutrals} \\
\boxed{\text{cne } c} & \stackrel{\text{def}}{=} & n \mid \mathrm{map}_{\mathbf{List}}\ f\ n & \text{compacted neutrals}
\end{array}
$$

Fig. 10. Weak-head normal and neutrals for $\mathrm{MLTT}_{\mathrm{map}}$ (extends fig. 4)

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. The mapped element is hence required to be a weak-head neutral term. To separate weak-head normal forms which might contain a map as their head from those which cannot, we split the usual neutral forms from the *compacted neutrals*, see fig. 10. Allais et al. [2013] also feature a decomposition of normal forms into three different classes rather than two akin to this one, although their normal forms for lists are more complex than ours as they validate more definitional equations than functorial laws.

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

As a result, just like for η on functions in rule CFun, we implement this rule as part of the 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 $\mathrm{map}_I\ f\ m$, whenever the neutrals n and m are convertible and f agree with the identity of $\mathrm{dom}\,(I)$ on any neutral term. See MapNeConvRedL in the next section for the exact rule.

Eliminators: fusion or not fusion? When considering the interaction between map and the eliminator $\mathrm{ind}_{\mathbf{List}}$, 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?

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

From the point of view of functorial equations, this is not necessary. In fig. 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).

$$\textsc{Fus}\ \frac{\begin{array}{c}\Gamma \vdash_{\mathrm{sub}} s \rhd_{\mathrm{h}} \mathbf{List}\ A \\ \Gamma, l : \mathbf{List}\ A \vdash_{\mathrm{sub}} P \rhd_{\mathrm{h}} \mathrm{Type} \qquad \ldots\end{array}}{\Gamma \vdash_{\mathrm{sub}} \mathrm{ind}_{\mathbf{List}}(s; l.P; \ldots) \rhd P[s]} \qquad \textsc{NoFus}\ \frac{\begin{array}{c}\Gamma \vdash_{\mathrm{sub}} A \lhd \qquad \Gamma \vdash_{\mathrm{sub}} s \lhd \mathbf{List}\ A \\ \Gamma, l : \mathbf{List}\ A \vdash_{\mathrm{sub}} P \rhd_{\mathrm{h}} \mathrm{Type} \qquad \ldots\end{array}}{\Gamma \vdash_{\mathrm{sub}} \mathrm{ind}_{\mathbf{List}\ A}(s; l.P; \ldots) \rhd P[s]}$$

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.

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 $\mathrm{MLTT}_{\mathrm{map}}$ with $\mathbf{0}, \mathbf{N}, \Pi, \Sigma, \mathbf{List}$[5] and a single universe. As it is heavily inspired by a previous Agda formalization [Abel et al. 2017], which has already been extended multiple times [Gilbert et al. 2019; Pujet and Tabareau 2022, 2023], we only give a high level overview of our main adaptations here, and direct the reader either to the Coq code or the original paper [Abel et al. 2017] for more details. The formalization spans ~25k lines of code, approximately 8k of which are specific to our extension with lists and definitionally functorial maps.

4.1 A logical relation with functor laws on list

The Coq development defines both the declarative and algorithmic presentations of $\mathrm{MLTT}_{\mathrm{map}}$ and proves their equivalence through a logical relation parametrized by a generic typing interface[6] instantiated by both presentations. Beyond generic variants of the typing and conversion judgement, the interface uses two extra judgements: $\Gamma \vdash_{\mathrm{map}} t \rightsquigarrow^\star t' : A$ stating that t reduces to t' and that they are both well typed at type A in context Γ; and $\Gamma \vdash_{\mathrm{map}} n \approx n' : A$ 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 logical relations 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 depending on a witness of the former. To do so, Abel et al. [2017] first define for each type former F what it means to be a type reducible as F, and 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 F. As we extend the logical relation to handle $\mathbf{List}$ and $\mathrm{map}_{\mathbf{List}}$, we focus on a high level description of the reducibility of types as lists and the reducible convertibility of terms of type $\mathbf{List}$.[7]

A type X is reducible as a list in context Γ, noted $\Gamma \Vdash_{\mathbf{List}} X$, if it weak-head reduces to $\mathbf{List}\ A$ for some parameter type A, itself reducible in any context Δ extending Γ through a weakening $\rho : \mathtt{Wk}(\Delta, \Gamma)$. If $\mathcal{L} : \Gamma \Vdash_{\mathbf{List}} X$ is a witness that X is reducible as a list, let us write $\mathbb{P}(\mathcal{L})$ for the parameter type A of this witness, and $\mathbb{P}_{\Vdash}(\mathcal{L}) : \Pi\{\rho : \mathtt{Wk}(\Delta, \Gamma)\}.\Delta \Vdash \mathbb{P}(\mathcal{L})[\rho]$ for its witness of reducibility.

Reducible convertibility of terms as lists $\Gamma \Vdash t \cong t' : \mathbf{List}\ A \mid \mathcal{L}$ is defined in fig. 11. Two terms t and t' are reducibly convertible as lists according to $\mathcal{L} : \Gamma \Vdash_{\mathbf{List}} X$ (ListRed) if they reduce to normal forms v, v' that are reducibly convertible as normal forms of type list $\Gamma \Vdash_{\mathrm{nf}} v \cong v' : \mathbf{List}\ A \mid \mathcal{L}$. 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 $\mathrm{map}_{\mathbf{List}}$. 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 $\mathrm{map}_{\mathbf{List}}$, relating $\mathrm{map}_{\mathbf{List}}\ f\ n$ and $\mathrm{map}_{\mathbf{List}}\ f'\ n'$ when the mapped list n and n' are convertible as neutrals and

[5] The formalization does not provide the eliminator for lists, but does provide it for $\mathbf{N}$, another inductive type with recursive occurrences.

[6] Defined in GenericTyping

[7] Available in file LogicalRelation.

$$\textsc{ListRed}\ \frac{\Gamma \vdash_{\mathrm{map}} t \rightsquigarrow^\star v : \mathbf{List}\,\mathbb{P}(\mathcal{L}) \qquad \Gamma \vdash_{\mathrm{map}} t' \rightsquigarrow^\star v' : \mathbf{List}\,\mathbb{P}(\mathcal{L}) \qquad \Gamma \Vdash_{\mathrm{nf}} v \cong v' : \mathbf{List}\,A \mid \mathcal{L}}{\Gamma \Vdash t \cong t' : \mathbf{List}\,A \mid \mathcal{L}}$$

$$\textsc{NeRed}\ \frac{\Gamma \vdash_{\mathrm{map}} n \approx n' : \mathbf{List}\,A}{\Gamma \Vdash_{\mathrm{nf}} n \cong n' : \mathbf{List}\,A \mid \mathcal{L}} \qquad \textsc{NilRed}\ \frac{\Gamma \Vdash A \cong P \mid \mathbb{P}_{\Vdash}(\mathcal{L}) \qquad \Gamma \Vdash A \cong P' \mid \mathbb{P}_{\Vdash}(\mathcal{L})}{\Gamma \Vdash_{\mathrm{nf}} \varepsilon_P \cong \varepsilon_{P'} : \mathbf{List}\,A \mid \mathcal{L}}$$

$$\textsc{ConsRed}\ \frac{\begin{array}{c}\Gamma \Vdash A \cong P \mid \mathbb{P}_{\Vdash}(\mathcal{L}) \qquad \Gamma \Vdash A \cong P' \mid \mathbb{P}_{\Vdash}(\mathcal{L}) \\ \Gamma \Vdash hd \cong hd' : A \mid \mathbb{P}_{\Vdash}(\mathcal{L}) \qquad \Gamma \Vdash tl \cong tl' : \mathbf{List}\,A \mid \mathbb{P}_{\Vdash}(\mathcal{L})\end{array}}{\Gamma \Vdash_{\mathrm{nf}} hd ::_{\mathrm{P}} tl \cong hd' ::_{\mathrm{P}'} tl' : \mathbf{List}\,A \mid \mathcal{L}}$$

$$\textsc{MapNeConvRedL}\ \frac{\Gamma \vdash_{\mathrm{map}} n \approx n' : \mathbf{List}\,A \qquad \Gamma, x : X \Vdash f\,x \cong x : A \mid \mathbb{P}_{\Vdash}(\mathcal{L})}{\Gamma \Vdash_{\mathrm{nf}} \mathrm{map}_{\mathbf{List}}\ f\,n \cong n' : \mathbf{List}\,A \mid \mathcal{L}} \qquad \textsc{NeMapConvRedR}\ \ldots$$

$$\textsc{MapMapConvRed}\ \frac{\Gamma \vdash_{\mathrm{map}} n \approx n' : \mathbf{List}\,X \qquad \Gamma, x : X \Vdash f\,x \cong f'\,x : A \mid \mathbb{P}_{\Vdash}(\mathcal{L})}{\Gamma \Vdash_{\mathrm{nf}} \mathrm{map}_{\mathbf{List}}\ f\,n \cong \mathrm{map}_{\mathbf{List}}\ f'\,n' : \mathbf{List}\,A \mid \mathcal{L}}$$

Fig. 11. Reducible convertibility of lists

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 $\mathbb{P}_{\Vdash}(\mathcal{L})$. 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 $\mathbb{P}_{\Vdash}(\mathcal{L})$, together ensuring that the recursive reducible conversion happens at a reducible type, namely a weakened instance of A. Finally, the symmetric rules NeMapConvRedR and MapNeConvRedL deal with the comparison of a $\mathrm{map}_{\mathbf{List}}$ against a neutral n, that can be morally thought as $\mathrm{map}_{\mathbf{List}}$ id n, and indeed the 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[8]. These properties are essential in order to show that the logical relation validates the functor laws on any reducible term. The proof proceeds through a 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 $\mathrm{map}_{\mathbf{List}}$ 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 should be definitionally associative and unital.

[8]Available in the directory LogicalRelation.

4.2 Equivalence between declarative and algorithmic typing

Instantiating the generic typing interface of the logical relation with declarative typing provides as consequences metatheoretical properties exhibiting the existence of normal forms, among which (weak-head) 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 that of Abel et al. [2017], with two important differences. First, reflecting the addition of compacted neutrals in our definition of normal forms, we must adapt algorithmic conversion accordingly, and introduce a third mutually defined relation to compare compacted neutrals. The main idea is presented in rules ListNeConv and NeMapListL below: when we need to compare compacted neutrals, we use the new relation $\approx_{\text{map}}$, which simulates the behaviour of the logical relation from fig. 11 on compacted neutrals. Second, Abel et al. [2017] stop at conversion, while we also provide algorithmic typing, which is necessary both to show decidability of type-checking and provide the necessary structure for sections 5.3 and 5.5.

$$\textsc{ListNeConv}\ \frac{\Gamma \vdash_{\text{map}} c \approx_{\text{map}} c' \lhd \mathbf{List}\ A}{\Gamma \vdash_{\text{map}} c \cong_{\text{h}} c' \lhd \mathbf{List}\ A} \qquad \textsc{NeMapListL}\ \frac{\Gamma \vdash_{\text{map}} n \approx_{\text{h}} n' \rhd \mathbf{List}\ A \quad \Gamma, x{:}A \vdash_{\text{map}} f\ x \cong x \lhd B}{\Gamma \vdash_{\text{map}} \text{map}_{\mathbf{List}}\ f\ n \approx_{\text{map}} n' \lhd \mathbf{List}\ B}$$

4.3 Practical algorithms to decide conversion and typing

Our final result is decidability of type checking.[9] This is not merely a theorem obtained by working in a constructive meta-theory. Rather, the Coq formalization defines implementations of type-checking and conversion[10] well-suited for extraction using the Equations plugin [Sozeau and Mangin 2019]. To separate the definition of these functions from their correctness, we define them as partial functions, building upon a Coq development by Winterhalter [2023] implementing the free recursion monad of McBride [2015]. Then, in a second phase, we separately show soundness, completeness and termination using the structure of algorithmic typing derivations, culminating in the decidability result.

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 is MLTT_{sub}, whose subtyping is subsumptive: whenever $\vdash_{\text{sub}} t : A \preccurlyeq A'$, then also $\vdash_{\text{sub}} t : A'$, leaving the use of subtyping implicit. The second is MLTT_{coe}, which features coercive subtyping, in the form of an operator $\text{coe}_{A,A'}\ t$, well-typed whenever $\vdash_{\text{coe}} t : A \preccurlyeq A'$ as before, but now explicitly marking where subtyping is used. 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} should 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} should serve as the flexible, user-facing system while MLTT_{coe} would be its well-behaved foundation, making it easier to build meta-theory and semantics.

[9] See file Decidability for the high-level function/theorem.
[10] In file Decidability/Functions.

$\boxed{\Gamma \vdash_{\mathrm{sub}} T \preccurlyeq_{\mathrm{h}} T' \triangleleft}$ Reduced type T is a subtype of reduced type T'

$$\textsc{UniSub}\ \frac{}{\Gamma \vdash_{\mathrm{sub}} \mathrm{Type}_i \preccurlyeq_{\mathrm{h}} \mathrm{Type}_i \triangleleft} \qquad \textsc{ProdSub}\ \frac{\Gamma \vdash_{\mathrm{sub}} A' \preccurlyeq A \triangleleft \qquad \Gamma, x\colon A' \vdash_{\mathrm{sub}} B \preccurlyeq B' \triangleleft}{\Gamma \vdash_{\mathrm{sub}} \Pi x\colon A.B \preccurlyeq_{\mathrm{h}} \Pi x\colon A'.B' \triangleleft}$$

$$\textsc{ListSub}\ \frac{\Gamma \vdash_{\mathrm{sub}} A \preccurlyeq A' \triangleleft}{\Gamma \vdash_{\mathrm{sub}} \mathbf{List}\ A \preccurlyeq_{\mathrm{h}} \mathbf{List}\ A' \triangleleft} \qquad \textsc{NeuSub}\ \frac{\Gamma \vdash_{\mathrm{sub}} n \approx_{\mathrm{h}} n' \triangleright T}{\Gamma \vdash_{\mathrm{sub}} n \preccurlyeq_{\mathrm{h}} n' \triangleleft}$$

Fig. 12. Algorithmic subtyping between reduced types (extends fig. 6)

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

Finally, section 5.4 discuss the implications of this equivalence for coherence and section 5.5 present the correspondence between $\mathrm{MLTT}_{\mathrm{coe}}$ and $\mathrm{MLTT}_{\mathrm{map}}$.

5.1 The type systems $\mathrm{MLTT}_{\mathrm{sub}}$ and $\mathrm{MLTT}_{\mathrm{coe}}$

5.1.1 Algorithmic $\mathrm{MLTT}_{\mathrm{sub}}$. $\mathrm{MLTT}_{\mathrm{sub}}$ replaces Check in MLTT from fig. 5 with the following rule, which uses subtyping $\preccurlyeq$ instead of conversion:

$$\textsc{CheckSub}\ \frac{\Gamma \vdash_{\mathrm{sub}} t \triangleright T' \qquad \Gamma \vdash_{\mathrm{sub}} T' \preccurlyeq T \triangleleft}{\Gamma \vdash_{\mathrm{sub}} t \triangleleft T}$$

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

5.1.2 A type of labels to model records. While the rule of fig. 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 that choice. Still, for our subtyping to be of any interest, we must fix something.

We thus choose a rather simple example, presented in fig. 13. We fix a countable set of labels Lbl, and for each finite subset $L \in \mathcal{P}_{\mathrm{f}}(\mathrm{Lbl})$ introduce a type $\boldsymbol{L}$, with inclusion as subtyping between these. For each $l \in \mathrm{Lbl}$, there is a term $\underline{l} : \boldsymbol{\{l\}}$.[11] Finally, $\boldsymbol{L}$ has a dependent eliminator $\mathrm{ind}_{\boldsymbol{L}}$ which behaves like that of an inductive enumeration: if we can construct an inhabitant of $P\colon \boldsymbol{L} \to \mathrm{Type}$ for every $l \in L$, then we can construct some inhabitant of $\Pi\, x\colon \boldsymbol{L}.P\ x$.

Thanks to this eliminator, a function $R\colon \boldsymbol{L} \to \mathrm{Type}$ can be viewed as a (non-dependent) record type, with $R\ \underline{l}$ the type of the field labelled by l. A term $r : \Pi\, x\colon \boldsymbol{L}.R\ x$ is then a record of that type, with field l given by $r\ \underline{l}$. Finally, the eliminator lets us construct a record by giving a value for each

[11] $\{l\}$ denotes a singleton set. Via subtyping, whenever $l \in L$ we will be able to construct an inhabitant of $\boldsymbol{L}$ corresponding to l, either $\underline{l}$ itself in $\mathrm{MLTT}_{\mathrm{sub}}$, or $\mathrm{coe}_{\boldsymbol{\{l\}},\boldsymbol{L}}\ \underline{l}$ in $\mathrm{MLTT}_{\mathrm{coe}}$. We abbreviate the latter as $\mathrm{coe}_{\boldsymbol{L}}\ \underline{l}$.

$$\textsc{LblTy}\ \frac{L \in \mathcal{P}_{\mathrm{f}}(\mathrm{Lbl})}{\Gamma \vdash_{\mathrm{sub}} \boldsymbol{L} \triangleright \mathrm{Type}_0} \qquad \textsc{LblSub}\ \frac{L \subseteq L'}{\Gamma \vdash_{\mathrm{sub}} \boldsymbol{L} \preccurlyeq_{\mathrm{h}} \boldsymbol{L}' \triangleleft} \qquad \textsc{LblTm}\ \frac{l \in \mathrm{Lbl}}{\Gamma \vdash_{\mathrm{sub}} \underline{l} \triangleright \{\boldsymbol{l}\}}$$

$$\textsc{LblTmConv}\ \frac{}{\Gamma \vdash_{\mathrm{sub}} \underline{l} \cong_{\mathrm{h}} \underline{l} \triangleleft \boldsymbol{L}} \qquad \textsc{LblElim}\ \frac{\Gamma \vdash_{\mathrm{sub}} s \triangleleft \boldsymbol{L} \qquad \Gamma, x{:}\,\boldsymbol{L} \vdash_{\mathrm{sub}} P \triangleright_{\mathrm{h}} \mathrm{Type}_i \qquad \Gamma \vdash_{\mathrm{sub}} b_l \triangleleft P[\underline{l}] \text{ for all } l \in L}{\Gamma \vdash_{\mathrm{sub}} \mathrm{ind}_{\boldsymbol{L}}(s; x.P; \overrightarrow{b_l}) \triangleright P[s]}$$

$$\textsc{NLblInd}\ \frac{\Gamma \vdash_{\mathrm{sub}} n \approx n' \triangleright S \qquad \Gamma, z{:}\,\boldsymbol{L} \vdash_{\mathrm{sub}} P \cong P' \triangleleft \qquad \Gamma \vdash_{\mathrm{sub}} b_l \cong b'_l \triangleleft P[\underline{l}] \text{ for all } l \in L}{\Gamma \vdash_{\mathrm{sub}} \mathrm{ind}_{\boldsymbol{L}}(n; z.P; \overrightarrow{b_l}) \approx \mathrm{ind}_{\boldsymbol{L}}(n'; z.P'; \overrightarrow{b_l}) \triangleright P[n]}$$

$$\textsc{LblRed}\ \frac{}{\mathrm{ind}_{\{\boldsymbol{l}\}}(\underline{l}; x.P; \overrightarrow{b_l}) \rightsquigarrow^1 b_l}$$

Fig. 13. Labels, typing and conversion (extends figs. 5, 6 and 12)

field. In this reading, structural subtyping on the domain of function types gives width subtyping of record types, and subtyping on the codomain induces depth subtyping, for free.

