Compiling Programs with Erased Univalence

(Extended version)

ANDREAS ABEL, University of Gothenburg, Sweden
NILS ANDERS DANIELSSON, University of Gothenburg, Sweden
ANDREA VEZZOSI, IT University Copenhagen, Denmark

We present a variant of cubical type theory with an erasure modality to mark data that should be erased by compilers. In this variant glue types—used to prove univalence—as well as higher constructors may only be used in compile-time code. We show, under certain assumptions, that every closed, run-time natural number reduces to the right value after erasure.

We have developed a variant of Cubical Agda based on the ideas presented here, thereby providing what appears to be the first compiler for some variant of cubical type theory, and we present a case study intended to illustrate that the resulting language is useful in practice.

1 INTRODUCTION

Programs written in dependently typed functional languages often contain "proofs" mixed up with the program logic. By proofs we mean code that does not actually influence the result of the program, yet is present to ensure that, say, all pattern matches are exhaustive, or that some invariant is not broken. Such proofs can affect the performance of a program, including its asymptotic complexity [Tejiščák 2020]. As a simple example, consider the following type of fixed-length lists in Agda [The Agda Team 2021]:

$$\begin{aligned}&\textbf{data}\ \mathit{Vec}\ (A : \mathit{Type}\ a) : \mathbb{N} \to \mathit{Type}\ a\ \textbf{where}\\ &\quad [\,]\quad : \mathit{Vec}\ A\ \mathsf{zero}\\ &\quad \_{::}\_ : \forall\ \{n\} \to A \to \mathit{Vec}\ A\ n \to \mathit{Vec}\ A\ (\mathsf{suc}\ n)\end{aligned} \tag{1}$$

Note the implicit argument n of the list constructor: this argument does not need to be given explicitly if it can be inferred from the context, but (depending on what optimisations are used) it might still be present at run-time.

Brady et al. [2004] have presented a way to automatically erase the argument n, but this method cannot handle every conceivable situation in which one might want to erase something [Tejiščák 2020]. One way to address this problem is to give programmers the means to state that certain data should be erased by the compiler, and to let the type-checker ensure that such marked data indeed does not affect the results of run-time computations. There are a number of variants of this approach [Paulin-Mohring 1989; Paulin-Mohring and Werner 1993; Van Raamsdonk and Severi 2002; Letouzey 2003; Fernandez et al. 2003; Barras and Bernardo 2008; Mishra-Linger and Sheard 2008; Mishra-Linger 2008; Gundry and McBride 2013; Bernardy and Moulin 2013; Gundry 2013; Sjöberg 2015; McBride 2016; Weirich et al. 2017; Atkey 2018; Tejiščák 2020; Brady 2021]. For instance, in Agda one can mark function and constructor arguments as well as certain definitions as erased using the attribute @0 (or *@erased*):

$$\begin{aligned}&\textbf{data}\ \mathit{Vec}\ (A : \mathit{Type}\ a) : \mathbb{N} \to \mathit{Type}\ a\ \textbf{where}\\ &\quad [\,]\quad : \mathit{Vec}\ A\ \mathsf{zero}\\ &\quad \_{::}\_ : \forall\ \{@0\ n\} \to A \to \mathit{Vec}\ A\ n \to \mathit{Vec}\ A\ (\mathsf{suc}\ n)\end{aligned} \tag{2}$$

Cubical Agda [Vezzosi et al. 2019] is a variant of Agda with support for higher inductive types (which can be used to define things like quotient types) and the univalence axiom [The Univalent Foundations Program 2013]. In fact, the univalence axiom is no axiom in Cubical Agda: it can be proved, and computes. This "axiom" can be used both in mathematics and in program verification.

For instance, in some situations it can be used to “transport” proofs from an inefficient but simple data structure to an efficient but more complicated one [Tabareau et al. 2021; Angiuli et al. 2021b].

If the univalence axiom computes, does this mean that one can use it without restriction for programming? No, at least not yet. Compiling Cubical Agda programs is nontrivial: the computation rules involve evaluation under binders, and typical compilation techniques for functional languages only support computation for closed terms. Perhaps this can be addressed following work on compiled execution of open Coq terms [Grégoire and Leroy 2002; Boespflug et al. 2011], but we suspect that such an approach would lead to overheads for code that does not use cubical features. Instead we take a different approach:

- We have implemented a variant of Cubical Agda in which certain cubical features may only be used in an erased setting. In particular, the feature which is used to prove univalence, *Glue*, may not be used in run-time code. However, the equality type (the type of *paths*) can be used in run-time code.
- We also introduce the concept of an erased constructor, i.e. a constructor which may only be used in an erased setting (see Section 2). Higher inductive types may only be used if all the higher constructors are erased.
- With these restrictions in place it is easy to compile Cubical Agda programs. However, we note that it is important to design the rules for erasure correctly: it is easy to end up with a broken system (see Section 3).
- A type-checker and a compiler are available as part of a currently unreleased version of Agda,[1] and the implementation is discussed in Section 5.
- We demonstrate by a larger case study[2] that a system with only erased univalence and only erased higher constructors can be useful in practice, see Section 6. In fact, we have also found a use for erased *regular* (point) constructors, see Section 6.2.
- In Section 4 we outline a correctness proof for a small language, $\mathsf{CTT}^{0\omega}$, with some key features of the Cubical Agda implementation. (We do not make any claims about the correctness of Cubical Agda, which is a large piece of software.) Note that the proof relies on some metatheoretical results, for instance injectivity of type formers, that we simply assume (see Conjecture 4.1). We believe that these assumptions can be proven by extending Huber’s work on canonicity [2019] to our theory. Note also that Sterling and Angiuli [2021] have recently presented a normalisation proof for cubical type theory, although not based on reduction.

To the best of our knowledge this piece of work provides the first integration of erasure and cubical type theory, as well as the first “reasonable” way to compile some variant of cubical type theory. (One could presumably “compile” a program by pairing up its source code with an interpreter.) We are also not aware of any previous work on erased constructors. Related work is discussed further in Section 7.

2 CUBICAL AGDA

Cubical Agda [Vezzosi et al. 2019] is a variant of Agda with support for a variant of cubical type theory (CTT) [Cohen et al. 2018b; Angiuli et al. 2021a]. This section contains an introduction to Cubical Agda, including its support for erasure.

Let us start by discussing erasure. We do not explain all the nuances here, see the typing rules in Section 4.1 for more details. As mentioned above one can mark arguments as erased using @0. For instance, the following function takes one erased argument, and one that is not erased:

[1] At the time of writing available at https://github.com/agda/agda/tree/5018aa45c18e3bd2b7de323c789daefc920cbbe7.

[2] The code for this case study is at the time of writing available at https://www.cse.chalmers.se/~nad/.

$$\begin{aligned}&\mathit{const} : \mathit{Bool} \to @0\ \mathit{Bool} \to \mathit{Bool}\\&\mathit{const}\ x\ \_ = x\end{aligned} \tag{3}$$

Variables corresponding to erased arguments can only be used in "erased contexts", for instance to construct erased arguments:

$$\begin{aligned}&\mathit{const\text{-}true} : @0\ \mathit{Bool} \to \mathit{Bool}\\&\mathit{const\text{-}true}\ y = \mathit{const}\ \mathsf{true}\ y\end{aligned} \tag{4}$$

Run-time decisions must not be made based on erased data, so the following piece of code is rejected:

$$\begin{aligned}&\mathit{not} : @0\ \mathit{Bool} \to \mathit{Bool}\\&\mathit{not}\ \mathsf{true}\ \ = \mathsf{false}\\&\mathit{not}\ \mathsf{false} = \mathsf{true}\end{aligned} \tag{5}$$

Top-level function definitions can also be marked as erased, in which case they can only be used in erased contexts, but on the other hand erased names can be used in the bodies of such definitions (with an exception related to pattern-matching lambdas that we prefer not to discuss here).

A key part of Cubical Agda is the notion of a *path*, which is a kind of equality. Paths of type $x \equiv y$ are functions from the *interval* I, subject to the restriction that they map the two endpoints of the interval, $\underline{0}$ and $\underline{1}$, to x and y, respectively. This makes it easy to prove that equality is a congruence, and that equality of functions is extensional:

$$\begin{aligned}&\mathit{cong} : (f : A \to B) \to x \equiv y \to f\ x \equiv f\ y\\&\mathit{cong}\ f\ \mathit{eq} = \lambda\ i \to f\ (\mathit{eq}\ i)\end{aligned} \tag{6}$$

$$\begin{aligned}&\mathit{ext} : (\forall\ x \to f\ x \equiv g\ x) \to f \equiv g\\&\mathit{ext}\ \mathit{eq} = \lambda\ i\ x \to \mathit{eq}\ x\ i\end{aligned} \tag{7}$$

To avoid clutter we use Agda's generalisable variables [The Agda Team 2021], which make it possible to specify once and for all what the types of undeclared variables like f should be. The full type of *cong* is (almost) the following one:

$$\{a : \mathit{Level}\}\ \{A : \mathit{Type}\ a\}\ \{b : \mathit{Level}\}\ \{B : \mathit{Type}\ b\}\ \{x\ y : A\}\ (f : A \to B) \to x \equiv y \to f\ x \equiv f\ y \tag{8}$$

Here *Level* is the type of *universe levels*, and *Type a* is the universe with universe level a. Arguments in braces are *implicit*, and do not have to be given explicitly if they can be inferred by Agda. Below we omit argument specifications, but for clarity we take care to never omit any *erased* argument.

One can use a path by applying it to an interval argument. One can also *transport* via a path, using the following function:

$$\mathit{transp} : \{p : I \to \mathit{Level}\}\ (P : (i : I) \to \mathit{Type}\ (p\ i)) \to I \to P\ \underline{0} \to P\ \underline{1} \tag{9}$$

This type signature comes with a side condition. For full details, see Section 4.1.4. However, let us consider two cases here: If the interval argument is $\underline{1}$, then the function returns its final argument, so P must be definitionally constant. If the interval argument is $\underline{0}$, then there is no side condition, and the computational behaviour depends on P.

Cubical Agda also supports *higher inductive types*. Such types are data types that can have both regular (*point*) constructors and *higher* constructors. Here is an example, the propositional truncation operator [The Univalent Foundations Program 2013]:

$$\begin{aligned}&\mathbf{data}\ \|\_\|\ (A : \mathit{Type}\ a) : \mathit{Type}\ a\ \mathbf{where}\\&\quad |\_|\ \ \ \ \ : A \to \|\ A\ \|\\&\quad \mathsf{trunc} : (x\ y : \|\ A\ \|) \to x \equiv y\end{aligned} \tag{10}$$

The higher constructor trunc ensures that all elements of the propositional truncation of a type are equal. The following map function illustrates how functions from the propositional truncation can be defined by pattern matching:

$$\begin{aligned} &map : (A \to B) \to \| A \| \to \| B \| \\ &map\, f \mid x \mid \qquad\qquad = \mid f\ x \mid \\ &map\, f\ (\mathsf{trunc}\ x\ y\ i) = \mathsf{trunc}\ (map\, f\ x)\ (map\, f\ y)\ i \end{aligned} \tag{11}$$

Note that the constructor trunc is applied to *three* arguments in the left-hand side. This ensures that the application has the same type, namely A, as the corresponding pattern on the previous line. Agda imposes two side conditions on the right-hand side of the final clause: If $\underline{0}$ is substituted for i, then the right-hand side must be definitionally equal to $map\, f\ (\mathsf{trunc}\ x\ y\ \underline{0})$, i.e. $map\, f\ x$. Similarly, if $\underline{1}$ is substituted for i, then the right-hand side must be definitionally equal to $map\, f\ (\mathsf{trunc}\ x\ y\ \underline{1})$, i.e. $map\, f\ y$.

As part of the work presented in this text we have made it possible to mark constructors as erased. Here is a contrived example:

$$\begin{aligned} &\mathbf{data}\ D : \mathit{Type}\ \mathbf{where} \\ &\quad \mathsf{here} \qquad : D \\ &\quad @0\ \mathsf{gone} : D \end{aligned} \tag{12}$$

An erased constructor cannot be used in run-time code, and conversely the right-hand side of a function clause that contains a match on an erased constructor *in a non-erased position* is not run-time code. For instance, the following code is allowed:

$$\begin{aligned} &ok : D \to @0\ \mathit{Bool} \to \mathit{Bool} \\ &ok\ \mathsf{here}\ \ \_ = \mathsf{true} \\ &ok\ \mathsf{gone}\ x = x \end{aligned} \tag{13}$$

Note that the erased argument x can be used in the right-hand side of the final clause. The constructor gone is not present at run-time, so this clause can only trigger at compile-time. However, the following code is not allowed:

$$\begin{aligned} &bad : @0\ D \to \mathit{Bool} \\ &bad\ \mathsf{here}\ \ = \mathsf{true} \\ &bad\ \mathsf{gone} = \mathsf{false} \end{aligned} \tag{14}$$

If *bad* were allowed, then both *bad* here and *bad* gone would be valid pieces of run-time code. The first one is equal to true, and the second one is equal to false, but the only difference between these two pieces of code is the first argument of *bad*, which is erased.

A prime motivation for erased constructors is to have higher inductive types in which the higher constructors are erased, and hence guaranteed not to interfere in the execution of run-time code. Here is one example, the propositional truncation operator with an erased truncation constructor:

$$\begin{aligned} &\mathbf{data}\ \|\_\|^{\mathrm{E}}\ (A : \mathit{Type}\ a) : \mathit{Type}\ a\ \mathbf{where} \\ &\quad |\_| \qquad\quad : A \to \| A \|^{\mathrm{E}} \\ &\quad @0\ \mathsf{trunc} : (x\ y : \| A \|^{\mathrm{E}}) \to x \equiv y \end{aligned} \tag{15}$$

The propositional truncation operator (10) is recursive, so it might be non-trivial to predict the performance of code that uses it. However, for this variant of truncation only the point constructor |_| is available at run-time.

One may wonder if there is any point in allowing erased *point* constructors. We have found a use for this, see Section 6.

The type of (half adjoint) *equivalences* [The Univalent Foundations Program 2013] from the type A to the type B can be defined in the following way:

$$\begin{aligned}&\mathit{Is\text{-}equivalence} : \{A : \mathit{Type}\ a\}\ \{B : \mathit{Type}\ b\} \to (A \to B) \to \mathit{Type}\ (a \sqcup b)\\ &\mathit{Is\text{-}equivalence}\ \{A = A\}\ \{B = B\}\ f =\\ &\quad (f^{-1} : B \to A) \times (f\text{-}f^{-1} : \forall\ x \to f\ (f^{-1}\ x) \equiv x) \times (f^{-1}\text{-}f : \forall\ x \to f^{-1}\ (f\ x) \equiv x) \times\\ &\quad \forall\ x \to \mathit{cong}\ f\ (f^{-1}\text{-}f\ x) \equiv f\text{-}f^{-1}\ (f\ x)\end{aligned} \tag{16}$$

$$\begin{aligned}&\_\simeq\_ : \mathit{Type}\ a \to \mathit{Type}\ b \to \mathit{Type}\ (a \sqcup b)\\ &A \simeq B = (f : A \to B) \times \mathit{Is\text{-}equivalence}\ f\end{aligned} \tag{17}$$

Here the function $\_\sqcup\_$ returns the least upper bound of two universe levels, and the notation $\{x = y\}$ is used to bind the variable y to the implicit argument x. We use the notation $(x : A) \times P\ x$—which is not currently valid Agda code—for Σ-types, i.e. pairs where the type of the second component can depend on the value of the first component. The two projections are called fst and snd.

One can map paths to equivalences using *transp* and the identity equivalence *id≃*:

$$\begin{aligned}&{\equiv}{\to}{\simeq} : (A\ B : \mathit{Type}\ a) \to A \equiv B \to A \simeq B\\ &{\equiv}{\to}{\simeq}\ A\ \_\ eq = \mathit{transp}\ (\lambda\ i \to A \simeq eq\ i)\ \underline{0}\ \mathit{id}{\simeq}\end{aligned} \tag{18}$$

Univalence can be stated in the following way (where *lsuc* is the successor function for universe levels):

$$\begin{aligned}&\mathit{Univalence} : (a : \mathit{Level}) \to \mathit{Type}\ (\mathit{lsuc}\ a)\\ &\mathit{Univalence}\ a = \{A\ B : \mathit{Type}\ a\} \to \mathit{Is\text{-}equivalence}\ ({\equiv}{\to}{\simeq}\ A\ B)\end{aligned} \tag{19}$$

A consequence of univalence is that equivalent types (in the same universe *Type a*) are equal. Univalence can be proved in Cubical Agda using the *Glue* type former [Cohen et al. 2018b; Vezzosi et al. 2019].

3 POSTULATING ERASED UNIVALENCE

In this text we discuss how one can have a system where univalence can be used in erased settings, but not at run-time. One might wonder if one could avoid the theoretical development of this paper by just postulating univalence, in such a way that the postulate could only be used in erased code. This would mean that univalence would not compute at compile-time, and one might also miss another benefit of our approach, namely that one can prove function extensionality in a non-erased setting (7). However, would anything worse happen?

This section contains some examples that show that certain typing rules are problematic in the presence of erased univalence (postulated or not): we construct pairs of terms where one is provably equal to true and one to false, but where the only differences are erased by the compiler.

Unless otherwise noted we use the typing rules of Agda 2.6.2, with the `--with-K` flag, in this section. However, we do not make use of the K rule, which is incompatible with (even postulated) univalence. Rather, when the K rule is turned on using `--with-K` Agda allows some other things that are potentially incompatible with univalence. Here we give some examples related to erasure that were discovered by us.

Let us consider the equality, or identity, type defined as an inductive family in the following way:

$$\begin{aligned}&\mathbf{data}\ \mathit{Id}\ \{A : \mathit{Type}\ a\}\ (x : A) : A \to \mathit{Type}\ a\ \mathbf{where}\\ &\quad \mathsf{refl} : \mathit{Id}\ x\ x\end{aligned} \tag{20}$$

Brady et al. [2004] show that, in a certain setting, one can erase values of types like this one. The idea is that, in a closed context, values of type *Id x y* must be constructed using the only constructor.

One might thus expect that it would be fine to use the following definition, where the identity proof argument is erased:

$$\begin{aligned}&\mathit{subst}_2 : (P : A \to \mathit{Type}\ p) \to @0\ \mathit{Id}\ x\ y \to P\ x \to P\ y\\&\mathit{subst}_2\ \_\ \mathsf{refl}\ p = p\end{aligned} \tag{21}$$

This is valid Agda code (when the `--with-K` flag is enabled), and Idris 2 (version 0.3.0-2287a7ff3) supports something similar. However, in the presence of erased univalence (expressed using *Id*) this definition is problematic. If we postulate erased univalence, then we can construct an erased proof corresponding to the not function:

$$@0\ \mathit{not} : \mathit{Id}\ \mathit{Bool}\ \mathit{Bool} \tag{22}$$

We can thus construct the following terms:

$$\begin{aligned}&\mathit{should\text{-}be\text{-}true} : \mathit{Bool}\\&\mathit{should\text{-}be\text{-}true} = \mathit{subst}_2\ (\lambda\ B \to B)\ \mathsf{refl}\ \mathsf{true}\end{aligned} \tag{23}$$

$$\begin{aligned}&\mathit{should\text{-}be\text{-}false} : \mathit{Bool}\\&\mathit{should\text{-}be\text{-}false} = \mathit{subst}_2\ (\lambda\ B \to B)\ \mathit{not}\ \mathsf{true}\end{aligned} \tag{24}$$

In an erased context we can also prove that the first term is equal to true, while the second term is equal to false. However, the second argument of subst_2 is erased, thus the only difference between these two terms is erased.

We break for the definition of the erasure type [Mishra-Linger 2008] that incorporates the erasure modality as a type constructor. It can be encoded in $\mathsf{CTT}^{0\omega}$ (see Section 4.1.2) and we shall use it for our case study (Section 6) and for the continued discussion of *subst*.

$$\begin{aligned}&\textbf{record}\ \mathit{Erased}\ (@0\ A : \mathit{Type}\ a) : \mathit{Type}\ a\ \textbf{where}\\&\quad\textbf{constructor}\ [\_]\\&\quad\textbf{field}\ @0\ \mathsf{erased} : A\end{aligned} \tag{25}$$

Note that the only field is erased. The erasure annotation @0 cannot be used everywhere. For instance, the following piece of code is not valid: $\mathit{Bool} \times (@0\ \mathit{Bool})$. In this case one can instead use *Erased*: $\mathit{Bool} \times \mathit{Erased}\ \mathit{Bool}$. Danielsson [2019] has investigated some of the theory of *Erased*, and the following is a key lemma (here expressed using *Id*):

$$\begin{aligned}&[]\text{-}\mathit{cong} : \{@0\ A : \mathit{Type}\ a\}\ \{@0\ x\ y : A\} \to \mathit{Erased}\ (\mathit{Id}\ x\ y) \to \mathit{Id}\ [\ x\ ]\ [\ y\ ]\\&[]\text{-}\mathit{cong}\ [\ \mathsf{refl}\ ] = \mathsf{refl}\end{aligned} \tag{26}$$

Again Idris 2 accepts similar code.

Picking up our discussion, the definition of subst_2 above, with an erased second argument, is problematic in the presence of erased univalence. What about the following variant, with an erased *first* argument?

$$\begin{aligned}&\mathit{subst}_1 : (@0\ P : A \to \mathit{Type}\ p) \to \mathit{Id}\ x\ y \to P\ x \to P\ y\\&\mathit{subst}_1\ \_\ \mathsf{refl}\ p = p\end{aligned} \tag{27}$$

(This definition is currently allowed by Cubical Agda, which does not have proper support for inductive families like *Id*, but there are plans to add proper support for inductive families and at the same time make Cubical Agda reject this definition.) Note that the first argument is unused. However, the combination of this definition and $[]$-*cong* is problematic. Consider the following two definitions:

$$\begin{aligned}&\mathit{should\text{-}be\text{-}true} : \mathit{Bool}\\&\mathit{should\text{-}be\text{-}true} = \mathit{subst}_1\ (\lambda\ ([\ B\ ]) \to B)\ ([]\text{-}\mathit{cong}\ [\ \mathsf{refl}\ ])\ \mathsf{true}\end{aligned} \tag{28}$$

$$\begin{aligned}&\mathit{should\text{-}be\text{-}false} : \mathit{Bool}\\&\mathit{should\text{-}be\text{-}false} = \mathit{subst}_1\ (\lambda\ ([\ B\ ]) \to B)\ ([\,]\text{-}\mathit{cong}\ [\ \mathit{not}\ ])\ \mathsf{true}\end{aligned} \tag{29}$$

Because the first argument of subst_1 is erased Agda allows us to return the erased variable B. Again we can prove, in an erased context, that the first term is equal to true, while the second term is equal to false. Furthermore the argument of [_] is erased, so the only difference between these two terms is erased (given the assumptions mentioned above).

