Extensionality of Ghost Dependent Types for Free

Théo Winterhalter

Extensionality of Ghost Dependent Types for Free

THÉO WINTERHALTER, Inria Saclay, France

We introduce ghost type theory (GTT) a dependent type theory extended with a new universe for ghost data that can safely be erased when running a program but which is not proof irrelevant like with a universe of (strict) propositions. Instead, ghost data carry information that can be used in proofs or to discard impossible cases in relevant computations. Casts can be used to replace ghost values by others that are propositionally equal, but crucially these casts can safely be ignored for conversion. We give a syntactical model of GTT using a program translation akin to the parametricity translation and thus show consistency of the theory. We further extend GTT to support equality reflection and show that we can eliminate its use without the need for the usual extra axioms of function extensionality and uniqueness of identity proofs. In particular we validate the intuition that indices of inductive types—such as the length index of vectors—do not matter for computation and can safely be considered modulo theory.

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

Additional Key Words and Phrases: dependent types, termination, consistency, extensionality

ACM Reference Format:
Théo Winterhalter. 2023. Extensionality of Ghost Dependent Types for Free. 1, 1 (July 2023), 29 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Dependent type theory—the underlying theory of numerous proof assistants such as Agda [Norell 2007] or Coq [Coq development team 2023]—typically feature two notions of equality: definitional equality that is external and determines which types are the same; and propositional equality which is internal and which allows the user to perform equational reasoning.

Definitional equality is crucial in practice as it makes the life of the user easier by identifying objects on the nose. For instance, it allows one to consider the type of lists of length $3 + 2$—usually written $\mathsf{vec}\ A\ (3 + 2)$—as being the same as the type of those of length 5, *i.e.* $\mathsf{vec}\ A\ 5$. This in turn makes it possible to state and prove a propositional equality between $[1; 2; 3; 4; 5]$ and the concatenation of $[1; 2; 3]$ and $[4; 5]$. That said, be it in Agda or in Coq, this notion of definitional equality is limited in order to remain decidable. A typical instance is that while $0 + n$ and n are identified, it is not the case in general for $n + 0$ to be equal to n. This can be somewhat alleviated by extending the proof assistant with rewrite rules [Cockx and Abel 2016; Cockx et al. 2021] but then we are still stuck when comparing $n + m$ and $m + n$. A remedy was once proposed in the form of Coq modulo theory (CoqMT) by Strub [2010] but this process cannot be complete while remaining decidable. As such, other proof assistants such as F* [Swamy et al. 2016], Andromeda 1 [Bauer et al. 2016] or NuPRL [Constable and Bates 2014] embrace undecidability and implement extensional type theories which make propositional and definitional equalities coincide, a phenomenon called equality reflection.

Extensional type theory (ETT) has been well studied and Hofmann [1995] showed it was conservative over intensional type theory (ITT, *i.e.* without equality reflection) extended with two equality principles: uniqueness of identity proofs (UIP), which equates any two proofs of the same propositional equality; and function extensionality (funext) which allows one to equate two functions as long as they are pointwise equal. This conservativity result was later made constructive

Author's address: Théo Winterhalter, theo.winterhalter@inria.fr, Inria Saclay, 1 Rue Honoré d'Estienne d'Orves, Palaiseau, France, 91120.

2023. XXXX-XXXX/2023/7-ART
https://doi.org/10.1145/nnnnnnn.nnnnnnn

and turned into an effective translation [Oury 2005; Winterhalter et al. 2019]. The need for the UIP axiom is at odds with homotopy type theory and in particular the univalence axiom as they are inconsistent together. The two can still be made to coexist in a single type theory using so-called two-level type theories as proposed by Altenkirch et al. [2016]. Thankfully, the translation proposed by Winterhalter et al. [2019] is general enough to apply to a two-level setting. Still it would be interesting to delineate a subsystem of ETT that does not require the use of funext and UIP so as to remain as agnostic as possible.