5.1.3 Algorithmic $MLTT_{\mathrm{coe}}$. In contrast with $\mathrm{MLTT}_{\mathrm{sub}}$, in $\mathrm{MLTT}_{\mathrm{coe}}$ rule Check is crucially *not* altered. Instead, subtyping is *only* allowed when *explicitly* marked by coe, as follows:

$$\textsc{Coe}\ \frac{\Gamma \vdash_{\mathrm{coe}} A \triangleleft \qquad \Gamma \vdash_{\mathrm{coe}} A' \triangleleft \qquad \Gamma \vdash_{\mathrm{coe}} t \triangleleft A \qquad \Gamma \vdash_{\mathrm{coe}} A \preccurlyeq A' \triangleleft}{\Gamma \vdash_{\mathrm{coe}} \mathrm{coe}_{A,A'}\, t \triangleright A'}$$

Inference rules for all other type and term formers are similar to those of figs. 5 and 13.

Reduction must of course be extended to give an operational behaviour to coe, and is given in fig. 14, together with normal forms. Operationally, $\mathrm{coe}_{A,A'}\, t$ reduces the types A and A' to head normal forms, then behaves like map according to these, propagating coe recursively on subterms. Since $\mathrm{coe}_{A,A'}\, t$ is well-typed only when A is a subtype of A', the type formers of their head normal forms cannot differ, ensuring that we can always rely on such a map-like behaviour to enact structural subtyping. At base type $\boldsymbol{L}$, $\mathrm{coe}_{\{\boldsymbol{l}\},\boldsymbol{L}}\, \underline{l}$ works as canonical constructors for the type $\boldsymbol{L}$. Finally, just as for map, rule CoeCoe let us compact a succession of stuck coe.

Neutral conversion is described at the top of fig. 15 and features an additional comparison between compacted neutrals similar to $\mathrm{MLTT}_{\mathrm{map}}$ (ListNeConv). Rule NCoe is the "congruence rule" for coercions, where the source and target types necessarily agree by typing invariants. Rules NCoeL and NCoeR handle identity coercions.

Accordingly, $\approx_{\mathrm{coe}}$ is carefully used whenever normal forms can be compacted neutrals, *i.e.* at neutral and list types, as shown at the bottom of fig. 15. Apart from this change, conversion at the term and type level and subtyping are similar to those of $\mathrm{MLTT}_{\mathrm{sub}}$.

5.1.4 Declarative $MLTT_{\mathrm{coe}}$. The declarative presentation of $\mathrm{MLTT}_{\mathrm{coe}}$, noted $\vdash_{\mathrm{coe}}$, straightforwardly extends MLTT (figs. 2 and 3) with typing rules for labels and coe similar to the inference rules from the algorithmic presentation, as well as reduction and conversion rules for $\mathrm{ind}_{\boldsymbol{L}}$ and coe from fig. 14. Most importantly, it satisfies the following two equations of identity and composition of coercions, definitionally.

$$\textsc{CoeId}\ \frac{\Gamma \vdash_{\mathrm{coe}} t : A}{\Gamma \vdash_{\mathrm{coe}} \mathrm{coe}_{A,A}\, t \cong t : A} \qquad \textsc{CoeTrans}\ \frac{\Gamma \vdash_{\mathrm{coe}} t : A \qquad \Gamma \vdash_{\mathrm{coe}} A \preccurlyeq A' \qquad \Gamma \vdash_{\mathrm{coe}} A' \preccurlyeq A''}{\Gamma \vdash_{\mathrm{coe}} \mathrm{coe}_{A',A''}\, \mathrm{coe}_{A,A'}\, t \cong \mathrm{coe}_{A,A''}\, t : A''}$$

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

$$\mathrm{coe}_{\Pi\, x:A'.B', \Pi\, x:A.B}(\lambda\, x\colon A''.t) \rightsquigarrow^1 \lambda\, x\colon A.\, \mathrm{coe}_{B'\left[\mathrm{coe}_{A,A'}\, x\right], B[x]}(t\left[\mathrm{coe}_{A,A'}\, x\right])$$

$$\textsc{RedCoeFunNe}\ \frac{\mathrm{ne}\, f}{(\mathrm{coe}_{\Pi\, x:A'.B', \Pi\, x:A.B}\, f)\ a \rightsquigarrow^1 \mathrm{coe}_{B'\left[\mathrm{coe}_{A,A'}\, a\right], B[a]}(f\ (\mathrm{coe}_{A,A'}\, a))}$$

$$\mathrm{coe}_{\mathrm{Type}_i, \mathrm{Type}_i}\, t \rightsquigarrow^1 t \qquad \mathrm{coe}_{\boldsymbol{L}, \boldsymbol{L}'}\, \mathrm{coe}_{\boldsymbol{L}}\, \underline{l} \rightsquigarrow^1 \mathrm{coe}_{\boldsymbol{L}'}\, \underline{l} \qquad \mathrm{coe}_{\mathbf{List}\, A, \mathbf{List}\, A'}\, \varepsilon \rightsquigarrow^1 \varepsilon$$

$$\mathrm{coe}_{\mathbf{List}\, A, \mathbf{List}\, A'}(h :: t) \rightsquigarrow^1 \mathrm{coe}_{A,A'}\, h :: \mathrm{coe}_{\mathbf{List}\, A, \mathbf{List}\, A'}\, t \qquad \textsc{CoeL}\ \frac{A \rightsquigarrow^1 A'}{\mathrm{coe}_{A,B}\, t \rightsquigarrow^1 \mathrm{coe}_{A',B}\, t}$$

$$\textsc{CoeR}\ \frac{\mathrm{nf}\, A \qquad B \rightsquigarrow^1 B'}{\mathrm{coe}_{A,B}\, t \rightsquigarrow^1 \mathrm{coe}_{A,B'}\, t} \qquad \textsc{CoeTm}\ \frac{\mathrm{nf}\, A \qquad \mathrm{nf}\, B \qquad t \rightsquigarrow^1 t'}{\mathrm{coe}_{A,B}\, t \rightsquigarrow^1 \mathrm{coe}_{A,B}\, t'}$$

$$\textsc{CoeCoe}\ \frac{\mathrm{nf}\, U \qquad \mathrm{nf}\, U' \qquad \mathrm{nf}\, T \qquad \mathrm{nf}\, T' \qquad \mathrm{ne}\, n}{\mathrm{coe}_{U,U'}\, \mathrm{coe}_{T,T'}\, n \rightsquigarrow^1 \mathrm{coe}_{T,U'}\, n}$$

Fig. 14. Weak-head reduction rules for coercion (extends fig. 4)

$\boxed{\Gamma \vdash_{\mathrm{coe}} t \approx_{\mathrm{coe}} t' \lhd T}$ Compacted neutrals t and t' are comparable at type T

$$\textsc{NCoe}\ \frac{\Gamma \vdash_{\mathrm{coe}} n \approx n' \rhd S''}{\Gamma \vdash_{\mathrm{coe}} \mathrm{coe}_{S,T}\, n \approx_{\mathrm{coe}} \mathrm{coe}_{S',T'}\, n' \lhd T''} \qquad \textsc{NCoeL}\ \frac{\Gamma \vdash_{\mathrm{coe}} n \approx n' \rhd S''}{\Gamma \vdash_{\mathrm{coe}} \mathrm{coe}_{S,T}\, n \approx_{\mathrm{coe}} n' \lhd T''}$$

$$\textsc{NCoeR}\ \frac{\Gamma \vdash_{\mathrm{coe}} n \approx n' \rhd S''}{\Gamma \vdash_{\mathrm{coe}} n \approx_{\mathrm{coe}} \mathrm{coe}_{S',T'}\, n' \lhd T''} \qquad \textsc{NNoCoe}\ \frac{\Gamma \vdash_{\mathrm{coe}} n \approx n' \rhd S''}{\Gamma \vdash_{\mathrm{coe}} n \approx_{\mathrm{coe}} n' \lhd T''}$$

$\boxed{\Gamma \vdash_{\mathrm{coe}} t \cong_{\mathrm{h}} t' \lhd T}$

$$\textsc{NeuList}\ \frac{\Gamma \vdash_{\mathrm{coe}} n \approx_{\mathrm{coe}} n' \lhd \mathbf{List}\, A}{\Gamma \vdash_{\mathrm{coe}} n \cong_{\mathrm{h}} n' \lhd \mathbf{List}\, A} \qquad \textsc{NeuNeu}\ \frac{\Gamma \vdash_{\mathrm{coe}} n \approx_{\mathrm{coe}} n' \lhd M \qquad \mathrm{ne}\, M}{\Gamma \vdash_{\mathrm{coe}} n \cong_{\mathrm{h}} n' \lhd M}$$

Fig. 15. Algorithmic comparison of neutrals, with explicit coercions (extends/replaces fig. 6)

5.2 Equivalence of algorithmic and declarative typing

In order to prove properties of correspondence between $\mathrm{MLTT}_{\mathrm{coe}}$ and $\mathrm{MLTT}_{\mathrm{sub}}$, we must first study $\mathrm{MLTT}_{\mathrm{coe}}$. All the meta-theoretic properties of $\mathrm{MLTT}_{\mathrm{map}}$ proved in section 4 carry over to $\mathrm{MLTT}_{\mathrm{coe}}$. We highlight the following two.

Theorem 5.1 (Weak-head normalization). *If $\Gamma \vdash_{\mathrm{coe}} t : T$, then there exists a weak-head normal form t' such that $t \rightsquigarrow^\star t'$.*

Theorem 5.2 (Soundness and completeness of algorithmic typing). *If $\vdash_{\mathrm{coe}} \Gamma$ and $\Gamma \vdash_{\mathrm{coe}} t \rhd T$ then $\Gamma \vdash_{\mathrm{coe}} t : T$, and similarly for the other judgements. Conversely, if $\Gamma \vdash_{\mathrm{coe}} t : T$, then $\Gamma \vdash_{\mathrm{coe}} t \lhd T$, and similarly for the other judgements.*

Most proof ideas from $\mathrm{MLTT}_{\mathrm{map}}$ carry over to $\mathrm{MLTT}_{\mathrm{coe}}$, and we did not mechanize this part of the paper, focusing our formalization effort on the most challenging aspect of the theory. Still, we give a sketch of how to extend the logical relation for $\mathrm{MLTT}_{\mathrm{map}}$ to $\mathrm{MLTT}_{\mathrm{coe}}$ – 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 $\mathrm{MLTT}_{\mathrm{coe}}$). $\mathrm{MLTT}_{\mathrm{coe}}$ has three main differences compared to $\mathrm{MLTT}_{\mathrm{map}}$, so let us treat them in order.

First, 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, rather than defining two separate judgement we can factor the two together by using a single judgement parametrized by a *conversion problem*,[12] 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.

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

We are left with the type formers $\boldsymbol{L}$, that behave like enumeration types. The formalized proof for $\mathbf{N}$ adapts, up to an additional case for compacted neutrals of shape $\mathrm{coe}_{\boldsymbol{L},\boldsymbol{L}'}\ n$. The reducibility of $\mathrm{coe}_{\boldsymbol{L},\boldsymbol{L}'}\ t$ uses CoeCoe on the compacted neutral case and follows that of the definable term $\mathrm{ind}_{\boldsymbol{L}}(t; x.\boldsymbol{L}'; (\mathrm{coe}_{\boldsymbol{L}'}\ \underline{l})_{l\in L})$ on other canonical forms. □

5.3 Elaboration and erasure

We can now turn to the correspondence between $\mathrm{MLTT}_{\mathrm{sub}}$ and $\mathrm{MLTT}_{\mathrm{coe}}$. The translation in the forward direction, *erasure* $|\cdot|$, simply removes coercions $|\mathrm{coe}_{A,A'}\ t| = t$ and is otherwise a congruence. It is lifted pointwise to contexts. We first show that erasure is sound, meaning that it preserves typing, and then that it is also invertible, *i.e.* that any well-typed $\mathrm{MLTT}_{\mathrm{sub}}$ term t' elaborates to a well-typed $\mathrm{MLTT}_{\mathrm{coe}}$ 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 but requires setting up reduction in the right way in fig. 14 so that the lemmas stated in this section 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.

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

For instance, the above rule is inadequate, as it would η-expand terms at function types more in $\mathrm{MLTT}_{\mathrm{coe}}$ than in $\mathrm{MLTT}_{\mathrm{sub}}$. It remains nonetheless admissible, as a conversion.

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

PROOF. By induction on the number of steps, and then on the derivation of one-step reduction. Coercion reduction in $\mathrm{MLTT}_{\mathrm{coe}}$ map to zero steps on the erased terms, while other reduction steps (β, ι, etc.) map to their counterpart after erasure, using that erasure commutes to substitution. □

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

[12] This technique is borrowed from the way cumulativity is handled in MetaCoq [Sozeau, Forster, et al. 2023].

(1) *if* $\Gamma \vdash_{\mathrm{coe}} t \approx_{\mathrm{coe}} u \lhd T$ *then there exists* T' *such that* $\Gamma \vdash_{\mathrm{coe}} T' \preccurlyeq T \lhd$ *and* $|\Gamma| \vdash_{\mathrm{sub}} |t| \approx |u| \rhd |T'|$;
(2) *if* $\Gamma \vdash_{\mathrm{coe}} T \preccurlyeq_{\mathrm{h}} U \lhd$, *then* $|\Gamma| \vdash_{\mathrm{sub}} |T| \preccurlyeq_{\mathrm{h}} |U| \lhd$;
(3) *and similarly for the other subtyping and conversion judgements.*

Proof. By mutual induction, each rule being mapped to their counterpart. Rules for $\Gamma \vdash_{\mathrm{coe}} n \approx_{\mathrm{coe}} n' \lhd T$ are simply dropped, as that judgement is replaced by $\Gamma \vdash_{\mathrm{sub}} n \approx n' \rhd S$ in $\mathrm{MLTT}_{\mathrm{sub}}$. Lemma 5.3 is employed whenever terms and types are reduced to normal forms. □

Theorem 5.5 (Soundness of erasure – induction). *The following implications hold, whenever the inputs of the premise are well-formed:*

- *if* $\Gamma \vdash_{\mathrm{coe}} t \rhd T$, *then there exists* T' *such that* $\Gamma \vdash_{\mathrm{coe}} T' \preccurlyeq T \lhd$ *and* $|\Gamma| \vdash_{\mathrm{sub}} |t| \rhd |T'|$;
- *if* $\Gamma \vdash_{\mathrm{coe}} t \rhd_{\mathrm{h}} T$, *then there exists* T' *such that* $\Gamma \vdash_{\mathrm{coe}} T' \preccurlyeq_{\mathrm{h}} T \lhd$ *and* $|\Gamma| \vdash_{\mathrm{sub}} |t| \rhd_{\mathrm{h}} |T'|$;
- *if* $\Gamma \vdash_{\mathrm{coe}} t \lhd T$, *then there exists* T' *such that* $\Gamma \vdash_{\mathrm{coe}} T' \preccurlyeq T \lhd$ *and* $|\Gamma| \vdash_{\mathrm{sub}} |t| \rhd |T'|$.

Proof. By mutual induction. Checking needs transitivity of $\preccurlyeq$. Reduced inference relies on lemma 5.3 to handle reduction. Finally, each rule for inference can be mapped to its counterpart, noting that $\Gamma \vdash_{\mathrm{coe}} T' \preccurlyeq T \lhd$ and $|\Gamma| \vdash_{\mathrm{sub}} |t| \rhd |T'|$ together imply, by theorem 5.4, $|\Gamma| \vdash_{\mathrm{sub}} |t| \lhd |T|$, so that induction hypothesis on checking premises in $\mathrm{MLTT}_{\mathrm{coe}}$ can be turned into checking premises in $\mathrm{MLTT}_{\mathrm{sub}}$. Finally, for the introduction of $\mathrm{coe}_{A,B}\, t$, subtyping between A and B is combined with the subtyping derivation obtained by induction hypothesis on t. □

In the end, we obtain the following high-level corollary, that erasure preserves typing.

Corollary 5.6 (Soundness of erasure). *If* $\Gamma \vdash_{\mathrm{coe}} t : T$, *then* $|\Gamma| \vdash_{\mathrm{sub}} |t| \lhd |T|$.

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* $\Gamma \vdash_{\mathrm{coe}} f\ a : B$ *and* $|f| = \lambda\, x{:}\, A'.\ t'$, *then there exists* t *such that* $|t| = t'$ *and* $f\ a \rightsquigarrow^\star t[a]$.

Lemma 5.8 (Catch up, label type). *If* $\Gamma \vdash_{\mathrm{coe}} t : \boldsymbol{L}$ *and* $|t| = \underline{l}$, *then* $t \rightsquigarrow^\star \underline{l}$ *or* $t \rightsquigarrow^\star \mathrm{coe}_{\boldsymbol{L}}\, \underline{l}$.

Lemma 5.9 (Catch up, positive types). *If* T *is a positive type (i.e. it is* Type_i, $\mathbf{N}$, $\mathbf{List}$*) and* $\Gamma \vdash_{\mathrm{coe}} t : T$ *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. □

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

Lemma 5.10 (Erasure is a backward simulation). *Assume that* $\Gamma \vdash_{\mathrm{coe}} t : T$. *If* $|t| \rightsquigarrow^\star u'$, *with* u' *a weak-head normal form, then* $t \rightsquigarrow^\star u$, *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* $|\Gamma| \vdash_{\mathrm{sub}} |T| \preccurlyeq |U| \lhd$, *then* $\Gamma \vdash_{\mathrm{coe}} T \preccurlyeq U \lhd$;
(2) *if* $|\Gamma| \vdash_{\mathrm{sub}} |t| \cong |u| \lhd |T|$, *then* $\Gamma \vdash_{\mathrm{coe}} t \cong u \lhd T$;
(3) *if* $|\Gamma| \vdash_{\mathrm{sub}} |t| \approx |u| \rhd T$, *then* $\Gamma \vdash_{\mathrm{coe}} t \approx u \rhd T$;
(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* $|\Gamma| \vdash_{\mathrm{sub}} t' \rhd T'$, *then there exists* t *and* T *such that* $t' = |t|$, $T' = |T|$, *and* $\Gamma \vdash_{\mathrm{coe}} t \rhd T$;
(2) *if* $|\Gamma| \vdash_{\mathrm{sub}} t' \rhd_{\mathrm{h}} T'$, *then there exists* t *and* T *such that* $t' = |t|$, $T' = |T|$, *and* $\Gamma \vdash_{\mathrm{coe}} t \rhd_{\mathrm{h}} T$;
(3) *if* $|\Gamma| \vdash_{\mathrm{sub}} t' \lhd |T|$, *then there exists* t *such that* $t' = |t|$ *and* $\Gamma \vdash_{\mathrm{coe}} t \lhd T$.

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. □

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

Corollary 5.13 (Elaboration). *If* $\Gamma \vdash_{\mathrm{coe}} T$ *and* $|\Gamma| \vdash_{\mathrm{sub}} t' \lhd |T|$, *then there exists* t *such that* $\Gamma \vdash_{\mathrm{coe}} t : T$, *and* $|t| = t'$.

Note that, to establish this equivalence we did *not* need to develop any meta-theory for $\mathrm{MLTT}_{\mathrm{sub}}$: having the meta-theory of $\mathrm{MLTT}_{\mathrm{coe}}$ 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 $\mathrm{MLTT}_{\mathrm{coe}}$ to $\mathrm{MLTT}_{\mathrm{sub}}$.

5.4 Coherence

An important property of elaboration is *coherence*, stating that the elaboration of a well-typed term should not depend on their 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 $\mathrm{MLTT}_{\mathrm{coe}}$ can still erase to the same $\mathrm{MLTT}_{\mathrm{sub}}$ term. While only one of them is the result of elaboration as defined in corollary 5.13, coherence means in our setting that all these distinct terms should 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.