Let us instead use the following standard variant of subst_1 and subst_2, with no erased arguments:

$$\begin{aligned}&\mathit{subst} : (P : A \to \mathit{Type}\ p) \to \mathit{Id}\ x\ y \to P\ x \to P\ y\\&\mathit{subst}\ \_\ \mathsf{refl}\ p = p\end{aligned} \tag{30}$$

Let us now consider the following type:

$$\begin{aligned}&\textbf{record}\ \mathit{Box}\ (@0\ A : \mathit{Type}\ a) : \mathit{Type}\ a\ \textbf{where}\\&\quad \textbf{constructor}\ [\_]\\&\quad \textbf{field}\ \mathsf{unbox} : A\end{aligned} \tag{31}$$

This type is also problematic, and again Idris 2 accepts similar code. The problem is that the erased type argument A is used in a non-erased context. We can use *Box* to construct the following definition, and prove (in an erased context) that it is equal to false:

$$\begin{aligned}&\mathit{should\text{-}be\text{-}false} : \mathit{Bool}\\&\mathit{should\text{-}be\text{-}false} = \mathsf{unbox}\ (\mathit{subst}\ (\lambda\ ([\ A\ ]) \to \mathit{Box}\ A)\ ([\,]\text{-}\mathit{cong}\ [\ \mathit{not}\ ])\ [\ \mathsf{true}\ ])\end{aligned} \tag{32}$$

If the arguments of the Π or Σ type constructors were erased, then we could also construct similar examples:

$$\begin{aligned}&\mathit{should\text{-}be\text{-}false} : \mathit{Bool}\\&\mathit{should\text{-}be\text{-}false} = \mathit{subst}\ (\lambda\ ([\ A\ ]) \to \top \to A)\ ([\,]\text{-}\mathit{cong}\ [\ \mathit{not}\ ])\ (\lambda\ \_ \to \mathsf{true})\ \mathsf{tt}\end{aligned} \tag{33}$$

$$\begin{aligned}&\mathit{should\text{-}be\text{-}false} : \mathit{Bool}\\&\mathit{should\text{-}be\text{-}false} = \mathit{snd}\ (\mathit{subst}\ (\lambda\ ([\ A\ ]) \to \top \times A)\ ([\,]\text{-}\mathit{cong}\ [\ \mathit{not}\ ])\ (\mathsf{tt}\ ,\ \mathsf{true}))\end{aligned} \tag{34}$$

(Here ⊤ is the unit type, with constructor tt. The first example is not valid Agda code, because the function type $\top \to A$, mentioning the erased A, is ill-formed in non-erased position. The second example could have been constructed—with the K rule turned on—if the Σ-type had been defined using erased type arguments.)

These examples show that, even though it might be possible to postulate erased univalence, running the resulting compiled code might not give the intended result if the typing rules are not designed correctly. We would also like to point out that problems like this could affect Cubical Agda. For instance, if the first explicit argument of *transp* (9) were erased, then we could define functions corresponding to subst_1 and subst_2, but expressed using paths instead of *Id*. Note that a variant of $[\,]$-*cong* expressed using paths can be proved in Cubical Agda:

$$\begin{aligned}&[\,]\text{-}\mathit{cong} : \{@0\ A : \mathit{Type}\ a\}\ \{@0\ x\ y : A\} \to \mathit{Erased}\ (x \equiv y) \to [\ x\ ] \equiv [\ y\ ]\\&[\,]\text{-}\mathit{cong}\ [\ \mathit{eq}\ ] = \lambda\ i \to [\ \mathit{eq}\ i\ ]\end{aligned} \tag{35}$$

After having marked pitfalls in our terrain, let us turn to the formal presentation of $\mathsf{CTT}^{0\omega}$.

4 CUBICAL TYPE THEORY WITH ERASURE

4.1 The Type Theory

In this section we present $\mathsf{CTT}^{0\omega}$, a variant of Cubical Type Theory [Cohen et al. 2018b] augmented with erasure annotations on variables and one higher inductive type, namely propositional truncation $\|A\|^{\mathrm{E}}$. The grammar is given in Figure 1 as an overview; the role of the individual constructs will become clearer with the typing rules.

p, q	$::=$	$0 \mid \omega$	erasure modalities
r, s	$::=$	$0 \mid 1 \mid r \vee s \mid r \wedge s \mid 1 - r \mid i$	coordinates (interval expressions)
φ, ψ	$::=$	$0_{\mathbb{F}} \mid 1_{\mathbb{F}} \mid \varphi \vee \psi \mid \varphi \wedge \psi \mid i = 0 \mid i = 1$	constraints (face expr.s, $_{\mathbb{F}}$ often omitted)
Γ, Δ	$::=$	ε	empty typing context (usually omitted)
	$\mid$	$\Gamma, x :^{p} A$	assumption (runtime-available if $p = \omega$)
	$\mid$	$\Gamma, i : \mathbb{I}$	interval variable declaration
	$\mid$	Γ, φ	assumption of constraint
$A, B, C, T, a, e, t, u, v, w$			types and terms
	$::=$	x	variable
	$\mid$	U_n	n-th universe
	$\mid$	$(x :^{p} A) \to B \mid \lambda x^{p}.\, t \mid t\,{}^{p} u$	Π formation, introduction, elimination
	$\mid$	$(x :^{p} A) \times B \mid \langle^{p} t_1,\ t_2\rangle \mid \pi_i\, t$	Σ form, intro, elim
	$\mid$	$\top \mid \langle\rangle$	unit form, intro
	$\mid$	$\bot \mid \bot\text{-elim}_A\, t$	empty form, elim
	$\mid$	$\mathbb{N} \mid \mathsf{zero} \mid \mathsf{suc}\, t \mid \mathbb{N}\text{-elim}_{z.C}\, u\, (x^{p} y^{q}.v)\, t$	Nat form, intros, elim
	$\mid$	$\mathsf{Path}\, A\, t\, u \mid \lambda i.\, t \mid t\, r$	path form, intro, elim
	$\mid$	$\mathsf{transp}^{i}\, A\, \varphi\, u_0$	transport
	$\mid$	$\mathsf{hcomp}^{i}_{A}\, [\varphi \mapsto u]\, u_0$	homogeneous composition
	$\mid$	$[\varphi_1 \hookrightarrow u_1, \ldots, \varphi_n \hookrightarrow u_n]$	system
	$\mid$	$\|A\|^{\mathrm{E}}$	propositional truncation formation
	$\mid$	$\mathsf{tr}\, t \mid \mathsf{trunc}\, t\, u\, r$	propositional truncation introductions
	$\mid$	$\|\|\|^{\mathrm{E}}\text{-elim}\, (x.C)\, (x.t)\, (x.y.i.u)\, w$	propositional truncation elimination
	$\mid$	$\mathsf{hcomp\text{-}intro}_{i.B}\, [\varphi \mapsto t]\, a$	homogeneous comp. introduction
	$\mid$	$\mathsf{hcomp\text{-}elim}\, [\varphi \mapsto i.B]\, t$	homogeneous comp. elimination
	$\mid$	$\mathsf{Glue}\, [\varphi \mapsto (T, e)]\, A$	glue formation
	$\mid$	$\mathsf{glue}\, [\varphi \mapsto t]\, a$	glue introduction
	$\mid$	$\mathsf{unglue}\, [\varphi \mapsto e]\, t$	glue elimination

Fig. 1. Syntax

Figure 2 shows how $\mathsf{CTT}^{0\omega}$-expressions are erased to an untyped λ-calculus with booleans, tuples, natural numbers and a wrapper construct $\mathsf{tr}\, v$ with elimination $\|\|\|^{\mathrm{E}}\text{-elim}\, (x.t)\, w$ for which $\|\|\|^{\mathrm{E}}\text{-elim}\, (x.t)\, (\mathsf{tr}\, v)$ reduces to $t[v/x]$. The n-clause conditional $[b_1 \hookrightarrow v_1, \ldots, b_n \hookrightarrow v_n]$ reduces to the first branch v_k for which the guard b_k evaluates to 1 (true).

The dummy $\lightning$ replaces certain erased subexpressions. Our typing rules aim to guarantee that $\lightning$ will never appear in evaluation position during execution of the compiled program (Corollary 4.3). Canonical types (U, Π, Σ, $\top$, $\bot$, $\mathbb{N}$, Path, $\|\_\|^{\mathrm{E}}$, Glue) compile to $\lightning$, as well as $\bot$-elim and expressions that can only appear or happen to appear in erased position. The operational semantics of the target language is weak head call-by-name reduction $v \rightsquigarrow^{*} w$, where $\lightning$ is an inert constant.

In the presentation to follow, the reader is invited to match the compilation function $|t|$ with the typing rules for runtime terms $\Gamma \vdash t :^{\omega} A$.

Target language.

u, v, w, b	$::=$	$\lightning$	dummy
	$\mid$	$x \mid \lambda x.\, v \mid v\, w \mid \langle\rangle \mid \langle v,\, w\rangle \mid \pi_k\, v$	lambda calculus with tuples
	$\mid$	$\mathsf{zero} \mid \mathsf{suc}\, v \mid \mathbb{N}\text{-}\mathsf{elim}\, u\, (xy.v)\, w$	natural numbers
	$\mid$	$\mathsf{tr}\, v \mid \|\|\|^{\mathrm{E}}\text{-}\mathsf{elim}\, (x.v)\, w$	truncation (wrapper)
	$\mid$	$0 \mid 1 \mid b_1 \vee b_2 \mid b_1 \wedge b_2 \mid 1 - b$	booleans
	$\mid$	$[b_1 \hookrightarrow v_1, \ldots, b_n \hookrightarrow v_n]$	n-clause conditional

Erasure $|t|$ of expressions t.

$\|T\|$	$= \lightning$ (T type)	$\|\mathbb{N}\text{-}\mathsf{elim}_{z.C}\, u\, (x^p y^q.v)\, t\|$	$= \mathbb{N}\text{-}\mathsf{elim}\, \|u\|\, (xy.\|v\|)\, \|t\|$
$\|x\|$	$= x$	$\|\lambda i.\, t\|$	$= \lambda i.\, \|t\|$
$\|\lambda x^0.\, t\|$	$= \lambda\_.\, \|t\|$	$\|t\, r\|$	$= \|t\|\, \|r\|$
$\|\lambda x^{\omega}.\, t\|$	$= \lambda x.\, \|t\|$	$\|\mathsf{transp}^i\, A\, \varphi\, u_0\|$	$= \|u_0\|$
$\|t\, {}^0 u\|$	$= \|t\|\, \lightning$	$\|\mathsf{hcomp}^i_A\, [\varphi \mapsto u]\, u_0\|$	$= \|u_0\|$
$\|t\, {}^{\omega} u\|$	$= \|t\|\, \|u\|$	$\|[\varphi_1 \hookrightarrow u_1, \ldots, \varphi_n \hookrightarrow u_n]\|$	$= [\|\varphi_1\| \hookrightarrow \|u_1\|, \ldots, \|\varphi_n\| \hookrightarrow \|u_n\|]$
$\|\langle^0 t,\, u\rangle\|$	$= \langle \lightning,\, \|u\|\rangle$	$\|\mathsf{tr}\, t\|$	$= \mathsf{tr}\, \|t\|$
$\|\langle^{\omega} t,\, u\rangle\|$	$= \langle \|t\|,\, \|u\|\rangle$	$\|\mathsf{trunc}\, t\, u\, r\|$	$= \lightning$
$\|\pi_k\, t\|$	$= \pi_k\, \|t\|$	$\|\|\|\|^{\mathrm{E}}\text{-}\mathsf{elim}\, (x.C)\, (x.t)\, (x.y.i.u)\, w\|$	$= \|\|\|^{\mathrm{E}}\text{-}\mathsf{elim}\, (x.\|t\|)\, \|w\|$
$\|\langle\rangle\|$	$= \langle\rangle$	$\|\mathsf{hcomp}\text{-}\mathsf{intro}_{i.B}\, [\varphi \mapsto t]\, a\|$	$= \|a\|$
$\|\bot\text{-}\mathsf{elim}_A\, t\|$	$= \lightning$	$\|\mathsf{hcomp}\text{-}\mathsf{elim}\, [\varphi \mapsto i.B]\, t\|$	$= \|t\|$
$\|\mathsf{zero}\|$	$= \mathsf{zero}$	$\|\mathsf{glue}\, [\varphi \mapsto t]\, a\|$	$= \lightning$
$\|\mathsf{suc}\, t\|$	$= \mathsf{suc}\, \|t\|$	$\|\mathsf{unglue}\, [\varphi \mapsto e]\, t\|$	$= \lightning$

Erasure $|r|$ of coordinates r to booleans is homeomorphic.
Erasure $|\varphi|$ of constraints φ to booleans is homeomorphic, except for $|i = 1| = i$ and $|i = 0| = 1 - i$.

Fig. 2. Erasure function

$\vdash \Gamma$	context Γ is well-formed		
$\Gamma \vdash t :^p A$	in context Γ, term t has type A and erasure status p		
$\Gamma \vdash r : \mathbb{I}$	in context Γ, coordinate r is well-formed		
$\Gamma \vdash \varphi : \mathbb{F}$	in context Γ, constraint φ is well-formed		
$\Gamma \vdash t = u : A$	in context Γ, terms t and u of type A are equal		
$\Gamma \vdash r = s : \mathbb{I}$	in context Γ, coordinates r and s are equal		
$\Gamma \vdash \varphi = \psi : \mathbb{F}$	in context Γ, constraints φ and ψ are equal		
$\Gamma \vdash A$	$\iff$	$\exists n.\, \Gamma \vdash A :^0 \mathsf{U}_n$	in context Γ, type A is well-formed
$\Gamma \vdash A = B$	$\iff$	$\exists n.\, \Gamma \vdash A = B : \mathsf{U}_n$	in context Γ, types A and B are equal

---

Fig. 3. Judgements

4.1.1 Judgments. Figure 3 lists the judgements of $\mathsf{CTT}^{0\omega}$. The judgements are designed to enjoy the following standard properties (but we have not proved this in detail):

(1) (Context well-formedness, weakening, substitution:) All judgements of the form $\Gamma \vdash J$ entail $\vdash \Gamma$ and are closed under weakening with wellformed context extensions and under well-typed substitution.
(2) (Syntactic validity/presupposition:) The judgements $\Gamma \vdash t :^p A$ and $\Gamma \vdash t = u : A$ entail $\Gamma \vdash A$. Judgement $\Gamma \vdash t = u : A$ is designed to entail both $\Gamma \vdash t :^0 A$ and $\Gamma \vdash u :^0 A$, and analogously for the other equality judgements. These entailments allow us to drop redundant premises from the typing rules.
(3) (Subsumption:) If $\Gamma \vdash t :^\omega A$ then $\Gamma \vdash t :^0 A$ and analogously for the other typing judgements. From this follows a context subsumption property: If $\Gamma, x :^0 A, \Delta \vdash J$ then $\Gamma, x :^\omega A, \Delta \vdash J$.

Judgemental equality is only defined for the sake of type conversion, and types to the right of the typing judgement's colon are not present at runtime, so premises of equality rules are typed in the erased world (0).

In the following we present typing and equality rules for $\mathsf{CTT}^{0\omega}$, starting with standard type formers from Martin-Löf Type Theory and moving on to the cubical parts. Due to lack of space we do not spell out all equality rules, e.g. not the congruence rules, or rules covered by the literature that are not essential for the discussion.

4.1.2 Standard Type Theory. Our augmentation of Martin-Löf Type Theory with erasure annotations is based on the presentations of McBride [2016] and Atkey [2018], but there are some notable differences: On the one hand, even though the erasure function erases all type constructors, types are not always treated as runtime irrelevant in the type system, because then we could construct problematic examples like some of those discussed in Section 3. Thus we sometimes require that types are well-formed in the non-erased world ($\Gamma \vdash A :^\omega \mathsf{U}_n$), and we assign erasure modalities more carefully in the type formation rules. On the other hand, because we do not support linearity but only erasure, we have full weakening and can freely project the components of a record.

Contexts, universes and conversion. $\mathsf{CTT}^{0\omega}$ has a non-cumulative infinite hierarchy of predicative universes U_n ($n \in \mathbb{N}$). Non-cumulativity is not essential, but simplifies the presentation as we do not need a subtyping relation.

$$\frac{}{\vdash \varepsilon} \qquad \frac{\vdash \Gamma \qquad \Gamma \vdash A}{\vdash \Gamma, x :^p A}\ x \notin \mathsf{dom}(\Gamma) \qquad \frac{\vdash \Gamma}{\vdash \Gamma, i : \mathbb{I}}\ i \notin \mathsf{dom}(\Gamma) \qquad \frac{\vdash \Gamma \qquad \Gamma \vdash \varphi : \mathbb{F}}{\vdash \Gamma, \varphi}$$

$$\textsc{univ}\ \frac{\vdash \Gamma}{\Gamma \vdash \mathsf{U}_n :^p \mathsf{U}_{n+1}} \qquad \textsc{conv}\ \frac{\Gamma \vdash t :^p A \qquad \Gamma \vdash A = B}{\Gamma \vdash t :^p B}$$

Functions. In $\mathsf{CTT}^{0\omega}$ we have two dependent function types, the ordinary $(x :^\omega A) \to B$, and the "erased Π-type" that types functions whose argument cannot be inspected at runtime, thus, can be erased to a dummy value by the compiler (erasing it completely might change the behaviour under call-by-value [Letouzey 2003], thus, we abstain). The following rules ensure the correctness of erasure annotations, e.g. in applications, so that compiled programs do not go wrong.

In the variable rule we use modality subsumption $q \leq p$ which holds unless $q = 0$ and $p = \omega$. Thus non-erased variables ($q = \omega$) can always be used, and any variable can be used in an erased position ($p = 0$). The product pq of two modalities is 0 unless both are ω. The product is used in the application rule (ΠE), where the argument u is checked at the quantity pq, which is 0 if either p or q is 0. Thus the argument is checked at quantity 0 if we are in an erased context ($p = 0$), or if the function treats its argument as erased ($q = 0$).

$$\text{VAR}\ \frac{\vdash \Gamma \qquad (x :^q A) \in \Gamma}{\Gamma \vdash x :^p A}\ q \leq p \qquad \Pi\text{F}\ \frac{\Gamma \vdash A :^{pq} \mathsf{U}_m \qquad \Gamma, x :^q A \vdash B :^p \mathsf{U}_n}{\Gamma \vdash (x :^q A) \to B :^p \mathsf{U}_{\max(m,n)}}$$

$$\Pi\text{I}\ \frac{\Gamma \vdash (x :^q A) \to B \qquad \Gamma, x :^q A \vdash t :^p B}{\Gamma \vdash \lambda x^q.\, t :^p (x :^q A) \to B} \qquad \Pi\text{E}\ \frac{\Gamma \vdash t :^p (x :^q A) \to B \qquad \Gamma \vdash u :^{pq} A}{\Gamma \vdash t\, {}^q u :^p B[u/x]}$$

$$\Pi\beta\ \frac{\Gamma, x :^q A \vdash t :^0 B \qquad \Gamma \vdash u :^0 A}{\Gamma \vdash (\lambda x^q.\, t)\, {}^q u = t[u/x] : B[u/x]} \qquad \Pi\eta\ \frac{\Gamma \vdash t :^0 (x :^q A) \to B}{\Gamma \vdash \lambda x^q.\, (t\, {}^q x) = t : (x :^q A) \to B}$$

Note that the modality of a variable is irrelevant in the erased world ($p = 0$). Thus, in rule Πβ we could change the hypothesis $x :^q A$ to $x :^{q'} A$ for any other q'. Likewise, we could change the hypothesis $x :^p A$ in rules ΠF and ΠI to $x :^{pq} A$ without changing the typing relation: If $q = \omega$, then $pq = p$, and if $q = 0$, then the modality of x does not matter at all.

Some comment is in order for the typing $\Gamma \vdash A :^{pq} \mathsf{U}_m$ of the domain in Π-formation (ΠF). If the function argument is erased ($q = 0$), then any cubical transport happens in erased context and is thus runtime-irrelevant. Thus, we then do not need A at runtime.

Pairing. Components of a tuple can be marked as erased; following Atkey [2018] we achieve this by putting an erasure annotation in the first component of a pair $\langle {}^p t_1,\, t_2 \rangle$. The second component can also be erased if it is wrapped in another pair (see Section 4.1.2). The formation of Σ-types follows Π-types, for analogous reasons: If the first component of a pair is erased, we do not need to transport it at runtime, thus, its type A can be erased as well.

$$\Sigma\text{F}\ \frac{\Gamma \vdash A :^{pq} \mathsf{U}_m \qquad \Gamma, x :^q A \vdash B :^p \mathsf{U}_n}{\Gamma \vdash (x :^q A) \times B :^p \mathsf{U}_{\max(m,n)}} \qquad \Sigma\text{E}_1\ \frac{\Gamma \vdash t :^{pq} (x :^q A) \times B}{\Gamma \vdash \pi_1\, t :^{pq} A}$$

$$\Sigma\text{I}\ \frac{\Gamma \vdash (x :^q A) \times B \qquad \Gamma \vdash t_1 :^{pq} A \qquad \Gamma \vdash t_2 :^p B[t_1/x]}{\Gamma \vdash \langle {}^q t_1,\, t_2 \rangle :^p (x :^q A) \times B} \qquad \Sigma\text{E}_2\ \frac{\Gamma \vdash t :^p (x :^q A) \times B}{\Gamma \vdash \pi_2\, t :^p B[\pi_1\, t/x]}$$

$$\Sigma\beta\ \frac{\Gamma \vdash t_k :^0 A_k \quad (\forall k = 1, 2)}{\Gamma \vdash \pi_k \langle {}^q t_1,\, t_2 \rangle = t_k : A_k} \qquad \Sigma\eta\ \frac{\Gamma \vdash t :^0 (x :^q A) \times B}{\Gamma \vdash t = \langle {}^q \pi_1\, t,\, \pi_2\, t \rangle : (x :^q A) \times B}$$

Unit and empty type. $\mathsf{CTT}^{0\omega}$ has a unit type ⊤ with η-equality and an empty type ⊥ with *ex falso quodlibet*. Note that even in the non-erased world ⊥-elim takes an *erased* proof of ⊥, this means that at compile-time information can flow from the erased to the non-erased world. Yet as there is no closed term of type ⊥, compilation can erase ⊥-elim as a whole, fixing the "leak".

$$\top\text{F}\ \frac{\vdash \Gamma}{\Gamma \vdash \top :^p \mathsf{U}_n} \qquad \top\text{I}\ \frac{\vdash \Gamma}{\Gamma \vdash \langle\rangle :^p \top} \qquad \top\eta\ \frac{\Gamma \vdash t :^0 \top}{\Gamma \vdash t = \langle\rangle : \top} \qquad \bot\text{F}\ \frac{\vdash \Gamma}{\Gamma \vdash \bot :^p \mathsf{U}_n} \qquad \bot\text{E}\ \frac{\Gamma \vdash A \qquad \Gamma \vdash t :^0 \bot}{\Gamma \vdash \bot\text{-}\mathsf{elim}_A\, t :^p A}$$

Natural numbers. The eliminator ℕ-elim for unary numbers can be configured such that the step term s may not use the current number x at runtime ($q = 0$), turning it into an iterator (so natural numbers act as Church numerals, cf. McBride [2016, Section 5]). Furthermore the recursive call y can be unavailable at runtime (when $r = 0$), then ℕ-elim is simply a case distinction on the scrutinee t.

$$\mathbb{N}\text{F}\ \frac{\vdash \Gamma}{\Gamma \vdash \mathbb{N} :^p \mathsf{U}_n} \qquad \mathbb{N}\text{I}_1\ \frac{\vdash \Gamma}{\Gamma \vdash \mathsf{zero} :^p \mathbb{N}} \qquad \mathbb{N}\text{I}_2\ \frac{\Gamma \vdash t :^p \mathbb{N}}{\Gamma \vdash \mathsf{suc}\, t :^p \mathbb{N}}$$

$$\mathbb{N}\text{E}\ \frac{\Gamma, x :^\omega \mathbb{N} \vdash A \qquad \Gamma \vdash z :^p A[\mathsf{zero}/x] \qquad \Gamma, x :^q \mathbb{N}, y :^r A \vdash s :^p A[\mathsf{suc}\, x/x] \qquad \Gamma \vdash t :^p \mathbb{N}}{\Gamma \vdash \mathbb{N}\text{-}\mathsf{elim}_{x.A}\, z\, (x^q y^r.s)\, t :^p A[t/x]}$$

The $\mathbb{N}$-eliminator comes with the usual β-equalities.