Furthermore, even though the lack of decidable type checking does not prevent systems such as F* to be usable in practice—through the use of SMT automation in the case of F*—the equality reflection rule has many drawbacks. First, in an inconsistent context, anything goes: since all equalities are provable, then all terms have all types; for instance, assuming $D = D \to D$ is enough to encode untyped λ-calculus and thus non-terminating functions. Second, even under consistent contexts one can construct nonsensical terms; for instance, assuming $\mathsf{nat} \to \mathsf{nat} = \mathsf{nat} \to \mathsf{bool}$ (validated by the cardinal model proposed by Bauer and Winterhalter [2020]), one can write the identity function for natural numbers $\lambda x.x$ and give it type $\mathsf{nat} \to \mathsf{bool}$ so that $(\lambda x.x)\ 0$ is of type bool when it should be equal to 0. The solution to this problem is to annotate both function abstraction and application by their domain and codomain and restrict β-reduction to the cases where they match individually, essentially bypassing the lack of injectivity of Π-types (which would in particular say that $\mathsf{nat} = \mathsf{bool}$ follows from $\mathsf{nat} \to \mathsf{nat} = \mathsf{nat} \to \mathsf{bool}$, a fact that usually holds in ITT) but as a result terms become exponential in size compared to their ITT counterparts. Third, for similar reasons, ETT has no power to discriminate between different type formers, there is no way for the type checker to conclude that a definitional equality between *e.g.* nat and $\mathsf{nat} \to \mathsf{nat}$ is not possible and as such it cannot terminate early and report an error. All these points show the difficulty for a checker to give useful feedback and error messages to the user. They also raise problems about extraction of programs written in ETT. In fact ETT, has no concrete evaluation semantics.

We propose a restriction of ETT that remedies several[1] of these drawbacks by allowing equality reflection for ghost values, *i.e.* those that can be erased at extraction. This extends the definitional proof irrelevance of Prop, introduced by Gilbert et al. [2019], which identifies all proofs of the same proposition. We argue (informally) that ghost expressions are precisely those for which a user should wish to use reflection in the first place. Let us illustrate this intuition by looking at the example of vectors—or length-indexed lists—introduced above. In order to write the reversal of vectors using an accumulator (to obtain a tail-recursive version) we have to consider both the length of the list to be reversed and that of the accumulator to express the length of the result. As such one would write the following:

$$
\begin{aligned}
&\mathsf{rev} : \forall\ n\ m.\ \mathsf{vec}\ A\ n \to \mathsf{vec}\ A\ m \to \mathsf{vec}\ A\ (n + m)\\
&\mathsf{rev}\ \ 0\ m\ \ \mathsf{vnil}\ \ acc := acc\\
&\mathsf{rev}\ (\mathsf{S}\ \ k)\ m\ (\mathsf{vcons}\ \ a\ k\ v)\ acc := \mathsf{rev}\ \ k\ \ (\mathsf{S}\ \ m)\ v\ \ (\mathsf{vcons}\ \ a\ m\ acc)
\end{aligned}
$$

While this definition is perfectly fine for regular lists, it suffers from two issues:

(1) The second clause of the pattern matching is not well typed in ITT as the recursive call is of type $\mathsf{vec}\ \ A\ \ (k + \mathsf{S}\ \ m)$ instead of the expected $\mathsf{vec}\ \ A\ \ (\mathsf{S}\ \ k + m)$ which is only propositionally equal to it. In ITT one would need to use a transport along said equality to make the term type check, at the cost of polluting the produced term. In ETT the definition would be accepted as

[1]We in fact conjecture that all of these drawbacks are remedied by our proposal but leave the question of evaluation to future work.

is by reflecting that same equality, but running the translation of Winterhalter et al. [2019] on it would also result in a transport.

(2) n and m appear in the function definition and will thus remain in the extracted program, even if they are essentially there for typing purposes and do not intervene in the computation. One could argue that extraction should figure it out, but bear in mind that there is nothing preventing the user from using n and m in a meaningful way. The same goes for the natural numbers contained in the vector (k in vcons a k v).

We thus argue that instead of taking n and m to be of type nat, they should instead be of type erased nat. The erased modality indicates that its values should be considered ghost and cannot therefore be used in a way that is relevant for computation. It is nevertheless different from the squash ‖nat‖ which identifies all its elements (meaning ‖nat‖ is equivalent to $\top$, the true proposition type with exactly one inhabitant). In particular, we are still able to distinguish different erased values such as erase 0 and erase (S 0); which is needed to write the function that takes the head of a vector whose length it not erase 0.

The rev function is then defined using extensionality for erased values. This can take two forms: either by using ghost equality reflection or by using ghost casts that are irrelevant for computation and for conversion. The intuition is that ghost casts can only go from $P\ u$ to $P\ v$ when the argument to predicate P is itself ghost, in other words when $P\ u$ and $P\ v$ both erase to the same type.