Theorem 5.14 (Coherence). *If* t, u *are such that* $\Gamma \vdash_{\mathrm{coe}} t \lhd T$ *and* $\Gamma \vdash_{\mathrm{coe}} u \lhd T$, *and moreover* $|t| = |u|$ *(i.e.* t *and* u *are both "elaborations" of the same* $\mathrm{MLTT}_{\mathrm{sub}}$ *term), then* $\Gamma \vdash_{\mathrm{coe}} t \cong u \lhd T$.

Proof. By reflexivity (obtained through the equivalence with the declarative system), $\Gamma \vdash_{\mathrm{coe}} t \cong t \lhd T$. Using theorem 5.5 (soundness of erasure), we get $|\Gamma| \vdash_{\mathrm{sub}} |t| \cong |t| \lhd |T|$, and so also $|\Gamma| \vdash_{\mathrm{sub}} |t| \cong |u| \lhd |T|$. But then by lemma 5.11 (elaboration preserving conversion), we can come back, and obtain $\Gamma \vdash_{\mathrm{coe}} t \cong u \lhd T$. □

5.5 From Coercions To Functorial Maps

$\mathrm{MLTT}_{\mathrm{coe}}$ terms contain enough information to capture entirely the subtyping derivations. Figure 16 exploits this information to define a relation $[\![t]\!] \simeq t'$ relating a $\mathrm{MLTT}_{\mathrm{coe}}$ term t and a $\mathrm{MLTT}_{\mathrm{map}}$ term t' that makes explicit the functorial nature of coercions. Most cases of the translation are left to appendix C.3, keeping only the key cases translating of coe on lists. This relation is a partial function containing well-typed terms in its domain. The definition of $[\![t]\!] \simeq t'$ employs

$$\textsc{TslCoeId}\ \frac{[\![A \rightsquigarrow B]\!] \simeq \star \qquad [\![t]\!] \simeq t'}{[\![\mathrm{coe}_{A,B}\, t]\!] \simeq t'} \qquad \textsc{TslCoe}\ \frac{[\![A \rightsquigarrow B]\!] \simeq f \qquad [\![t]\!] \simeq t'}{[\![\mathrm{coe}_{A,B}\, t]\!] \simeq f\, t'}$$

$$\textsc{TslCoeNf}\ \frac{A \rightsquigarrow^\star A'\ \mathrm{nf} \qquad B \rightsquigarrow^\star B'\ \mathrm{nf} \qquad [\![A' \rightsquigarrow B']\!] \simeq x}{[\![A \rightsquigarrow B]\!] \simeq x}$$

$$\textsc{TslCoeListId}\ \frac{[\![A \rightsquigarrow B]\!] \simeq \star}{[\![\mathbf{List}\ A \rightsquigarrow \mathbf{List}\ B]\!] \simeq \star} \qquad \textsc{TslCoeList}\ \frac{[\![A \rightsquigarrow B]\!] \simeq f}{[\![\mathbf{List}\ A \rightsquigarrow \mathbf{List}\ B]\!] \simeq \mathrm{map}_{\mathbf{List}}\, f}$$

Fig. 16. Translation from $\mathrm{MLTT}_{\mathrm{coe}}$ to $\mathrm{MLTT}_{\mathrm{map}}$

an auxiliar relation $[\![A \rightsquigarrow B]\!] \simeq x$ to translate coercions from A to B, where x is either the special value $\star$ or a $\mathrm{MLTT}_{\mathrm{map}}$ term f. The value $\star$ arises in the case of an identity coercion that should be erased by the translation.

Since both $\mathrm{MLTT}_{\mathrm{coe}}$ and $\mathrm{MLTT}_{\mathrm{map}}$ share similar conversion rules, we conjecture that the translation not only preserves typing and conversion but also reflects it.

Conjecture 5.15 (Embedding). $[\![-]\!]$ *embeds* $\mathit{MLTT}_{\mathrm{coe}}$ *into* $\mathit{MLTT}_{\mathrm{map}}$.

This embedding can be combined with the algorithm deciding type-checking for $\mathrm{MLTT}_{\mathrm{map}}$, producing an alternative decision procedure for $\mathrm{MLTT}_{\mathrm{coe}}$. In practice, this means that a proof assistant could reuse parts of an implementation of $\mathrm{MLTT}_{\mathrm{map}}$ in its kernel to provide support for structural coercive subtyping.

6 RELATED AND FUTURE WORK

Logical relations, new equations for neutral terms. Allais et al. [2013] propose to add a variety of fusion laws for lists, including our functor laws, but only in a simply-typed setting, although they sketch how their work could be extended to handle dependency. 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 figs. 10 and 14) instead of the usual normal/neutral forms.

Formalizing logical relations for MLTT is a difficult exercise, pioneered by Abel et al. [2017] in Agda using inductive-recursive definitions, and Wieczorek and Biernacki [2018] in Coq using impredicativity. We build upon and extend a Coq reimplementation of the former. In particular, we provide an effective algorithm to decide type-checking directly on syntactic terms, getting closer to an actual implementation.

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, guarded by an internal proof of equality. Their `cast` does not satisfy a definitional transitivity law, and we give some evidence in appendix A showing that an extension in that direction would be difficult. Another primitive with a rather similar operational behaviour also 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].

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 $\mathrm{MLTT}_{\mathrm{coe}}$.

Subtyping, dependent types and algorithmic derivations. 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 logically rather weak type system.

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 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 and conjecture that injectivity of type constructors hold in presence of χ-rules, a result that we prove and formalize for **List**. Both of these papers employ a strict order for subtyping and do not consider the functor law for the identity, nor tackle decidability of type-checking.

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.

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. Refinement subtyping are heavily used in F* but also in Russel [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[13] propositions [Gilbert et al. 2019] to preserve coherence 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 whether extra definitional equations are needed—and can be realized—to make structural cumulativity work seamlessly.

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.

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.

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

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/.

Benedikt Ahrens and Peter LeFanu Lumsdaine. 2019. "Displayed Categories." *Log. Methods Comput. Sci.*, 15, 1. DOI: 10.23638/LMCS-15(1:20)2019.

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.

Thorsten Altenkirch, Neil Ghani, Peter G. Hancock, Conor McBride, and Peter Morris. 2015. "Indexed containers." *J. Funct. Program.*, 25. DOI: 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.

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.

Freiric 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.

Jean Benabou. 1985. "Fibered Categories and the Foundations of Naive Category Theory." *J. Symb. Log.*, 50, 1, 10–37. DOI: 10.2307/2273784.

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.

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.

Jesper Cockx. 2020. *Disable all subtyping by default?* https://github.com/agda/agda/issues/4474. Accessed: 2023-07-10. (2020).

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.

Cvetan Dunchev, Ferruccio Guidi, Claudio Sacerdoti Coen, and Enrico Tassi. Nov. 2015. "ELPI: fast, Embeddable, λProlog 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.

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.

Lucas Escot, Josselin Poiret, 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.

Cătălin Hriţcu. 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.

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.

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.

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.

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.

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.

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.

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.

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. 2015. "Turing-Completeness Totally Free." In: *Mathematics of Program Construction*. Ed. by Ralf Hinze and Janis Voigtländer. Springer International Publishing, Cham, 257–275.

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.

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.

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.

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.

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.

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.

Matthieu Sozeau and Cyprien Mangin. July 2019. "Equations Reloaded: High-Level Dependently-Typed Functional Programming and Proving in Coq." *Proc. ACM Program. Lang.*, 3, ICFP, Article 86, (July 2019), 29 pages. DOI: 10.1145/3341690.

Jonathan Sterling. 2019. "Algebraic Type Theory and Universe Hierarchies." *CoRR*, abs/1902.08848. arXiv: 1902.08848.

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 Bodík and Rupak Majumdar. ACM, 256–270. ISBN: 978-1-4503-3549-2. DOI: 10.1145/2837614.2837655.

Enrico Tassi. Jan. 2018. "Elpi: an extension language for Coq (Metaprogramming Coq in the Elpi λProlog 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, 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.

Eric Wieser. 2023. *Multiple inheritance hazards in algebraic typeclass hierarchies*. (2023). arXiv: 2306.00617 [cs.LO].

Théo Winterhalter. 2023. “Composable partial functions in Coq, totally for free.” In: *29th International Conference on Types for Proofs and Programs*.

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* sub $A\,B$ *for any two types A and B, equipped with reflexivity witnesses* $\mathtt{refl}_A : \mathrm{sub}\,A\,A$ *and transitivity witnesses* $\mathtt{trans}\,w\,w' : \mathrm{sub}\,A\,C$ *for* $w : \mathrm{sub}\,A\,B$ *and* $w' : \mathrm{sub}\,B\,C$*, as well as a coercion function* $\mathrm{coe}_{A,B} : \mathrm{sub}\,A\,B \to A \to B$*, such that* $\mathrm{coe}_{A,A}\ \mathtt{refl}_A \cong \mathrm{id}_A$ *and* $\mathrm{coe}_{B,C}\ w \circ \mathrm{coe}_{A,B}\ w' \cong \mathrm{coe}_{A,C}(\mathtt{trans}\,w_{A,B}\,w_{B,C})$*. Then $\mathcal{T}$ embeds definitional models of the untyped λ-calculus, and in particular divergent terms.*

Indeed, whenever a context provides inhabitants of both $\mathrm{sub}\,A\,B$ and $\mathrm{sub}\,B\,A$, $\mathrm{coe}_{A,B}$ and $\mathrm{coe}_{B,A}$ provide a definitional isomorphism $A \cong B$. In particular any context inhabiting $\mathrm{sub}\,A\,A \to A$ and $\mathrm{sub}\,A \to A\,A$, for instance an inconsistent one, provides a definitional retraction of $A \to A$ 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

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

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

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

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

$$\textsc{El}\ \frac{\Gamma \vdash A : \mathrm{Type}_i}{\Gamma \vdash A} \qquad \textsc{FunTy}\ \frac{\Gamma \vdash A \qquad \Gamma, x : A \vdash B}{\Gamma \vdash \Pi\, x : A.B} \qquad \textsc{ListTy}\ \frac{\Gamma \vdash A}{\Gamma \vdash \mathbf{List}\ A}$$

$$\textsc{SigTy}\ \frac{\Gamma \vdash A \qquad \Gamma, x : A \vdash B}{\Gamma \vdash \Sigma\, x : A.B} \qquad \textsc{TreeTy}\ \frac{\Gamma \vdash A \qquad \Gamma, x : A \vdash B}{\Gamma \vdash \mathbf{W}\, x : A.B}$$

$$\textsc{IdTy}\ \frac{\Gamma \vdash A \qquad \Gamma \vdash a : A \qquad \Gamma \vdash a' : A}{\Gamma \vdash \mathbf{Id}_A\ a\, a'}$$

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

$$\textsc{Conv}\ \frac{\Gamma \vdash t : A \qquad \Gamma \vdash A \cong B}{\Gamma \vdash t : B} \qquad \textsc{Var}\ \frac{\vdash \Gamma \qquad (x : A \in \Gamma)}{\Gamma \vdash x : A} \qquad \textsc{Sort}\ \frac{\vdash \Gamma}{\Gamma \vdash \mathrm{Type}_i : \mathrm{Type}_{i+1}}$$

$$\textsc{FunUni}\ \frac{\Gamma \vdash A : \mathrm{Type}_i \quad \Gamma, x : A \vdash B : \mathrm{Type}_i}{\Gamma \vdash \Pi\, x : A.B : \mathrm{Type}_i} \qquad \textsc{Abs}\ \frac{\Gamma \vdash A \qquad \Gamma, x : A \vdash B \quad \Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda\, x : A.t : \Pi\, x : A.B} \qquad \textsc{App}\ \frac{\Gamma \vdash t : \Pi\, x : A.B \quad \Gamma \vdash u : A}{\Gamma \vdash t\ u : B[u]}$$

$$\textsc{ListUni}\ \frac{\Gamma \vdash A : \mathrm{Type}_i}{\Gamma \vdash \mathbf{List}\ A : \mathrm{Type}_i} \qquad \textsc{Nil}\ \frac{\Gamma \vdash A}{\Gamma \vdash \varepsilon_A : \mathbf{List}\ A} \qquad \textsc{Cons}\ \frac{\Gamma \vdash A \qquad \Gamma \vdash a : A \quad \Gamma \vdash l : \mathbf{List}\ A}{\Gamma \vdash a ::_A l : \mathbf{List}\ A}$$

$$\textsc{ListInd}\ \frac{\Gamma \vdash A \qquad \Gamma \vdash s : \mathbf{List}\ A \quad \Gamma, x : \mathbf{List}\ A \vdash P \qquad \Gamma \vdash b_\varepsilon : P[\varepsilon_A] \qquad \Gamma, x : A, y : \mathbf{List}\ A, z : P[y] \vdash b_:: : P[x ::_A y]}{\Gamma \vdash \mathrm{ind}_{\mathbf{List}\ A}(s; z.P; b_\varepsilon, x.y.z.b_::) : P[s]}$$

$$\textsc{EmptyUni}\ \frac{}{\Gamma \vdash \mathbf{0} : \mathrm{Type}_0} \qquad \textsc{UnitUni}\ \frac{}{\Gamma \vdash \mathbf{1} : \mathrm{Type}_0} \qquad \textsc{UnitTm}\ \frac{}{\Gamma \vdash () : \mathbf{1}}$$

$$\textsc{EmptyInd}\ \frac{\Gamma \vdash s : \mathbf{0} \qquad \Gamma \vdash P}{\Gamma \vdash \mathrm{ind}_{\mathbf{0}}(s; P) : P} \qquad \textsc{UnitInd}\ \frac{\Gamma \vdash s : \mathbf{1} \qquad \Gamma, z : \mathbf{1} \vdash P \qquad \Gamma \vdash b_() : P[()]}{\Gamma \vdash \mathrm{ind}_{\mathbf{1}}(s; z.P; b_()) : P[s]}$$

$$\textsc{SigUni}\ \frac{\Gamma \vdash A : \mathrm{Type}_i \qquad \Gamma, x : A \vdash B : \mathrm{Type}_i}{\Gamma \vdash \Sigma\, x : A.B : \mathrm{Type}_i} \qquad \textsc{Pair}\ \frac{\Gamma \vdash t : A \qquad \Gamma \vdash u : B[t]}{\Gamma \vdash (t, u)_{x.B} : \Sigma\, x : A.B}$$

$$\textsc{Proj}_1\ \frac{\Gamma \vdash p : \Sigma\, x : A.B}{\Gamma \vdash \pi_1\, p : A} \qquad \textsc{Proj}_2\ \frac{\Gamma \vdash p : \Sigma\, x : A.B}{\Gamma \vdash \pi_2\, p : B[u]}$$

$$\textsc{TreeUni}\ \frac{\Gamma \vdash A : \mathrm{Type}_i \qquad \Gamma, x : A \vdash B : \mathrm{Type}_i}{\Gamma \vdash \mathbf{W}\, x : A.B : \mathrm{Type}_i} \qquad \textsc{Sup}\ \frac{\Gamma, x : A \vdash B \quad \Gamma \vdash a : A \qquad \Gamma \vdash k : B[a] \to \mathbf{W}\, x : A.B}{\Gamma \vdash \mathrm{sup}_{x.B}\, a\, k : \mathbf{W}\, x : A.B}$$

$$\textsc{TreeInd}\ \frac{\Gamma \vdash A \qquad \Gamma, x : A \vdash B \qquad \Gamma \vdash s : \mathbf{W}\, x : A.B \qquad \Gamma, z : \mathbf{W}\, x : A.B \vdash P \quad \Gamma, x : A, y : B[x] \to W x : A.B, h : \Pi\, z : B[x].P[y\ z] \vdash b : P\left[\mathrm{sup}_{x.B}\, x\ y\right]}{\Gamma \vdash \mathrm{ind}_{\mathbf{W}\, x : A.B}(s; z.P; x.y.z.b) : P[s]}$$

$$\textsc{BoolUni}\ \frac{}{\Gamma \vdash \mathbf{B} : \mathrm{Type}_0} \qquad \textsc{True}\ \frac{}{\Gamma \vdash \mathrm{tt} : \mathbf{B}} \qquad \textsc{False}\ \frac{}{\Gamma \vdash \mathrm{ff} : \mathbf{B}}$$

$$\textsc{BoolInd}\ \frac{\Gamma \vdash s : \mathbf{B} \qquad \Gamma, z : \mathbf{B} \vdash P \qquad \Gamma \vdash b_{\mathrm{tt}} : P[\mathrm{tt}] \qquad \Gamma \vdash b_{\mathrm{ff}} : P[\mathrm{ff}]}{\Gamma \vdash \mathrm{ind}_{\mathbf{B}}(s; z.P; b_{\mathrm{tt}}, b_{\mathrm{ff}}) : P[s]}$$

$$\textsc{IdTy}\ \frac{\Gamma \vdash A \qquad \Gamma \vdash a : A \qquad \Gamma \vdash a' : A}{\Gamma \vdash \mathbf{Id}_A\ a\ a'} \qquad \textsc{ReflTm}\ \frac{\Gamma \vdash A \qquad \Gamma \vdash a : A}{\Gamma \vdash \mathrm{refl}_{A,a} : \mathbf{Id}_A\ a\ a}$$

$$\textsc{IdInd}\ \frac{\begin{array}{c}\Gamma \vdash A \qquad \Gamma \vdash a : A \qquad \Gamma \vdash a' : A \\ \Gamma \vdash s : \mathbf{Id}_A\ a\ a' \qquad \Gamma, x\colon A, y\colon A, z\colon \mathbf{Id}_A\ x\ y \vdash P \qquad \Gamma, x\colon A \vdash b : P\big[\mathrm{id}, x, x, \mathrm{refl}_{A,x}\big]\end{array}}{\Gamma \vdash \mathrm{ind}_{\mathbf{Id}_A}(s; x.y.z.P; x.b) : P[\mathrm{id}, a, a', s]}$$

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

$$\textsc{ReflTy}\ \frac{\Gamma \vdash A}{\Gamma \vdash A \cong A} \qquad \textsc{TransTy}\ \frac{\Gamma \vdash A \cong B \qquad \Gamma \vdash B \cong C}{\Gamma \vdash A \cong C} \qquad \textsc{ElC}\ \frac{\Gamma \vdash A \cong A' : \mathrm{Type}_i}{\Gamma \vdash A \cong A'}$$

$$\textsc{FunTyC}\ \frac{\Gamma \vdash A \cong A' \qquad \Gamma, x\colon A \vdash B \cong B'}{\Gamma \vdash \Pi\, x\colon A.B \cong \Pi\, x\colon A'.B'} \qquad \textsc{ListTyC}\ \frac{\Gamma \vdash A \cong A'}{\Gamma \vdash \mathbf{List}\ A \cong \mathbf{List}\ A'}$$

$$\textsc{SigTyC}\ \frac{\Gamma \vdash A \cong A' \qquad \Gamma, x\colon A \vdash B \cong B'}{\Gamma \vdash \Sigma\, x\colon A.B \cong \Sigma\, x\colon A'.B'} \qquad \textsc{TreeTyC}\ \frac{\Gamma \vdash A \cong A' \qquad \Gamma, x\colon A \vdash B \cong B'}{\Gamma \vdash \mathbf{W}\, x\colon A.B \cong \mathbf{W}\, x\colon A'.B'}$$

$$\textsc{IdTy}\ \frac{\begin{array}{c}\Gamma \vdash A \cong A' \qquad \Gamma \vdash t \cong t' : A \\ \Gamma \vdash u \cong u' : A\end{array}}{\Gamma \vdash \mathbf{Id}_A\ t\ u \cong \mathbf{Id}_{A'}\ t'\ u'}$$