Erasure type. Record types can be encoded as nested Σ-types ending in $\top$. This way, each field can be given an erasure status. In particular, we can encode the erasure type (25) of Section 3 as a record with a single erased field:

$$\begin{array}{lcl} \mathsf{Erased}\, A & = & (\_ :^0 A) \times \top \\ [t] & = & \langle {}^0 t,\ \langle\rangle\rangle \\ \mathsf{erased}\, t & = & \pi_1\, t \end{array}$$

The reader is invited to check that the following typing and equality rules for Erased are derivable.

$$\frac{\Gamma \vdash A :^0 \mathsf{U}_n}{\Gamma \vdash \mathsf{Erased}\, A :^p \mathsf{U}_n} \qquad \frac{\Gamma \vdash t :^0 A}{\Gamma \vdash [t] :^p \mathsf{Erased}\, A} \qquad \frac{\Gamma \vdash t :^0 \mathsf{Erased}\, A}{\Gamma \vdash \mathsf{erased}\, t :^0 A}$$

$$\frac{\Gamma \vdash t :^0 A}{\Gamma \vdash \mathsf{erased}\, [t] = t : A} \qquad \frac{\Gamma \vdash t :^0 \mathsf{Erased}\, A}{\Gamma \vdash [\mathsf{erased}\, t] = t : \mathsf{Erased}\, A}$$

4.1.3 Coordinates, Constraints and Partial Elements. Following Cohen et al. [2018b], a coordinate r is wellformed, $\Gamma \vdash r : \mathbb{I}$, if $(i : \mathbb{I}) \in \Gamma$ for all free variables i in r. The same holds for well-formed constraints $\Gamma \vdash \varphi : \mathbb{F}$.

Equalities $\Gamma \vdash r = s : \mathbb{I}$ are derivable if they hold by the laws of De Morgan algebras, i.e., any Boolean algebra minus the laws of excluded middle ($r \vee (1 - r) = 1$) and noncontradiction ($r \wedge (1-r) = 0$). The standard model is the real interval $[0; 1]$ with $\vee$ being maximum and $\wedge$ minimum. Equalities $\Gamma \vdash \varphi = \psi : \mathbb{F}$ follow from the laws of distributive lattices and $(i = 0) \wedge (i = 1) = 0_{\mathbb{F}}$. Moreover both equality judgments include the congruence induced by context *restrictions* Γ, φ. Examples of valid equalities are:

$$\begin{array}{lcl} i : \mathbb{I}, j : \mathbb{I}, i = 0, i = 1 & \vdash & j = 1 : \mathbb{I} \\ i : \mathbb{I}, j : \mathbb{I}, i = 0 \vee j = 0, i = 1 & \vdash & (j = 0) = 1 : \mathbb{F} \end{array}$$

A collection of constraints $\{\varphi_k\}_{k=1..n}$ is *covering* in Γ if $\Gamma \vdash (\varphi_1 \vee \ldots \vee \varphi_n) = 1 : \mathbb{F}$. The rule Cover allows judgements to branch on covering constraints:

$$\textsc{Cover}\ \frac{\Gamma, \varphi_1 \vdash J \quad \ldots \quad \Gamma, \varphi_n \vdash J}{\Gamma \vdash J}\ \Gamma \vdash (\varphi_1 \vee \ldots \vee \varphi_n) = 1 : \mathbb{F}$$

A so-called *system* is a n-ary conditional $[\varphi_1 \hookrightarrow t_1, \ldots, \varphi_n \hookrightarrow t_n]$ whose conditions are covering. Systems reduce to one of their branches t_k if the condition φ_k holds. Systems need to be consistent, so if two conditions φ_k and φ_l hold, the respective branches t_k and t_l must be judgementally equal.

$$\textsc{SysF}\ \frac{\Gamma, \varphi_k \vdash t_k :^p A\ (\forall k) \qquad \Gamma, \varphi_k \wedge \varphi_l \vdash t_k = t_l : A\ (\forall k \neq l)}{\Gamma \vdash [\varphi_1 \hookrightarrow t_1, \ldots, \varphi_n \hookrightarrow t_n] :^p A}\ \Gamma \vdash (\varphi_1 \vee \ldots \vee \varphi_n) = 1 : \mathbb{F}$$

$$\textsc{Sys}\beta\ \frac{\Gamma \vdash [\varphi_1 \hookrightarrow t_1, \ldots, \varphi_n \hookrightarrow t_n] :^0 A \qquad \Gamma \vdash \varphi_k = 1 : \mathbb{F}}{\Gamma \vdash [\varphi_1 \hookrightarrow t_1, \ldots, \varphi_n \hookrightarrow t_n] = t_k : A}\Gamma \vdash (\varphi_1 \vee \ldots \vee \varphi_n) = 1 : \mathbb{F}$$

For a *partial element* $\Gamma, \varphi \vdash u :^p A$, existing under the constraint φ, $\boxed{\Gamma \vdash t :^p A[\varphi \mapsto u]}$ is an abbreviation for the conjunction of $\Gamma \vdash t :^p A$ and $\Gamma, \varphi \vdash t = u : A$. This notation allows us to constrain terms and we will use it both in premises and conclusions of rules (where it specifies a second consequence). Further details and motivations are presented by Cohen et al. [2018b].

4.1.4 Basic Path Primitives. Figure 4 lists rules for paths, the cubical replacement for propositional equality. On top of the Path type former itself and the transport operation transp, already discussed previously, we also introduce the *homogeneous composition* operator $\mathsf{hcomp}^i_A\,[\varphi \mapsto u]\,u_0$ which allows to compose paths together, e.g. to implement transitivity of path equality: given $p : \mathsf{Path}\,A\,x\,y$ and $q : \mathsf{Path}\,A\,y\,z$ we can define a path $\mathsf{Path}\,A\,x\,z$ by $\lambda i.\,\mathsf{hcomp}^j_A\,[(i = 0) \mapsto x, (i = 1) \mapsto q j]\,(p i)$. While a more comprehensive introduction to these primitives is given by Vezzosi et al. [2019], let us note that the A argument of transp needs to be provided at quantity ω when the operation is used at quantity ω, preventing the problematic variants of *subst* discussed in Section 3.

Both transp and hcomp also have associated judgmental equalities which depend on the type A. For these we mostly adopt the formulation presented by Huber [2017], with the term typing premises at quantity 0. The only exception is $\mathsf{hcomp}_{\mathsf{U}_n}$, which we handle as a dedicated type former with its own associated judgmental equalities for transp and hcomp. These equalities can be constructed from the reduction rules in Section 4.2.7 by dropping the premise $\Gamma \vdash \varphi \neq 1_{\mathbb{F}} : \mathbb{F}$.

4.1.5 Glue Types and Homogeneous Composition in the Universe. Cohen et al. [2018b] reduce composition in the universe to a use of Glue by turning the (partial) line in the universe into an equivalence. In the prototype implementation by Cohen et al. [2018a] this was found to be quite inefficient and a dedicated type former was introduced as an optimization instead. For our purposes having $\mathsf{hcomp}^i_{\mathsf{U}_n}\,[\varphi \mapsto B]\,A$ as its own type former allows transports over compositions in the universe to compute without the use of Glue, so that reduction (Section 4.2) of a term typed at ω will always result in a term typed at ω. The introduction and elimination forms for $\mathsf{hcomp}^i_{\mathsf{U}_n}\,[\varphi \mapsto B]\,A$ are designed (see Figure 4) so that they are compatible with the equality $\mathsf{hcomp}^i_{\mathsf{U}_n}\,[1_{\mathbb{F}} \mapsto B]\,A = B[1/i]$ and so that we can prove that the type is equivalent to A. In particular $\mathsf{hcomp\text{-}elim}\,[\varphi \mapsto i.B]$ is the equivalence map, which agrees with transport along B when $\varphi = 1_{\mathbb{F}}$. The rules for Glue, glue, and unglue are taken directly from Cohen et al. [2018b], as those terms are in the 0 fragment of the theory.

(See Figure 5.)

4.1.6 Propositional Truncation. The following formalizes the higher inductive type $\|\_\|^{\mathrm{E}}$ (15), albeit with a dedicated eliminator.

$$\textsc{TruncF}\quad \frac{\Gamma \vdash A :^p \mathsf{U}_m}{\Gamma \vdash \|A\|^{\mathrm{E}} :^p \mathsf{U}_m} \qquad \textsc{TruncI}_1\quad \frac{\Gamma \vdash t :^p A}{\Gamma \vdash \mathsf{tr}\,t :^p \|A\|^{\mathrm{E}}}$$

$$\textsc{TruncI}_2\quad \frac{\Gamma \vdash A \qquad \Gamma \vdash t, u :^0 \|A\|^{\mathrm{E}} \qquad \Gamma \vdash r : \mathbb{I}}{\Gamma \vdash \mathsf{trunc}\,t\,u\,r :^0 \|A\|^{\mathrm{E}} \begin{bmatrix} r = 0 & \mapsto & t \\ r = 1 & \mapsto & u \end{bmatrix}}$$

$$\textsc{TruncE}\quad \frac{\begin{matrix} \Gamma \vdash A \qquad \Gamma, x :^p \|A\|^{\mathrm{E}} \vdash C :^p \mathsf{U}_k \qquad \Gamma, x :^p A \vdash t :^p C[\mathsf{tr}\,x/x] \\ \Gamma, x\,y :^0 \|A\|^{\mathrm{E}}, i : \mathbb{I} \vdash u :^0 C[\mathsf{trunc}\,x\,y\,i/x][i = 0 \mapsto x, i = 1 \mapsto y] \qquad \Gamma \vdash w :^p \|A\|^{\mathrm{E}} \end{matrix}}{\Gamma \vdash \|\|^{\mathrm{E}}\text{-}\mathsf{elim}\,(x.C)\,(x.t)\,(x.y.i.u)\,w :^p C[w/x]}$$

$$\textsc{PathF}\ \frac{\Gamma \vdash A :^p \mathsf{U}_n \qquad \Gamma \vdash t, u :^p A}{\Gamma \vdash \mathsf{Path}\, A\, t\, u :^p \mathsf{U}_n}$$

$$\textsc{PathI}\ \frac{\Gamma, i : \mathbb{I} \vdash t :^p A}{\Gamma \vdash \lambda i.t :^p \mathsf{Path}\, A\ t[0/i]\ t[1/i]} \qquad \textsc{PathE}\ \frac{\Gamma \vdash A \qquad \Gamma \vdash t :^p \mathsf{Path}\, A\, a_0\, a_1 \qquad \Gamma \vdash r : \mathbb{I}}{\Gamma \vdash t\, r :^p A}$$

$$\textsc{PathB}\ \frac{\Gamma \vdash A \qquad \Gamma \vdash t :^0 \mathsf{Path}\, A\, a_0\, a_1}{\Gamma \vdash t\, b = a_b : A}\ b \in \{0,1\} \qquad \textsc{Path}\beta\ \frac{\Gamma, i : \mathbb{I} \vdash t :^0 A \qquad \Gamma \vdash r : \mathbb{I}}{\Gamma \vdash (\lambda i.t) r = t[r/i] : A}$$

$$\textsc{Path}\eta\ \frac{\Gamma \vdash t, u :^0 \mathsf{Path}\, A\, a_0\, a_1 \qquad \Gamma, i : \mathbb{I} \vdash t\, i = u\, i : A}{\Gamma \vdash t = u : \mathsf{Path}\, A\, a_0\, a_1}$$

$$\textsc{Transp}\ \frac{\Gamma, i : \mathbb{I} \vdash A :^p \mathsf{U}_n \qquad \Gamma \vdash \varphi : \mathbb{F} \qquad \Gamma, i : \mathbb{I}, \varphi \vdash A[0/i] = A : \mathsf{U}_n \qquad \Gamma \vdash u :^p A[0/i]}{\Gamma \vdash \mathsf{transp}^i\, A\, \varphi\, u :^p A[1/i][\varphi \mapsto u]}$$

$$\textsc{HCompF}\ \frac{\Gamma \vdash A :^p \mathsf{U}_n \qquad \Gamma \vdash \varphi : \mathbb{F} \qquad \Gamma, i : \mathbb{I}, \varphi \vdash u :^p A \qquad \Gamma \vdash u_0 :^p A[\varphi \mapsto u[0/i]]}{\Gamma \vdash \mathsf{hcomp}^i_A\ [\varphi \mapsto u]\ u_0 :^p A[\varphi \mapsto u[1/i]]}$$

$$\textsc{HCompI}\ \frac{\begin{array}{c}\Gamma, i : \mathbb{I}, \varphi \vdash B :^p \mathsf{U}_n \qquad \Gamma, \varphi \vdash B[0/i] = A : \mathsf{U}_n \\ \Gamma, \varphi \vdash t :^p B[1/i] \qquad \Gamma \vdash a :^p A \qquad \Gamma, \varphi \vdash \mathsf{transp}^j\, B[1-j/i]\, 0\, t = a : A\end{array}}{\Gamma \vdash \mathsf{hcomp\text{-}intro}_{i.B}\ [\varphi \mapsto t]\ a :^p (\mathsf{hcomp}^i_{\mathsf{U}_n}\ [\varphi \mapsto B]\ A)[\varphi \mapsto t]}$$

$$\textsc{HCompE}\ \frac{\Gamma, i : \mathbb{I}, \varphi \vdash B :^p \mathsf{U}_n \qquad \Gamma \vdash B[0/i] = A : \mathsf{U}_n \qquad \Gamma \vdash u :^p \mathsf{hcomp}^i_{\mathsf{U}_n}\ [\varphi \mapsto B]\ A}{\Gamma \vdash \mathsf{hcomp\text{-}elim}\ [\varphi \mapsto i.B]\ u :^p A[\varphi \mapsto \mathsf{transp}^j\, B[1-j/i]\, 0\, u]}$$

$$\textsc{HComp}\beta\ \frac{\begin{array}{c}\Gamma, i : \mathbb{I}, \varphi \vdash B :^0 \mathsf{U}_n \qquad \Gamma \vdash B[0/i] = A : \mathsf{U}_n \\ \Gamma, \varphi \vdash t :^0 B[1/i] \qquad \Gamma \vdash a :^0 A \qquad \Gamma, \varphi \vdash \mathsf{transp}^j\, B[1-j/i]\, 0\, t = a : A\end{array}}{\Gamma \vdash \mathsf{hcomp\text{-}elim}\ [\varphi \mapsto i.B]\ (\mathsf{hcomp\text{-}intro}_{i.B}\ [\varphi \mapsto t]\ a) = a : A}$$

$$\textsc{HComp}\eta\ \frac{\Gamma, i : \mathbb{I}, \varphi \vdash B :^0 \mathsf{U}_n \qquad \Gamma \vdash B[0/i] = A : \mathsf{U}_n \qquad \Gamma \vdash u :^0 \mathsf{hcomp}^i_{\mathsf{U}_n}\ [\varphi \mapsto B]\ A}{\Gamma \vdash \mathsf{hcomp\text{-}intro}_{i.B}\ [\varphi \mapsto u]\ (\mathsf{hcomp\text{-}elim}\ [\varphi \mapsto i.B]\ u) = u : \mathsf{hcomp}^i_{\mathsf{U}_n}\ [\varphi \mapsto B]\ A}$$

Fig. 4. Paths and homogeneous composition.

Writing f for $\|\|^{\mathrm{E}}$-elim $(x.C)\ (x.t)\ (x.y.i.u)$ we have the following β-rules:

$$\begin{aligned} f\ (\mathsf{tr}\, a) &= t[a/x] \\ f\ (\mathsf{trunc}\, t_0\, t_1\, r) &= u[t_0/x, t_1/y, r/i] \\ f\ (\mathsf{hcomp}^i_{\|A\|^{\mathrm{E}}}\ [\varphi \mapsto u]\ u_0) &= \mathsf{comp}^i\, C[\mathsf{hfill}^i_{\|A\|^{\mathrm{E}}}\ [\varphi \mapsto u]\ u_0]\ [\varphi \mapsto f\, u]\ (f\, u_0) \end{aligned}$$

$$\textsc{GlueF}\ \frac{\Gamma \vdash A :^0 \mathsf{U}_n \qquad \Gamma \vdash \varphi : \mathbb{F} \qquad \Gamma, \varphi \vdash T :^0 \mathsf{U}_n \qquad \Gamma, \varphi \vdash e :^0 \mathsf{Equiv}\, T\, A}{\Gamma \vdash \mathsf{Glue}\, [\varphi \mapsto (T, e)]\, A :^0 \mathsf{U}_n[\varphi \mapsto \mathsf{U}_n]}$$

$$\textsc{GlueI}\ \frac{\begin{array}{c}\Gamma \vdash \varphi : \mathbb{F} \\ \Gamma, \varphi \vdash e :^0 \mathsf{Equiv}\, T\, A \qquad \Gamma, \varphi \vdash t :^0 T \qquad \Gamma \vdash a :^0 A \qquad \Gamma, \varphi \vdash (\pi_1\, e)\ {}^{\omega} t = a : A\end{array}}{\Gamma \vdash \mathsf{glue}\, [\varphi \mapsto t]\, a :^0 (\mathsf{Glue}\, [\varphi \mapsto (T, e)]\, A)[\varphi \mapsto t]}$$

$$\textsc{GlueE}\ \frac{\Gamma, \varphi \vdash T \qquad \Gamma \vdash A \qquad \Gamma, \varphi \vdash e :^0 \mathsf{Equiv}\, T\, A \qquad \Gamma \vdash u :^0 \mathsf{Glue}\, [\varphi \mapsto (T, e)]\, A}{\Gamma \vdash \mathsf{unglue}\, [\varphi \mapsto e]\, u :^0 A[\varphi \mapsto (\pi_1\, e)\ {}^{\omega} u}$$

$$\textsc{Glue}\beta\ \frac{\begin{array}{c}\Gamma \vdash \varphi : \mathbb{F} \\ \Gamma, \varphi \vdash e :^0 \mathsf{Equiv}\, T\, A \qquad \Gamma, \varphi \vdash t :^0 T \qquad \Gamma \vdash a :^0 A \qquad \Gamma, \varphi \vdash (\pi_1\, e)\ {}^{\omega} t = a : A\end{array}}{\Gamma \vdash \mathsf{unglue}\, [\varphi \mapsto e]\, (\mathsf{glue}\, [\varphi \mapsto t]\, a) = a : A}$$

$$\textsc{Glue}\eta\ \frac{\Gamma, \varphi \vdash T \qquad \Gamma \vdash A \qquad \Gamma, \varphi \vdash e :^0 \mathsf{Equiv}\, T\, A \qquad \Gamma \vdash u :^0 \mathsf{Glue}\, [\varphi \mapsto (T, e)]\, A}{\Gamma \vdash \mathsf{glue}\, [\varphi \mapsto u]\, (\mathsf{unglue}\, [\varphi \mapsto e]\, u) = u : \mathsf{Glue}\, [\varphi \mapsto (T, e)]\, A}$$

Fig. 5. Glue.

4.2 Reduction

The operational semantics of $\mathsf{CTT}^{0\omega}$ is given by a weak head call-by-name typed reduction relation. While judgemental equality is only needed in the erased world for type conversion, we are interested to reduce in the non-erased world for the sake of computation. Thus, the relation takes the form $\boxed{\Gamma \vdash t \Rightarrow u :^p A}$. We have designed this relation so that if it holds, then $\Gamma \vdash t, u :^p A$ and $\Gamma \vdash t = u : A$ also hold (but again we have not proved this in detail). Ignoring modalities, reduction follows Huber [2019, 2017], except for composition in the universe, which does not reduce to Glue, as explained before. It may seem odd that we define reduction for terms typed at 0, however our proof of correctness for erasure uses the fact that types reduce to canonical forms at quantity 0.

Reduction includes all β-equalities and the congruence rules for evaluation contexts, as usual in weak head reduction. Rule Sysβ is non-deterministic, so in the reduction we make it deterministic by picking the first branch of the system whose constraint evaluates to 1. For terms such as $\mathsf{transp}^i\, A\, \varphi\, u$, $\mathsf{hcomp}^i\, A\, \varphi\, u$, or applications of glue/unglue, hcomp-intro/hcomp-elim, which have overlapping equality rules when φ is $1_{\mathbb{F}}$, we make reduction deterministic by requiring $\Gamma \vdash \varphi \neq 1_{\mathbb{F}} : \mathbb{F}$ for the more general rule. Below we present only non-obvious rules or rules that we did not cover in the case of equality.

4.2.1 Conversion.

$$\frac{\Gamma \vdash u \Rightarrow v :^p A \qquad \Gamma \vdash A = B}{\Gamma \vdash u \Rightarrow v :^p B}$$

4.2.2 Partial Terms.

$$\frac{\begin{array}{c}\Gamma \vdash \vee_i\, \varphi_i = 1 : \mathbb{F} \qquad \Gamma \vdash A \qquad \Gamma, \varphi_i \vdash t_i :^p A\ (\forall i \in \{1 \dots n\}) \\ \Gamma, \varphi_i \wedge \varphi_j \vdash t_i = t_j : A\ (\forall i, j \in \{1 \dots n\}) \qquad k \text{ minimal with } \Gamma \vdash \varphi_k = 1 : \mathbb{F}\end{array}}{\Gamma \vdash [\varphi_1 \hookrightarrow t_1, \dots, \varphi_n \hookrightarrow t_n] \Rightarrow t_k :^p A}$$

4.2.3 Universe.

$$\frac{\Gamma \vdash A :^0 \mathsf{U}_n \qquad \Gamma, \varphi \vdash T :^0 \mathsf{U}_n \qquad \Gamma, \varphi \vdash e :^0 \mathsf{Equiv}\, T\, A \qquad \Gamma \vdash \varphi = 1 : \mathbb{F}}{\Gamma \vdash \mathsf{Glue}\, [\varphi \mapsto (T, e)]\, A \Rightarrow T :^0 \mathsf{U}_n}$$

Note: $\mathsf{hcomp}_{\mathsf{U}_n}$ is covered by the general rule for hcomp.