We in fact take inspiration from this intuition coming from erasure and extraction to build the model of our ghost type theory with casts by adapting a translation of Pédrot and Tabareau [2018] that handles exceptions in dependent type theory: all ghost and proof informations are erased and inaccessible branches locally correspond to raised exceptions; then a parametricity translation ensures that these exceptions are never actually raised.

Outline of the paper. In Section 2 we introduce GTT, a dependent type theory with a universe of ghost types along with ghost casts. Then in Section 3 we give a model of GTT by means of three translations accounting for the three levels of relevance (for propositions, ghost types and regular types). We derive consistency and type former discrimination from it in Section 4. We then show how to extend both GTT and its model with inductive types such as vectors in Section 5, allowing us to show some concrete examples. Finally we give in Section 6 a definition of GRTT, a version of GTT with ghost reflection instead of casts and show how we can adapt (and simplify!) the proof of Oury [2005]; Winterhalter et al. [2019] to translate GRTT to GTT.

2 GHOST TYPE THEORY

Ghost type theory (GTT) is essentially a version of Martin-Löf type theory (MLTT) with a universe of definitional proof-irrelevant propositions Prop [Gilbert et al. 2019] to which we add a hierarchy of universes of ghost types written Ghost_i. The only way to construct a ghost type is through the erased type former and the inhabitants of erased A are obtained from values v of type A, written erase v. One can also eliminate erased values thanks to the eliminator reveal; of course erased values may not be used to produce relevant information, only ghost or propositional. Note that we follow F* notation [Swamy et al. 2016][2] for those. Equality of ghost expressions $u, v : A$ is a proposition written $u \approx_A v$, which is reflexive (as shown by gh-refl) and which is eliminated through the cast operator.

[2]Our reveal is an eliminator rather than a projection however.

2.1 Syntax of GTT

The syntax of GTT is mostly standard and described below.

$$
\begin{array}{rcll}
\Gamma, \Delta & ::= & \bullet \mid \Gamma, x^{\mathfrak{s}} : A & \\
t, u, A, B & ::= & x^{\mathfrak{s}} \mid \mathsf{Kind}_i \mid \mathsf{Type}_i \mid \mathsf{Ghost}_i \mid \mathsf{Prop} & (i \in \mathbb{N}) \\
& \mid & \forall^{\mathfrak{r}}(x^{\mathfrak{s}} : A).B \mid \lambda(x^{\mathfrak{s}} : A).t \mid t\ u & \\
& \mid & \mathsf{erased}\ A \mid \mathsf{erase}\ t \mid \mathsf{reveal}(t, P, p) \mid \mathsf{reveal}_*(t, p) & \\
& \mid & u \approx_A v \mid \mathsf{gh\text{-}refl}_A\ u \mid \mathsf{cast}(e, P, t) & \\
& \mid & \bot \mid \mathsf{exfalso}_{\mathfrak{s}}(A, p) & \\
\mathfrak{r}, \mathfrak{s} & ::= & \mathbb{K} \mid \mathbb{T} \mid \mathbb{G} \mid \mathbb{P} &
\end{array}
$$

Several things might appear surprising so we explain them one by one.

First, notice how, beyond Ghost and Prop, we have two universe hierarchies, Kind and Type. The main reason for this is that for our model, we need the parametricity translation to produce propositions, even on Type, which would be ill typed if we had $\mathsf{Type}_i : \mathsf{Type}_{i+1}$ and we thus implement the solution of Keller and Lasson [2012] of separating regular types from universes and thus having $\mathsf{Type}_i : \mathsf{Kind}_{i+1}$ but also $\mathsf{Prop} : \mathsf{Kind}$ and $\mathsf{Ghost}_i : \mathsf{Kind}_{i+1}$.

A related observation is that we also have extra annotations on binders as well as on variables and on the eliminator for the empty type $\bot$. These *modes* have to be $\mathbb{K}, \mathbb{T}, \mathbb{G}$ or $\mathbb{P}$ and respectively correspond to Kind, Type, Ghost and Prop without the universe levels. Their presence on variables may appear be redundant with the binders / context. This is mainly a technicality for us to build the model without having to explicitly refer to contexts lookups. We will omit them when they are not important and can be inferred from the context in order to improve readability.

Finally, one might notice how we duplicate the eliminator for erased into reveal and reveal_*. The idea is that reveal can only be used in mode $\mathbb{P}$ or $\mathbb{G}$ since it *reveals* relevant content that is supposed to be erased. There is however the need to be able to construct propositions from erased values in order to be able to discriminate *e.g.* $\mathsf{erase}\ 0$ and $\mathsf{erase}\ 1$.