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

$$\textsc{Refl}\ \frac{\Gamma \vdash t : A}{\Gamma \vdash t \cong t : A} \qquad \textsc{Trans}\ \frac{\Gamma \vdash t \cong u : A \qquad \Gamma \vdash u \cong v : A}{\Gamma \vdash t \cong v : A} \qquad \textsc{Conv}\ \frac{\begin{array}{c}\Gamma \vdash t \cong t' : A \\ \Gamma \vdash A \cong B\end{array}}{\Gamma \vdash t \cong t' : B}$$

$$\beta\textsc{Fun}\ \frac{\begin{array}{c}\Gamma \vdash A \qquad \Gamma, x\colon A \vdash B \\ \Gamma, x\colon A \vdash t : B \qquad \Gamma \vdash u : A\end{array}}{\Gamma \vdash (\lambda\, x\colon A.t)\ u \cong t[u] : B[u]} \qquad \eta\textsc{Fun}\ \frac{\Gamma, x\colon A \vdash f\ x \cong g\ x : B}{\Gamma \vdash f \cong g : \Pi\, x\colon A.B}$$

$$\beta\textsc{Sig}_1\ \frac{\begin{array}{c}\Gamma \vdash A \qquad \Gamma, x\colon A \vdash B \\ \Gamma \vdash t : A \qquad \Gamma \vdash u : B[t]\end{array}}{\Gamma \vdash \pi_1\,(t, u)_{x.B} \cong t : A} \qquad \beta\textsc{Sig}_2\ \frac{\begin{array}{c}\Gamma \vdash A \qquad \Gamma, x\colon A \vdash B \\ \Gamma \vdash t : A \qquad \Gamma \vdash u : B[t]\end{array}}{\Gamma \vdash \pi_2\,(t, u)_{x.B} \cong u : B[t]}$$

$$\eta\textsc{Sig}\ \frac{\Gamma \vdash A \qquad \Gamma, x\colon A \vdash B \qquad \Gamma \vdash p : \Sigma\, x\colon A.B}{\Gamma \vdash p \cong (\pi_1\, p, \pi_2\, p)_{x.B} : \Sigma\, x\colon A.B}$$

$$\iota\textsc{Nil}\ \frac{\begin{array}{c}\Gamma \vdash A \qquad \Gamma, x\colon \mathbf{List}\ A \vdash P \\ \Gamma \vdash b_\varepsilon : P[\varepsilon_A] \qquad \Gamma, x\colon A, y\colon \mathbf{List}\ A, z\colon P[y] \vdash b_:: : P[x ::_A y]\end{array}}{\Gamma \vdash \mathrm{ind}_{\mathbf{List}\ A}(\varepsilon_A; z.P; b_\varepsilon, x.y.z.b_::) \cong b_\varepsilon : P[\varepsilon_A]}$$

$$\iota\textsc{Cons}\ \frac{\Gamma \vdash A \qquad \Gamma \vdash a : A \qquad \Gamma \vdash l : \mathbf{List}\ A \qquad \Gamma, x\colon \mathbf{List}\ A \vdash P \qquad \Gamma \vdash b_\varepsilon : P[\varepsilon_A] \qquad \Gamma, x\colon A, y\colon \mathbf{List}\ A, z\colon P[y] \vdash b_:: : P[x ::_A y]}{\Gamma \vdash \mathrm{ind}_{\mathbf{List}\ A}(a ::_A l; z.P; b_\varepsilon, x.y.z.b_::) \cong b_::[\mathrm{id}, a, l, \mathrm{ind}_{\mathbf{List}\ A}(l; z.P; b_\varepsilon, x.y.z.b_::)] : P[a ::_A l]}$$

$$\iota\textsc{True}\ \frac{\Gamma, z\colon \mathbf{B} \vdash P \qquad \Gamma \vdash b_{\mathrm{tt}} : P[\mathrm{tt}] \qquad \Gamma \vdash b_{\mathrm{ff}} : P[\mathrm{ff}]}{\Gamma \vdash \mathrm{ind}_{\mathbf{B}}(\mathrm{tt}; z.P; b_{\mathrm{tt}}, b_{\mathrm{ff}}) \cong b_{\mathrm{tt}} : P[s]} \qquad \iota\textsc{False}\ \frac{\Gamma, z\colon \mathbf{B} \vdash P \qquad \Gamma \vdash b_{\mathrm{tt}} : P[\mathrm{tt}] \qquad \Gamma \vdash b_{\mathrm{ff}} : P[\mathrm{ff}]}{\Gamma \vdash \mathrm{ind}_{\mathbf{B}}(\mathrm{ff}; z.P; b_{\mathrm{tt}}, b_{\mathrm{ff}}) \cong b_{\mathrm{ff}} : P[s]}$$

$$\iota\textsc{Tree}\ \frac{\Gamma, x\colon A \vdash B \qquad \Gamma \vdash a : A \qquad \Gamma \vdash k : B[a] \to \mathbf{W}\, x\colon A.B \qquad \Gamma, z\colon \mathbf{W}\, x\colon A.B \vdash P \qquad \Gamma, x\colon A, y\colon B[x] \to W x\colon A.B, h\colon \Pi\, z\colon B[x].P[y\ z] \vdash b : P\left[\sup_{x.B} x\ y\right]}{\Gamma \vdash \mathrm{ind}_{\mathbf{W}\, x:A.B}(\sup_{x.B} a\ k; z.P; x.y.z.b) \cong b[\mathrm{id}, a, k, (\lambda\, z\colon B[x].\, \mathrm{ind}_{\mathbf{W}\, x:A.B}(k\, z; z.P; x.y.z.b))] : P[s]}$$

$$\iota\textsc{Refl}\ \frac{\Gamma \vdash A \qquad \Gamma \vdash a : A \qquad \Gamma, x\colon A, y\colon A, z\colon \mathbf{Id}_A\ x\ y \vdash P \qquad \Gamma, x\colon A \vdash b : P\left[\mathrm{id}, x, x, \mathrm{refl}_{A,x}\right]}{\Gamma \vdash \mathrm{ind}_{\mathbf{Id}_A}(\mathrm{refl}_{A,a}; x.y.z.P; x.b) \cong b[a] : P\left[\mathrm{id}, a, a, \mathrm{refl}_{A,a}\right]}$$

$$\textsc{FunCong}\ \frac{\Gamma \vdash A \cong A' : \mathrm{Type}_i \qquad \Gamma, x\colon A \vdash B \cong B' : \mathrm{Type}_i}{\Gamma \vdash \Pi\, x\colon A.B \cong \Pi\, x\colon A'.B' : \mathrm{Type}_i} \qquad \text{other congruences omitted}$$

B.2 Algorithmic MLTT

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

$$\beta\textsc{Fun}\ \frac{}{(\lambda\, x\colon A.t)\ u \rightsquigarrow^1 t[u]} \qquad \beta\textsc{Sig}_1\ \frac{}{\pi_1\ (t, u)_{x.B} \rightsquigarrow^1 t} \qquad \beta\textsc{Sig}_2\ \frac{}{\pi_2\ (t, u)_{x.B} \rightsquigarrow^1 u}$$

$$\iota\textsc{RedNil}\ \frac{}{\mathrm{ind}_{\mathbf{List}\ A}(\varepsilon_A; x.P; b_\varepsilon, x.y.z.b_::) \rightsquigarrow^1 b_\varepsilon}$$

$$\iota\textsc{RedCons}\ \frac{}{\mathrm{ind}_{\mathbf{List}\ A}(a ::_A l; x.P; b_\varepsilon, x.y.z.b_::) \rightsquigarrow^1 b_::[\mathrm{id}, a, l, \mathrm{ind}_{\mathbf{List}\ A}(l; z.P; b_\varepsilon, x.y.z.b_::)]}$$

$$\iota\textsc{Tree}\ \frac{}{\mathrm{ind}_{\mathbf{W}\, x:A.B}(\sup_{x.B} a\ k; z.P; x.y.z.b) \rightsquigarrow^1 b[\mathrm{id}, a, k, (\lambda\, z\colon B[x].\, \mathrm{ind}_{\mathbf{W}\, x:A.B}(k\, z; z.P; x.y.z.b))]}$$

$$\iota\textsc{True}\ \frac{}{\mathrm{ind}_{\mathbf{B}}(\mathrm{tt}; z.P; b_{\mathrm{tt}}, b_{\mathrm{ff}}) \rightsquigarrow^1 b_{\mathrm{tt}}} \qquad \iota\textsc{False}\ \frac{}{\mathrm{ind}_{\mathbf{B}}(\mathrm{ff}; z.P; b_{\mathrm{tt}}, b_{\mathrm{ff}}) \rightsquigarrow^1 b_{\mathrm{ff}}}$$

$$\iota\textsc{Refl}\ \frac{}{\mathrm{ind}_{\mathbf{Id}_A}(\mathrm{refl}_{A,a}; x.z.P; x.b) \rightsquigarrow^1 b[a]} \qquad \textsc{RedApp}\ \frac{t \rightsquigarrow^1 t'}{t\ u \rightsquigarrow^1 t'\ u} \qquad \textsc{RedSig}_1\ \frac{t \rightsquigarrow^1 t'}{\pi_1\, t \rightsquigarrow^1 \pi_1\, t'}$$

$$\textsc{RedSig}_2\ \frac{t \rightsquigarrow^1 t'}{\pi_2\, t \rightsquigarrow^1 \pi_2\, t'} \qquad \textsc{RedInd}\ \frac{t \rightsquigarrow^1 t'}{\mathrm{ind}_T(t; P; \vec{b}) \rightsquigarrow^1 \mathrm{ind}_T(t'; P; \vec{b})}$$

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

$$\textsc{RedBase}\ \frac{}{t \rightsquigarrow^\star t} \qquad \textsc{RedStep}\ \frac{t \rightsquigarrow^\star t' \qquad t' \rightsquigarrow^1 t''}{t \rightsquigarrow^\star t''}$$

$\boxed{\text{nf } f}$ $\stackrel{\text{def}}{=}$ $n \mid \text{Type}_i \mid \Pi\, x{:}\, t.t \mid \lambda\, x{:}\, t.t \mid \mathbf{List}\, t \mid \varepsilon_t \mid t ::_{\text{t}} t \mid$ weak-head normal forms
$\Sigma\, x{:}\, t.t \mid \pi_1\, t \mid \pi_2\, t \mid \mathbf{W}\, x{:}\, t.t \mid \text{sup}_t\, t\, t \mid \mathbf{0} \mid \mathbf{1} \mid () \mid$
$\mathbf{B} \mid \text{tt} \mid \text{ff} \mid \mathbf{Id}_t\ t\, t' \mid \text{refl}_{t,t}$

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

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

$$\textsc{FunTy}\ \frac{\Gamma \vdash A \lhd \qquad \Gamma, x{:}\, A \vdash B \lhd}{\Gamma \vdash \Pi\, x{:}\, A.B \lhd} \qquad \textsc{ListTy}\ \frac{\Gamma \vdash A \lhd}{\Gamma \vdash \mathbf{List}\, A \lhd}$$

$$\textsc{SigTy}\ \frac{\Gamma \vdash A \lhd \qquad \Gamma, x{:}\, A \vdash B \lhd}{\Gamma \vdash \Sigma\, x{:}\, A.B \lhd} \qquad \textsc{TreeTy}\ \frac{\Gamma \vdash A \lhd \qquad \Gamma, x{:}\, A \vdash B \lhd}{\Gamma \vdash \mathbf{W}\, x{:}\, A.B \lhd} \qquad \textsc{EmptyTy}\ \frac{}{\Gamma \vdash \mathbf{0} \lhd}$$

$$\textsc{UnitTy}\ \frac{}{\Gamma \vdash \mathbf{1} \lhd} \qquad \textsc{BoolTy}\ \frac{}{\Gamma \vdash \mathbf{B} \lhd} \qquad \textsc{IdTy}\ \frac{\Gamma \vdash A \lhd \qquad \Gamma \vdash a \lhd A \qquad \Gamma \vdash a' \lhd A}{\Gamma \vdash \mathbf{Id}_A\ a\, a' \lhd}$$

$$\textsc{El}\ \frac{\Gamma \vdash A \rhd_{\text{h}} \text{Type}_i \qquad A \text{ is not a canonical form}}{\Gamma \vdash A \lhd}$$

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

$$\textsc{Sort}\ \frac{}{\Gamma \vdash \text{Type}_i \rhd \text{Type}_{i+1}} \qquad \textsc{Var}\ \frac{(x{:}\, T) \in \Gamma}{\Gamma \vdash x \rhd T} \qquad \textsc{Fun}\ \frac{\Gamma \vdash A \rhd_{\text{h}} \text{Type}_i \qquad \Gamma, x{:}\, A \vdash B \rhd_{\text{h}} \text{Type}_i}{\Gamma \vdash \Pi\, x{:}\, A.B \rhd \text{Type}_i}$$

$$\textsc{Abs}\ \frac{\Gamma \vdash A \lhd \qquad \Gamma, x{:}\, A \vdash t \rhd B}{\Gamma \vdash \lambda\, x{:}\, A.t \rhd \Pi\, x{:}\, A.B} \qquad \textsc{App}\ \frac{\Gamma \vdash t \rhd_{\text{h}} \Pi\, x{:}\, A.B \qquad \Gamma \vdash u \lhd A}{\Gamma \vdash t\, u \rhd B[u]}$$

$$\textsc{List}\ \frac{\Gamma \vdash A \rhd_{\text{h}} \text{Type}_i}{\Gamma \vdash \mathbf{List}\, A \rhd \text{Type}_i} \qquad \textsc{Nil}\ \frac{\Gamma \vdash A \lhd}{\Gamma \vdash \varepsilon_A \rhd \mathbf{List}\, A} \qquad \textsc{Cons}\ \frac{\Gamma \vdash A \lhd \qquad \Gamma \vdash a \lhd A \qquad \Gamma \vdash l \lhd \mathbf{List}\, A}{\Gamma \vdash a ::_{\text{A}} l \rhd \mathbf{List}\, A}$$

$$\textsc{ListInd}\ \frac{\Gamma \vdash A \lhd \qquad \Gamma \vdash s \lhd \mathbf{List}\, A \qquad \Gamma, x{:}\, \mathbf{List}\, A \vdash P \rhd \qquad \Gamma \vdash b_\varepsilon \lhd P[\varepsilon_A] \qquad \Gamma, x{:}\, A, y{:}\, \mathbf{List}\, A, z{:}\, P[y] \vdash b_{::} \lhd P[x ::_{\text{A}} y]}{\Gamma \vdash \text{ind}_{\mathbf{List}\, A}(s; z.P; b_\varepsilon, x.y.z.b_{::}) \rhd P[s]}$$

$$\textsc{Empty}\ \frac{}{\Gamma \vdash \mathbf{0} \rhd \text{Type}_0} \qquad \textsc{EmptyInd}\ \frac{\Gamma \vdash s \lhd \mathbf{0} \qquad \Gamma \vdash P \lhd}{\Gamma \vdash \text{ind}_{\mathbf{0}}(s; P) \rhd P} \qquad \textsc{UnitUni}\ \frac{}{\Gamma \vdash \mathbf{1} \rhd \text{Type}_0}$$

$$\textsc{UnitTm}\ \frac{}{\Gamma \vdash () \triangleright \mathbf{1}} \qquad \textsc{UnitInd}\ \frac{\Gamma \vdash s \triangleleft \mathbf{1} \qquad \Gamma, z\colon \mathbf{1} \vdash P \triangleleft \qquad \Gamma \vdash b_{()} \triangleleft P[()]}{\Gamma \vdash \mathrm{ind}_{\mathbf{1}}(s; z.P; b_{()}) \triangleright P[s]}$$

$$\textsc{Sig}\ \frac{\Gamma \vdash A \triangleright_{\mathrm{h}} \mathrm{Type}_i \qquad \Gamma, x\colon A \vdash B \triangleright_{\mathrm{h}} \mathrm{Type}_i}{\Gamma \vdash \Sigma\, x\colon A.B \triangleright \mathrm{Type}_i} \qquad \textsc{Pair}\ \frac{\Gamma \vdash t \triangleright A \qquad \Gamma, x\colon A \vdash B \triangleleft \qquad \Gamma \vdash u \triangleleft B[t]}{\Gamma \vdash (t, u)_{x.B} \triangleright \Sigma\, x\colon A.B}$$

$$\textsc{Proj}_1\ \frac{\Gamma \vdash p \triangleright_{\mathrm{h}} \Sigma\, x\colon A.B}{\Gamma \vdash \pi_1\, p \triangleright A} \qquad \textsc{Proj}_2\ \frac{\Gamma \vdash p \triangleright \Sigma\, x\colon A.B}{\Gamma \vdash \pi_2\, p \triangleright B[u]}$$

$$\textsc{Tree}\ \frac{\Gamma \vdash A \triangleright_{\mathrm{h}} \mathrm{Type}_i \qquad \Gamma, x\colon A \vdash B \triangleright_{\mathrm{h}} \mathrm{Type}_i}{\Gamma \vdash \mathbf{W}\, x\colon A.B \triangleright \mathrm{Type}_i}$$

$$\textsc{Sup}\ \frac{\Gamma \vdash a \triangleright A \qquad \Gamma, x\colon A \vdash B \triangleleft \qquad \Gamma \vdash k \triangleleft B[a] \to \mathbf{W}\, x\colon A.B}{\Gamma \vdash \mathrm{sup}_{x.B}\, a\, k \triangleright \mathbf{W}\, x\colon A.B}$$

$$\textsc{TreeInd}\ \frac{\begin{array}{c}\Gamma \vdash A \triangleleft \qquad \Gamma, x\colon A \vdash B \triangleleft \qquad \Gamma \vdash s \triangleleft \mathbf{W}\, x\colon A.B \qquad \Gamma, z\colon \mathbf{W}\, x\colon A.B \vdash P \triangleleft \\ \Gamma, x\colon A, y\colon B[x] \to W\, x\colon A.B, h\colon \Pi\, z\colon B[x].P[y\, z] \vdash b \triangleleft P\left[\mathrm{sup}_{x.B}\, x\, y\right]\end{array}}{\Gamma \vdash \mathrm{ind}_{\mathbf{W}\, x:A.B}(s; z.P; x.y.z.b) \triangleright P[s]}$$

$$\textsc{BoolUni}\ \frac{}{\Gamma \vdash \mathbf{B} \triangleright \mathrm{Type}_0} \qquad \textsc{True}\ \frac{}{\Gamma \vdash \mathrm{tt} \triangleright \mathbf{B}} \qquad \textsc{False}\ \frac{}{\Gamma \vdash \mathrm{ff} \triangleright \mathbf{B}}$$

$$\textsc{BoolInd}\ \frac{\Gamma \vdash s \triangleleft \mathbf{B} \qquad \Gamma, z\colon \mathbf{B} \vdash P \triangleleft \qquad \Gamma \vdash b_{\mathrm{tt}} \triangleleft P[\mathrm{tt}] \qquad \Gamma \vdash b_{\mathrm{ff}} \triangleleft P[\mathrm{ff}]}{\Gamma \vdash \mathrm{ind}_{\mathbf{B}}(s; z.P; b_{\mathrm{tt}}, b_{\mathrm{ff}}) \triangleright P[s]}$$

$$\textsc{IdTy}\ \frac{\Gamma \vdash A \triangleright_{\mathrm{h}} \mathrm{Type}_i \qquad \Gamma \vdash a \triangleleft A \qquad \Gamma \vdash a' \triangleleft A}{\Gamma \vdash \mathbf{Id}_A\ a\, a' \triangleright \mathrm{Type}_i} \qquad \textsc{ReflTm}\ \frac{\Gamma \vdash A \triangleleft \qquad \Gamma \vdash a \triangleleft A}{\Gamma \vdash \mathrm{refl}_{A,a} \triangleright \mathbf{Id}_A\ a\, a}$$

$$\textsc{IdInd}\ \frac{\begin{array}{c}\Gamma \vdash A \triangleleft \\ \Gamma \vdash s \triangleright_{\mathrm{h}} \mathbf{Id}_{A'}\ a\, a' \qquad \Gamma, x, y\colon A, z\colon \mathbf{Id}_A\ x\, y \vdash P \triangleleft \qquad \Gamma, x\colon A \vdash b \triangleleft P\left[\mathrm{id}, x, x, \mathrm{refl}_{A,x}\right]\end{array}}{\Gamma \vdash \mathrm{ind}_{\mathbf{Id}_A}(s; x.y.z.P; x.b) \triangleright P[\mathrm{id}, a, a', s]}$$