4.2.4 Path.

$$\frac{\Gamma \vdash A \qquad \Gamma, i : \mathbb{I} \vdash t :^p A \qquad \Gamma \vdash r : \mathbb{I}}{\Gamma \vdash (\lambda i.t)\, r \Rightarrow t[r/i] :^p A} \qquad \frac{\Gamma \vdash t \Rightarrow t' :^p \mathsf{Path}\, A\, u\, v \qquad \Gamma \vdash r : \mathbb{I}}{\Gamma \vdash t\, r \Rightarrow t'\, r :^p A}$$

4.2.5 Glue. The reduction rules are taken from Huber [2019], placing both conclusion and premises in the 0 fragment.

$$\frac{\Gamma, \varphi \vdash e :^0 \mathsf{Equiv}\, A\, T \qquad \Gamma, \varphi \vdash t :^0 T \qquad \Gamma \vdash a :^0 A[\varphi \mapsto \mathsf{fst}\, e \cdot^\omega t] \qquad \Gamma \vdash \varphi = 1 : \mathbb{F}}{\Gamma \vdash \mathsf{glue}\, [\varphi \mapsto t]\, a \Rightarrow t :^0 T}$$

$$\frac{\Gamma, \varphi \vdash e :^0 \mathsf{Equiv}\, A\, T \qquad \Gamma, \varphi \vdash t :^0 T \qquad \Gamma \vdash a :^0 A[\varphi \mapsto \mathsf{fst}\, e \cdot^\omega t] \qquad \Gamma \vdash \varphi \neq 1 : \mathbb{F}}{\Gamma \vdash \mathsf{unglue}\, [\varphi \mapsto e]\, (\mathsf{glue}\, [\varphi \mapsto t]\, a) \Rightarrow a :^0 A}$$

$$\frac{\Gamma, \varphi \vdash e :^0 \mathsf{Equiv}\, A\, T \qquad \Gamma \vdash u :^0 \mathsf{Glue}\, [\varphi \mapsto (T, e)]\, A \qquad \Gamma \vdash \varphi = 1 : \mathbb{F}}{\Gamma \vdash \mathsf{unglue}\, [\varphi \mapsto e]\, u \Rightarrow \mathsf{fst}\, e \cdot^\omega u :^0 A}$$

$$\frac{\Gamma \vdash u \Rightarrow u' :^0 \mathsf{Glue}\, [\varphi \mapsto (T, e)]\, A \qquad \Gamma \vdash \varphi \neq 1 : \mathbb{F}}{\Gamma \vdash \mathsf{unglue}\, [\varphi \mapsto e]\, u \Rightarrow \mathsf{unglue}\, [\varphi \mapsto e]\, u' :^0 A}$$

4.2.6 Homogeneous Composition in the Universe.

$$\frac{\begin{array}{c}\Gamma, i : \mathbb{I}, \varphi \vdash B :^p \mathsf{U}_n \qquad \Gamma \vdash A[\varphi \mapsto B[0/i]] \\ \Gamma, \varphi \vdash t :^p B[1/i] \qquad \Gamma \vdash a :^p A[\varphi \mapsto \mathsf{transp}^j\, B[1-j/i]\, 0\, t] \qquad \Gamma \vdash \varphi = 1 : \mathbb{F}\end{array}}{\Gamma \vdash \mathsf{hcomp\text{-}intro}_{i.B}\, [\varphi \mapsto t]\, a \Rightarrow t :^p B[1/i]}$$

$$\frac{\Gamma, \varphi, i : \mathbb{I} \vdash B :^p \mathsf{U}_n \qquad \Gamma \vdash u :^p \mathsf{hcomp}^i_{\mathsf{U}_n}\, [\varphi \mapsto B]\, A \qquad \Gamma \vdash \varphi = 1 : \mathbb{F}}{\Gamma \vdash \mathsf{hcomp\text{-}elim}\, [\varphi \mapsto i.B]\, u \Rightarrow \mathsf{transp}^j\, B[1-j/i]\, 0\, u :^0 A}$$

$$\frac{\begin{array}{c}\Gamma, i : \mathbb{I}, \varphi \vdash B, B' :^p \mathsf{U}_n \qquad \Gamma, i : \mathbb{I}, \varphi \vdash B = B' : \mathsf{U}_n \qquad \Gamma \vdash A[\varphi \mapsto B[0/i]] \\ \Gamma, \varphi \vdash t :^p B[1/i] \qquad \Gamma \vdash a :^p A[\varphi \mapsto \mathsf{transp}^j\, B[1-j/i]\, 0\, t] \qquad \Gamma \vdash \varphi \neq 1 : \mathbb{F}\end{array}}{\Gamma \vdash \mathsf{hcomp\text{-}elim}\, [\varphi \mapsto i.B']\, (\mathsf{hcomp\text{-}intro}_{i.B}\, [\varphi \mapsto t]\, a) \Rightarrow a :^p A}$$

$$\frac{\Gamma, \varphi, i : \mathbb{I} \vdash B :^p \mathsf{U}_n \qquad \Gamma \vdash u \Rightarrow u' :^p \mathsf{hcomp}^i_{\mathsf{U}_n}\, [\varphi \mapsto B]\, A \qquad \Gamma \vdash \varphi \neq 1 : \mathbb{F}}{\Gamma \vdash \mathsf{hcomp\text{-}elim}\, [\varphi \mapsto i.B]\, u \Rightarrow \mathsf{hcomp\text{-}elim}\, [\varphi \mapsto i.B]\, u' :^0 A}$$

4.2.7 transp and hcomp.

$$\frac{\Gamma \vdash A :^p \mathsf{U}_n \qquad \Gamma \vdash \varphi : \mathbb{F} \qquad \Gamma, i : \mathbb{I}, \varphi \vdash u :^p A \qquad \Gamma \vdash u_0 :^p A[\varphi \mapsto u[0/i]] \qquad \Gamma \vdash \varphi = 1 : \mathbb{F}}{\Gamma \vdash \mathsf{hcomp}^i_A\, [\varphi \mapsto u]\, u_0 \Rightarrow u[1/i] :^p A}$$

$$\frac{\begin{array}{c}\Gamma \vdash A \Rightarrow A' :^p \mathsf{U}_n \\ \Gamma \vdash \varphi : \mathbb{F} \qquad \Gamma, i : \mathbb{I}, \varphi \vdash u :^p A \qquad \Gamma \vdash u_0 :^p A[\varphi \mapsto u[0/i]] \qquad \Gamma \vdash \varphi \neq 1 : \mathbb{F}\end{array}}{\Gamma \vdash \mathsf{hcomp}^i_A\, [\varphi \mapsto u]\, u_0 \Rightarrow \mathsf{hcomp}^i_{A'}\, [\varphi \mapsto u]\, u_0 :^p A'}$$

$$\frac{\begin{array}{c}\Gamma, i : \mathbb{I} \vdash A :^p \mathsf{U}_n \\ \Gamma \vdash \varphi : \mathbb{F} \qquad \Gamma, i : \mathbb{I}, \varphi \vdash A[0/i] = A : \mathsf{U}_n \qquad \Gamma \vdash u :^p A[0/i] \qquad \Gamma \vdash \varphi = 1 : \mathbb{F}\end{array}}{\Gamma \vdash \mathsf{transp}^i\, A\, \varphi\, u \Rightarrow u :^p A[1/i]}$$

$$\frac{\begin{array}{c}\Gamma, i : \mathbb{I} \vdash A \Rightarrow A' :^p \mathsf{U}_n \\ \Gamma \vdash \varphi : \mathbb{F} \qquad \Gamma, i : \mathbb{I}, \varphi \vdash A[0/i] = A : \mathsf{U}_n \qquad \Gamma \vdash u :^p A[0/i] \qquad \Gamma \vdash \varphi \neq 1 : \mathbb{F}\end{array}}{\Gamma \vdash \mathsf{transp}^i\, A\, \varphi\, u \Rightarrow \mathsf{transp}^i\, A'\, \varphi\, u :^p A'[1/i]}$$

The congruence rule for $\mathsf{transp}^i\, A\, \varphi\, u$ above shows why, even for an initially empty Γ, we need to consider reduction in a context with interval variables. When φ is not $1_{\mathbb{F}}$ we want to reduce A to a canonical form because for each type former we have reduction rules for transp (and hcomp), obtained by orienting the corresponding judgmental equalities, and requiring the necessary premises for well-typedness. We present the reduction rules for transp and hcomp for homogeneous composition in the universe, as even their plain CTT version does not appear in the literature. We also provide the rule for transp for function types as it further motivates our choice of erasure annotations for this type former.

Function Types. Let $C := (x :^q A) \to B$.

$$\frac{\begin{array}{c}\Gamma, i : \mathbb{I} \vdash A :^{qp} \mathsf{U}_n \qquad \Gamma, i : \mathbb{I}, x :^q A \vdash B :^p \mathsf{U}_m \\ \Gamma \vdash \varphi : \mathbb{F} \qquad \Gamma, i : \mathbb{I}, \varphi \vdash C[0/i] = C : \mathsf{U}_{\max(n,m)} \qquad \Gamma \vdash u :^p C[0/i] \qquad \Gamma \vdash \varphi \neq 1 : \mathbb{F}\end{array}}{\Gamma \vdash \mathsf{transp}^i\, C\, \varphi\, u \Rightarrow \lambda^q y.\, \mathsf{transp}^i\, B[v[1-i/i]/x]\, \varphi\, (u \cdot^q v[1/i]) :^p C[1/i]}$$

where $v := \mathsf{transpFill}^i\, A[1-i/i]\, \varphi\, y$, being a path connecting y to $\mathsf{transp}^i\, A[1-i/i]\, \varphi\, y$, see Huber [2017] for the definition of transpFill, the typing of the arguments is the same as transp.

Note that, when $p = \omega$, the term $B[v[1-i/i]/x]$ is well-typed only if $y :^q A[1/i]$ is usable in an argument to B. So in particular when $q = 0$ we need $\Gamma, i : \mathbb{I}, x :^0 A \vdash B :^p \mathsf{U}_n$. When $q = \omega$ we have a choice, but the most permissive is to let $\Gamma, i : \mathbb{I}, x :^\omega A \vdash B :^p \mathsf{U}_n$. The domain type A is required to be available at ω only when both p and q are ω, as otherwise it is only used in an erased context. Overall this matches the premises for formation of function types.

Homogeneous Composition in the Universe. The following two are some of the most complex reduction rules, only matched by the corresponding ones for Glue which they are based on. We give them as a reference, and to substantiate our claim that such reductions preserve the modality p. For both of them all the premises are typed at p, and the right hand side uses term formers and operations available at any modality. In particular we make use of $\mathsf{hfill}^i\, A\, [\varphi \mapsto u]\, u_0$, which creates a path between u_0 and an hcomp with the same arguments, and heterogenous composition, comp, which is a combination of transp and hcomp. They are defined as in Huber [2017]. In the following, let $C := \mathsf{hcomp}^j_{\mathsf{U}_n}\, [\psi \mapsto B]\, A$.

Rule for hcomp.

$$\frac{\begin{array}{c}\Gamma \vdash \psi : \mathbb{F} \qquad \Gamma, \psi, j : \mathbb{I} \vdash B :^p \mathsf{U}_n \qquad \Gamma \vdash A :^p \mathsf{U}_n[\psi \mapsto B[1/i]] \\ \Gamma \vdash \varphi : \mathbb{F} \qquad \Gamma, i : \mathbb{I}, \varphi \vdash u :^p C \qquad \Gamma \vdash u_0 :^p C[\varphi \mapsto u[0/i]] \qquad \Gamma \vdash \varphi \neq 1 : \mathbb{F}\end{array}}{\Gamma \vdash \mathsf{hcomp}^i_C\,[\varphi \mapsto u]\,u_0 \Rightarrow \mathsf{hcomp\text{-}intro}_{j.B}\,[\psi \mapsto t[1/i]]\,a_1 :^p C}$$

where

$\Gamma, \psi, i : \mathbb{I} \vdash t := \mathsf{hfill}^i_{B[1/j]}\,\varphi\,u\,u_0 :^p B[1/j][\varphi \mapsto u, i = 0 \mapsto u_0]$

$a_1 := \mathsf{hcomp}^i_A\,[\varphi \mapsto \mathsf{hcomp\text{-}elim}\,[\psi \mapsto j.B]\,u, \psi \mapsto \mathsf{transp}^j\,B[1-j]\,0\,t](\mathsf{hcomp\text{-}elim}\,[\psi \mapsto j.B]\,u_0).$

Rule for transp.

$$\frac{\begin{array}{c}\Gamma, i : \mathbb{I} \vdash \psi : \mathbb{F} \qquad \Gamma, i : \mathbb{I}, \psi, j : \mathbb{I} \vdash B :^p \mathsf{U}_n \\ \Gamma, i : \mathbb{I} \vdash A :^p \mathsf{U}_n[\psi \mapsto B[1/i]] \qquad \Gamma \vdash \varphi : \mathbb{F} \qquad \Gamma \vdash u :^p C[0/i] \qquad \Gamma \vdash \varphi \neq 1 : \mathbb{F}\end{array}}{\Gamma \vdash \mathsf{transp}^i\,C\,\varphi\,u \Rightarrow \mathsf{hcomp\text{-}intro}_{j.B[1/i]}\,[\psi[1/i] \mapsto t'_1]\,a'_1 :^p C[1/i]}$$

where

$a_0 := \mathsf{hcomp\text{-}elim}\,[\psi[0/i] \mapsto j.B[0/i]]\,u$

$i \vdash t := \mathsf{transpFill}^i\,B[1/j]\,\varphi\,u$

$a_1 := \mathsf{comp}^i\,A\,[\varphi \mapsto a_0, \forall i.\psi \mapsto \mathsf{transp}^j\,B[1-j/j]\,t]\,a_0$

$\psi[1/i] \vdash (t'_1, \alpha) := \mathsf{prf}^i\,(B[1/i, 1-i, j])\,[\varphi \mapsto u_0, \forall i.\psi \mapsto t[1/i]]\,a_1$

$a'_1 := \mathsf{hcomp}^j_{A[1/i]}\,[\varphi \mapsto a_1, \psi[1/i] \mapsto \alpha\,j]\,a_1$

and $\mathsf{prf}^i\,E\,[\varphi \mapsto a]\,b$ can be derived to fit the following typing:

$$\frac{\Gamma, i : \mathbb{I} \vdash E :^p \mathsf{U}_n \qquad \Gamma \vdash \varphi : \mathbb{F} \qquad \Gamma \vdash a :^p E[0/i] \qquad \Gamma \vdash b :^p (E[1/i])[\varphi \mapsto \mathsf{transp}^i\,E\,0\,a]}{\Gamma \vdash \mathsf{prf}^i\,E\,[\varphi \mapsto a]\,b :^p \mathsf{fiber}\,(\mathsf{transp}^i\,E\,0)\,b}$$

4.3 Logical Relation

To prove our main result we will define a realizability relation $t \circledR w : A$ between closed terms and programs at a particular type A. As usual for dependent types, we cannot simply induct on the type expression A. Instead, we induct on a derivation D of $\Vdash A : \mathsf{U}_m$, witnessing that A is *forced*.

In this section, we need typed parallel substitutions $\boxed{\Gamma \vdash \sigma :^p \Delta}$ which are defined by axiom $\Gamma \vdash \varepsilon :^p \varepsilon$ and the following rules:

$$\frac{\Gamma \vdash \sigma :^p \Delta \qquad \Gamma \vdash t :^{pq} A\sigma}{\Gamma \vdash (\sigma, t/x) :^p (\Delta, x :^q A)} \qquad \frac{\Gamma \vdash \sigma :^p \Delta \qquad \Gamma \vdash r : \mathbb{I}}{\Gamma \vdash (\sigma, r/i) :^p (\Delta, i : \mathbb{I})} \qquad \frac{\Gamma \vdash \sigma :^p \Delta \qquad \Gamma \vdash \varphi\sigma = 1 : \mathbb{F}}{\Gamma \vdash \sigma :^p (\Delta, \varphi)}$$

4.3.1 Forcing. Let H, I, J, K range over pure interval contexts, i.e., contexts of the form $i_0 : \mathbb{I}, \ldots, i_n : \mathbb{I}$. We define predicates $I \Vdash A :^p m$ (for universe m) and $I \Vdash t :^p A/_D$ where D is a witness of $I \Vdash A :^p m$. We will often omit the subscript $/_D$ as the particular witness does not affect the relation. The forcing predicates are defined akin to Huber's computability predicates [2019], adapted to account for erasure annotations and homogeneous composition in the universe as a canonical form. Moreover our forcing relation is quite simplified compared to the one in *loc. cit.*, as we retain only the information necessary for the definition of realizability.

The predicates are defined by an outer well-founded induction on m. Figure 6 gives the inductive definition of forced types $\boxed{I \Vdash A :^p m}$. We then define term forcing $\boxed{I \Vdash t :^p A/_D}$ by cases on the derivation D of $I \Vdash A :^p m$.

Univ: $I \Vdash t :^p n$

Glue: $I, \varphi \Vdash t :^p T \qquad I \Vdash \mathsf{unglue}\,[\varphi \mapsto e]\,t :^p T_0$

HComp: $I, \varphi \Vdash t :^p B[1/i] \qquad I, i : \mathbb{I}, \varphi \vdash B :^p \mathsf{U}_n \qquad I \Vdash \mathsf{hcomp\text{-}elim}\,[\varphi \mapsto i.B]\,t :^p B_0$

Pi: $\forall \sigma\,s.\, J \vdash \sigma :^p I \implies J \vdash s :^{pq} S\sigma \implies J \Vdash t\sigma\,{}^q s :^p T[\sigma, s/x]$

$$\textsc{Univ}\ \frac{I \vdash A \Rightarrow^* \mathsf{U}_n :^p \mathsf{U}_m \qquad n < m}{I \Vdash A :^p m}$$

$$\textsc{Glue}\ \frac{\begin{array}{c} I \vdash A \Rightarrow^* \mathsf{Glue}\,[\varphi \mapsto (T, e)]\, T_0 :^p \mathsf{U}_m \qquad I \vdash \varphi \neq 1 : \mathbb{F} \\ I \Vdash T_0 :^p m \qquad I, \varphi \Vdash T :^p m \qquad I, \varphi \Vdash e :^p \mathsf{Equiv}\, T\, T_0 \end{array}}{I \Vdash A :^p m}$$

$$\textsc{HComp}\ \frac{\begin{array}{c} I \vdash A \Rightarrow^* \mathsf{hcomp}^i_{\mathsf{U}_n}\,[\varphi \mapsto B]\, B_0 :^p \mathsf{U}_m \qquad n < m \qquad I \vdash \varphi \neq 1 : \mathbb{F} \\ I, i : \mathbb{I}, \varphi \Vdash B :^p n \qquad I \Vdash B_0 :^p n \qquad I, \varphi \vdash B[0/i] = B_0 : \mathsf{U}_n \end{array}}{I \Vdash A :^p m}$$

$$\textsc{Pi}\ \frac{I \vdash A \Rightarrow^* (x :^q S) \to T :^p \mathsf{U}_m \qquad I \Vdash S :^p m \qquad \forall \sigma.\, J \vdash \sigma :^p (I, x :^q S) \implies J \Vdash T\sigma :^p m}{I \Vdash A :^p m}$$

$$\textsc{Sigma}\ \frac{I \vdash A \Rightarrow^* (x :^q S) \times T :^p \mathsf{U}_m \qquad I \Vdash S :^p m \qquad \forall \sigma.\, J \vdash \sigma :^p (I, x :^q S) \implies J \Vdash T\sigma :^p m}{I \Vdash A :^p m}$$

$$\textsc{Nat}\ \frac{I \vdash A \Rightarrow^* \mathbb{N} :^p \mathsf{U}_m}{I \Vdash A :^p m} \qquad \textsc{Trunc}\ \frac{I \vdash A \Rightarrow^* \|B\|^{\mathrm{E}} :^p \mathsf{U}_m \qquad I \Vdash B :^p m}{I \Vdash A :^p m}$$

$$\textsc{Path}\ \frac{I \vdash A \Rightarrow^* \mathsf{Path}\, B\, a_0\, a_1 :^p \mathsf{U}_m \qquad I \Vdash B :^p m \qquad I \vdash a_0 :^p B \qquad I \vdash a_1 :^p B}{I \Vdash A :^p m}$$

$$\textsc{Unit}\ \frac{I \vdash A \Rightarrow^* \top :^p \mathsf{U}_m}{I \Vdash A :^p m} \qquad \textsc{Empty}\ \frac{I \vdash A \Rightarrow^* \bot :^p \mathsf{U}_m}{I \Vdash A :^p m}$$

Herein, we write $\boxed{I, \varphi \Vdash A :^p m}$ to mean $\forall J \vdash \sigma :^p I.\ J \vdash \varphi\sigma = 1_{\mathbb{F}} : \mathbb{F} \implies J \Vdash A\sigma :^p m$.

Fig. 6. Forcing types.

Sigma: $I \Vdash \pi_1\, t :^{pq} S \qquad I \Vdash \pi_2\, t :^p T[\pi_1\, t/x]$
Nat: Inductively generated by

$$\textsc{Zero}\ \frac{\vdash t \Rightarrow^* 0 :^p \mathbb{N}}{I \Vdash t :^p \mathbb{N}} \qquad \textsc{Suc}\ \frac{\vdash t \Rightarrow^* \mathsf{suc}\, t' :^p \mathbb{N}}{I \Vdash t :^p \mathbb{N}}$$

TRUNC: Inductively generated by

$$\text{TR}\ \frac{I \vdash t \Rightarrow^* \mathsf{tr}\, t' :^p \|B\|^{\mathrm{E}} \qquad I \Vdash t' :^p B}{I \Vdash t :^p \|B\|^{\mathrm{E}}}$$

$$\text{TRUNC-HCOMP}\ \frac{\begin{array}{c} I \vdash t \Rightarrow^* \mathsf{hcomp}^i_{\|B'\|^{\mathrm{E}}}\, [\varphi \mapsto u]\, u_0 :^p \|B\|^{\mathrm{E}} \\ I \vdash \varphi \neq 0 : \mathbb{F} \qquad I, j : \mathbb{I}, \varphi \Vdash u :^p \|B\|^{\mathrm{E}} \qquad I \Vdash u_0 :^p \|B\|^{\mathrm{E}} \end{array}}{I \Vdash t :^p \|B\|^{\mathrm{E}}}$$

$$\text{PROP}\ \frac{\begin{array}{c} I \vdash t \Rightarrow^* \mathsf{trunc}\, u_0\, u_1\, r :^0 \|B\|^{\mathrm{E}} \\ I \vdash r \neq 0 : \mathbb{I} \qquad I \vdash r \neq 1 : \mathbb{I} \qquad I \Vdash u_0 :^0 \|B\|^{\mathrm{E}} \qquad I \Vdash u_1 :^0 \|B\|^{\mathrm{E}} \end{array}}{I \Vdash t :^0 \|B\|^{\mathrm{E}}}$$

PATH:
- $\forall \sigma\, s.\ J \vdash \sigma :^p I \implies J \vdash r : \mathbb{I} \implies J \Vdash t\, r :^p B$
- $I \vdash t\, 0 = a_0 : B$
- $I \vdash t\, 1 = a_1 : B$

UNIT: Unconditionally true.
EMPTY: False.

where we write $\boxed{I, \varphi \Vdash t :^p A}$ to mean $\forall J \vdash \sigma :^p I.\ J \vdash \varphi\sigma = 1_{\mathbb{F}} : \mathbb{F} \implies J \Vdash t\sigma :^p A\sigma$.