As usual, we write $A \to B$ for $\forall^{\mathfrak{r}}(x^{\mathfrak{s}} : A).B$ when B does not depend on A and when the modes are inferable from the context.

2.2 Cast-free syntax

We want to ensure that ghost casts do not get in the way of equality. Consider for instance the expression $\mathsf{cast}(e, \lambda x.\, \mathsf{nat}, 0)$ which is to be understood as taking some ghost equality $e : u \approx v$ to be able to cast 0 of type $(\lambda x.\, \mathsf{nat})\ u$ to type $(\lambda x.\, \mathsf{nat})\ v$. As both sides are in fact nat, we would like to be able to state that this is equal to 0. Even more so, we would like to conclude that $\mathsf{cast}(e, \lambda x.\, \mathsf{nat}, 0) + n$ is equal to n. The problem is not limited to cases where the cast is essentially doing nothing, consider the cast of a λ-abstraction: $\mathsf{cast}(e, \lambda x.A\ x \to B\ x, \lambda(y : A\ u).\ t)$, where $e : u \approx v$, thus mapping the function of type $A\ u \to B\ u$ to $A\ v \to B\ v$. We would like to apply this function and have it compute, in other words we would like it be a λ-abstraction itself, typically by casting inside the body of the function to move the argument from $A\ v$ to $A\ u$ and the result from $B\ u$ to $B\ v$:

$$\lambda(z : A\ v).\ \mathsf{cast}(e, B, t[y := \mathsf{cast}(e^{-1}, A, z)])$$

where $e^{-1} : v \approx u$ is just symmetry applied to e.

To achieve this we introduce an operator $|-|$ that removes all casts from a term (or a context). We are going to use this cast removal inside conversion to obtain the desired equalities we mentioned above. The model is going to justify this approach. The definition of $|-|$ is straightforward so we only show a few examples, the full definition can be found in Figure 15 in Appendix B.

$$|\mathsf{cast}(e, P, t)| := |t| \qquad |t\ u| := |t|\ |u| \qquad |\forall^{\mathfrak{r}}(x^{\mathfrak{s}} : A).B| := \forall^{\mathfrak{r}}(x^{\mathfrak{s}} : |A|).|B|$$

2.3 Mode of a term

Similarly to Gilbert et al. [2019, Section 4.5] we want to distinguish relevant and irrelevant computations syntactically, and the same goes for ghost computations. Like for them, it allows us to define conversion syntactically, while handling a priori semantic rules like proof irrelevance. For us it also fills another role as the definition of the translations of Section 3 uses this information. We thus define a function which determines the *mode* of a term, *i.e.* whether its type lives in Prop, Ghost, Type or Kind. It is connected to a similar function $\lfloor s \rfloor$, which returns the sort class of a sort, by the idea that when $t : A : s$ then $\mathrm{md}(t) = \lfloor s \rfloor$. This fact will be established after we give the model (Lemma 4.3).

$$\mathrm{md}(x^{\mathfrak{s}}) := \mathfrak{s} \qquad \mathrm{md}(\mathrm{Kind}_i) := \mathbb{K} \qquad \mathrm{md}(\mathrm{Type}) := \mathbb{K} \qquad \mathrm{md}(\mathrm{Ghost}) := \mathbb{K} \qquad \mathrm{md}(\mathrm{Prop}) := \mathbb{K}$$

$$\mathrm{md}(\forall^{\mathfrak{r}}(x^{\mathfrak{s}} : A).B) := \mathbb{K} \qquad \mathrm{md}(\lambda(x^{\mathfrak{s}} : A).t) := \mathrm{md}(t) \qquad \mathrm{md}(t\ u) := \mathrm{md}(t)$$

$$\mathrm{md}(\mathrm{erased}\ A) := \mathbb{K} \qquad \mathrm{md}(\mathrm{erase}\ t) := \mathbb{G} \qquad \mathrm{md}(\mathrm{reveal}(t, P, p)) := \mathrm{if}\ \mathrm{md}(p) = \mathbb{G}\ \mathrm{then}\ \mathbb{G}\ \mathrm{else}\ \mathbb{P}$$