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

$$\textsc{Check}\ \frac{\Gamma \vdash t \triangleright T' \qquad \Gamma \vdash T' \cong T \triangleleft}{\Gamma \vdash t \triangleleft T}$$

$\boxed{\Gamma \vdash t \triangleright_{\mathrm{h}} T}$ Term t infers the reduced type T

$$\textsc{InfRed}\ \frac{\Gamma \vdash t \triangleright T \qquad \Gamma \vdash T \rightsquigarrow^{\star} T'}{\Gamma \vdash t \triangleright_{\mathrm{h}} T'}$$

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

$$\textsc{TyRed}\ \frac{T \rightsquigarrow^\star U \qquad T' \rightsquigarrow^\star U' \qquad \Gamma \vdash U \cong_{\mathrm{h}} U' \lhd}{\Gamma \vdash T \cong T' \lhd}$$

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

$$\textsc{TmRed}\ \frac{t \rightsquigarrow^\star u \qquad t' \rightsquigarrow^\star u' \qquad T \rightsquigarrow^\star U \qquad \Gamma \vdash u \cong_{\mathrm{h}} u' \lhd U}{\Gamma \vdash t \cong t' \lhd T}$$

$\boxed{\Gamma \vdash T \cong_{\mathrm{h}} T' \lhd}$ Reduced types T and T' are convertible

$$\textsc{CUniTy}\ \frac{}{\Gamma \vdash \mathrm{Type}_i \cong_{\mathrm{h}} \mathrm{Type}_i \lhd} \qquad \textsc{CProdTy}\ \frac{\Gamma \vdash A \cong A' \lhd \qquad \Gamma, x\colon A' \vdash B \cong B' \lhd}{\Gamma \vdash \Pi\, x\colon A.B \cong_{\mathrm{h}} \Pi\, x\colon A'.B' \lhd}$$

$$\textsc{CListTy}\ \frac{\Gamma \vdash A \cong A' \lhd}{\Gamma \vdash \mathbf{List}\ A \cong_{\mathrm{h}} \mathbf{List}\ A' \lhd} \qquad \textsc{CSigTy}\ \frac{\Gamma \vdash A \cong A' \lhd \qquad \Gamma, x\colon A \vdash B \cong B' \lhd}{\Gamma \vdash \Sigma\, x\colon A.B \cong_{\mathrm{h}} \Sigma\, x\colon A'.B' \lhd}$$

$$\textsc{CTreeTy}\ \frac{\Gamma \vdash A \cong A' \lhd \quad \Gamma, x\colon A \vdash B \cong B' \lhd}{\Gamma \vdash \mathbf{W}\, x\colon A.B \cong_{\mathrm{h}} \mathbf{W}\, x\colon A'.B' \lhd} \qquad \textsc{CIdTy}\ \frac{\Gamma \vdash A \cong A' \lhd \qquad \Gamma \vdash t \cong t' \lhd A \quad \Gamma \vdash u \cong u' \lhd A}{\Gamma \vdash \mathbf{Id}_A\ t\ u \cong_{\mathrm{h}} \mathbf{Id}_{A'}\ t'\ u' \lhd}$$

$$\textsc{CReflTy}\ \frac{T \text{ is } \mathbf{0}, \mathbf{1} \text{ or } \mathbf{B}}{\Gamma \vdash T \cong_{\mathrm{h}} T \lhd} \qquad \textsc{NeuTy}\ \frac{\Gamma \vdash n \approx n' \rhd T}{\Gamma \vdash n \cong_{\mathrm{h}} n' \lhd}$$

$\boxed{\Gamma \vdash t \cong_{\mathrm{h}} t' \lhd A}$ Reduced terms t and t' are convertible at type A

$$\textsc{CUni}\ \frac{}{\Gamma \vdash \mathrm{Type}_i \cong_{\mathrm{h}} \mathrm{Type}_j \lhd \mathrm{Type}_k} \qquad \textsc{CFun}\ \frac{\Gamma \vdash A \cong A' \lhd \mathrm{Type}_i \quad \Gamma, x\colon A' \vdash B \cong B' \lhd \mathrm{Type}_i}{\Gamma \vdash \Pi\, x\colon A.B \cong_{\mathrm{h}} \Pi\, x\colon A'.B' \lhd \mathrm{Type}_i}$$

$$\textsc{CFunEta}\ \frac{\Gamma, x\colon A \vdash f\ x \cong f'\ x \lhd B}{\Gamma \vdash f \cong_{\mathrm{h}} f' \lhd \Pi\, x\colon A.B} \qquad \textsc{CSig}\ \frac{\Gamma \vdash A \cong A' \lhd \mathrm{Type}_i \quad \Gamma, x\colon A' \vdash B \cong B' \lhd \mathrm{Type}_i}{\Gamma \vdash \Sigma\, x\colon A.B \cong_{\mathrm{h}} \Sigma\, x\colon A'.B' \lhd \mathrm{Type}_i}$$

$$\textsc{CSigEta}\ \frac{\Gamma \vdash \pi_1\, p \cong \pi_1\, p' \lhd A \quad \Gamma \vdash \pi_2\, p \cong \pi_2\, p' \lhd B[\pi_1\, p]}{\Gamma \vdash p \cong_{\mathrm{h}} p' \lhd \Sigma\, x\colon A.B} \qquad \textsc{CList}\ \frac{\Gamma \vdash A \cong A' \lhd \mathrm{Type}_i}{\Gamma \vdash \mathbf{List}\ A \cong_{\mathrm{h}} \mathbf{List}\ A' \lhd \mathrm{Type}_i}$$

$$\textsc{CNil}\ \frac{}{\Gamma \vdash \varepsilon_A \cong_{\mathrm{h}} \varepsilon_{A'} \lhd \mathbf{List}\ A''} \qquad \textsc{CCons}\ \frac{\Gamma \vdash a \cong a' \lhd A'' \qquad \Gamma \vdash l \cong l' \lhd \mathbf{List}\ A''}{\Gamma \vdash a ::_A l \cong_{\mathrm{h}} a' ::_{A'} l' \lhd \mathbf{List}\ A''}$$

$$\textsc{CReflUni}\ \frac{T \text{ is } \mathbf{0}, \mathbf{1} \text{ or } \mathbf{B}}{\Gamma \vdash T \cong_{\mathrm{h}} T \lhd \mathrm{Type}_0} \qquad \textsc{CUnitK}\ \frac{}{\Gamma \vdash () \cong_{\mathrm{h}} () \lhd \mathbf{1}} \qquad \textsc{CReflBool}\ \frac{t \text{ is } \mathrm{tt} \text{ or } \mathrm{ff}}{\Gamma \vdash t \cong_{\mathrm{h}} t \lhd \mathbf{B}}$$

$$\textsc{CSup}\ \frac{\Gamma \vdash a \cong a' \lhd A'' \qquad \Gamma \vdash k \cong k' \lhd B''[a] \to \mathbf{W}\, x{:}\, A''.B''}{\Gamma \vdash \mathsf{sup}_{x.B}\, a\, k \cong_{\mathrm{h}} \mathsf{sup}_{x.B'}\, a'\, k' \lhd \mathbf{W}\, x{:}\, A''.B''}$$

$$\textsc{ReflRefl}\ \frac{}{\Gamma \vdash \mathsf{refl}_{A,a} \cong \mathsf{refl}_{A',a'} \lhd \mathbf{Id}_{A''}\ t\, u} \qquad \textsc{NeuNeu}\ \frac{\Gamma \vdash n \approx n' \rhd S \qquad \mathrm{ne}\, M}{\Gamma \vdash n \cong_{\mathrm{h}} n' \lhd M}$$

$$\textsc{NeuPos}\ \frac{\Gamma \vdash n \approx n' \rhd S \qquad T \text{ is } \mathsf{Type}_i, \mathbf{0}, \mathbf{1}, \mathbf{B}, \mathbf{List}\, A, \mathbf{W}\, x{:}\, A.B \text{ or } \mathbf{Id}_A\ a\, a'}{\Gamma \vdash n \cong_{\mathrm{h}} n' \lhd T}$$

$\boxed{\Gamma \vdash t \approx_{\mathrm{h}} t' \rhd T}$ Neutrals t and t' are comparable, inferring the reduced type T

$$\textsc{NRed}\ \frac{\Gamma \vdash n \approx n' \rhd T \qquad T \rightsquigarrow^{\star} S}{\Gamma \vdash n \approx_{\mathrm{h}} n' \rhd S}$$

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

$$\textsc{NVar}\ \frac{(x{:}\, T \in \Gamma)}{\Gamma \vdash x \approx x \rhd T} \qquad \textsc{NApp}\ \frac{\Gamma \vdash n \approx_{\mathrm{h}} n' \rhd \Pi\, x{:}\, A.B \qquad \Gamma \vdash u \cong u' \lhd A}{\Gamma \vdash n\, u \approx n'\, u' \rhd B[u]}$$

$$\textsc{NListInd}\ \frac{\begin{array}{c}\Gamma \vdash A \cong A' \lhd \qquad \Gamma \vdash s \approx s' \rhd S \qquad \Gamma, z{:}\, \mathbf{List}\, A \vdash P \cong P' \lhd \\ \Gamma \vdash b_{\varepsilon} \cong b'_{\varepsilon} \lhd P[\varepsilon_A] \qquad \Gamma, x{:}\, A, y{:}\, \mathbf{List}\, A, z{:}\, P[y] \vdash b_{::} \cong b'_{::} \lhd P[x ::_A y]\end{array}}{\Gamma \vdash \mathsf{ind}_{\mathbf{List}\, A}(s; z.P; b_{\varepsilon}, x.y.z.b_{::}) \approx \mathsf{ind}_{\mathbf{List}\, A'}(s'; z.P'; b'_{\varepsilon}, x.y.z.b'_{::}) \rhd P[s]}$$

$$\textsc{NEmptyInd}\ \frac{\Gamma \vdash s \approx_{\mathrm{h}} s' \rhd \mathbf{0} \qquad \Gamma \vdash P \cong P' \lhd}{\Gamma \vdash \mathsf{ind}_{\mathbf{0}}(s; P) \approx \mathsf{ind}_{\mathbf{0}}(s'; P') \rhd P}$$

$$\textsc{NUnitInd}\ \frac{\Gamma \vdash s \approx_{\mathrm{h}} s' \rhd \mathbf{1} \qquad \Gamma, z{:}\, \mathbf{1} \vdash P \cong P' \lhd \qquad \Gamma \vdash b \cong b' \lhd P[()]}{\Gamma \vdash \mathsf{ind}_{\mathbf{1}}(s; z.P; b) \approx \mathsf{ind}_{\mathbf{0}}(s'; z.P'; b') \rhd P[s]}$$

$$\textsc{NSig}_1\ \frac{\Gamma \vdash n \approx_{\mathrm{h}} n' \rhd \Sigma\, x{:}\, A.B}{\Gamma \vdash \pi_1\, n \approx \pi_1\, n' \rhd A} \qquad \textsc{NSig}_2\ \frac{\Gamma \vdash n \approx_{\mathrm{h}} n' \rhd \Sigma\, x{:}\, A.B}{\Gamma \vdash \pi_2\, n \approx \pi_2\, n' \rhd B[\pi_1\, n]}$$

$$\textsc{NTreeInd}\ \frac{\begin{array}{c}\Gamma \vdash A \cong A' \lhd \\ \Gamma, x{:}\, A \vdash B \cong B' \lhd \qquad \Gamma \vdash s \approx s' \rhd S \qquad \Gamma, z{:}\, \mathbf{W}\, x{:}\, A.B \vdash P \cong P' \lhd \\ \Gamma, x{:}\, A, y{:}\, B[x] \to W\, x{:}\, A.B, h{:}\, \Pi\, z{:}\, B[x].P[y\, z] \vdash b \cong b' \lhd P\left[\mathsf{sup}_{x.B}\, x\, y\right]\end{array}}{\Gamma \vdash \mathsf{ind}_{\mathbf{W}\, x:A.B}(s; z.P; x.y.z.b) \approx \mathsf{ind}_{\mathbf{W}\, x:A'.B'}(s'; z.P'; x.y.z.b') \rhd P[s]}$$

$$\textsc{NBoolInd}\ \frac{\begin{array}{c}\Gamma \vdash s \approx_{\mathrm{h}} s' \rhd \mathbf{B} \\ \Gamma, z{:}\, \mathbf{B} \vdash P \cong P' \lhd \qquad \Gamma \vdash b_{\mathsf{tt}} \cong b'_{\mathsf{tt}} \lhd P[\mathsf{tt}] \qquad \Gamma \vdash b_{\mathsf{ff}} \cong b_{\mathsf{ff}} \lhd P[\mathsf{ff}]\end{array}}{\Gamma \vdash \mathsf{ind}_{\mathbf{B}}(s; z.P; b_{\mathsf{tt}}, b_{\mathsf{ff}}) \approx \mathsf{ind}_{\mathbf{B}}(s'; z.P'; b'_{\mathsf{tt}}, b'_{\mathsf{ff}}) \rhd P[s]}$$

$$\textsc{IdInd}\ \frac{\begin{array}{c}\Gamma \vdash A \cong A' \lhd \qquad \Gamma \vdash s \approx_{\mathrm{h}} s' \rhd \mathbf{Id}_{A''}\ a\, a' \\ \Gamma, x{:}\, A, y{:}\, A, z{:}\, \mathbf{Id}_A\ x\, y \vdash P \cong P' \lhd \qquad \Gamma, x{:}\, A \vdash b \cong b' \lhd P\left[\mathsf{id}, x, x, \mathsf{refl}_{A,x}\right]\end{array}}{\Gamma \vdash \mathsf{ind}_{\mathbf{Id}_A}(s; x.y.z.P; x.b) \approx \mathsf{ind}_{\mathbf{Id}_{A'}}(s'; x.y.z.P'; x.b') \rhd P[\mathsf{id}, a, a', s]}$$

B.3 Declarative $\mathsf{MLTT}_{\mathrm{map}}$

Extend the rules of appendix B.1.

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

$$\textsc{Map}\ \frac{\Gamma \vdash_{\mathrm{map}} X, Y : \mathrm{dom}(F) \quad \Gamma \vdash_{\mathrm{map}} f : \mathrm{Hom}_F(X,Y) \quad \Gamma \vdash_{\mathrm{map}} t : F\,X}{\Gamma \vdash_{\mathrm{map}} \mathrm{map}_F\, f\, t : F\,Y} \qquad \textsc{MapId}\ \frac{\Gamma \vdash_{\mathrm{map}} X : \mathrm{dom}(F) \quad \Gamma \vdash_{\mathrm{map}} t : F\,X}{\Gamma \vdash_{\mathrm{map}} \mathrm{map}_F\, \mathrm{id}_{F\,X}\, t \cong t : F\,X}$$

$$\textsc{MapComp}\ \frac{\Gamma \vdash_{\mathrm{map}} X, Y, Z : \mathrm{dom}(F) \quad \Gamma \vdash_{\mathrm{map}} g : \mathrm{Hom}_F(X,Y) \quad \Gamma \vdash_{\mathrm{map}} f : \mathrm{Hom}_F(Y,Z) \quad \Gamma \vdash_{\mathrm{map}} t : F\,X}{\Gamma \vdash_{\mathrm{map}} \mathrm{map}_F\ f\,(\mathrm{map}_F\, g\, t) \cong \mathrm{map}_F(f \circ g)\, t : F\,Z}$$

$$\textsc{Map-Cong}\ \frac{\Gamma \vdash_{\mathrm{map}} X, Y : \mathrm{dom}(F) \quad \Gamma \vdash_{\mathrm{map}} f \cong f' : \mathrm{Hom}_F(X,Y) \quad \Gamma \vdash_{\mathrm{map}} t \cong t' : F\,X}{\Gamma \vdash_{\mathrm{map}} \mathrm{map}_F\ f\, t \cong \mathrm{map}_F\ f'\, t' : F\,Y}$$

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

$$\textsc{mapFun}\ \frac{\Gamma \vdash_{\mathrm{map}} (f,g) : \mathrm{Hom}_\Pi((A,B),(A',B')) \quad \Gamma \vdash_{\mathrm{map}} h : \Pi\, x{:}\, A.B \quad \Gamma \vdash_{\mathrm{map}} a : A'}{\Gamma \vdash_{\mathrm{map}} \mathrm{map}_\Pi\,(f,g)\, h\, a \cong g\,(h\,(f\,a)) : B'}$$

$$\textsc{mapSig}_1\ \frac{\Gamma \vdash_{\mathrm{map}} (f,g) : \mathrm{Hom}_\Sigma((A,B),(A',B')) \quad \Gamma \vdash_{\mathrm{map}} p : \Sigma\, x{:}\, A.B}{\Gamma \vdash_{\mathrm{map}} \pi_1\,(\mathrm{map}_\Sigma\,(f,g)\,p) \cong f\,(\pi_1\,p) : A'}$$

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

$$\textsc{mapListNil}\ \frac{\Gamma \vdash_{\mathrm{map}} f : \mathrm{Hom}_{\mathbf{List}}(A,A')}{\Gamma \vdash_{\mathrm{map}} \mathrm{map}_{\mathbf{List}}\ f\, \varepsilon_A \cong \varepsilon_{A'} : \mathbf{List}\,A'}$$

$$\textsc{mapListCons}\ \frac{\Gamma \vdash_{\mathrm{map}} f : \mathrm{Hom}_{\mathbf{List}}(A,A') \quad \Gamma \vdash_{\mathrm{map}} hd : A \quad \Gamma \vdash_{\mathrm{map}} tl : \mathbf{List}\,A}{\Gamma \vdash_{\mathrm{map}} \mathrm{map}_{\mathbf{List}}\ f\,(hd ::_{\mathrm{A}} tl) \cong (f\,hd) ::_{\mathrm{A}'} (\mathrm{map}_{\mathbf{List}}\, f\, tl) : \mathbf{List}\,A'}$$

$$\textsc{mapW}\ \frac{\Gamma \vdash_{\mathrm{map}} (f,g) : \mathrm{Hom}_{\mathbf{W}}((A,B),(A',B')) \quad \Gamma \vdash_{\mathrm{map}} a : A \quad \Gamma \vdash_{\mathrm{map}} k : B\,a \to \mathbf{W}\,x{:}\,A.B}{\Gamma \vdash_{\mathrm{map}} \mathrm{map}_{\mathbf{W}}(f,g)\,(\sup_{x.B} a\, k) \cong \sup_{x.B'}(f\,a)\ (\lambda\, x{:}\, B'[f\,a].\, \mathrm{map}_{\mathbf{W}}(f,g)\,(k\,(g\,x))) : \mathbf{W}\,x{:}\,A'.B'}$$

$$\textsc{mapId}\ \frac{\Gamma \vdash_{\mathrm{map}} f : \mathrm{Hom}_{\mathbf{Id}}(A,A') \quad \Gamma \vdash_{\mathrm{map}} a : A}{\Gamma \vdash_{\mathrm{map}} \mathrm{map}_{\mathbf{Id}}\ f\ \mathrm{refl}_{A,a} \cong \mathrm{refl}_{A',f\,a} : \mathbf{Id}\ A'\,(f\,a)\,(f\,a)}$$

B.4 Algorithmic $\mathsf{MLTT}_{\mathrm{map}}$

Extend appendix B.2. Replaces the rules already named with the same name in appendix B.2.