A type A is *valid* in context Γ, $\boxed{\Gamma \Vdash^{\mathrm{v}} A}$, iff there is a universe level m such that $I \Vdash A\sigma :^0 m$ for all $I \vdash \sigma :^0 \Gamma$. We extend the forcing relation to contexts $\boxed{\Vdash \Gamma}$ in the standard way.

$$\frac{}{\Vdash \varepsilon} \qquad \frac{\Vdash \Gamma \qquad \Gamma \Vdash^{\mathrm{v}} A}{\Vdash \Gamma, x :^p A} \qquad \frac{\Vdash \Gamma}{\Vdash \Gamma, i : \mathbb{I}} \qquad \frac{\Vdash \Gamma}{\Vdash \Gamma, \varphi}$$

We make some unproved assumptions that we conjecture can be proved by extending Huber's canonicity proof for CTT:

CONJECTURE 4.1. *We assume the following:*

(1) *Whenever* $I \vdash A :^0 \mathsf{U}_n$ *we have* $I \Vdash A :^0 n$.
(2) *Whenever* $I \vdash t :^p A$ *we have* $I \Vdash t :^p A$
(3) *In contexts* I, *canonical forms are injective and disjoint with regard to judgmental equality.*

4.3.2 Realizability. The forcing relation gives us an inductive structure on types that we exploit to define the logical relation we are interested in. In Figure 7, we define a "realizability" relation $\boxed{t \circledR v : A/_D}$ between closed non-erased typed terms $\vdash t :^\omega A$ of $\mathrm{CTT}^{0\omega}$ and closed terms v of the target language. Theorem 4.2 will show $t \circledR |t| : A$, i.e., that a term is related to is erasure. The definition of ® proceeds by recursion on $D : \ \Vdash A :^0 m$.

We also define $\boxed{r \circledR v : \mathbb{I}}$ and $\boxed{\varphi \circledR v : \mathbb{F}}$ as follows:

- $r \circledR v : \mathbb{I}$ iff $\vdash r = b : \mathbb{I}$ and $v \rightsquigarrow^* b$ for some $b \in \{0, 1\}$.
- $\varphi \circledR v : \mathbb{F}$ iff $\vdash \varphi = b : \mathbb{F}$ and $v \rightsquigarrow^* b'$ for some $(b, b') \in \{(0_{\mathbb{F}}, 0), (1_{\mathbb{F}}, 1)\}$.

Univ, Glue, Unit: Relation holds unconditionally.
Empty: Relation never holds.
Pi ($A \Rightarrow^* (x :^q S) \to T$): Holds when $t\ {}^q s \circledR v\ w : T[s/x]$ for all w and $\vdash s :^q S$ that, if $q = \omega$, satisfy $s \circledR w : S$.
Sigma ($A \Rightarrow^* (x :^q S) \times T$): Holds when $\pi_2\, t \circledR \pi_2\, v : T[\pi_1\, t/x]$, and $\pi_1\, t \circledR \pi_1\, v : S$ if $q = \omega$.
Nat: Inductively generated by

$$\textsc{Zero}\ \frac{\vdash t \Rightarrow^* 0 :^\omega \mathbb{N} \qquad v \rightsquigarrow^* 0}{t \circledR v : \mathbb{N}} \qquad \textsc{Suc}\ \frac{\vdash t \Rightarrow^* \mathsf{suc}\, t' :^\omega \mathbb{N} \qquad v \rightsquigarrow^* \mathsf{suc}\, v' \qquad t' \circledR v' : \mathbb{N}}{t \circledR v : \mathbb{N}}$$

Trunc ($A \Rightarrow^* \|B\|^{\mathrm{E}}$): Inductively generated by

$$\textsc{tr}\ \frac{\vdash t \Rightarrow^* \mathsf{tr}\, t' :^\omega \|B\|^{\mathrm{E}} \qquad v \rightsquigarrow^* \mathsf{tr}\, v' \qquad t' \circledR v' : B}{t \circledR v : \|B\|^{\mathrm{E}}}$$

$$\textsc{Trunc-hcomp}\ \frac{\vdash t \Rightarrow^* \mathsf{hcomp}^i_{\|B'\|^{\mathrm{E}}}\ [\varphi \mapsto u]\ u_0 :^\omega \|B\|^{\mathrm{E}} \qquad \vdash \varphi = 0 : \mathbb{F} \qquad u_0 \circledR v : \|B\|^{\mathrm{E}}}{t \circledR v : \|B\|^{\mathrm{E}}}$$

HComp ($A \Rightarrow^* \mathsf{hcomp}^i_{\mathsf{U}_n}\ [\varphi \mapsto B]\ B_0 \wedge \varphi = 0_{\mathbb{F}}$): Holds when $\mathsf{hcomp\text{-}elim}\ [0 \mapsto i.[]]\ t \circledR v : B_0$.
Path ($A \Rightarrow^* \mathsf{Path}\, B\, a_0\, a_1$): Holds when $t\, r \circledR v\, w : B$ for all w and $\vdash r : \mathbb{I}$ with $r \circledR w : \mathbb{I}$.

Fig. 7. Realizability $t \circledR v : A$.

We extend realizability to substitutions of terms σ and programs ρ: Given $D : \Vdash \Gamma$, relation $\boxed{\sigma \circledR \rho : \Gamma/_D}$ entails $\vdash \sigma :^\omega \Gamma$ and essentially holds if it holds pointwise.

$$\frac{}{\varepsilon \circledR \varepsilon : \varepsilon} \qquad \frac{\sigma \circledR \rho : \Gamma \qquad \vdash t :^q A\sigma \qquad t \circledR v : A\sigma \text{ when } q = \omega}{(\sigma, t/x) \circledR (\rho, v/x) : (\Gamma, x :^q A)}$$

$$\frac{\sigma \circledR \rho : \Gamma \qquad \vdash r : \mathbb{I} \qquad r \circledR b : \mathbb{I}}{(\sigma, r/i) \circledR (\rho, b/x) : (\Gamma, i : \mathbb{I})} \qquad \frac{\sigma \circledR \rho : \Gamma \qquad \vdash \varphi\sigma = 1 : \mathbb{F} \qquad |\varphi|\rho \rightsquigarrow^* 1}{\sigma \circledR \rho : (\Gamma, \varphi)}$$

(Here, we omitted D but it is easy to fill in correctly.)

Given $D_\Gamma : \Vdash \Gamma$ and $D_A : \Gamma \Vdash^{\mathsf{V}} A$ we define

$$\boxed{\Gamma \models t : A} \text{ iff } \Gamma \vdash t :^\omega A \text{ and } \forall \sigma\, \rho.\ \sigma \circledR \rho : \Gamma \implies t\sigma \circledR |t|\rho : A\sigma.$$

We also extend $\Gamma \models t : A$ to $\mathbb{I}$ or $\mathbb{F}$ in place of A in the obvious way.

Theorem 4.2 (Fundamental Theorem). *$\Gamma \vdash t :^\omega A$ implies $\Gamma \models t : A$*

Proof. By induction on the typing derivation of $\Gamma \vdash t :^\omega A$, using the lemmas from Section 4.3.4. The proof makes essential use of path closure (see Section 4.3.3). □

Corollary 4.3 (Soundness of compilation). *$\vdash t :^\omega \mathbb{N}$ implies that t and $|t|$ reduce to the same numeral.*

Proof. By induction on the proof of $t \circledR |t| : \mathbb{N}$ obtained by Theorem 4.2. □

Lemma 4.4 (Expansion).

$$\frac{\vdash A \Rightarrow^* B :^0 \mathsf{U}_m \qquad t \circledR v : B}{t \circledR v : A} \qquad \frac{\vdash t \Rightarrow^* u :^\omega A \qquad v \rightsquigarrow^* w \qquad u \circledR w : A}{t \circledR v : A}$$

Proof. For the first implication we proceed by case splitting on $\Vdash B :^0 m$. All of the possible cases mention B only in a premise of the form $\vdash B \Rightarrow^* T :^0 \mathsf{U}_m$, which implies $\vdash A \Rightarrow^* T :^0 \mathsf{U}_m$ by transitivity, so we can derive $\Vdash A :^0 m$ with the same rule, which means $t \circledR v : A$ is equivalent to $t \circledR v : B$.

For the second implication we proceed by induction on $\Vdash A :^0 m$. In the cases for $\mathbb{N}$ and $\|B\|^{\mathrm{E}}$, we conclude directly, as they are clearly closed under expansion. In the cases for $\mathsf{Path}\, B\, a_0\, a_1$, $(x :^q A) \to B$, $(x :^q A) \times B$, and $\mathsf{hcomp}^i\, [\varphi \mapsto B]\, B_0$ with $\varphi = 0_{\mathbb{F}}$, we proceed by using the congruence rules for reduction of the respective elimination forms, and conclude by I.H. The other cases are trivially true. □

4.3.3 Path Closure. In the proof of Theorem 4.2 we have to handle the fact that both $\mathsf{transp}^i\, A\, \varphi\, u_0$ and $\mathsf{hcomp}^i_A\, [\varphi \mapsto u]\, u_0$ are erased to $|u_0|$, while the terms themselves might not be judgmentally equal to u_0. For example when φ is equal $1_{\mathbb{F}}$ we have that the homogeneous composition reduces to $u[1/i]$, which is only equal to u_0 up to a path. Fortunately paths typed at ω are relatively simple, because their use of Glue or path constructors is limited to subterms at 0, so we are able to prove that realizability is closed under such paths in Lemma 4.8. In the following we say that a term $i : \mathbb{I} \vdash t :^p A$ connects t_0 to t_1 if $i : \mathbb{I} \vdash t[b/i] = t_b : A[b/i]$ for $b \in \{0, 1\}$.

We write $\mathsf{hcomp}^{(n)}_A$ for n iterations of $\mathsf{hcomp}^i_A\, [\varphi \mapsto u]$ where $\varphi = 0_{\mathbb{F}}$. In particular hcomp^0_A is the identity. The notation overlooks the different possible u terms but this is benign as they are all judgmentally equal to the partial element [] and do not contribute to reduction.

Closed elements of U_n that are path equal at ω can still differ in the amount of $\mathsf{hcomp}^{(n)}$ present. Indeed paths in U_n, applied to some interval variable i, can have $\mathsf{hcomp}^j_{\mathsf{U}_n}\, [\varphi \mapsto B]\, B_0$ as WHNF, as long as φ is not 1. These represent the composition of at most 3 paths, B_0 and $B[1/i]$ and $B[0/i]$. To ease our proof of the Path Closure lemma we want to simplify out such compositions, and be left with only paths of the form $\mathsf{hcomp}^j\, [\varphi \mapsto B_0]\, B_0$ where φ is either 0 or $i = b$ for some $b \in \{0, 1\}$.

Definition 4.5 (Simple Path). We say a term $i : \mathbb{I} \vdash P :^\omega X$ is a *simple path* if it is of the form

$$
\begin{array}{lcl}
P, Q & ::= & C \mid \mathsf{hcomp}^j_X\, [\tilde{\varphi} \mapsto P]\, P \quad \text{(identical } P\text{s!)} \\
C & ::= & \mathsf{tr}\, t \mid (x :^p A) \to B \mid (x :^p A) \times B \mid \mathsf{U}_n \mid \mathbb{N} \mid \|A\|^{\mathrm{E}} \mid \bot \mid \top \\
X, Y & ::= & \mathsf{U}_n \mid \|A\|^{\mathrm{E}} \\
\tilde{\varphi} & ::= & 0_{\mathbb{F}} \mid (i = 0) \mid (i = 1)
\end{array}
$$

We call paths of the form $i :^\omega I \vdash X :^\omega \mathsf{U}_m$ *very simple.* We observe that for any two very simple paths X, Y such that $\vdash X[1/i] = Y[0/i] : \mathsf{U}_m$ there is a very simple path XY which is path-equal to the homotopy composition of X and Y. Moreover whenever one of X or Y is constant in i we will choose XY to be equal to the other.

Lemma 4.6. *Given two simple paths $i : \mathbb{I} \vdash P :^\omega X$ and $i : \mathbb{I} \vdash Q :^\omega Y$ such that $i : \mathbb{I} \vdash P[1/i] = Q[0/i] : X[1/i]$ there is a simple path $i : \mathbb{I} \vdash PQ :^\omega XY$ which is path-equal to the homotopy composition of P and Q.*

Proof. We proceed by induction on P and Q. If P is of the form C_P and Q is of the form C_Q then they must be the same term former and we can compose their arguments with hcomp to obtain the desired PQ, as we would do for composition in an inductive type.

If $P = \mathsf{hcomp}^j_X\, [(i = 1) \mapsto P_0]\, P_0$, we recurse on P_0 and Q to obtain P_0Q and define PQ to be $\mathsf{hcomp}^j_{XY}\, [(i = 1) \mapsto P_0Q]\, P_0Q$. The symmetric case where $Q = \mathsf{hcomp}^j_Y\, [(i = 0) \mapsto Q_0]\, Q_0$ is analogous.

If $P = \mathsf{hcomp}^j_X\, [0_{\mathbb{F}} \mapsto P_0]\, P_0$ and $Q = \mathsf{hcomp}^j_Y\, [0_{\mathbb{F}} \mapsto Q_0]\, Q_0$ then we recurse on P_0 and Q_0 to obtain PQ_0 and define PQ as $\mathsf{hcomp}^j_{XY}\, [0_{\mathbb{F}} \mapsto PQ_0]\, PQ_0$.

If $P = \mathsf{hcomp}^j_X\ [0_{\mathbb{F}} \mapsto P_0]\ P_0$ and $Q = \mathsf{hcomp}^j_Y\ [(i = 1) \mapsto Q_0]\ Q_0$ then we recurse on P_0 and Q_0 to obtain PQ_0 and define PQ as $\mathsf{hcomp}^j_{XY}\ [(i = 1) \mapsto PQ_0]\ PQ_0$.

If $P = \mathsf{hcomp}^j_X\ [(i = 0) \mapsto P_0]\ P_0$ and $Q = \mathsf{hcomp}^j_Y\ [0_{\mathbb{F}} \mapsto Q_0]\ Q_0$ then we recurse on P_0 and Q_0 to obtain PQ_0 and define PQ as $\mathsf{hcomp}^j_{XY}\ [(i = 0) \mapsto PQ_0]\ PQ_0$.

If $P = \mathsf{hcomp}^j_X\ [(i = 0) \mapsto P_0]\ P_0$ and $Q = \mathsf{hcomp}^j_{\mathsf{U}_n}\ [(i = 1) \mapsto Q_0]\ Q_0$ then we recurse on P_0 and Q_0 to obtain PQ_0 which we use as the definition of PQ.

The other cases are ruled out by $i : \mathbb{I} \vdash P[1/i] = Q[0/i]$.

In each of the above cases it is easily checked that the endpoints are preserved and the resulting path PQ is path equal to the composition. □

Lemma 4.7 (Path Simplification). *Given $i : \mathbb{I} \vdash A :^{\omega} X$ there is a simple path B which is path equal to A and has the same endpoints.*

Proof. We proceed by induction on $i : \mathbb{I} \Vdash A :^{\omega} X$. If A reduces to a term of the form C we are done. If A reduces to $\mathsf{hcomp}^j_X\ [\varphi \mapsto A_\varphi]\ A_0$ then we have the following cases

(a) $0_{\mathbb{F}}$
(b) $(i = 0) \vee (i = 1)$
(c) $i = 0$
(d) $i = 1$

In case (a) we recurse on A_0 to obtain B_0 then define B as $\mathsf{hcomp}^j_X\ [0_{\mathbb{F}} \mapsto B_0]\ B_0$ In the other cases we will also recurse on the relevant $A_\varphi[b/i]$ to obtain $B_{b/i}$ with $\vdash B_{b/i}[0/j] = B_0[b/i] : X[b/i]$, and use lemma 4.6 as explained below.

In case (c) we have $j : \mathbb{I} \vdash B_{0/i} :^{\omega} X[0/i]$ and $\vdash B_{0/i}[0/j] = B_0[0/i] : X[0/i]$ so we can compose $B_{0/i}[1 - j/j]$ with B_0 to obtain a simple path Q and then define B as $\mathsf{hcomp}^j_X\ [i = 0 \mapsto Q]\ Q$. Case (d) is symmetric.

In case (b) we have both $B_{0/i}$ and $B_{1/i}$. We compose $B_{0/i}[1 - j/j]$ with B_0 to obtain a simple path Q as before. We have $Q[1/i]$ judgmentally equal to $B_{1/i}[0/j]$ so we can compose the two paths to obtain the desired simple path B. □

Lemma 4.8 (Path Closure). *Given $i : \mathbb{I} \vdash A :^{\omega} \mathsf{U}_n$ connecting A_0 to A_1 and $i : \mathbb{I} \vdash t :^{\omega} A$, connecting t_0 to t_1 we have that $t_0 \circledR v : A_0$ implies $t_1 \circledR v : A_1$.*

Proof. We proceed by well-founded induction on the maximum of the height of $\Vdash A_0 :^{\omega} \mathsf{U}_n$ and $\Vdash A_1 :^{\omega} \mathsf{U}_n$. We start by applying lemma 4.7 to A obtaining a simple path B connecting A_0 to A_1 and path equal to A. By comp we also obtain $i : \mathbb{I} \vdash u :^{\omega} B$ connecting t_0 to t_1. We proceed by cases on B.

If B is of the form $\mathsf{hcomp}^j\ [\tilde{\varphi} \mapsto B_0]\ B_0$ we proceed by cases on $\tilde{\varphi}$. In case $0_{\mathbb{F}}$ we have that both A_b must reduce to types of the form $\mathsf{hcomp}^(1)\ A'_b$ and so $t_b \circledR v : A_b$ is equivalent to $(\mathsf{hcomp\text{-}elim}\ [0 \mapsto []])\ t_b \circledR v : A'_b$. By applying $(\mathsf{hcomp\text{-}elim}\ [0 \mapsto []])$ to u we obtain a path between the new endpoints and we conclude by I.H. on A'_b. In case $(i = 0)$ we have that A_1 must reduce to a type of the form $\mathsf{hcomp}^(1)\ A'_1$, and so $t_1 \circledR v : A_1$ is equivalent to $(\mathsf{hcomp\text{-}elim}\ [0 \mapsto []])\ t_1 \circledR v : A'_1$. We have that B_0 connects A_0 and A'_1. Applying $(\mathsf{hcomp\text{-}elim}\ [(i = 0) \mapsto_{.} B_0])$ to u we obtain a term in B_0 that connects $\mathsf{transp}^j\ A_0\ t_0$ to $(\mathsf{hcomp\text{-}elim}\ [0 \mapsto []])\ t_1$. By transFill we obtain a path with t_0 as left endpoint instead. With this last path and B_0 we can conclude by I.H. on A_0 and A'_1. The case $(i = 1)$ is symmetric.

In the other cases we have that B is built with a canonical type former other than hcomp, so that both A_b also reduce to this same type former.

If $B = (x :^q S) \to T$ then we have A_b reducing to $(x :^q S_b) \to T_b$ with S connecting S_0 to S_1 and T connecting T_0 to T_1. We are given w and $\vdash s_1 :^q S_1$ and have to show $t_0\, {}^q s_1 \circledR v\, w : T_1[s_1/x]$. By transpFill we have $i : \mathbb{I} \vdash s :^q S$ connecting some s_0 to s_1, so we can derive $t_0\, {}^q s_0 \circledR v\, w : T_0[s_0/x]$ from the assumption about t_0. If $q = \omega$ this last step additionally requires us to show $s_0 \circledR w : S_0$ which we do by I.H. on S_b, which is possible because in this case S is available at ω. Using u and s we build a path between the two $t_b\, {}^q s_b$ over $T[s/x]$, and conclude by I.H. on the two $T_b[s_b/x]$.

If $B = (x :^q S) \times T$ then we have A_b reducing to $(x :^q S_b) \times T_b$ with S connecting S_0 to S_1 and T connecting T_0 to T_1. If $q = \omega$ then we can use $\pi_1\, u$ and $\pi_2\, u$ to create paths at ω connecting the respective projections of t_0 and t_1, then we conclude by I.H. on S_b and $T_b[\pi_1\, t_b/x]$ respectively. If $q = 0$ then we still have the path $\pi_2\, u$ at ω, so again we can conclude by I.H. on $T_b[\pi_1\, t_b/x]$.

If $B = \mathsf{Path}\, S\, s_0\, s_1$ then we have A_b reducing to $\mathsf{Path}\, S_b\, s_{0b}\, s_{1b}$ with the subterms connected by the subterms of B. We are then given $r \circledR w : \mathbb{I}$ and have to show $t_1\, r \circledR v\, w : S_1$. By the assumption about t_0 we have $t_0\, r \circledR v\, w : S_0$. We have that $i : \mathbb{I} \vdash u\, r :^\omega S$ connects the two $t_b\, r$, so we can conclude by I.H. on S_b.

If $B = \mathbb{N}$ then t_0 and t_1 reduce to the same numeral, hence they have the same realizers.

If $B = \mathsf{U}_m$ or $B = \top$ or $B = \bot$ then $t_0 \circledR v : A_0$ is equivalent to $t_1 \circledR v : A_1$.

If $B = \|S\|^{\mathrm{E}}$ then we have A_b reducing to $\|S_b\|^{\mathrm{E}}$ with S connecting S_0 to S_1. By the Path Simplification lemma 4.7 applied to u we obtain a simple path P between t_0 and t_1. We then proceed by a local recursion on the simple path structure of P. If $P = \mathsf{tr}\, s$ then we have t_b reducing to tr, s_b with s connecting s_0 to s_1, from $t_0 \circledR v : A_0$ we then have $v \leadsto^* \mathsf{tr}\, v'$ and $s_0 \circledR v' : S_0$ and we conclude by I.H on S_b using the paths s and S. If $P = \mathsf{hcomp}^j_{\|S\|^{\mathrm{E}}}\, [\tilde{\varphi} \mapsto P']\, P'$ then we proceed by cases on $\tilde{phi}$. If $\tilde{phi} = 0_{\mathbb{F}}$ then we have t_b reducing to $\mathsf{hcomp}^{(1)}\, t'_b$ with P' connecting the two t'_b. We then have $t_0 \circledR v : \|S_0\|^{\mathrm{E}}$ equivalent to $t'_0 \circledR v : \|S_0\|^{\mathrm{E}}$ by rule HCOMP. By the same rule, to show $t_1 \circledR v : \|S_1\|^{\mathrm{E}}$ then it is sufficient to show $t'_1 \circledR b : \|S_1\|^{\mathrm{E}}$, which we do by our local I.H. on P'. The other cases for $\tilde{\varphi}$ are analogous, except only one of the two t_b will reduce to an hcomp whose base is connected to the other by P'. □

4.3.4 *Semantic Typing Lemmas.*

To prove the fudamental theorem we give lemmas corresponding to each typing rule with an ω conclusion.

transp.