$$\mathrm{md}(\mathrm{reveal}_*(t, p)) := \mathbb{K} \qquad \mathrm{md}(u \approx_A v) := \mathbb{K} \qquad \mathrm{md}(\text{gh-refl}_A\ u) := \mathbb{P}$$

$$\mathrm{md}(\mathrm{cast}(e, P, t)) := \mathrm{md}(t) \qquad \mathrm{md}(\bot) := \mathbb{K} \qquad \mathrm{md}(\mathrm{exfalso}_{\mathfrak{s}}(A, p)) := \mathfrak{s}$$

$$\lfloor \mathrm{Kind} \rfloor := \mathbb{K} \qquad \lfloor \mathrm{Type} \rfloor := \mathbb{T} \qquad \lfloor \mathrm{Ghost} \rfloor := \mathbb{G} \qquad \lfloor \mathrm{Prop} \rfloor := \mathbb{P}$$

2.4 Typing rules of GTT

GTT features three kinds of judgements: $\vdash \Gamma$ (context formation), $\Gamma \vdash t : A$ (typing), and $u \equiv v$ (definitional equality, or conversion). Context formation and typing rules are given in Figure 1, while an excerpt of definitional equality rules is given in Figure 2. We omit the structural and congruence rules and instead focus on computation rules, proof irrelevance, and on the *recasting rule* which exploits the cast removal of Section 2.2; the remaining rules are found in Appendix A. We do not include η-rules or cumulativity for now as we believe them to be mostly orthogonal to the problem at hand.

Universes. In order to factorise the different rules involving universes we define $s \cdot r$ denoting the universe of a dependent function type whose domain lives in universe s and codomain in r.

$$s \cdot \mathrm{Prop} := \mathrm{Prop} \qquad \mathrm{Prop} \cdot s := s \qquad \mathrm{Ghost}_i \cdot \mathrm{Ghost}_j := \mathrm{Ghost}_{\max(i,j)}$$

$$\mathrm{Type}_i \cdot \mathrm{Ghost}_j := \mathrm{Ghost}_{\max(i,j)} \qquad \mathrm{Kind}_i \cdot \mathrm{Ghost}_j := \mathrm{Ghost}_{\max(i,j)}$$

$$\mathrm{Ghost}_i \cdot \mathrm{Type}_j := \mathrm{Type}_{\max(i,j)} \qquad \mathrm{Type}_i \cdot \mathrm{Type}_j := \mathrm{Type}_{\max(i,j)} \qquad \mathrm{Ghost}_i \cdot \mathrm{Kind}_j := \mathrm{Kind}_{\max(i,j)}$$

$$\mathrm{Type}_i \cdot \mathrm{Kind}_j := \mathrm{Kind}_{\max(i,j)} \qquad \mathrm{Kind}_i \cdot \mathrm{Kind}_j := \mathrm{Kind}_{\max(i,j)}$$

Notice how the first rule corresponds to impredicativity of Prop and more generally how the sort on the right always determines the sort class of the whole expression.

Ghost-specific rules. You can see how reveal is used to produce proofs (of a proposition) or ghost values, but also how reveal_* extends this to producing propositions directly. We shall see in Section 5.1 how it is useful to discriminate constructors of an erased inductive type which is a specificity of ghost types. We highlight the rules regarding casts in blue. Note how we cannot use cast in sort Kind as our model does not support it. The recasting rule says that we can change u for any v that is *syntactically* equal ($=_\alpha$) to it after removal of casts. Note that it subsumes any

computation rule for casts. In fact, we could have equivalently considered conversion only on cast-free syntax and instead have $|A| \equiv |B|$ as a premise to the conversion rule which would probably correspond to what an implementation would do. We choose to keep it this way to better highlight the difference between GTT and GRTT which we will introduce in Section 6.

Redundant information. In some rules, one can see extra premises highlighted in gray. They are redundant, and we will show later (in Theorem 4.4) that they can be safely removed. To do so, we need some knowledge that will come from the model, so we require them to help in establishing said model. Note that not all premises talking about mode are actually redundant: those in conversion are in fact needed, and essentially represent a lightweight way of determining whether the proof irrelevance or recasting rules apply, similar to the theory presented by Gilbert et al. [2019, Section 4.5] for the implementation of proof irrelevance in Coq. There is also one mode premise for the conversion rule that does not seem to follow from any of the other hypotheses and it would be interesting to investigate whether this constraint can be lifted although it should not matter much in practice.