$\boxed{\Gamma \vdash_{\mathrm{map}} t \cong_{\mathrm{h}} t' \lhd T}$

$$\textsc{NeuPosMap}\ \frac{\Gamma \vdash_{\mathrm{map}} n \approx_{\mathrm{map}} n' \lhd T \qquad T \text{ is } \mathrm{Type}_i, \mathbf{List}\ A, \mathbf{W}\ x{:}A.B \text{ or } \mathbf{Id}_A\ a\ a'}{\Gamma \vdash_{\mathrm{map}} n \cong_{\mathrm{h}} n' \lhd T}$$

$\boxed{\Gamma \vdash_{\mathrm{map}} n \approx n' \rhd T}$

$$\textsc{NListInd}\ \frac{\begin{array}{c}\Gamma \vdash_{\mathrm{map}} A \cong A' \lhd \qquad \Gamma \vdash_{\mathrm{map}} s \approx_{\mathrm{map}} s' \lhd \mathbf{List}\ A \qquad \Gamma, z{:}\mathbf{List}\ A \vdash_{\mathrm{map}} P \cong P' \lhd \\ \Gamma \vdash_{\mathrm{map}} b_\varepsilon \cong b'_\varepsilon \lhd P[\varepsilon_A] \qquad \Gamma, x{:}A, y{:}\mathbf{List}\ A, z{:}P[y] \vdash_{\mathrm{map}} b_{::} \cong b'_{::} \lhd P[x ::_A y]\end{array}}{\Gamma \vdash_{\mathrm{map}} \mathrm{ind}_{\mathbf{List}\ A}(s; z.P; b_\varepsilon, x.y.z.b_{::}) \approx \mathrm{ind}_{\mathbf{List}\ A'}(s'; z.P'; b'_\varepsilon, x.y.z.b'_{::}) \rhd P[s]}$$

$$\textsc{NTreeInd}\ \frac{\begin{array}{c}\Gamma \vdash_{\mathrm{map}} A \cong A' \lhd \qquad \Gamma, x{:}A \vdash_{\mathrm{map}} B \cong B' \lhd \\ \Gamma \vdash_{\mathrm{map}} s \approx_{\mathrm{map}} s' \lhd \mathbf{W}\ x{:}A.B \qquad \Gamma, z{:}\mathbf{W}\ x{:}A.B \vdash_{\mathrm{map}} P \cong P' \lhd \\ \Gamma, x{:}A, y{:}B[x] \to Wx{:}A.B, h{:}\Pi\, z{:}B[x].P[y\,z] \vdash_{\mathrm{map}} b \cong b' \lhd P\left[\mathrm{sup}_{x.B}\, x\ y\right]\end{array}}{\Gamma \vdash_{\mathrm{map}} \mathrm{ind}_{\mathbf{W}\ x:A.B}(s; z.P; x.y.z.b) \approx \mathrm{ind}_{\mathbf{W}\ x:A'.B'}(s'; z.P'; x.y.z.b') \rhd P[s]}$$

$$\textsc{IdInd}\ \frac{\begin{array}{c}\Gamma \vdash_{\mathrm{map}} A \cong A' \lhd \qquad \Gamma \vdash_{\mathrm{map}} s \approx_{\mathrm{map}} s' \lhd \mathbf{Id}_A \rhd a, a' \\ \Gamma, x, y{:}A, z{:}\mathbf{Id}_A\ x\ y \vdash_{\mathrm{map}} P \cong P' \lhd \qquad \Gamma, x{:}A \vdash_{\mathrm{map}} b \cong b' \lhd P\left[\mathrm{id}, x, x, \mathrm{refl}_{A,x}\right]\end{array}}{\Gamma \vdash_{\mathrm{map}} \mathrm{ind}_{\mathbf{Id}_A}(s; x.z.P; b) \approx \mathrm{ind}_{\mathbf{Id}_{A'}}(s'; x.z.P'; b') \rhd P[\mathrm{id}, a, a', s]}$$

$\boxed{\mathsf{unmap}, \mathsf{unmapfun}, \mathsf{unmapfunW}_1, \mathsf{unmapfunW}_2}$

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

$\boxed{\Gamma \vdash_{\mathrm{map}} n \approx_{\mathrm{map}} n' \lhd T}$

$$\textsc{UnmapList}\ \frac{\begin{array}{c}\Gamma \vdash_{\mathrm{map}} \mathsf{unmap}(n) \approx_{\mathrm{h}} \mathsf{unmap}(n') \rhd \mathbf{List}\ A \\ \Gamma, x{:}A \vdash_{\mathrm{map}} \mathsf{unmapfun}(n, x) \cong \mathsf{unmapfun}(n', x) \lhd B\end{array}}{\Gamma \vdash_{\mathrm{map}} n \approx_{\mathrm{map}} n' \lhd \mathbf{List}\ B}$$

$$\textsc{UnmapTree}\ \frac{\begin{array}{c}\Gamma \vdash_{\mathrm{map}} \mathsf{unmap}(n) \approx_{\mathrm{h}} \mathsf{unmap}(n') \rhd \mathbf{W}\ x{:}A.B \\ \Gamma, x{:}A \vdash_{\mathrm{map}} \mathsf{unmapfunW}_1(n, x) \cong \mathsf{unmapfunW}_1(n', x) \lhd A' \\ \Gamma, x{:}A, y{:}B'\left[\mathsf{unmapfunW}_1(n, x)\right] \vdash_{\mathrm{map}} \mathsf{unmapfunW}_2(n, y) \cong \mathsf{unmapfunW}_2(n', y) \lhd B\, x\end{array}}{\Gamma \vdash_{\mathrm{map}} n \approx_{\mathrm{map}} n' \lhd \mathbf{W}\ x{:}A.B}$$

$$\textsc{UnmapId}\ \frac{\begin{array}{c}\Gamma \vdash_{\mathrm{map}} \mathsf{unmap}(n) \approx_{\mathrm{h}} \mathsf{unmap}(n') \rhd \mathbf{Id}_A\ a\ a' \\ \Gamma, x{:}A \vdash_{\mathrm{map}} \mathsf{unmapfun}(n, x) \cong \mathsf{unmapfun}(n', x) \lhd A'\end{array}}{\Gamma \vdash_{\mathrm{map}} s \approx_{\mathrm{map}} s' \lhd \mathbf{Id}_{A'} \rhd a, a'}$$

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

$$\mathrm{map}_{\Pi}\ f\,h\,t \rightsquigarrow^1 (\pi_2\, f)\,(h\,((\pi_1\, f)\,t)) \qquad \pi_1\,(\mathrm{map}_{\Sigma}\ f\,p) \rightsquigarrow^1 \pi_1\, f\,(\pi_1\, p)$$

$$\pi_2\,(\mathrm{map}_{\Sigma}\ f\,p) \rightsquigarrow^1 \pi_2\, f\,(\pi_2\, p) \qquad \mathrm{map}_{\mathbf{List}}\, f\,\varepsilon \rightsquigarrow^1 \varepsilon$$

$$\mathrm{map}_{\mathbf{List}}\, f\,(hd :: tl) \rightsquigarrow^1 f\,hd :: \mathrm{map}_{\mathbf{List}}\, f\,tl$$

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

$$\mathrm{map}_{\mathbf{Id}}\ f\ \mathrm{refl}_{A,a} \rightsquigarrow^1 \mathrm{refl}_{B,f\,a} \qquad \textsc{RedMapComp}\ \frac{\mathrm{ne}\ n}{\mathrm{map}_{\mathbf{List}}\, f\,(\mathrm{map}_{\mathbf{List}}\, g\, n) \rightsquigarrow^1 \mathrm{map}_{\mathbf{List}}(f \circ g)\, n}$$

B.5 Declarative label types

Extend appendix B.1.

$$\textsc{LblTy}\ \frac{L \in \mathcal{P}_{\mathrm{f}}(\mathrm{Lbl})}{\Gamma \vdash \boldsymbol{L}} \qquad \textsc{LblUni}\ \frac{L \in \mathcal{P}_{\mathrm{f}}(\mathrm{Lbl})}{\Gamma \vdash \boldsymbol{L} : \mathrm{Type}_0} \qquad \textsc{LblTm}\ \frac{l \in \mathrm{Lbl}}{\Gamma \vdash \underline{l} : \{\boldsymbol{l}\}}$$

$$\textsc{LblElim}\ \frac{\Gamma \vdash s : \boldsymbol{L} \qquad \Gamma, x{:}\,\boldsymbol{L} \vdash P \qquad \Gamma \vdash b_l : P[\underline{l}]\ \text{for all}\ l \in L}{\Gamma \vdash \mathrm{ind}_{\boldsymbol{L}}(s; x.P; \overrightarrow{b_l}) : P[s]}$$

$$\textsc{NLblInd}\ \frac{\Gamma \vdash t \cong t' : \boldsymbol{L} \qquad \Gamma, z{:}\,\boldsymbol{L} \vdash P \cong P' \qquad \Gamma \vdash b_l \cong b'_l : P[\underline{l}]\ \text{for all}\ l \in L}{\Gamma \vdash \mathrm{ind}_{\boldsymbol{L}}(t; z.P; \overrightarrow{b_l}) \cong \mathrm{ind}_{\boldsymbol{L}}(t'; z.P'; \overrightarrow{b'_l}) : P[t]}$$

$$\beta\textsc{Lbl}\ \frac{l \in L \qquad \Gamma \vdash s : \boldsymbol{L} \qquad \Gamma, x{:}\,\boldsymbol{L} \vdash P \qquad \Gamma \vdash b_l : P[\underline{l}]\ \text{for all}\ l \in L}{\Gamma \vdash \mathrm{ind}_{\{\boldsymbol{l}\}}(\underline{l}; x.P; \overrightarrow{b_l}) \cong b_l : P[\underline{l}]}$$

B.6 Algorithmic label types

Extend appendix B.2. The term $\underline{l}$ is a normal form, and $\mathrm{ind}_{\boldsymbol{L}}(n; x.P; \vec{b})$ is neutral whenever n is.

$$\textsc{LblTy}\ \frac{L \in \mathcal{P}_{\mathrm{f}}(\mathrm{Lbl})}{\Gamma \vdash \boldsymbol{L} \triangleright \mathrm{Type}_0} \qquad \textsc{LblTm}\ \frac{l \in \mathrm{Lbl}}{\Gamma \vdash \underline{l} \triangleright \{\boldsymbol{l}\}} \qquad \textsc{LblTmConv}\ \frac{}{\Gamma \vdash \underline{l} \cong_{\mathrm{h}} \underline{l} \triangleleft \boldsymbol{L}}$$

$$\textsc{LblElim}\ \frac{\Gamma \vdash s \triangleleft \boldsymbol{L} \qquad \Gamma, x{:}\,\boldsymbol{L} \vdash P \triangleright_{\mathrm{h}} \mathrm{Type}_i \qquad \Gamma \vdash b_l \triangleleft P[\underline{l}]\ \text{for all}\ l \in L}{\Gamma \vdash \mathrm{ind}_{\boldsymbol{L}}(s; x.P; \overrightarrow{b_l}) \triangleright P[s]}$$

$$\textsc{NLblInd}\ \frac{\Gamma \vdash n \approx n' \triangleright S \qquad \Gamma, z{:}\,\boldsymbol{L} \vdash P \cong P' \triangleleft \qquad \Gamma \vdash b_l \cong b'_l \triangleleft P[\underline{l}]\ \text{for all}\ l \in L}{\Gamma \vdash \mathrm{ind}_{\boldsymbol{L}}(n; z.P; \overrightarrow{b_l}) \approx \mathrm{ind}_{\boldsymbol{L}}(n'; z.P'; \overrightarrow{b'_l}) \triangleright P[n]}$$

$$\textsc{LblRed}\ \frac{}{\mathrm{ind}_{\{\boldsymbol{l}\}}(\underline{l}; x.P; \overrightarrow{b_l}) \rightsquigarrow^1 b_l}$$

B.7 Algorithmic $\mathsf{MLTT}_{\mathrm{sub}}$

Extend appendices B.2 and B.6, with rule CheckSub replacing Check.

$\boxed{\Gamma \vdash_{\mathrm{sub}} t \lhd T}$

$$\textsc{CheckSub}\ \frac{\Gamma \vdash_{\mathrm{sub}} t \rhd T' \qquad \Gamma \vdash_{\mathrm{sub}} T' \preccurlyeq T \lhd}{\Gamma \vdash_{\mathrm{sub}} t \lhd T}$$

$\boxed{\Gamma \vdash_{\mathrm{sub}} T \preccurlyeq T' \lhd}$ Type T is a subtype of type T'

$$\textsc{TyRed}\ \frac{T \rightsquigarrow^\star U \qquad T' \rightsquigarrow^\star U' \qquad \Gamma \vdash_{\mathrm{sub}} U \preccurlyeq_{\mathrm{h}} U' \lhd}{\Gamma \vdash_{\mathrm{sub}} T \preccurlyeq T' \lhd}$$

$\boxed{\Gamma \vdash_{\mathrm{sub}} T \preccurlyeq_{\mathrm{h}} T' \lhd}$ Reduced type T is a subtype of reduced type T'

$$\textsc{LblSub}\ \frac{L \subseteq L'}{\Gamma \vdash_{\mathrm{sub}} \boldsymbol{L} \preccurlyeq \boldsymbol{L}' \lhd} \qquad \textsc{ProdSub}\ \frac{\Gamma \vdash_{\mathrm{sub}} A' \preccurlyeq A \lhd \qquad \Gamma, x\colon A' \vdash_{\mathrm{sub}} B \preccurlyeq B' \lhd}{\Gamma \vdash_{\mathrm{sub}} \Pi\, x\colon A.B \preccurlyeq_{\mathrm{h}} \Pi\, x\colon A'.B' \lhd}$$

$$\textsc{ListSub}\ \frac{\Gamma \vdash_{\mathrm{sub}} A \preccurlyeq A' \lhd}{\Gamma \vdash_{\mathrm{sub}} \mathbf{List}\ A \preccurlyeq_{\mathrm{h}} \mathbf{List}\ A' \lhd} \qquad \textsc{SigSub}\ \frac{\Gamma \vdash_{\mathrm{sub}} A \preccurlyeq A' \lhd \qquad \Gamma, x\colon A \vdash_{\mathrm{sub}} B \preccurlyeq B' \lhd}{\Gamma \vdash_{\mathrm{sub}} \Sigma\, x\colon A.B \preccurlyeq_{\mathrm{h}} \Sigma\, x\colon A'.B' \lhd}$$

$$\textsc{TreeSub}\ \frac{\Gamma \vdash_{\mathrm{sub}} A \preccurlyeq A' \lhd \qquad \Gamma, x\colon A \vdash_{\mathrm{sub}} B' \preccurlyeq B \lhd}{\Gamma \vdash_{\mathrm{sub}} \mathbf{W}\, x\colon A.B \preccurlyeq_{\mathrm{h}} \mathbf{W}\, x\colon A'.B' \lhd} \qquad \textsc{IdSub}\ \frac{\Gamma \vdash_{\mathrm{sub}} A \preccurlyeq A' \lhd \qquad \Gamma \vdash_{\mathrm{sub}} t \cong t' \lhd A' \qquad \Gamma \vdash_{\mathrm{sub}} u \cong u' \lhd A'}{\Gamma \vdash_{\mathrm{sub}} \mathbf{Id}_A\ t\, u \preccurlyeq_{\mathrm{h}} \mathbf{Id}_{A'}\ t'\, u' \lhd}$$

$$\textsc{SubRefl}\ \frac{T \text{ is } \mathrm{Type}_i, \mathbf{0}, \mathbf{1} \text{ or } \mathbf{B}}{\Gamma \vdash_{\mathrm{sub}} T \preccurlyeq_{\mathrm{h}} T \lhd} \qquad \textsc{NeuSub}\ \frac{\Gamma \vdash_{\mathrm{sub}} n \approx_{\mathrm{h}} n' \rhd T}{\Gamma \vdash_{\mathrm{sub}} n \preccurlyeq_{\mathrm{h}} n' \lhd}$$

Admissible

$$\textsc{ConvSub}\ \frac{\Gamma \vdash_{\mathrm{sub}} A \cong A' \lhd}{\Gamma \vdash_{\mathrm{sub}} A \preccurlyeq A' \lhd} \qquad \textsc{SubAntiSym}\ \frac{\Gamma \vdash_{\mathrm{sub}} A \preccurlyeq A' \lhd \qquad \Gamma \vdash_{\mathrm{sub}} A' \preccurlyeq A \lhd}{\Gamma \vdash_{\mathrm{sub}} A \cong A' \lhd}$$

$$\textsc{SubTrans}\ \frac{\Gamma \vdash_{\mathrm{sub}} A \preccurlyeq A' \lhd \qquad \Gamma \vdash_{\mathrm{sub}} A' \preccurlyeq A'' \lhd}{\Gamma \vdash_{\mathrm{sub}} A \preccurlyeq A'' \lhd}$$

B.8 Algorithmic $\mathsf{MLTT}_{\mathrm{coe}}$

Extend appendix B.2 and appendix B.6.