LEMMA 4.9 (SEMANTIC TYPING OF transp).

$$\frac{\Gamma, i : \mathbb{I} \models A : \mathsf{U}_n \qquad \Gamma \models \varphi : \mathbb{F} \qquad \Gamma, i : \mathbb{I}, \varphi \vdash A[0/i] = A : \mathsf{U}_n \qquad \Gamma \models u : A[0/i]}{\Gamma \models \mathsf{transp}^i_A\, \varphi\, u : A}$$

PROOF. Given $\sigma \circledR \rho : \Gamma$ we have to show $(\mathsf{transp}^i_A\, \varphi\, u)\sigma \circledR |u|\rho : A\sigma$. By assumption we have $u\sigma \circledR |u|\rho : A\sigma$, so we can conclude by the Path Closure Lemma 4.8 and transpFill. □

hcomp.

LEMMA 4.10 (SEMANTIC TYPING OF hcomp).

$$\frac{\Gamma \models A : \mathsf{U}_n \qquad \Gamma \models \varphi : \mathbb{F} \qquad \Gamma, i : \mathbb{I}, \varphi \models u : A \qquad \Gamma \models u_0 : A \qquad \Gamma \vdash u_0 = u[0/i] : A}{\Gamma \models \mathsf{hcomp}^i_A\, [\varphi \mapsto u]\, u_0 : A}$$

PROOF. Given $\sigma \circledR \rho : \Gamma$ we have to show $(\mathsf{hcomp}^i_A\, [\varphi \mapsto u]\, u_0)\sigma \circledR |u_0|\rho : A\sigma$. By assumption we have $u_0\sigma \circledR |u_0|\rho : A\sigma$, so we can conclude by the Path Closure Lemma 4.8 and hfill. □

Lemma 4.11 (Semantic typing of hcomp-intro).

$$\frac{\Gamma,\varphi \models t : B[1/i] \qquad \Gamma \models a : A \qquad \Gamma,\varphi \vdash \mathsf{transp}^j\, B[1-j/i]\, 0\, t = a : A \qquad \Gamma,\varphi, i :^{\omega} \models B : \mathsf{U}_n}{\Gamma \models \mathsf{hcomp\text{-}intro}_{n,i.B}\, [\varphi \mapsto t]\, a : \mathsf{hcomp}^i\, [\varphi \mapsto B]\, A}$$

Proof. Given $\sigma \circledR \rho : \Gamma$ we have to show $(\mathsf{hcomp\text{-}intro}_{n,i.B}\, [\varphi \mapsto t]\, a)\sigma \circledR |a|\rho : (\mathsf{hcomp}^i\, [\varphi \mapsto B]\, A)\sigma$. We proceed by cases on $\vdash \varphi\sigma :^{\omega} \mathbb{F}$.

Case $\varphi\sigma = 0$: then $\mathsf{hcomp}^i\, [\varphi\sigma \mapsto B[\sigma,i]]\, A\sigma$ is canonical and we have to show

$$\mathsf{hcomp\text{-}elim}\, [\varphi\sigma \mapsto i.B[\sigma,i]]\, ((\mathsf{hcomp\text{-}intro}_{n,i.B}\, [\varphi \mapsto t]\, a)\sigma) \circledR |a|\rho : \mathsf{hcomp}^i\, [\varphi\sigma \mapsto B[\sigma,i]]\, A\sigma$$

by the Expansion Lemma 4.4 it is suffienct to show $a\sigma$ is related to $|a|\rho$, which we have by assumption.

Case $\varphi\sigma = 1$: then $\mathsf{hcomp}^i\, [\varphi\sigma \mapsto B[\sigma,i]]\, A\sigma$ reduces to $B[\sigma,1]$, and $\mathsf{hcomp\text{-}intro}_{n,i.B}\, [\varphi \mapsto t]\, a)\sigma$ reduces to $t\sigma$, so by the Expansion Lemma 4.4 it is sufficient to show $t\sigma \circledR |a|\rho : B[\sigma,1]$. What we have is $a\sigma \circledR |a|\rho : A$ and $\Gamma \vdash \mathsf{transp}^j\, B[1-j/i]\, 0\, t = a : A$, so by transpFill we also have a term $i : \mathbb{I} \vdash p :^{\omega} B[\sigma,i]$ such that $\vdash p[0/i] = a : A\sigma$ and $\vdash p[1/i] = t\sigma : B[\sigma,1]$ and we conclude with the Path Closure Lemma 4.8. □

Lemma 4.12 (Semantic typing of hcomp-elim).

$$\frac{\Gamma, i:\mathbb{I},\varphi \models B : \mathsf{U}_n \qquad \Gamma \models A : \mathsf{U}_n \qquad \Gamma,\varphi \vdash B[0/i] = A \qquad \Gamma \models u : \mathsf{hcomp}^i_{\mathsf{U}_n}\, [\varphi \mapsto B]\, A}{\Gamma \models \mathsf{hcomp\text{-}elim}\, [\varphi \mapsto i.B]\, u : A}$$

Proof. Given $\sigma \circledR \rho : \Gamma$ we have to show $(\mathsf{hcomp\text{-}elim}\, [\varphi \mapsto i.B]\, u)\sigma \circledR |u|\rho : A$. We proceed by cases on $\varphi\sigma$.

Case $\varphi\sigma = 0$, then $(\mathsf{hcomp}^i_{\mathsf{U}_n}\, [\varphi \mapsto B]\, A)\sigma$ is canonical and the lemma follows by the assumption about u and the definition of realizability for this type.

Case $\varphi\sigma = 1$, then $(\mathsf{hcomp}^i_{\mathsf{U}_n}\, [\varphi \mapsto B]\, A)\sigma$ reduces to $B[\sigma,1]$ and $(\mathsf{hcomp\text{-}elim}\, [\varphi \mapsto i.B]\, u)\sigma$ reduces to $\mathsf{transp}^j\, B[\sigma, 1-j/i]\, 0\, u\sigma$, so by the Expansion Lemma 4.4 it is sufficient to show $\mathsf{transp}^j\, B[\sigma,1-j/i]\, 0\, u\sigma \circledR |u|\rho : B[\sigma,1]$. We have $u\sigma \circledR |u|\rho : B[\sigma,0]$ so we conclude by transpFill and the Path Closure Lemma 4.8. □

Propositional Truncation.

Lemma 4.13. $\Gamma \models A : \mathsf{U}_n$ *implies* $\Gamma \models \|A\|^E : \mathsf{U}_n$

Proof. We have to show $\|A\|^{\mathrm{E}}\sigma \circledR |\|A\|^{\mathrm{E}}|\rho : \mathsf{U}_n$, i.e., $\|A\|^{\mathrm{E}}\sigma \circledR \lightning : \mathsf{U}_n$ which holds by definition. □

Lemma 4.14. $\Gamma \models t : A$ *implies* $\Gamma \models \mathsf{tr}\, t : \|A\|^E$

Proof. Given $\sigma \circledR \rho : \Gamma$ we have to show $\mathsf{tr}\, t\sigma \circledR \mathsf{tr}\, |t|\rho : \|A\|^{\mathrm{E}}$. This follows by rule TR. □

Lemma 4.15.

$$\frac{\Gamma \vdash A \qquad \Gamma, x :^{\omega} \|A\|^E \models C : \mathsf{U}_n \qquad \Gamma, x :^{\omega} A \models t : C[\mathsf{tr}\, x/x] \quad \Gamma, x\, y :^0 \|A\|^E, i:\mathbb{I} \vdash u :^0 C[\mathsf{trunc}\, x\, y\, i/x][i=0 \mapsto x, i=1 \mapsto y] \qquad \Gamma \models w : \|A\|^E}{\Gamma \models \|\|^E\text{-}\mathsf{elim}\, (x.C)\, (x.t)\, (x.y.i.u)\, w : C[w/x]}$$

Proof. Let us write f for $\|\|^{\mathrm{E}}\text{-}\mathsf{elim}\, (x.C[\sigma,x])\, (x.t[\sigma,x])\, (x.y.i.u[\sigma,x,y,i])$. Given $\sigma \circledR \rho : \Gamma$ we have to show $f\, w\sigma \circledR \|\|^{\mathrm{E}}\text{-}\mathsf{elim}\, (x.|t|[\rho,x])\, |w|\rho : C[\sigma, w\sigma]$, let us call the two terms c and v. We have $w\sigma \circledR |w|\rho : \|A\|^{\mathrm{E}}$, we proceed by induction on its proof.

Case TR: we derive $\vdash c \Rightarrow^* t[\sigma,t'] :^{\omega} C[\sigma, \mathsf{tr}\, t']$ and $v \rightsquigarrow^* |t|[\rho, |v'|]$. By the Expansion Lemma 4.4, the assumption for $x.t$ and $t' \circledR v' : \|A\|^{\mathrm{E}}$ we are done.

Case TRUNC-HCOMP**:** we derive $\vdash c \Rightarrow^* c' :^\omega C[\sigma, \mathrm{tr}\, t']$ where c' is a composition with base $f\, u_0$. From $u_0 \circledR v : \|B\|^{\mathrm{E}}$ we obtain $f\, u_0 \circledR v : C[\sigma, u_0]$ by IH. Since c' is path equal to $f\, u_0$ we conclude by applying the Path Closure Lemma 4.8 and then the Expansion Lemma 4.4.

□

Path.

LEMMA 4.16 (SEMANTIC TYPING OF Path). $\Gamma \models \mathsf{Path}\, A\, a_0\, a_1 : \mathsf{U}_n$

PROOF. trivial □

LEMMA 4.17 (SEMANTIC TYPING OF PATH ABSTRACTION).

$$\frac{\Gamma \vdash A \qquad \Gamma \vdash a_0, a_1 :^0 A \qquad \Gamma, i : \mathbb{I} \models t : A \qquad \Gamma \vdash t[b/i] = a_b : A}{\Gamma \models \lambda i.t : \mathsf{Path}\, A\, a_0\, a_1}$$

PROOF. Given $\sigma \circledR \rho : \Gamma$ and $r \circledR w : \mathbb{I}$ we have to show $(\lambda i.\, t)\sigma\, r \circledR |\lambda i.\, t|\rho\, w : A$. By definition of $|\lambda i.\, t|$ and the Expansion Lemma 4.4 it is sufficient to show $t(\sigma, r) \circledR |t|(\rho, w) : A$, which follows from the premise about t. □

LEMMA 4.18 (SEMANTIC TYPING OF PATH APPLICATION).

$$\frac{\Gamma \vdash A \qquad \Gamma \vdash a_0, a_1 :^0 A \qquad \Gamma \models t : \mathsf{Path}\, A\, a_0\, a_1 \qquad \Gamma \models r : \mathbb{I}}{\Gamma \models t\, r : A}$$

PROOF. Given $\sigma \circledR \rho : \Gamma$ we have to show $t\sigma\, r\sigma \circledR |t|\rho\, |r|\rho : A$, which follows directly by the assumptions about t and r. □

Empty and Unit Types.

LEMMA 4.19 (SEMANTIC TYPING OF $\bot$ AND $\top$). $\Gamma \models \bot : \mathsf{U}_n$ *and* $\Gamma \models \top : \mathsf{U}_n$

PROOF. trivial. □

LEMMA 4.20 (SEMANTIC TYPING OF $\langle\rangle$). $\Gamma \models \langle\rangle : \top$

PROOF. trivial. □

LEMMA 4.21 (SEMANTIC TYPING OF $\bot\text{-elim}_A\, t$).

$$\frac{\Gamma \models t : \bot}{\Gamma \models \bot\text{-elim}_A\, t : A}$$

PROOF. Given $\sigma \circledR \rho : \Gamma$ we have to show $(\bot\text{-elim}_A\, t)\sigma \circledR |\bot\text{-elim}_A\, t|\rho : A\sigma$. From $\Gamma \models t : \bot$ we obtain $t\sigma \circledR |t|\rho : \bot$ which is a contradiction. □

Function Types.

LEMMA 4.22.

$$\frac{\Gamma \vdash A :^q \mathsf{U}_m \qquad \Gamma, x :^q A \models B : \mathsf{U}_n}{\Gamma \models (x :^q A) \to B : \mathsf{U}_{\max(m,n)}}$$

PROOF. trivial □

LEMMA 4.23.

$$\frac{\Gamma, x :^q A \models t : B}{\Gamma \models \lambda x^q.\, t : (x :^q A) \to B}$$

Proof. Given $\sigma \circledR \rho : \Gamma$ we have to show $(\lambda x^q.\, t)\sigma \circledR |\lambda x^q.\, t|\rho : ((x :^q A) \to B)\sigma$. We proceed by cases on q.

$q = \omega$**:** then given $a \circledR w : A\sigma$, by Expansion (Lemma 4.4) we have to show $t(\sigma, a) \circledR |t|(\rho, w) : B[\sigma, a/x]$ which follows from the assumption about t.

$q = 0$**:** then given $\vdash a :^0 A\sigma$, by Expansion (Lemma 4.4) we have to show $t(\sigma, a) \circledR |t|(\rho, \lightning) : B[\sigma, a/x]$ which follows from the assumption about t.

□

Lemma 4.24.

$$\frac{\Gamma \models t : (x :^\omega A) \to B \qquad \Gamma \models u : A}{\Gamma \models t\, {}^\omega u : B[u/x]}$$

Proof. Given $\sigma \circledR \rho : \Gamma$ we have to show $t\sigma\, {}^\omega u\sigma \circledR |t|\rho\, |u|\rho : B[u/x]\sigma$. This follows directly by the definition of $\Gamma \models t : (x :^\omega A) \to B$. □

Lemma 4.25.

$$\frac{\Gamma \models t : (x :^0 A) \to B \qquad \Gamma \vdash u :^0 A}{\Gamma \models t\, {}^0 u : B[u/x]}$$

Proof. Given $\sigma \circledR \rho : \Gamma$ we have to show $t\sigma\, {}^0 u\sigma \circledR |t|\rho\, \lightning : B[u/x]\sigma$. This follows directly by the definition of $\Gamma \models t : (x :^0 A) \to B$. □

Product Types.

Lemma 4.26.

$$\frac{\Gamma \vdash A :^p \mathsf{U}_m \qquad \Gamma, x :^q A \vdash B :^p \mathsf{U}_n}{\Gamma \models (x :^q A) \times B : \mathsf{U}_{\max(m,n)}}$$

Proof. trivial. □

Lemma 4.27.

$$\frac{\Gamma \models t_1 : A \qquad \Gamma \models t_2 : B[t_1/x]}{\Gamma \models \langle^\omega t_1,\, t_2\rangle : (x :^\omega A) \times B} \qquad \frac{\Gamma \vdash t_1 :^0 A \qquad \Gamma \models t_2 : B[t_1/x]}{\Gamma \models \langle^0 t_1,\, t_2\rangle : (x :^0 A) \times B}$$

Proof. In either case we are given $\sigma \circledR \rho : \Gamma$ and have to show $\langle^q t_1,\, t_2\rangle\sigma \circledR |\langle^q t_1,\, t_2\rangle|\rho : ((x :^q A) \times B)\sigma$. If $q = \omega$ we then have to show both

(1) $\pi_1\, \langle^q t_1,\, t_2\rangle\sigma \circledR \pi_1\, |\langle^q t_1,\, t_2\rangle|\rho : A\sigma$
(2) $\pi_2\, \langle^q t_1,\, t_2\rangle\sigma \circledR \pi_2\, |\langle^q t_1,\, t_2\rangle|\rho : B[\sigma, \pi_1\, \langle^q t_1,\, t_2\rangle\sigma]$.

By Expansion (Lemma 4.4) it is sufficient to show $t_1\sigma \circledR |t_1|\rho : A$ and $t_2\sigma \circledR |t_2|\rho : B[\sigma, \pi_1\, \langle^q t_1,\, t_2\rangle\sigma]$, which follow from the assumptions. If $q = 0$ then we have only to show $\pi_2\, \langle^q t_1,\, t_2\rangle\sigma \circledR \pi_2\, |\langle^q t_1,\, t_2\rangle|\rho : B[\sigma, \pi_1\, \langle^q t_1,\, t_2\rangle\sigma]$, which follows for the same reason as above. □

Lemma 4.28.

$$\frac{\Gamma \models t : (x :^\omega A) \times B}{\Gamma \models \pi_1\, t : A} \qquad \frac{\Gamma \models t : (x :^q A) \times B}{\Gamma \models \pi_2\, t : B[\pi_1\, t/x]}$$

Proof. Both implications follow directly from the definition of $\Gamma \models t : (x :^q A) \times B$. □

Variables.

Lemma 4.29.

$$\frac{\vdash \Gamma \qquad (x :^q A) \in \Gamma \qquad q \leq \omega}{\Gamma \models x : A}$$

Proof. Given $\sigma \circledR \rho : \Gamma$ we have to show $x\sigma \circledR x\rho : A\sigma$ We have that $q \leq \omega$ implies $q = \omega$, so we only have to lookup what we need in $\sigma \circledR \rho : \Gamma$. □

Interval and Formulas.

Lemma 4.30.

$$\frac{\Gamma \vdash r : \mathbb{I}}{\Gamma \models r : \mathbb{I}} \qquad \frac{\Gamma \vdash \varphi : \mathbb{F}}{\Gamma \models \varphi : \mathbb{F}}$$

Proof. Given $\sigma \circledR \rho : \Gamma$ we have to show $r\sigma \circledR \lightning r\rho : \mathbb{I}$ and $\varphi\sigma \circledR \lightning \varphi\rho : \mathbb{F}$. For any interval variable i in Γ we have that $\sigma(i)$ and $\rho(i)$ agree on whether it is mapped to 0 or 1, and the rest is about calculating with boolean algebra expressions. □

Systems.

Lemma 4.31.

$$\frac{\Gamma, \varphi_1 \models t : A \quad \dots \quad \Gamma, \varphi_n \models t : A \qquad \Gamma \vdash (\varphi_1 \vee \dots \vee \varphi_n) = 1_{\mathbb{F}} : \mathbb{F}}{\Gamma \models t : A}$$

Proof. Given $\sigma \circledR \rho : \Gamma$ we have to show $t\sigma \circledR \lightning t\rho : A\sigma$. We have $\varepsilon \vdash (\varphi_1 \vee \dots \vee \varphi_n)\sigma = 1_{\mathbb{F}} : \mathbb{F}$, which implies there is a φ_k such that $\varepsilon \vdash \varphi_k\sigma = 1_{\mathbb{F}} : \mathbb{F}$, hace we also have $\sigma \circledR \rho : \Gamma, \varphi_k$ so we conclude with the correponding premise. □

Lemma 4.32.

$$\frac{\Gamma, \varphi_k \models t_k : A \ (\forall k) \qquad \Gamma, \varphi_k \wedge \varphi_l \vdash t_k = t_l : A \ (\forall k \neq l) \qquad \Gamma \vdash (\varphi_1 \vee \dots \vee \varphi_n) = 1_{\mathbb{F}} : \mathbb{F}}{\Gamma \models [\varphi_1 \hookrightarrow t_1, \dots, \varphi_n \hookrightarrow t_n] : A}$$

Proof. Given $\sigma \circledR \rho : \Gamma$ we have to show

$$[\varphi_1\sigma \hookrightarrow t_1\sigma, \dots, \varphi_n\sigma \hookrightarrow t_n\sigma] \circledR [|\varphi_1|\rho \hookrightarrow |t_1|\rho, \dots, |\varphi_n|\rho \hookrightarrow |t_n|\rho] : A\sigma$$

as before we have a φ_k such that $\varphi_k\sigma$ is equal to $1_{\mathbb{F}}$, in particular we have a minimal such k. Then also $|\varphi_k|\rho \rightsquigarrow^* 1$ while $|\varphi_j|\rho \rightsquigarrow^* 0$ for $j < k$. We conclude by the Expansion lemma 4.4 and the premise for t_k. □

Naturals.

Lemma 4.33.

$$\frac{\vdash \Gamma}{\Gamma \models \mathbb{N} : \mathsf{U}_n} \qquad \frac{\vdash \Gamma}{\Gamma \models \mathsf{zero} : \mathbb{N}} \qquad \frac{\Gamma \models t : \mathbb{N}}{\Gamma \models \mathsf{suc}\, t : \mathsf{U}_n}$$

Proof. The first implication is trivial. The other two follow directly by the definition of $t \circledR v : \mathbb{N}$. □

Lemma 4.34.

$$\frac{\Gamma, x :^\omega \mathbb{N} \vdash A \qquad \Gamma \models z : A[\mathsf{zero}/x] \qquad \Gamma, x :^q \mathbb{N}, y :^r A \models s : A[\mathsf{suc}\, x/x] \qquad \Gamma \models t : \mathbb{N}}{\Gamma \models \mathbb{N}\text{-}\mathsf{elim}_{x.A}\, z\, (x^q y^r.s)\, t : A[t/x]}$$

Proof. Given $\sigma \circledR \rho \ : \ \Gamma$ we have to show $\mathbb{N}\text{-elim}_{x.A}\, z\, (x^q y^r.s)\, t\sigma \circledR \mathbb{N}\text{-elim}\, |z|\, (xy.|s|)\, |t|\rho \ :\ A[\sigma, t]$. We proceed by induction on $t\sigma \circledR |t|\rho : \mathbb{N}$.

Case Zero here both $t\sigma$ and $|t|\rho$ reduce to zero so by the Expansion Lemma 4.4 we conclude with $z\sigma \circledR |z|\sigma : A\sigma$ which we get by assumption.

Case Suc $\vdash t\sigma \Rightarrow^* \mathrm{suc}\, t' :^{\omega} \mathbb{N}$ and $|t|\rho \rightsquigarrow^* \mathrm{suc}\, v'$ with $t' \circledR v' : \mathbb{N}$. Then by the Expansion Lemma 4.4 it is sufficient to show

$$s[\sigma, t', \mathbb{N}\text{-elim}_{x.A[\sigma,x}\,]\, (z\sigma^x q^y.r)\, s[\sigma, x, y] t'] \circledR |s|[\rho, v', \mathbb{N}\text{-elim}\, |z|\rho\, (x^q y^r.|s|[\sigma, x, y])\, v'] : A[\sigma, \mathrm{suc}\, t']$$

which follows from the assumption about s and the I.H. on $t' \circledR v' : \mathbb{N}$.

□

5 THE IMPLEMENTATION

We have implemented a variant of Cubical Agda based on the ideas presented in this text. This variant is activated through the use of the flag `--erased-cubical`. We do not present all details of the implementation here, just some key points.

The implementation uses typing rules similar to those presented in Section 4.1. Agda has several compiler backends, and we have focused mainly on the one that generates Haskell code. The implementation of this backend is based on the erasure function presented in Figure 2. For instance, the interval is compiled as the type of booleans: $\underline{0}$ is turned into `False`, and $\underline{1}$ to `True`.

6 A CASE STUDY

Let us now present a case study that shows that one can do something useful with the variant of Cubical Agda that we have introduced, in which univalence can only be used in erased contexts, and higher constructors must be erased. First we present equivalences with erased proofs and a non-recursive variant of $\|\_\|^{\mathrm{E}}$ (15), and then we move on to the main focus of the case study: higher lenses with erased proofs.

For every numbered definition or type signature in this section there is a corresponding piece of code in the accompanying Agda code. Some, but not all, numbered definitions above are also included in the accompanying code. (There are small differences between the code and the text.)

6.1 Equivalences with Erased Proofs

Recall that the type of equivalences was defined above (17). The following definition states that a function is an *equivalence with erased proofs*:

$$\begin{aligned}
&\mathit{Is\text{-}equivalence}^{\mathrm{E}} : \{A : \mathit{Type}\ a\}\ \{B : \mathit{Type}\ b\} \to (A \to B) \to \mathit{Type}\ (a \sqcup b)\\
&\mathit{Is\text{-}equivalence}^{\mathrm{E}}\ \{A = A\}\ \{B = B\}\ f =\\
&\quad (f^{-1} : B \to A) \times\\
&\quad \mathit{Erased}\ (\ (f\text{-}f^{-1} : \forall\ x \to f\ (f^{-1}\ x) \equiv x) \times (f^{-1}\text{-}f : \forall\ x \to f^{-1}\ (f\ x) \equiv x) \times\\
&\qquad\qquad \forall\ x \to \mathit{cong}\ f\ (f^{-1}\text{-}f\ x) \equiv f\text{-}f^{-1}\ (f\ x))
\end{aligned} \tag{36}$$

The definition uses *Erased* (25). If *Erased* is removed, then we get the usual definition of what it means to be a half adjoint equivalence (16). The type $A \simeq^{\mathrm{E}} B$ states that there is an equivalence with erased proofs from A to B:

$$\begin{aligned}
&_\simeq^{\mathrm{E}}_ : \mathit{Type}\ a \to \mathit{Type}\ b \to \mathit{Type}\ (a \sqcup b)\\
&A \simeq^{\mathrm{E}} B = (f : A \to B) \times \mathit{Is\text{-}equivalence}^{\mathrm{E}}\ f
\end{aligned} \tag{37}$$

If *eq* has type $A \simeq^{\mathrm{E}} B$, then we use *to eq* to denote the function (the first component), and *from eq* to denote the inverse (the second component). Note that the remaining components are erased. They can still be used in erased contexts:

$$@0\ \mathit{to\text{-}from} : (eq : A \simeq^{\mathrm{E}} B) \to \forall\ x \to \mathit{to}\ eq\ (\mathit{from}\ eq\ x) \equiv x$$
$$\mathit{to\text{-}from}\ (\_ , \_ , [\ (eq , \_)\ ]) = eq \tag{38}$$

$$@0\ \mathit{from\text{-}to} : (eq : A \simeq^{\mathrm{E}} B) \to \forall\ x \to \mathit{from}\ eq\ (\mathit{to}\ eq\ x) \equiv x$$
$$\mathit{from\text{-}to}\ (\_ , \_ , [\ (\_ , eq , \_)\ ]) = eq \tag{39}$$

In erased contexts *Erased* A is equivalent to A:

$$@0\ \mathit{Erased}{\simeq} : \mathit{Erased}\ A \simeq A \tag{40}$$

Thus $A \simeq^{\mathrm{E}} B$ and $A \simeq B$ are equivalent in erased contexts. Another thing to note is that, whenever there is an erased equivalence between A and B, then *Erased* A and *Erased* B are equivalent [Danielsson 2019]:

$$\{@0\ A : \mathit{Type}\ a\}\ \{@0\ B : \mathit{Type}\ b\} \to @0\ A \simeq B \to \mathit{Erased}\ A \simeq \mathit{Erased}\ B \tag{41}$$

This fact is used in several proofs discussed below.

The proof of the following preservation result makes use of the fact that the argument of Q is erased:

$$\{Q : @0\ B \to \mathit{Type}\ q\}\ (eq_1 : A \simeq^{\mathrm{E}} B)\ (eq_2 : \forall\ x \to P\ x \simeq^{\mathrm{E}} Q\ (\mathit{to}\ eq_1\ x)) \to$$
$$((x : A) \times P\ x) \simeq^{\mathrm{E}} ((x : B) \times Q\ x) \tag{42}$$

The right-to-left direction is defined in the following way:

$$\lambda\ (x , y) \to \mathit{from}\ eq_1\ x , \mathit{from}\ (eq_2\ (\mathit{from}\ eq_1\ x))\ (\mathit{subst}^{\mathrm{E}}\ Q\ (\mathit{sym}\ (\mathit{to\text{-}from}\ eq_1\ x))\ y)$$

Here *sym* is a proof of symmetry for paths, and $\mathit{subst}^{\mathrm{E}}$ is defined in the following way, using $[\,]$-*cong* (35) and a variant of *subst* (30) for paths:

$$\mathit{subst}^{\mathrm{E}} : \{@0\ A : \mathit{Type}\ a\}\ \{@0\ x\ y : A\}\ (P : @0\ A \to \mathit{Type}\ p) \to @0\ x \equiv y \to P\ x \to P\ y$$
$$\mathit{subst}^{\mathrm{E}}\ P\ eq = \mathit{subst}\ (\lambda\ ([\ x\ ]) \to P\ x)\ ([\,]\text{-}\mathit{cong}\ [\ eq\ ]) \tag{43}$$

$$\mathit{subst} : (P : A \to \mathit{Type}\ p) \to x \equiv y \to P\ x \to P\ y$$
$$\mathit{subst}\ P\ eq\ p = \mathit{transp}\ (\lambda\ i \to P\ (eq\ i))\ \underline{0}\ p \tag{44}$$

Note that, while the definition of $\mathit{subst}^{\mathrm{E}}$ is similar to some problematic definitions from Section 3, it is fine because P takes an erased argument. In the implementation of the right-to-left direction above we can use the erased definition *to-from* because the path argument of $\mathit{subst}^{\mathrm{E}}$ is erased.

As an example of what Lemma 42 can be used for we have the following lemma:

$$\{P : @0\ A \to \mathit{Type}\ p\} \to (eq : A \simeq^{\mathrm{E}} \top) \to ((x : A) \times P\ x) \simeq^{\mathrm{E}} P\ (\mathit{from}\ eq\ \mathsf{tt}) \tag{45}$$

We can calculate in the following way, using Lemma 42 in the first step:

$$((x : A) \times P\ x) \quad \simeq^{\mathrm{E}} \quad ((x : \top) \times P\ (\mathit{from}\ eq\ x)) \quad \simeq \quad P\ (\mathit{from}\ eq\ \mathsf{tt})$$

Lemma 42 requires that the argument of Q is erased. If this is not the case, then one can in some cases use the following lemma instead (the proof is omitted):

$$(f : A \to B)\ (f^{-1} : B \to A) \to (\forall\ x \to f\ (f^{-1}\ x) \equiv x) \to @0\ (\forall\ x \to f^{-1}\ (f\ x) \equiv x) \to$$
$$(\forall\ x \to P\ x \simeq^{\mathrm{E}} Q\ (f\ x)) \to ((x : A) \times P\ x) \simeq^{\mathrm{E}} ((x : B) \times Q\ x) \tag{46}$$

However, this lemma may be harder to use: note that one of the equality proof arguments is not erased. The lemma can be used to prove the following variant of Lemma 45:

$$(eq : A \simeq^{\mathrm{E}} \top) \to (\forall\ x\ y \to P\ x \simeq^{\mathrm{E}} P\ y) \to ((x : A) \times P\ x) \simeq^{\mathrm{E}} P\ (\mathit{from}\ eq\ \mathsf{tt}) \tag{47}$$

Here P is not required to take an erased argument, but in return one has to prove that P is weakly constant, up to equivalences with erased proofs. (When P is omitted from a type signature its type is an instance of $A \to \mathit{Type}\ p$.)

6.2 A Non-recursive Definition of the Propositional Truncation Operator

Van Doorn [2016] presents a definition of the propositional truncation operator that does not use any recursive higher inductive types (it uses the natural numbers). This definition makes use of two non-recursive higher inductive types, the *one-step truncation* and the *sequential colimit*:

$$\begin{array}{l} \textbf{data}\ \|\_\|^1\ (A : \mathit{Type}\ a) : \mathit{Type}\ a\ \textbf{where} \\ \quad |\_| \qquad\qquad : A \to \|\ A\ \|^1 \\ \quad \|\text{-constant} : \forall\ x\ y \to |\ x\ | \equiv |\ y\ | \end{array} \tag{48}$$

$$\begin{array}{l} \textbf{data}\ \mathit{Colimit}\ (P : \mathbb{N} \to \mathit{Type}\ p)\ (\mathit{step} : \forall\ \{n\} \to P\ n \to P\ (\mathrm{suc}\ n)) : \mathit{Type}\ p\ \textbf{where} \\ \quad |\_| \quad\ : P\ n \to \mathit{Colimit}\ P\ \mathit{step} \\ \quad |\mathrm{step}| : (x : P\ n) \to |\ \mathit{step}\ x\ | \equiv |\ x\ | \end{array} \tag{49}$$

The sequential colimit has the following universal property:

$$\begin{array}{l} \{\mathit{step} : \forall\ \{n\} \to P\ n \to P\ (\mathrm{suc}\ n)\} \to \\ (\mathit{Colimit}\ P\ \mathit{step} \to B) \simeq ((f : \forall\ n \to P\ n \to B) \times \forall\ n\ x \to f\ (\mathrm{suc}\ n)\ (\mathit{step}\ x) \equiv f\ n\ x) \end{array} \tag{50}$$

The one-step truncation can be iterated (we include "out" in the name, following Capriotti et al. [2021], because the final application of the one-step truncation is on the outside):

$$\begin{array}{l} \|\_\|^1\text{-}\mathit{out} : \mathit{Type}\ a \to \mathbb{N} \to \mathit{Type}\ a \\ \|\ A\ \|^1\text{-}\mathit{out}\ \mathrm{zero} \qquad = A \\ \|\ A\ \|^1\text{-}\mathit{out}\ (\mathrm{suc}\ n) = \|\ \|\ A\ \|^1\text{-}\mathit{out}\ n\ \|^1 \end{array} \tag{51}$$

The non-recursive definition of the propositional truncation operator can now be defined in the following way:

$$\begin{array}{l} \|\_\|^{\mathrm{N}} : \mathit{Type}\ a \to \mathit{Type}\ a \\ \|\ A\ \|^{\mathrm{N}} = \mathit{Colimit}\ \|\ A\ \|^1\text{-}\mathit{out}\ |\_| \end{array} \tag{52}$$

This definition is equivalent to the recursive one presented above (10).

Above we defined a variant of the propositional truncation operator with an erased higher constructor, $\|\_\|^{\mathrm{E}}$ (15). Let us now present a non-recursive variant of this operator, following Van Doorn. We use the following variant of the sequential colimit:

$$\begin{array}{l} \textbf{data}\ \mathit{Colimit}^{\mathrm{E}}\ (P_0 : \mathit{Type}\ p_0)\ (@0\ P_+ : \mathbb{N} \to \mathit{Type}\ p_+)\ (@0\ \mathit{step}_0 : P_0 \to P_+\ \mathrm{zero}) \\ \qquad\qquad\qquad (@0\ \mathit{step}_+ : \forall\ \{n\} \to P_+\ n \to P_+\ (\mathrm{suc}\ n)) : \mathit{Type}\ (p_0 \sqcup p_+)\ \textbf{where} \\ \quad |\_|_0 \qquad\quad\ : P_0 \qquad\quad \to \mathit{Colimit}^{\mathrm{E}}\ P_0\ P_+\ \mathit{step}_0\ \mathit{step}_+ \\ \quad @0\ |\_|_+ \qquad : P_+\ n \qquad \to \mathit{Colimit}^{\mathrm{E}}\ P_0\ P_+\ \mathit{step}_0\ \mathit{step}_+ \\ \quad @0\ |\mathrm{step}_0|_+\ : (x : P_0) \quad\ \to |\ \mathit{step}_0\ x\ |_+ \equiv |\ x\ |_0 \\ \quad @0\ |\mathrm{step}_+|_+\ : (x : P_+\ n) \to |\ \mathit{step}_+\ x\ |_+ \equiv |\ x\ |_+ \end{array} \tag{53}$$

Note that both higher constructors are erased, as well as one of the point constructors. This variant of the sequential colimit has the following universal property:

$$\begin{aligned}
&\{@0\ P_+ : \mathbb{N} \to \mathit{Type}\ p_+\} \\
&\{@0\ \mathit{step}_0 : P_0 \to P_+\ \mathsf{zero}\}\ \{@0\ \mathit{step}_+ : \forall\ \{n\} \to P_+\ n \to P_+\ (\mathsf{suc}\ n)\} \to \\
&(\mathit{Colimit}^{\mathrm{E}}\ P_0\ P_+\ \mathit{step}_0\ \mathit{step}_+ \to B) \simeq \\
&((f_0 : P_0 \to B) \times \mathit{Erased}\ (\ (f_+ : \forall\ n \to P_+\ n \to B) \times \\
&\qquad (\forall\ x \to f_+\ \mathsf{zero}\ (\mathit{step}_0\ x) \equiv f_0\ x) \times \\
&\qquad (\forall\ n\ x \to f_+\ (\mathsf{suc}\ n)\ (\mathit{step}_+\ x) \equiv f_+\ n\ x)))
\end{aligned} \tag{54}$$

A function from the sequential colimit $\mathit{Colimit}^{\mathrm{E}}\ P_0\ P_+\ \mathit{step}_0\ \mathit{step}_+$ is equivalent to a function from P_0, along with some erased information.

We can now define a non-recursive variant of $\|\_\|^{\mathrm{E}}$ in the following way:

$$\begin{aligned}
&\|_\|^{\mathrm{NE}} : \mathit{Type}\ a \to \mathit{Type}\ a \\
&\|\ A\ \|^{\mathrm{NE}} = \mathit{Colimit}^{\mathrm{E}}\ A\ (\|\ A\ \|^1\text{-}\mathit{out} \circ \mathsf{suc})\ |_|\ |_|
\end{aligned} \tag{55}$$

Note that it is fine to use the one-step truncation with a non-erased higher constructor in the second argument of $\mathrm{Colimit}^{\mathrm{E}}$, because this argument is erased. This definition is pointwise equivalent to the other one:

$$\|\ A\ \|^{\mathrm{NE}} \simeq \|\ A\ \|^{\mathrm{E}} \tag{56}$$

6.3 Higher Lenses with Erased Proofs

Capriotti et al. [2021] introduce *higher lenses*, variants of total, very well-behaved lenses [Foster et al. 2005] for which proofs that imply the lens laws are included in the data structures. These data structures are intended to work well in homotopy type theory/univalent foundations.

Capriotti et al. present several definitions of higher lenses. Here is one of them:

$$\begin{aligned}
&\textbf{record}\ \mathit{Lens}^{\mathrm{E}}\ (A : \mathit{Type}\ a)\ (B : \mathit{Type}\ b) : \mathit{Type}\ (\mathit{lsuc}\ (a \sqcup b))\ \textbf{where} \\
&\quad \textbf{field}\ \mathsf{R} : \mathit{Type}\ (a \sqcup b);\ \mathsf{equiv} : A \simeq \mathsf{R} \times B;\ \mathsf{inhabited} : \mathsf{R} \to \|\ B\ \|
\end{aligned} \tag{57}$$

A higher lens based on equivalences ("E") from the source type A to the view type B is a remainder type R, an equivalence between A and the Cartesian product of R and B, and a function from R to the propositional truncation of B.

This data structure contains data that might not be needed at run-time. Let us assume that all we need at run-time is to be able to use the associated getter and setter, which can be defined using the two directions of the equivalence (following Van Laarhoven [2011]):

$$\mathit{get} : \mathit{Lens}^{\mathrm{E}}\ A\ B \to A \to B \tag{58}$$

$$\mathit{set}\ : \mathit{Lens}^{\mathrm{E}}\ A\ B \to A \to B \to A \tag{59}$$

In that case we can make some parts of the data structure erased (the second "E" stands for "erased"):

$$\begin{aligned}
&\textbf{record}\ \mathit{Lens}^{\mathrm{EE}}\ (A : \mathit{Type}\ a)\ (B : \mathit{Type}\ b) : \mathit{Type}\ (\mathit{lsuc}\ (a \sqcup b))\ \textbf{where} \\
&\quad \textbf{field}\ \mathsf{R} : \mathit{Type}\ (a \sqcup b);\ \mathsf{equiv} : A \simeq^{\mathrm{E}} \mathsf{R} \times B;\ @0\ \mathsf{inhabited} : \mathsf{R} \to \|\ B\ \|
\end{aligned} \tag{60}$$

All that remains at run-time is the type R and the two directions of the equivalence. One might have hoped that it would be possible to mark R as erased, but as noted in Section 3 our system does not allow this (even though the erasure function in Figure 2 does erase types).

Note that the types of higher lenses given above are large (the resulting universe level includes *lsuc*). Capriotti et al. also present a small definition. This definition uses the notion of a fibre [The Univalent Foundations Program 2013], a kind of proof-relevant preimage:

$$\begin{aligned}
&_^{-1}_ : \{A : \mathit{Type}\ a\}\ \{B : \mathit{Type}\ b\} \to (A \to B) \to B \to \mathit{Type}\ (a \sqcup b) \\
&f^{-1}\ y = \exists\ x \times f\ x \equiv y
\end{aligned} \tag{61}$$

(We use the notation $\exists\ x \times P\ x$ for Σ-types when we do not want to write out the type of the first component.) A coinductive higher lens consists of a getter and a proof showing that the family of fibres of the getter is coherently constant:

$$\begin{aligned}&\mathit{Lens}^{\mathrm{C}} : \mathit{Type}\ a \to \mathit{Type}\ b \to \mathit{Type}\ (a \sqcup b)\\&\mathit{Lens}^{\mathrm{C}}\ A\ B = (\mathit{get} : A \to B) \times \mathit{Coherently\text{-}constant}^{\mathrm{C}}\ (\mathit{get}\ {}^{-1}\_)\end{aligned} \tag{62}$$

This definition is called coinductive because *Coherently-constant*$^{\mathrm{C}}$ is defined coinductively. We do not include the definition here, only its type signature:

$$\mathit{Coherently\text{-}constant}^{\mathrm{C}} : \{A : \mathit{Type}\ a\} \to (A \to \mathit{Type}\ p) \to \mathit{Type}\ (a \sqcup p) \tag{63}$$

Let us now present a variant of the small, coinductive higher lenses which at run-time consists of nothing but a getter and a setter. We build on the definition of *Lens*$^{\mathrm{C}}$ above: we include a getter, and an erased proof cc showing that the family of fibres of the getter is constant. However, we also want to include a non-erased setter. We do this, and then we add an erased field ensuring that this setter is equal to the setter obtained from get and cc (using the function *Lens*$^{\mathrm{C}}$*.set*):

$$\begin{aligned}&\textbf{record}\ \mathit{Lens}^{\mathrm{CE}}\ (A : \mathit{Type}\ a)\ (B : \mathit{Type}\ b) : \mathit{Type}\ (a \sqcup b)\ \textbf{where}\\&\quad\textbf{field}\ \mathrm{get} : A \to B;\ \mathrm{set} : A \to B \to A;\ @0\ \mathrm{cc} : \mathit{Coherently\text{-}constant}^{\mathrm{C}}\ (\mathrm{get}\ {}^{-1}\_)\\&\qquad\quad @0\ \mathrm{set{\equiv}set} : \mathrm{set} \equiv \mathit{Lens}^{\mathrm{C}}.\mathit{set}\ (\mathrm{get}\ ,\ \mathrm{cc})\end{aligned} \tag{64}$$

6.4 The Definitions Are Equivalent

How is *Lens*$^{\mathrm{EE}}$ related to *Lens*$^{\mathrm{CE}}$? These two types are pointwise equivalent (with erased proofs):

$$\mathit{Lens}^{\mathrm{EE}}\ A\ B \simeq^{\mathrm{E}} \mathit{Lens}^{\mathrm{CE}}\ A\ B \tag{65}$$

We do not include all details of the proof of this equivalence, but only some highlights intended to illustrate some lemmas and techniques that can be used when proving things in this setting. The full proof is available in the accompanying code, along with a proof (in an erased context) showing that the equivalence preserves getters and setters.

We proved the equivalence by going via two other representations of lenses:

$$\mathit{Lens}^{\mathrm{EE}}\ A\ B \quad \simeq^{\mathrm{E}} \quad \mathit{Lens}_1^{\mathrm{E}}\ A\ B \quad \simeq^{\mathrm{E}} \quad \mathit{Lens}_2^{\mathrm{E}}\ A\ B \quad \simeq^{\mathrm{E}} \quad \mathit{Lens}^{\mathrm{CE}}\ A\ B$$

Both of these representations are similar to *Lens*$^{\mathrm{C}}$ (but large):

$$\begin{aligned}&\mathit{Lens}_1^{\mathrm{E}} : \mathit{Type}\ a \to \mathit{Type}\ b \to \mathit{Type}\ (\mathit{lsuc}\ (a \sqcup b))\\&\mathit{Lens}_1^{\mathrm{E}}\ A\ B = (\mathit{get} : A \to B) \times \mathit{Coherently\text{-}constant}_1^{\mathrm{E}}\ (\mathit{get}\ {}^{-1\mathrm{E}}\_)\end{aligned} \tag{66}$$

$$\begin{aligned}&\mathit{Lens}_2^{\mathrm{E}} : \mathit{Type}\ a \to \mathit{Type}\ b \to \mathit{Type}\ (\mathit{lsuc}\ (a \sqcup b))\\&\mathit{Lens}_2^{\mathrm{E}}\ A\ B = (\mathit{get} : A \to B) \times \mathit{Coherently\text{-}constant}_2^{\mathrm{E}}\ (\mathit{get}\ {}^{-1\mathrm{E}}\_)\end{aligned} \tag{67}$$

They use a variant of the definition of fibres (61) with an erased equality proof:

$$\begin{aligned}&\_{}^{-1\mathrm{E}}\_ : \{A : \mathit{Type}\ a\}\ \{@0\ B : \mathit{Type}\ b\} \to @0\ (A \to B) \to @0\ B \to \mathit{Type}\ (a \sqcup b)\\&f\ {}^{-1\mathrm{E}}\ y = \exists\ x \times \mathit{Erased}\ (f\ x \equiv y)\end{aligned} \tag{68}$$

The type family *Coherently-constant*$_1^{\mathrm{E}}$ is defined in the following way:

$$\begin{aligned}&\mathit{Coherently\text{-}constant}_1^{\mathrm{E}} : \{A : \mathit{Type}\ a\} \to (A \to \mathit{Type}\ p) \to \mathit{Type}\ (a \sqcup \mathit{lsuc}\ p)\\&\mathit{Coherently\text{-}constant}_1^{\mathrm{E}}\ \{p = p\}\ \{A = A\}\ P =\\&\quad (Q : \|\ A\ \|^{\mathrm{E}} \to \mathit{Type}\ p) \times (\forall\ x \to P\ x \simeq^{\mathrm{E}} Q \mid x \mid) \times (f : \forall\ x\ y \to Q\ x \to Q\ y) \times\\&\quad \mathit{Erased}\ (\forall\ x\ y \to f\ x\ y \equiv \mathit{subst}\ Q\ (\mathsf{trunc}\ x\ y))\end{aligned} \tag{69}$$

Note the use of the variant of the propositional truncation operator with an erased higher constructor (15). Compare this definition to the following standard definition of coherent (or conditional) constancy [Shulman 2015], which does not use erasure:

$$\begin{aligned}&\mathit{Coherently\text{-}constant} : \{A : \mathit{Type}\ a\}\ \{B : \mathit{Type}\ b\} \to (A \to B) \to \mathit{Type}\ (a \sqcup b)\\&\mathit{Coherently\text{-}constant}\ \{A = A\}\ \{B = B\}\ f = (g : \|\ A\ \| \to B) \times f \equiv g \circ |\_|\end{aligned} \tag{70}$$

$\mathit{Coherently\text{-}constant}_1^{\mathrm{E}}$ is restricted to type-valued functions, for which equalities can be expressed using equivalences (in the presence of univalence): the definition uses a family of equivalences with erased proofs. Because the constructor trunc of $\|\_\|^{\mathrm{E}}$ is erased the definition also includes a non-erased part, f, which in erased contexts could be proved using trunc. The final, erased part ensures that f is pointwise equal to such a proof, which is defined using *subst* (44).