$\boxed{\Gamma \vdash_{\mathrm{coe}} t \rhd T}$

$$\textsc{Coe}\ \frac{\Gamma \vdash_{\mathrm{coe}} A \lhd \qquad \Gamma \vdash_{\mathrm{coe}} A' \lhd \qquad \Gamma \vdash_{\mathrm{coe}} t \lhd A \qquad \Gamma \vdash_{\mathrm{coe}} A \preccurlyeq A' \lhd}{\Gamma \vdash_{\mathrm{coe}} \mathrm{coe}_{A,A'}\ t \rhd A'}$$

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

$$\mathrm{coe}_{\Pi\, x{:}A.B, \Pi\, x{:}A'.B'}(\lambda\, x{:}\, A''.t) \rightsquigarrow^1 \lambda\, x{:}\, A'.\, \mathrm{coe}_{B\left[\mathrm{coe}_{A',A}\, x\right], B'[x]}(t\left[\mathrm{coe}_{A',A}\, x\right])$$

$$\textsc{RedCoeFunNe}\ \frac{\mathrm{ne}\, f}{(\mathrm{coe}_{\Pi\, x{:}A.B, \Pi\, x{:}A'.B'}\, f)\ a \rightsquigarrow^1 \mathrm{coe}_{B\left[\mathrm{coe}_{A',A}\, a\right], B'[a]}(f\ (\mathrm{coe}_{A',A}\, a))}$$

$$\mathrm{coe}_{\Sigma\, x{:}A.B, \Sigma\, x{:}A'.B'}(t, u) \rightsquigarrow^1 ((\mathrm{coe}_{A,A'}\, t), (\mathrm{coe}_{B[t], B'\left[\mathrm{coe}_{A,A'}\, t\right]}\, u))_{x.B'}$$

$$\textsc{RedCoeSigNe1}\ \frac{\mathrm{ne}\, p}{\pi_1\ (\mathrm{coe}_{\Sigma\, x{:}A.B, \Sigma\, x{:}A'.B'}\, p) \rightsquigarrow^1 \mathrm{coe}_{A,A'}(\pi_1\, p)}$$

$$\textsc{RedCoeSigNe2}\ \frac{\mathrm{ne}\, p}{\pi_2\ (\mathrm{coe}_{\Sigma\, x{:}A.B, \Sigma\, x{:}A'.B'}\, p) \rightsquigarrow^1 \mathrm{coe}_{B[\pi_1\, p], B'\left[\mathrm{coe}_{A,A'}(\pi_1\, p)\right]}(\pi_2\, p)}$$

$$\textsc{CoeRedId}\ \frac{T \text{ is } \mathrm{Type}_i,\ \mathbf{0},\ \mathbf{1} \text{ or } \mathbf{B}}{\mathrm{coe}_{T,T}\, t \rightsquigarrow^1 t} \qquad \mathrm{coe}_{\boldsymbol{L},\boldsymbol{L}'}\ \mathrm{coe}_{\boldsymbol{L}}\ \underline{l} \rightsquigarrow^1 \mathrm{coe}_{\boldsymbol{L}'}\ \underline{l} \qquad \mathrm{coe}_{\mathbf{List}\ A, \mathbf{List}\ A'}\ \varepsilon \rightsquigarrow^1 \varepsilon_{A'}$$

$$\mathrm{coe}_{\mathbf{List}\ A, \mathbf{List}\ A'}(h :: t) \rightsquigarrow^1 \mathrm{coe}_{A,A'}\ h ::_{A'} \mathrm{coe}_{\mathbf{List}\ A, \mathbf{List}\ A'}\ t$$

$$\mathrm{coe}_{\mathbf{W}\, x{:}A.B, \mathbf{W}\, x{:}A.B'}(\sup a\ l) \rightsquigarrow^1 \sup_{x.B'}(\mathrm{coe}_{A,A'}\ a)\ (\lambda\, x{:}\, B'\left[\mathrm{coe}_{A,A'}\ a\right].\, \mathrm{coe}_{\mathbf{W}\, x{:}A.B, \mathbf{W}\, x{:}A.B'}(k\ (\mathrm{coe}_{B'\left[\mathrm{coe}_{A,A'}\ a\right], B[a]}\ x)))$$

$$\mathrm{coe}_{\mathbf{Id}_A\ a\, b, \mathbf{Id}_{A'}\ a'\ b'}\ \mathrm{refl}_{A,a} \rightsquigarrow^1 \mathrm{refl}_{A', (\mathrm{coe}_{A,A'}\ a)}$$

$$\textsc{CoeL}\ \frac{A \rightsquigarrow^1 A'}{\mathrm{coe}_{A,B}\, t \rightsquigarrow^1 \mathrm{coe}_{A',B}\, t} \qquad \textsc{CoeR}\ \frac{\mathrm{nf}\, A \qquad B \rightsquigarrow^1 B'}{\mathrm{coe}_{A,B}\, t \rightsquigarrow^1 \mathrm{coe}_{A,B'}\, t}$$

$$\textsc{CoeTm}\ \frac{\mathrm{nf}\, A \qquad \mathrm{nf}\, B \qquad t \rightsquigarrow^1 t'}{\mathrm{coe}_{A,B}\, t \rightsquigarrow^1 \mathrm{coe}_{A,B}\, t'} \qquad \textsc{CoeCoe}\ \frac{\mathrm{nf}\, U \qquad \mathrm{nf}\, U' \qquad \mathrm{nf}\, T \qquad \mathrm{nf}\, T' \qquad \mathrm{ne}\, n}{\mathrm{coe}_{U,U'}\ \mathrm{coe}_{T,T'}\ n \rightsquigarrow^1 \mathrm{coe}_{T,U'}\ n}$$

$$\begin{array}{llll} \mathrm{nf} & f & \stackrel{\mathrm{def}}{=} & \cdots \mid c \mid \underline{l} \mid \mathrm{coe}_{\boldsymbol{L}}\ \underline{l} \quad \text{weak-head normal forms} \\ \mathrm{ne} & n & \stackrel{\mathrm{def}}{=} & \cdots \mid \mathrm{ind}_{\boldsymbol{L}}(t; c; \vec{t}) \quad \text{weak-head neutrals} \\ \mathrm{cne} & c & \stackrel{\mathrm{def}}{=} & \cdots \mid \mathrm{coe}_{f,f}\, n \quad \text{compacted neutrals} \end{array}$$

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

$$\textsc{NCoe}\ \frac{\Gamma \vdash_{\mathrm{coe}} n \approx n' \triangleright S''}{\Gamma \vdash_{\mathrm{coe}} \mathrm{coe}_{S,T}\, n \approx_{\mathrm{coe}} \mathrm{coe}_{S',T'}\, n' \triangleleft T''} \qquad \textsc{NCoeL}\ \frac{\Gamma \vdash_{\mathrm{coe}} n \approx n' \triangleright S''}{\Gamma \vdash_{\mathrm{coe}} \mathrm{coe}_{S,T}\, n \approx_{\mathrm{coe}} n' \triangleleft T''}$$

$$\textsc{NCoeR}\ \frac{\Gamma \vdash_{\mathrm{coe}} n \approx n' \triangleright S''}{\Gamma \vdash_{\mathrm{coe}} n \approx_{\mathrm{coe}} \mathrm{coe}_{S',T'}\, n' \triangleleft T''} \qquad \textsc{NNoCoe}\ \frac{\Gamma \vdash_{\mathrm{coe}} n \approx n' \triangleright S''}{\Gamma \vdash_{\mathrm{coe}} n \approx_{\mathrm{coe}} n' \triangleleft T''}$$

$\boxed{\Gamma \vdash_{\mathrm{coe}} t \cong_{\mathrm{h}} t' \triangleleft T}$

$$\textsc{LblTmConv}\ \frac{}{\Gamma \vdash_{\mathrm{coe}} \mathrm{coe}_{\boldsymbol{L}}\ \underline{l} \cong_{\mathrm{h}} \mathrm{coe}_{\boldsymbol{L}}\ \underline{l} \triangleleft \boldsymbol{L}} \qquad \textsc{NeuList}\ \frac{\Gamma \vdash_{\mathrm{coe}} n \approx_{\mathrm{coe}} n' \triangleleft \mathbf{List}\ A}{\Gamma \vdash_{\mathrm{coe}} n \cong_{\mathrm{h}} n' \triangleleft \mathbf{List}\ A}$$

$$\textsc{NeuTree}\ \frac{\Gamma \vdash_{\mathrm{coe}} n \approx_{\mathrm{coe}} n' \triangleleft \mathbf{W}\, x{:}\, A.B}{\Gamma \vdash_{\mathrm{coe}} n \cong_{\mathrm{h}} n' \triangleleft \mathbf{W}\, x{:}\, A.B} \qquad \textsc{NeuId}\ \frac{\Gamma \vdash_{\mathrm{coe}} n \approx_{\mathrm{coe}} n' \triangleleft \mathbf{Id}_A\ a\, a'}{\Gamma \vdash_{\mathrm{coe}} n \cong_{\mathrm{h}} n' \triangleleft \mathbf{Id}_A\ a\, a'}$$

$$\textsc{NeuNeu}\ \frac{\Gamma \vdash_{\mathrm{coe}} n \approx_{\mathrm{coe}} n' \triangleleft M \qquad \mathrm{ne}\ M}{\Gamma \vdash_{\mathrm{coe}} n \cong_{\mathrm{h}} n' \triangleleft M}$$

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

$$\textsc{TyRed}\ \frac{T \rightsquigarrow^{\star} U \qquad T' \rightsquigarrow^{\star} U' \qquad \Gamma \vdash_{\mathrm{coe}} U \preccurlyeq_{\mathrm{h}} U' \triangleleft}{\Gamma \vdash_{\mathrm{coe}} T \preccurlyeq T' \triangleleft}$$

$\boxed{\Gamma \vdash_{\mathrm{coe}} T \preccurlyeq_{\mathrm{h}} T' \triangleleft}$

$$\textsc{LblSub}\ \frac{L \subseteq L'}{\Gamma \vdash_{\mathrm{coe}} \boldsymbol{L} \preccurlyeq_{\mathrm{h}} \boldsymbol{L}' \triangleleft} \qquad \textsc{ProdSub}\ \frac{\Gamma \vdash_{\mathrm{coe}} A' \preccurlyeq A \triangleleft \qquad \Gamma, x{:}\, A' \vdash_{\mathrm{coe}} B\left[\mathrm{coe}_{A',A}\, x\right] \preccurlyeq B' \triangleleft}{\Gamma \vdash_{\mathrm{coe}} \Pi x{:}\, A.B \preccurlyeq_{\mathrm{h}} \Pi x{:}\, A'.B' \triangleleft}$$

$$\textsc{ListSub}\ \frac{\Gamma \vdash_{\mathrm{coe}} A \preccurlyeq A' \triangleleft}{\Gamma \vdash_{\mathrm{coe}} \mathbf{List}\ A \preccurlyeq_{\mathrm{h}} \mathbf{List}\ A' \triangleleft} \qquad \textsc{SigSub}\ \frac{\Gamma \vdash_{\mathrm{coe}} A \preccurlyeq A' \triangleleft \qquad \Gamma, x{:}\, A \vdash_{\mathrm{coe}} B \preccurlyeq B'\left[\mathrm{coe}_{A,A'}\, x\right] \triangleleft}{\Gamma \vdash_{\mathrm{coe}} \Sigma x{:}\, A.B \preccurlyeq_{\mathrm{h}} \Sigma x{:}\, A'.B' \triangleleft}$$

$$\textsc{TreeSub}\ \frac{\Gamma \vdash_{\mathrm{coe}} A \preccurlyeq A' \triangleleft \qquad \Gamma, x{:}\, A \vdash_{\mathrm{coe}} B'\left[\mathrm{coe}_{A,A'}\, x\right] \preccurlyeq B \triangleleft}{\Gamma \vdash_{\mathrm{coe}} \mathbf{W}\, x{:}\, A.B \preccurlyeq_{\mathrm{h}} \mathbf{W}\, x{:}\, A'.B' \triangleleft}$$

$$\textsc{IdSub}\ \frac{\Gamma \vdash_{\mathrm{coe}} A \preccurlyeq A' \triangleleft \qquad \Gamma \vdash_{\mathrm{coe}} \mathrm{coe}_{A,A'}\, t \cong t' \triangleleft A' \qquad \Gamma \vdash_{\mathrm{coe}} \mathrm{coe}_{A,A'}\, u \cong u' \triangleleft A'}{\Gamma \vdash_{\mathrm{coe}} \mathbf{Id}_A\ t\, u \preccurlyeq_{\mathrm{h}} \mathbf{Id}_{A'}\ t'\, u' \triangleleft}$$

$$\textsc{SubRefl}\ \frac{T \text{ is } \mathrm{Type}_i, \mathbf{0}, \mathbf{1} \text{ or } \mathbf{B}}{\Gamma \vdash_{\mathrm{coe}} T \preccurlyeq T \triangleleft}$$

B.9 Declarative MLTT$_{\mathrm{coe}}$

Extend appendix B.1 and appendix B.5.

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

$$\textsc{Coe}\ \frac{\Gamma \vdash_{\mathrm{coe}} A \qquad \Gamma \vdash_{\mathrm{coe}} A' \qquad \Gamma \vdash_{\mathrm{coe}} t : A \qquad \Gamma \vdash_{\mathrm{coe}} A \preccurlyeq A'}{\Gamma \vdash_{\mathrm{coe}} \mathrm{coe}_{A,A'}\, t : A'}$$

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

$$\textsc{CoeId}\ \frac{\Gamma \vdash_{\mathsf{coe}} t : A}{\Gamma \vdash_{\mathsf{coe}} \mathsf{coe}_{A,A}\, t \cong t : A} \qquad \textsc{CoeTrans}\ \frac{\Gamma \vdash_{\mathsf{coe}} t : A \qquad \Gamma \vdash_{\mathsf{coe}} A \preccurlyeq A' \qquad \Gamma \vdash_{\mathsf{coe}} A' \preccurlyeq A''}{\Gamma \vdash_{\mathsf{coe}} \mathsf{coe}_{A',A''}\, \mathsf{coe}_{A,A'}\, t \cong \mathsf{coe}_{A,A''}\, t : A''}$$

$$\textsc{CoeCong}\ \frac{\Gamma \vdash_{\mathsf{coe}} t \cong t' : A \qquad \Gamma \vdash_{\mathsf{coe}} A \cong A' \qquad \Gamma \vdash_{\mathsf{coe}} B \cong B'}{\Gamma \vdash_{\mathsf{coe}} \mathsf{coe}_{A,B}\, t \cong \mathsf{coe}_{A',B'}\, t' : B}$$

$$\textsc{CoeFun}\ \frac{\begin{array}{c}\Gamma \vdash_{\mathsf{coe}} A' \preccurlyeq A \qquad \Gamma, x\colon A' \vdash_{\mathsf{coe}} B\left[\mathsf{coe}_{A',A}\, x\right] \preccurlyeq B' \\ \Gamma \vdash_{\mathsf{coe}} f : \Pi\, x\colon A.B \qquad \Gamma \vdash_{\mathsf{coe}} a : A'\end{array}}{\Gamma \vdash_{\mathsf{coe}} (\mathsf{coe}_{\Pi\, x:A.B, \Pi\, x:A'.B'}\, f)\ a \cong \mathsf{coe}_{B\left[\mathsf{coe}_{A',A}\, a\right], B'[x]}(f\ (\mathsf{coe}_{A',A}\, a)) : \Pi\, x\colon A'.B'}$$

$$\textsc{CoeSig1}\ \frac{\Gamma \vdash_{\mathsf{coe}} A \preccurlyeq A' \qquad \Gamma, x\colon A \vdash_{\mathsf{coe}} B \preccurlyeq B'\left[\mathsf{coe}_{A,A'}\, x\right] \qquad \Gamma \vdash_{\mathsf{coe}} p : \Sigma\, x\colon A.B}{\Gamma \vdash_{\mathsf{coe}} \pi_1\, (\mathsf{coe}_{\Sigma\, x:A.B, \Sigma\, x:A'.B'}\, p) \cong \mathsf{coe}_{A,A'}\, (\pi_1\, p) : \Sigma\, x\colon A'.B'}$$

$$\textsc{CoeSig2}\ \frac{\Gamma \vdash_{\mathsf{coe}} A \preccurlyeq A' \qquad \Gamma, x\colon A \vdash_{\mathsf{coe}} B \preccurlyeq B'\left[\mathsf{coe}_{A,A'}\, x\right] \qquad \Gamma \vdash_{\mathsf{coe}} p : \Sigma\, x\colon A.B}{\Gamma \vdash_{\mathsf{coe}} \pi_2\, (\mathsf{coe}_{\Sigma\, x:A.B, \Sigma\, x:A'.B'}\, p) \cong \mathsf{coe}_{B[\pi_1\, p], B'\left[\mathsf{coe}_{A,A'}\, (\pi_1\, p)\right]}\, (\pi_2\, p) : \Sigma\, x\colon A'.B'}$$

$$\textsc{CoeNil}\ \frac{\Gamma \vdash_{\mathsf{coe}} A \preccurlyeq A'}{\Gamma \vdash_{\mathsf{coe}} \mathsf{coe}_{\mathbf{List}\ A, \mathbf{List}\ A'}\, \varepsilon_A \cong \varepsilon_{A'} : \mathbf{List}\ A'}$$

$$\textsc{CoeCons}\ \frac{\Gamma \vdash_{\mathsf{coe}} A \preccurlyeq A' \qquad \Gamma \vdash_{\mathsf{coe}} a : A \qquad \Gamma \vdash_{\mathsf{coe}} l : \mathbf{List}\ A}{\Gamma \vdash_{\mathsf{coe}} \mathsf{coe}_{\mathbf{List}\ A, \mathbf{List}\ A'}\, (a ::_{\mathrm{A}} l) \cong (\mathsf{coe}_{A,A'}\, a) ::_{\mathrm{A}'} (\mathsf{coe}_{\mathbf{List}\ A, \mathbf{List}\ A'}\, l) : \mathbf{List}\ A'}$$

$$\textsc{CoeTree}\ \frac{\begin{array}{c}\Gamma \vdash_{\mathsf{coe}} A \preccurlyeq A' \\ \Gamma, x\colon A \vdash_{\mathsf{coe}} B'\left[\mathsf{coe}_{A,A'}\, x\right] \preccurlyeq B \qquad \Gamma \vdash_{\mathsf{coe}} a : A \qquad \Gamma \vdash_{\mathsf{coe}} k : B\, a \to \mathbf{W}\, x\colon A.B\end{array}}{\begin{array}{l}\Gamma \vdash_{\mathsf{coe}} \mathsf{coe}_{\mathbf{W}\, x:A.B, \mathbf{W}\, x:A.B'}(\mathsf{sup}_{x.B}\, a\, l) \cong \\ \quad \mathsf{sup}_{x.B'}(\mathsf{coe}_{A,A'}\, a)\ (\lambda\, x\colon B'\left[\mathsf{coe}_{A,A'}\, a\right].\, \mathsf{coe}_{\mathbf{W}\, x:A.B, \mathbf{W}\, x:A.B'}(k\, (\mathsf{coe}_{B'\left[\mathsf{coe}_{A,A'}\, a\right], B[a]}\, x))) \\ \qquad\qquad : \mathbf{W}\, x\colon A'.B'\end{array}}$$

$$\textsc{CoeId}\ \frac{\Gamma \vdash_{\mathsf{coe}} A \preccurlyeq A' \qquad \Gamma \vdash_{\mathsf{coe}} a : A}{\Gamma \vdash_{\mathsf{coe}} \mathsf{coe}_{\mathbf{Id}_A\ a\, a, \mathbf{Id}_{A'}\ (\mathsf{coe}_{A,A'}\, a)\ (\mathsf{coe}_{A,A'}\, a)}\, \mathsf{refl}_{A,a} \cong \mathsf{refl}_{A', (\mathsf{coe}_{A,A'}\, a)} : \mathbf{Id}_A\ (\mathsf{coe}_{A,A'}\, a)\ (\mathsf{coe}_{A,A'}\, a)}$$

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

$$\textsc{LblSub}\ \frac{L \subseteq L'}{\Gamma \vdash_{\mathsf{coe}} \boldsymbol{L} \preccurlyeq \boldsymbol{L'}} \qquad \textsc{ProdSub}\ \frac{\Gamma \vdash_{\mathsf{coe}} A' \preccurlyeq A \qquad \Gamma, x\colon A' \vdash_{\mathsf{coe}} B\left[\mathsf{coe}_{A',A}\, x\right] \preccurlyeq B'}{\Gamma \vdash_{\mathsf{coe}} \Pi\, x\colon A.B \preccurlyeq \Pi\, x\colon A'.B'}$$

$$\textsc{ListSub}\ \frac{\Gamma \vdash_{\mathsf{coe}} A \preccurlyeq A'}{\Gamma \vdash_{\mathsf{coe}} \mathbf{List}\ A \preccurlyeq \mathbf{List}\ A'} \qquad \textsc{SigSub}\ \frac{\begin{array}{c}\Gamma \vdash_{\mathsf{coe}} A \preccurlyeq A' \\ \Gamma, x\colon A \vdash_{\mathsf{coe}} B \preccurlyeq B'\left[\mathsf{coe}_{A,A'}\, x\right]\end{array}}{\Gamma \vdash_{\mathsf{coe}} \Sigma\, x\colon A.B \preccurlyeq \Sigma\, x\colon A'.B'}$$

$$\textsc{TreeSub}\ \frac{\Gamma \vdash_{\mathrm{coe}} A \preccurlyeq A' \qquad \Gamma, x\colon A \vdash_{\mathrm{coe}} B'\left[\mathrm{coe}_{A,A'}\, x\right] \preccurlyeq B}{\Gamma \vdash_{\mathrm{coe}} \mathbf{W}\, x\colon A.B \preccurlyeq \mathbf{W}\, x\colon A'.B'}$$

$$\textsc{IdSub}\ \frac{\Gamma \vdash_{\mathrm{coe}} A \preccurlyeq A' \qquad \Gamma \vdash_{\mathrm{coe}} \mathrm{coe}_{A,A'}\, t \cong t' : A' \qquad \Gamma \vdash_{\mathrm{coe}} \mathrm{coe}_{A,A'}\, u \cong u' : A'}{\Gamma \vdash_{\mathrm{coe}} \mathbf{Id}_A\ t\, u \preccurlyeq \mathbf{Id}_{A'}\ t'\, u'}$$

$$\textsc{SubRefl}\ \frac{\Gamma \vdash_{\mathrm{coe}} A \cong A'}{\Gamma \vdash_{\mathrm{coe}} A \preccurlyeq A'} \qquad \textsc{SubTrans}\ \frac{\Gamma \vdash_{\mathrm{coe}} A \preccurlyeq A' \qquad \Gamma \vdash_{\mathrm{coe}} A' \preccurlyeq A''}{\Gamma \vdash_{\mathrm{coe}} A \preccurlyeq A''}$$

C PROOFS OF LEMMAS

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

C.1 From section 3.2

$$\mathrm{map}_\Pi\ ((g,f)\colon \mathrm{Hom}_\Pi((A,B),(A',B')))\ (h\colon \Pi(x\colon A)B) \stackrel{\text{def}}{=} \lambda\, x\colon A'.f\ (h\ (g\ x))$$
$$\mathrm{map}_\Sigma\ ((g,f)\colon \mathrm{Hom}_\Pi((A,B),(A',B')))\ (p\colon \Sigma(x\colon A)B) \stackrel{\text{def}}{=} (g\ (\pi_1\, p), f\ (\pi_2\, p))$$

Lemma C.1. map_Π *and* map_Σ *satisfy the functor laws* MapId *and* MapComp.