The type family $\mathit{Coherently\text{-}constant}_2^{\mathrm{E}}$ is defined in the following way:

$$\begin{aligned}&\mathit{Coherently\text{-}constant}_2^{\mathrm{E}} : \{A : \mathit{Type}\ a\} \to (A \to \mathit{Type}\ p) \to \mathit{Type}\ (a \sqcup \mathit{lsuc}\ p)\\&\mathit{Coherently\text{-}constant}_2^{\mathrm{E}}\ P =\\&\quad (f : \forall\ x\ y \to P\ x \to P\ y) \times\\&\quad \mathit{Erased}\ ((c : \mathit{Coherently\text{-}constant}_2^{\mathrm{C}}\ P) \times \forall\ x\ y \to f\ x\ y \equiv \mathit{subst}\ \mathit{id}\ (\mathit{constant}\ c\ x\ y))\end{aligned} \tag{71}$$

This definition uses yet another (coinductive) definition of coherent constancy, taken from the work of Capriotti et al. [2021, Definition 77]:

$$\mathit{Coherently\text{-}constant}_2^{\mathrm{C}} : \{A : \mathit{Type}\ a\}\ \{B : \mathit{Type}\ b\}\ (f : A \to B) \to \mathit{Type}\ (a \sqcup b) \tag{72}$$

This definition uses a higher inductive type with a non-erased higher constructor (48), and is not used at run-time. We only include the type signature, but note that coherently constant functions are weakly constant:

$$\mathit{constant} : \mathit{Coherently\text{-}constant}_2^{\mathrm{C}}\ f \to \forall\ x\ y \to f\ x \equiv f\ y \tag{73}$$

The definition of $\mathit{Coherently\text{-}constant}_2^{\mathrm{E}}$ includes a non-erased part, f, which in erased contexts could be proved using the erased proof c. The final, erased part ensures that f is pointwise equal to such a proof.

The first step of Lemma 65 ($\mathit{Lens}^{\mathrm{EE}}\ A\ B \simeq^{\mathrm{E}} \mathit{Lens}_1^{\mathrm{E}}\ A\ B$) is proved by giving functions in both directions and proving that these functions are inverses of each other. The proof is similar to one presented by Capriotti et al. [2021, Lemma 67]. It uses erased univalence. The right-to-left direction makes use of the fact that the "f" component of $\mathit{Coherently\text{-}constant}_1^{\mathrm{E}}$ is not erased. It also uses the following variant of the standard result that singletons are contractible [The Univalent Foundations Program 2013, Lemma 3.11.8]:

$$((y : A) \times \mathit{Erased}\ (x \equiv y)) \simeq^{\mathrm{E}} \top \tag{74}$$

This lemma says that singletons with erased proofs are equivalent, with erased proofs, to the unit type. (Note that the value x is not assumed to be erased.)

The second step of the proof ($\mathit{Lens}_1^{\mathrm{E}}\ A\ B \simeq^{\mathrm{E}} \mathit{Lens}_2^{\mathrm{E}}\ A\ B$) uses the following lemma:

$$(\|\ A\ \|^{\mathrm{E}} \to B) \simeq ((f : A \to B) \times \mathit{Erased}\ (\mathit{Coherently\text{-}constant}_2^{\mathrm{C}}\ f)) \tag{75}$$

Capriotti et al. prove a corresponding result that does not use erasure [2021, Lemma 82]. Our proof is similar:

$$\begin{aligned}&(\|\ A\ \|^{\mathrm{E}} \to B) && \simeq\\&(\|\ A\ \|^{\mathrm{NE}} \to B) && \simeq\\&((f_0 : A \to B) \times \mathit{Erased}\ (\ (f_+ : \forall\ n \to \|\ A\ \|^1\text{-}\mathit{out}\ (1 + n) \to B) \times\end{aligned}$$

$$(\forall\, x \to f_+\ 0 \mid x \mid \equiv f_0\ x) \times (\forall\, n\ x \to f_+\ (1 + n) \mid x \mid \equiv f_+\ n\ x))) \ \simeq$$
$$((f : A \to B) \times \mathit{Erased}\ (\mathit{Coherently\text{-}constant}^{\mathrm{C}}_2\ f))$$

The first two steps are based on work by Van Doorn [2016]: in the first step Lemma 56 is used to replace $\|\_\|^{\mathrm{E}}$ with the non-recursive variant defined above (55), and the second step uses the universal property of the sequential colimit (54). The final step makes use of some results proved by Capriotti et al. Note that this proof makes use of the fact that one can have types with erased point constructors (in this case $\|\_\|^{\mathrm{NE}}$).

The second step of Lemma 65 also uses Lemmas 42 and 45, as well as the following two lemmas related to erasure:

$$((Q : A \to \mathit{Type}\ p) \times \forall\, x \to P\ x \simeq^{\mathrm{E}} Q\ x) \simeq^{\mathrm{E}} \top \tag{76}$$

$$\begin{aligned}&\{@0\ g : (x : \|\ A\ \|^{\mathrm{E}}) \to P\ x\} \to\\ &((f : (x : \|\ A\ \|^{\mathrm{E}}) \to P\ x) \times \mathit{Erased}\ (f \equiv g)) \simeq ((f : (x : A) \to P \mid x \mid) \times \mathit{Erased}\ (f \equiv g \circ |\_|))\end{aligned} \tag{77}$$

The first lemma is a variant of Lemma 74, proved using erased univalence. The second lemma makes it possible to, in some cases, replace a function from $\|\ A\ \|^{\mathrm{E}}$ with a function from A. It is proved using $\|\_\|^{\mathrm{NE}}$ (55) and a dependent variant of the universal property of $\mathit{Colimit}^{\mathrm{E}}$ (54).

6.5 Compilation of Lenses

Let us define a lens for the second projection of a non-dependent pair. We use the type $\mathit{Lens}^{\mathrm{EE}}$, because this makes it easy to define the lens (for simplicity the types A and B are assumed to be in the same universe):

$$\mathit{snd}^{\mathrm{E}} : \{A\ B : \mathit{Type}\ a\} \to \mathit{Lens}^{\mathrm{EE}}\ (A \times B)\ B \tag{78}$$

We let the R field be $A \times \mathit{Erased}\ \|\ B\ \|$. This makes it easy to implement the inhabited field. It remains to prove that $A \times B$ is equivalent to $(A \times \mathit{Erased}\ \|\ B\ \|) \times B$, which follows from the following lemma:

$$\mathit{Erased}\ \|\ A\ \| \times A \simeq A \tag{79}$$

Using Lemma 65 we can convert the lens to the type $\mathit{Lens}^{\mathrm{CE}}$ that, at run-time, consists of nothing but a getter and a setter:

$$\mathit{snd}^{\mathrm{C}} : \{A\ B : \mathit{Type}\ a\} \to \mathit{Lens}^{\mathrm{CE}}\ (A \times B)\ B \tag{80}$$

If we instruct Agda to normalise every application of the conversion function that we get from Lemma 65 before the code is compiled, compile the code using Agda's (non-strict) GHC backend, and inspect the intermediate code at one point of GHC 9.0.1's compilation pipeline, then we get something like the following:

```
sndᶜ = \_ _ _ -> Lens
  (\p -> case p of { Pair _ y -> y })
  (\p y -> Pair (case p of { Pair x _ -> x }) y)
```

(Names have been changed, code related to coercions and casts has been removed, and some definitions have been inlined.) One thing to note is that the code above could lead to a space leak: when the setter is applied to a pair p and a new second component y the entire pair p might be retained until the first component of the result is demanded. This could perhaps be fixed by changing our implementation of $\mathit{snd}^{\mathrm{C}}$, or by switching to a strict backend. However, we choose to demonstrate another way to address this problem. The following lemma can be used to change the implementation of the setter, as long as the new implementation is extensionally equal to the old one:

$$(l : \mathit{Lens}^{\mathrm{CE}}\ A\ B)\ (\mathit{set} : A \to B \to A) \to @0\ \mathit{set} \equiv \mathsf{Lens}^{\mathrm{CE}}.\mathsf{set}\ l \to \mathit{Lens}^{\mathrm{CE}}\ A\ B \tag{81}$$

If we change the implementation to λ { $(x , \_)\ y \to (x , y)$ }, then we obtain the following code (edited as described above):

```
snd^C = \_ _ _ -> Lens
  (\p -> case p of { Pair _ y -> y })
  (\p y -> case p of { Pair x _ -> Pair x y })
```

7 RELATED WORK

We are not aware of any previous work on compiling cubical type theory, nor on combining cubical type theory with an erasure modality. However, as mentioned in the introduction there is plenty of work on erasure. The work of McBride [2016] and Atkey [2018] has been influential recently. Let us highlight some differences between the typing rules presented by Atkey and those used here. Atkey presents a type system that can be instantiated with different semirings (building on the work by McBride), we focus on the instantiation with a semiring with two elements, corresponding to the two quantities 0 and ω.

One difference is that in Atkey's type system a variable may only be used if all other variables in the context have quantity zero. This is presumably in order to support linear types, which we do not support. Our variable typing rule does not have any restrictions on the rest of the context.

Another difference is that, in Atkey's type system, elements of the universe can only be constructed in erased contexts. This is not much of a limitation, because the term constructor El, which takes elements of the universe to types, takes an erased argument. In contrast, we allow elements of the universe to be constructed in non-erased contexts. Furthermore one of the premises of the typing rule for transp is an element of the universe, and this premise uses the same quantity as the rule's conclusion. If the quantity of this premise were 0, then we could construct something akin to the problematic terms subst_1 (27) and subst_2 (21). (The typing rules for hcomp, hcomp-intro, hcomp-elim and the propositional truncation's eliminator also include premises such as the one discussed above.)

Atkey does not include a dedicated empty type (even though such a type could perhaps be encoded in his system). We include an empty type $\bot$, along with an "escape hatch", a function $\bot\text{-}\mathit{elim} : @0\ \bot \to A$ that takes an erased argument of the empty type and returns a (possibly) *non-erased* result of the arbitrary type A. With this function one can use an *erased* proof to discard an impossible branch. For instance, consider the following safe head function:

$$\begin{aligned}
&\mathit{head} : (\mathit{xs} : \mathit{List}\ A) \to @0\ (\mathit{xs} \not\equiv []) \to A \\
&\mathit{head}\ (x :: \mathit{xs})\ _ = x \\
&\mathit{head}\ []\qquad\ \ p = \bot\text{-}\mathit{elim}\ (p\ \mathit{refl})
\end{aligned} \tag{82}$$

(Here $x \not\equiv y$ is equal to $x \equiv y \to \bot$, and *refl* is a proof of reflexivity.)

A system for which the empty type's eliminator has type $(@0\ x : \bot)\ (@0\ P : \bot \to \mathit{Type}) \to P\ x$ (rephrased using this paper's notation) supports *empty type target erasure* [Mishra-Linger 2008]. If the erasure translation removes erased arguments and corresponding applications entirely, then one can end up with closed terms that get stuck. For instance, consider any closed term of type $@0\ \bot \to \bot$. For this reason Mishra-Linger argues against giving the eliminator this type. We instead do not remove erased lambda abstractions, and replace corresponding arguments with dummy values. Letouzey [2003] takes a similar approach, with an extra optimisation intended to reduce the number of lambda abstractions in the final code.

Mishra-Linger also argues against *token type target erasure*, where pattern matching is allowed for erased arguments if there is exactly one constructor with zero non-erased arguments. He constructs a piece of code for which the corresponding erased term loops forever, by employing token type target erasure for a type similar to *Id* (20). Again this example relies on erasure removing erased arguments entirely. The Agda implementation supports an extension of token type target erasure where the single constructor is not required to have zero non-erased arguments, but all arguments are treated as erased on the right-hand side. We did not include this feature in the theoretical development, instead choosing to focus on other things.

We are not aware of any previous work on erased constructors.

8 CONCLUSION

To the best of our knowledge this piece of work provides the first integration of erasure and cubical type theory, as well as the first "reasonable" way to compile some variant of cubical type theory.

We restrict certain cubical features: Glue and higher constructors may only be used in code that will be erased by the compiler. This makes the language more restrictive than full cubical type theory, but we have shown using a larger case study that the resulting language is useful. Furthermore it might be the case that one cannot compile full cubical type theory without some kind of performance overhead, due to the fact that some computation rules involve computation under binders. With the approach described here one can use standard compilation techniques.

Our theoretical development depends on assumptions that have not been proved. As mentioned above we believe that these assumptions can be proved by extending Huber's work on canonicity [2019] to our theory. We acknowledge that the fact that the meta-theory is not mechanised increases the risk that some definition or proof is incorrect. If work is undertaken to mechanise the meta-theory, then it may make sense to build on the formalisations presented by Abel et al. [2018] and Eriksson [2021].

ACKNOWLEDGEMENTS

Andreas Abel acknowledges support by the Swedish Research Council (Vetenskapsrådet) under grant 2019-04216 *Modal Dependent Type Theory*.

Andrea Vezzosi was supported by a research grant (13156) from VILLUM FONDEN.

REFERENCES

Andreas Abel, Joakim Öhman, and Andrea Vezzosi. 2018. Decidability of Conversion for Type Theory in Type Theory. *Proceedings of the ACM on Programming Languages* 2, POPL (2018), 23:1–23:29. https://doi.org/10.1145/3158111

Carlo Angiuli, Guillaume Brunerie, Thierry Coquand, Kuen-Bang Hou (Favonia), Robert Harper, and Daniel R. Licata. 2021a. Syntax and Models of Cartesian Cubical Type Theory. (2021). https://github.com/dlicata335/cart-cube/raw/f80af5cf6638c85ffda2397a8e097b3e9decac1b/cart-cube.pdf Unpublished.

Carlo Angiuli, Evan Cavallo, Anders Mörtberg, and Max Zeuner. 2021b. Internalizing Representation Independence with Univalence. *Proceedings of the ACM on Programming Languages* 5, POPL (2021), 12:1–12:30. https://doi.org/10.1145/3434293

Robert Atkey. 2018. Syntax and Semantics of Quantitative Type Theory. In *LICS '18 Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*. 56–65. https://doi.org/10.1145/3209108.3209189

Bruno Barras and Bruno Bernardo. 2008. The Implicit Calculus of Constructions as a Programming Language with Dependent Types. In *Foundations of Software Science and Computational Structures, 11th International Conference, FOSSACS 2008*. 365–379. https://doi.org/10.1007/978-3-540-78499-9_26

Jean-Philippe Bernardy and Guilhem Moulin. 2013. Type-Theory In Color. In *ICFP'13, Proceedings of the 2013 ACM SIGPLAN International Conference on Functional Programming*. 61–71. https://doi.org/10.1145/2500365.2500577

Mathieu Boespflug, Maxime Dénès, and Benjamin Grégoire. 2011. Full Reduction at Full Throttle. In *Certified Programs and Proofs, First International Conference, CPP 2011*. 362–377. https://doi.org/10.1007/978-3-642-25379-9_26

Edwin Brady. 2021. Idris 2: Quantitative Type Theory in Practice. In *35th European Conference on Object-Oriented Programming, ECOOP 2021*. 9:1–9:26. https://doi.org/10.4230/LIPIcs.ECOOP.2021.9

Edwin Brady, Conor McBride, and James McKinna. 2004. Inductive Families Need Not Store Their Indices. In *TYPES 2003: Types for Proofs and Programs*. 115–129. https://doi.org/10.1007/978-3-540-24849-1_8

Paolo Capriotti, Nils Anders Danielsson, and Andrea Vezzosi. 2021. Higher Lenses. Accepted for publication in LICS 2021, the 36th Annual Symposium on Logic in Computer Science. https://gup.ub.gu.se/publication/304854

Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. 2015–2018a. Cubicaltt. (2015–2018). https://github.com/mortberg/cubicaltt.

Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. 2018b. Cubical Type Theory: A Constructive Interpretation of the Univalence Axiom. In *21st International Conference on Types for Proofs and Programs, TYPES 2015*. 5:1–5:34. https://doi.org/10.4230/LIPIcs.TYPES.2015.5

Nils Anders Danielsson. 2019. Logical properties of a modality for erasure. (2019). http://www.cse.chalmers.se/~nad/publications/danielsson-erased.html

Oskar Eriksson. 2021. *An Agda formalization of modalities and erasures in a dependently typed language.* Master's thesis. Chalmers University of Technology.

Maribel Fernandez, Ian Mackie, Paula Severi, and Nora Szasz. 2003. Reduction Strategies for Program Extraction. *CLEI Electronic Journal* 6, 1 (2003). https://doi.org/10.19153/cleiej.6.1.2

J. Nathan Foster, Michael B. Greenwald, Jonathan T. Moore, Benjamin C. Pierce, and Alan Schmitt. 2005. Combinators for Bi-Directional Tree Transformations: A Linguistic Approach to the View Update Problem. In *POPL® 2005: The 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages®*. 233–246. https://doi.org/10.1145/1040305.1040325

Benjamin Grégoire and Xavier Leroy. 2002. A Compiled Implementation of Strong Reduction. In *Proceedings of the Seventh ACM SIGPLAN International Conference on Functional Programming (ICFP '02)*. 235–246. https://doi.org/10.1145/581478.581501

Adam Gundry and Conor McBride. 2013. Phase Your Erasure. (2013). https://personal.cis.strath.ac.uk/conor.mcbride/pub/phtt.pdf

Adam Michael Gundry. 2013. *Type Inference, Haskell and Dependent Types.* Ph.D. Dissertation. University of Strathclyde. http://adam.gundry.co.uk/pub/thesis/

Simon Huber. 2017. A Cubical Type Theory for Higher Inductive Types. (2017). http://www.cse.chalmers.se/~simonhu/misc/hcomp.pdf

Simon Huber. 2019. Canonicity for Cubical Type Theory. *J. Autom. Reason.* 63, 2 (2019), 173–210. https://doi.org/10.1007/s10817-018-9469-1

Pierre Letouzey. 2003. A New Extraction for Coq. In *Types for Proofs and Programs, International Workshop, TYPES 2002*. 200–219. https://doi.org/10.1007/3-540-39185-1_12

Conor McBride. 2016. I Got Plenty o' Nuttin'. In *A List of Successes That Can Change the World*. 207–233. https://doi.org/10.1007/978-3-319-30936-1_12

Nathan Mishra-Linger and Tim Sheard. 2008. Erasure and Polymorphism in Pure Type Systems. In *Foundations of Software Science and Computational Structures, 11th International Conference, FOSSACS 2008*. 350–364. https://doi.org/10.1007/978-3-540-78499-9_25

Richard Nathan Mishra-Linger. 2008. *Irrelevance, Polymorphism, and Erasure in Type Theory.* Ph.D. Dissertation. Portland State University. https://doi.org/10.15760/etd.2669

Christine Paulin-Mohring. 1989. Extracting F_w's Programs from Proofs in the Calculus of Constructions. In *POPL '89 Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. 89–104. https://doi.org/10.1145/75277.75285

Christine Paulin-Mohring and Benjamin Werner. 1993. Synthesis of ML programs in the system Coq. *Journal of Symbolic Computation* 15, 5–6 (1993), 607–640. https://doi.org/10.1016/S0747-7171(06)80007-6

Mike Shulman. 2015. Not every weakly constant function is conditionally constant. Blog post. Retrieved 2021-07-07 from https://homotopytypetheory.org/2015/06/11/not-every-weakly-constant-function-is-conditionally-constant/

Vilhelm Sjöberg. 2015. *A dependently typed language with nontermination.* Ph.D. Dissertation. University of Pennsylvania. https://repository.upenn.edu/dissertations/AAI3709556

Jonathan Sterling and Carlo Angiuli. 2021. Normalization for Cubical Type Theory. arXiv:2101.11479v1 [cs.LO] Accepted for publication in LICS 2021, the 36th Annual Symposium on Logic in Computer Science.

Nicolas Tabareau, Éric Tanter, and Matthieu Sozeau. 2021. The Marriage of Univalence and Parametricity. *J. ACM* 68, 1 (2021), 5:1–5:44. https://doi.org/10.1145/3429979

Matúš Tejiščák. 2020. A Dependently Typed Calculus with Pattern Matching and Erasure Inference. *Proceedings of the ACM on Programming Languages* 4, ICFP (2020), 91:1–91:29. https://doi.org/10.1145/3408973

The Agda Team. 2021. *Agda User Manual, Release 2.6.2.* https://agda.readthedocs.io/_/downloads/en/v2.6.2/pdf/

The Univalent Foundations Program. 2013. *Homotopy Type Theory* (1 ed.). https://homotopytypetheory.org/book/

Floris van Doorn. 2016. Constructing the Propositional Truncation using Non-recursive HITs. In *CPP'16, Proceedings of the 5th ACM SIGPLAN Conference on Certified Programs and Proofs*. 122–129. https://doi.org/10.1145/2854065.2854076

Twan van Laarhoven. 2011. Isomorphism lenses. Blog post. Retrieved 2021-07-07 from https://www.twanvl.nl/blog/haskell/isomorphism-lenses

Femke van Raamsdonk and Paula Severi. 2002. Eliminating Proofs from Programs. *Electronic Notes in Theoretical Computer Science* 70, 2 (2002), 42–59. https://doi.org/10.1016/S1571-0661(04)80505-X

Andrea Vezzosi, Anders Mörtberg, and Andreas Abel. 2019. Cubical Agda: A Dependently Typed Programming Language with Univalence and Higher Inductive Types. *Proceedings of the ACM on Programming Languages* 3, ICFP (2019), 87:1–87:29. https://doi.org/10.1145/3341691

Stephanie Weirich, Antoine Voizard, Pedro Henrique Azevedo de Amorim, and Richard Eisenberg. 2017. A Specification for Dependent Types in Haskell. *Proceedings of the ACM on Programming Languages* 1, ICFP (2017), 31:1–31:30. https://doi.org/10.1145/3110275