Proof. For the preservation of identities, we have:

$$\mathrm{map}_\Pi\ (\mathrm{id}_A, \lambda\{x\colon A\}.\,\mathrm{id}_{B\,x})\ h \cong \lambda\, x\colon A.\,\mathrm{id}_{B\,x}\ (h\ (\mathrm{id}_A\ x)) \cong \lambda\, x\colon A.g\ x \cong g$$
$$\mathrm{map}_\Sigma\ (\mathrm{id}_A, \lambda\{x\colon A\}.\,\mathrm{id}_{B\,x})\ p \cong (\mathrm{id}_A\ (\pi_1\, p), \mathrm{id}_{B\,(\pi_1\, p)}\ (\pi_2\, p)) \cong (\pi_1\, p, \pi_2\, p) \cong p$$

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

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

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

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

□

C.2 From section 5.3

Lemma C.2 (Catch up, function type (lemma 5.7)). *If* $\Gamma \vdash_{\mathrm{coe}} f\ a : B$ *and* $|f| = \lambda\, x\colon A'.\, t'$, *then there exists* t *such that* $|t| = t'$ *and* $f\ a \rightsquigarrow^\star t[a]$.

Proof. We must have that $f = (\mathrm{coe}_{T_1,\dots,T_n}(\lambda\, x\colon A.t_0))$[14] 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 $\Gamma, x\colon A \vdash_{\mathrm{coe}}$

[14] That is, a string of coercions $\mathrm{coe}_{T_{n-1},T_n}(\dots \mathrm{coe}_{T_1,T_2}(\lambda\, x\colon A.t_0))$.

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

$$f\ a \rightsquigarrow^\star \text{coe}_{B_0',B_1',\ldots,B_n'}((\lambda x{:}A.t_0)\ (\text{coe}_{A_n,\ldots,A_1,A}\ a))$$

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

$$f\ a \rightsquigarrow^\star \text{coe}_{B_0'',B_1'',\ldots,B_n''}(t_0\left[\text{coe}_{A_n,\ldots,A_1,A}\ a\right])$$

Now we can conclude, as indeed

$$\begin{aligned}\left|\text{coe}_{B_0'',B_1'',\ldots,B_n''}(t_0\left[\text{coe}_{A_n,\ldots,A_1,A}\ x\right])\right| &= \left|t_0\left[\text{coe}_{A_n,\ldots,A_1,A}\ x\right]\right| \\ &= |t_0|\ \left[\left|\text{coe}_{A_n,\ldots,A_1,A}\ x\right|\right] \\ &= |t_0|\ [|x|] = |t_0| = t'\end{aligned}$$

□

Lemma C.3 (Erasure is a backward simulation (lemma 5.10)). *Assume that* $\Gamma \vdash_{\text{coe}} t : T$. *If* $|t| \rightsquigarrow^\star u'$, *with* u' *a weak-head normal form, then* $t \rightsquigarrow^\star u$, *with* u *a weak-head normal form such that* $|u| = u'$.

Proof. First, if $|t| \rightsquigarrow^\star u'$, then there exists u such that $t \rightsquigarrow^\star u$ 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 $u \rightsquigarrow^\star v$ 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* $|\Gamma| \vdash_{\text{sub}} |T| \preccurlyeq^{\text{m}}_{\text{h}} |U| \triangleleft$, *then* $\Gamma \vdash_{\text{coe}} T \preccurlyeq^{\text{m}}_{\text{h}} U \triangleleft$;
(2) *if* $|\Gamma| \vdash_{\text{sub}} |T| \preccurlyeq^{\text{m}} |U| \triangleleft$, *then* $\Gamma \vdash_{\text{coe}} T \preccurlyeq U \triangleleft$;
(3) *if* $|\Gamma| \vdash_{\text{sub}} |t| \cong_{\text{h}} |u| \triangleleft |T|$, *then* $\Gamma \vdash_{\text{coe}} t \cong_{\text{h}} u \triangleleft T$;
(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* $|\Gamma| \vdash_{\text{sub}} |t| \approx_{\text{h}} |u| \triangleright T$, *then* $\Gamma \vdash_{\text{coe}} t \approx u \triangleright T$.

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 fig. 14, this can happen in three situations. If the coercions are between label types, then rule LblTmConv applies. If the coercions are between function 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). Finally, compacted neutrals can appear exactly in

the places where $\mathrm{MLTT}_{\mathrm{coe}}$ uses the comparison of the compacted neutrals, which strips away the possibly present coercions, as expected. □

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* $|\Gamma| \vdash_{\mathrm{sub}} t' \rhd T'$, *then there exists* t *and* T *such that* $t' = |t|$, $T' = |T|$, *and* $\Gamma \vdash_{\mathrm{coe}} t \rhd T$;
(2) *if* $|\Gamma| \vdash_{\mathrm{sub}} t' \rhd_{\mathrm{h}} T'$, *then there exists* t *and* T *such that* $t' = |t|$, $T' = |T|$, *and* $\Gamma \vdash_{\mathrm{coe}} t \rhd_{\mathrm{h}} T$;
(3) *if* $|\Gamma| \vdash_{\mathrm{sub}} t' \lhd |T|$, *then there exists* t *such that* $t' = |t|$ *and* $\Gamma \vdash_{\mathrm{coe}} t \lhd T$.

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. □

C.3 Translation from $\mathrm{MLTT}_{\mathrm{coe}}$ to $\mathrm{MLTT}_{\mathrm{map}}$

$$\textsc{TslTy}\ \frac{}{[\![\mathrm{Type}_i]\!] \simeq \mathrm{Type}_i} \qquad \textsc{TslLbl}\ \frac{}{[\![\boldsymbol{L}]\!] \simeq \boldsymbol{L}} \qquad \textsc{TslList}\ \frac{[\![A]\!] \simeq A'}{[\![\mathbf{List}\ A]\!] \simeq \mathbf{List}\ A'}$$

$$\textsc{TslPi}\ \frac{[\![A]\!] \simeq A' \qquad [\![B]\!] \simeq B'}{[\![\Pi\, x{:}\, A.B]\!] \simeq \Pi\, x{:}\, A'.B'} \qquad \textsc{TslVar}\ \frac{}{[\![x]\!] \simeq x} \qquad \textsc{TslLam}\ \frac{[\![A]\!] \simeq A' \qquad [\![t]\!] \simeq t'}{[\![\lambda\, x{:}\, A.t]\!] \simeq \lambda\, x{:}\, A'.t'}$$

$$\textsc{TslApp}\ \frac{[\![u]\!] \simeq u' \qquad [\![v]\!] \simeq v'}{[\![u\ v]\!] \simeq u'\ v'}$$

$$\textsc{TslCoeId}\ \frac{[\![A \rightsquigarrow B]\!] \simeq \star \qquad [\![t]\!] \simeq t'}{[\![\mathrm{coe}_{A,B}\, t]\!] \simeq t'} \qquad \textsc{TslCoe}\ \frac{[\![A \rightsquigarrow B]\!] \simeq f \qquad [\![t]\!] \simeq t'}{[\![\mathrm{coe}_{A,B}\, t]\!] \simeq f\, t'}$$

$$\textsc{TslCoeNf}\ \frac{A \rightsquigarrow^{\star} A'\ \mathrm{nf} \qquad B \rightsquigarrow^{\star} B'\ \mathrm{nf} \qquad [\![A' \rightsquigarrow B']\!] \simeq x \qquad A \neq A' \text{ or } B \neq B'}{[\![A \rightsquigarrow B]\!] \simeq x}$$

$$\textsc{TslCoeLblId}\ [\![\boldsymbol{L} \rightsquigarrow \boldsymbol{L}]\!] \simeq \star \qquad \textsc{TslCoeLbl}\ \frac{L \subsetneq L'}{[\![\boldsymbol{L} \rightsquigarrow \boldsymbol{L}']\!] \simeq \mathtt{inj}_{\boldsymbol{L},\boldsymbol{L}'}}$$

$$\textsc{TslCoeListId}\ \frac{[\![A \rightsquigarrow B]\!] \simeq \star}{[\![\mathbf{List}\ A \rightsquigarrow \mathbf{List}\ B]\!] \simeq \star} \qquad \textsc{TslCoeList}\ \frac{[\![A \rightsquigarrow B]\!] \simeq f}{[\![\mathbf{List}\ A \rightsquigarrow \mathbf{List}\ B]\!] \simeq \mathrm{map}_{\mathbf{List}}\, f}$$

$$\textsc{TslCoePiIdBoth}\ \frac{[\![A_2 \rightsquigarrow A_1]\!] \simeq \star \qquad [\![B_1 \rightsquigarrow B_2]\!] \simeq \star}{[\![\Pi\, x{:}\, A_1.B_1 \rightsquigarrow \Pi\, x{:}\, A_2.B_2]\!] \simeq \star}$$

$$\textsc{TslCoePiIdDom}\ \frac{[\![A_2 \rightsquigarrow A_1]\!] \simeq \star \qquad [\![B_1 \rightsquigarrow B_2]\!] \simeq g \qquad [\![A_1]\!] \simeq A_1'}{[\![\Pi\, x{:}\, A_1.B_1 \rightsquigarrow \Pi\, x{:}\, A_2.B_2]\!] \simeq \mathrm{map}_{\Pi}\, (\mathrm{id}_{A_1'}, \lambda\{x{:}\, A_1'\}.g)}$$

$$\textsc{TslCoePiIdCod}\ \frac{\begin{array}{c}[\![A_2 \rightsquigarrow A_1]\!] \simeq f \\ [\![B_1\left[\mathrm{coe}_{A_2,A_1}\ x\right] \rightsquigarrow B_2]\!] \simeq \star \qquad [\![A_2]\!] \simeq A_2' \qquad [\![B_2]\!] \simeq B_2'\end{array}}{[\![\Pi\, x{:}\, A_1.B_1 \rightsquigarrow \Pi\, x{:}\, A_2.B_2]\!] \simeq \mathrm{map}_\Pi(f, \lambda\{x{:}\, A_2'\}.\, \mathrm{id}_{B_2'})}$$

$$\textsc{TslCoePi}\ \frac{[\![A_2 \rightsquigarrow A_1]\!] \simeq f \qquad [\![B_1\left[\mathrm{coe}_{A_2,A_1}\ x\right] \rightsquigarrow B_2]\!] \simeq g \qquad [\![A_2]\!] \simeq A_2'}{[\![\Pi\, x{:}\, A_1.B_1 \rightsquigarrow \Pi\, x{:}\, A_2.B_2]\!] \simeq \mathrm{map}_\Pi(f, \lambda\{x{:}\, A_2'\}.g)}$$

$$\textsc{TslCoeTy}\ [\![\mathrm{Type}_i \rightsquigarrow \mathrm{Type}_i]\!] \simeq \star \qquad\qquad \textsc{TslCoeNe}\ \frac{\mathrm{ne}\ N \qquad \mathrm{ne}\ M}{[\![N \rightsquigarrow M]\!] \simeq \star}$$

The translation is extended to contexts pointwise.

$$\frac{}{[\![\cdot]\!] \simeq \cdot} \qquad\qquad \frac{[\![\Gamma]\!] \simeq \Gamma' \qquad [\![A]\!] \simeq A'}{[\![\Gamma, x{:}\, A]\!] \simeq \Gamma', x{:}\, A'}$$

We note $[\![t]\!]\downarrow$ when t is in the domain of the relation and $[\![t]\!]$ for the image of t when it is defined.

Lemma C.6 (Determinism of translation). *The translation relation $[\![t]\!] \simeq t'$ is a partial function, i.e. it is deterministic: for any $t\ t_1'\ t_2'$, if $[\![t]\!] \simeq t_1'$ and $[\![t]\!] \simeq t_2'$ then $t_1' = t_2'$.*

Proof. We show by mutual induction on a derivation that $[\![A \rightsquigarrow B]\!] \simeq x$ is a partial function as well from pairs of $\mathrm{MLTT}_{\mathrm{coe}}$ types to either $\star$ or a $\mathrm{MLTT}_{\mathrm{map}}$ term. In the key case TslCoeNf, note that the reduction relation $\rightsquigarrow^\star$ 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. □

Lemma C.7 (Stability of translation by weakening). *If ρ is a substitution that maps variables to variables then $[\![t]\!][\rho] = [\![t[\rho]]\!]$.*

Proof. Immediate by induction on t, the only case interesting case being the translation of variables, with a similar lemma for $[\![A \rightsquigarrow B]\!] \simeq x$ using that neutrals are preserved. □

Lemma C.8 (Well-typed terms translate). *If $\Gamma \vdash_{\mathrm{coe}} t : A$ then $[\![\Gamma]\!]\downarrow$, $[\![A]\!]\downarrow$ and $[\![t]\!]\downarrow$.*

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

- If $\vdash_{\mathrm{coe}} \Gamma$ then $[\![\Gamma]\!]\downarrow$;
- If $\Gamma \vdash_{\mathrm{coe}} A \triangleleft$ and $[\![\Gamma]\!]\downarrow$ then $[\![A]\!]\downarrow$;
- If $\Gamma \vdash_{\mathrm{coe}} t \triangleleft A$ and $[\![\Gamma]\!]\downarrow$ then $[\![t]\!]\downarrow$;
- If $\Gamma \vdash_{\mathrm{coe}} t \triangleright A$ and $[\![\Gamma]\!]\downarrow$ then $[\![t]\!]\downarrow$;
- If $\Gamma \vdash_{\mathrm{coe}} A \preccurlyeq B \triangleleft$ or $\Gamma \vdash_{\mathrm{coe}} A \preccurlyeq_{\mathrm{h}} B \triangleleft$ then there exists x such that $[\![A \rightsquigarrow B]\!] \simeq x$.

□

Lemma C.9 (Identity coercions). *If $\Gamma \vdash_{\mathrm{coe}} A \cong B \triangleleft$ or $\Gamma \vdash_{\mathrm{coe}} A \cong_{\mathrm{h}} B \triangleleft$ then $[\![A \rightsquigarrow B]\!] \simeq \star$.*

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

Lemma C.10 (Stability of translation by substitution). *If $\Gamma \vdash_{\mathrm{coe}} t : A$ and $\Delta \vdash_{\mathrm{coe}} \sigma : \Gamma$ then $[\![t]\!][[\![\sigma]\!]] = [\![t[\sigma]]\!]$ and similarly for typing.*
If $\Gamma \vdash_{\mathrm{coe}} A \preccurlyeq B \triangleleft$, $\Delta \vdash_{\mathrm{coe}} \sigma : \Gamma$ and

- *$[\![A \rightsquigarrow B]\!] \simeq \star$ then $[\![A[\sigma] \rightsquigarrow B[\sigma]]\!] \simeq \star$;*

- $[\![A \rightsquigarrow B]\!] \simeq f$ *then* $[\![A[\sigma] \rightsquigarrow B[\sigma]]\!] \simeq f[\sigma]$.

Proof. Straightforward mutual induction on the bidirectional derivation. □

Forward simulation. Following the proof strategy employed for the equivalence between subsumptive and coercive subtyping, the nest step would require to prove that the translation is a forward simulate, i.e. if $\Gamma \vdash_{\mathrm{coe}} t : A$ and $t \rightsquigarrow^1 t'$ then $[\![t]\!] \rightsquigarrow^\star [\![t']\!]$. As stated, this lemma does not hold. Indeed, the rule CoeCoe 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 $\mathrm{MLTT}_{\mathrm{map}}$ should hold, that is if $\Gamma \vdash_{\mathrm{coe}} t : A$ and $t \rightsquigarrow^1 t'$ then $[\![\Gamma]\!] \vdash_{\mathrm{map}} [\![t]\!] \cong [\![t']\!] : [\![A]\!]$. 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* $\vdash_{\mathrm{coe}} \Gamma$ *then* $\vdash_{\mathrm{map}} [\![\Gamma]\!]$
(2) *If* $\Gamma \vdash_{\mathrm{coe}} A$ *then* $[\![\Gamma]\!] \vdash_{\mathrm{map}} [\![A]\!]$
(3) *If* $\Gamma \vdash_{\mathrm{coe}} t : A$ *then* $[\![\Gamma]\!] \vdash_{\mathrm{map}} [\![t]\!] : [\![A]\!]$
(4) *If* $\Gamma \vdash_{\mathrm{coe}} A \cong B$ *then* $[\![\Gamma]\!] \vdash_{\mathrm{map}} [\![A]\!] \cong [\![B]\!]$
(5) *If* $\Gamma \vdash_{\mathrm{coe}} t \cong u : A$ *then* $[\![\Gamma]\!] \vdash_{\mathrm{map}} [\![t]\!] \cong [\![u]\!] : [\![A]\!]$
(6) *If* $\Gamma \vdash_{\mathrm{coe}} A \preccurlyeq B$ *then either*
   (a) $[\![A \rightsquigarrow B]\!] \simeq \star$ *and* $[\![\Gamma]\!] \vdash_{\mathrm{map}} [\![A]\!] \cong [\![B]\!]$
   (b) $[\![A \rightsquigarrow B]\!] \simeq f$ *and* $[\![\Gamma]\!] \vdash_{\mathrm{map}} f : [\![A]\!] \to [\![B]\!]$

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

Conjecture C.12 (Embedding). $[\![-]\!]$ *embeds* $MLTT_{\mathrm{coe}}$ *into* $MLTT_{\mathrm{map}}$*: well-typed* $MLTT_{\mathrm{coe}}$ *terms translate to well-typed* $MLTT_{\mathrm{map}}$ *terms, preserving and reflecting conversion.*