Substitutions. We write $t[x^{\mathfrak{s}} := u]$ for the substitution of $x^{\mathfrak{s}}$ by u in term t. For our purposes we will assume that a same term will not contain both *e.g.* $x^{\mathbb{G}}$ and $x^{\mathbb{T}}$ and we will thus omit the relevance annotation of variables when it is clear from the context.

2.5 Preliminary results

Without the model we can already establish various lemmas, in particular lemmas that connect semantic information (typing and conversion) and syntactic information (modes).

Lemma 2.1. *Substitution preserves modes: if* $\mathrm{md}(u) = \mathfrak{s}$ *then* $\mathrm{md}(t[x^{\mathfrak{s}}] := u) = \mathrm{md}(t)$.

Lemma 2.2. *Cast removal preserves modes: if* $|u| =_\alpha |v|$ *then* $\mathrm{md}(u) = \mathrm{md}(v)$.

Lemma 2.3. *Cast removal commutes with substitution:* $|t[x^{\mathfrak{s}} := u]| =_\alpha |t|[x^{\mathfrak{s}} := |u|]$.

Lemma 2.4. *Conversion entails mode equality: if* $\Gamma \vdash u \equiv v$ *then* $\mathrm{md}(u) = \mathrm{md}(v)$.

Proof. The proof is simple, by induction on the derivation, using Lemmas 2.1 and 2.2. □

We also state the usual substitution lemma for GTT, as well as a uniqueness of type theorem. Note that the latter would rather be a principle type lemma if we were to handle cumulativity.

Lemma 2.5 (Substitution). *If* $\Gamma, x^{\mathfrak{s}} : A \vdash t : B$ *and* $\Gamma \vdash u : A$ *with* $\mathrm{md}(u) = \mathfrak{s}$ *then we have* $\Gamma \vdash t[x^{\mathfrak{s}} := u] : A[x^{\mathfrak{s}} := u]$.

Proof. The proof is standard and requires generalising to $\Gamma, x^{\mathfrak{s}} : A, \Delta \vdash t : B$. □

Lemma 2.6 (Uniqueness of type). *If* $\Gamma \vdash t : A$ *and* $\Gamma \vdash t : B$ *then* $A \equiv B$.

Proof. The idea is to do a case analysis on t and perform inversion of typing for both hypotheses and to recursively build a conversion proof, which is also standard. □

We also show a variant of *validity* (sometimes called *presupposition*) that states that types are themselves well typed, with some additional information extracted from modes. Note how this lemma requires the context to be well formed as well since GTT typing derivations do not contain context formation assumptions.

Lemma 2.7 (Validity). *If* $\vdash \Gamma$ *and* $\Gamma \vdash t : A$ *then there exists a sort* s *such that* $\Gamma \vdash A : s$ *and* $\lfloor s \rfloor = \mathrm{md}(t)$.

$$\frac{}{\vdash \bullet} \qquad \frac{\vdash \Gamma \qquad \Gamma \vdash A : s}{\vdash \Gamma, x^{\lfloor s \rfloor} : A} \qquad \frac{(x^{\mathfrak{s}} : A) \in \Gamma}{\Gamma \vdash x^{\mathfrak{s}} : A}$$

$$\frac{\mathrm{md}(B) = \mathbb{K} \qquad \mathrm{md}(t) = \lfloor s \rfloor \qquad \Gamma \vdash t : A \qquad A \equiv B \qquad \Gamma \vdash B : s}{\Gamma \vdash t : B}$$

$$\frac{}{\Gamma \vdash \mathrm{Prop} : \mathrm{Kind}_i} \qquad \frac{i < j}{\Gamma \vdash \mathrm{Ghost}_i : \mathrm{Kind}_j} \qquad \frac{i < j}{\Gamma \vdash \mathrm{Type}_i : \mathrm{Kind}_j} \qquad \frac{i < j}{\Gamma \vdash \mathrm{Kind}_i : \mathrm{Kind}_j}$$

$$\frac{\mathrm{md}(A) = \mathrm{md}(B) = \mathbb{K} \qquad \Gamma \vdash A : s \qquad \Gamma, x^{\lfloor s \rfloor} : A \vdash B : r}{\Gamma \vdash \forall^{\lfloor r \rfloor}(x^{\lfloor s \rfloor} : A).B : s \cdot r}$$

$$\frac{\mathrm{md}(A) = \mathbb{K} \qquad \mathrm{md}(t) = \lfloor r \rfloor \qquad \Gamma \vdash A : s \qquad \Gamma, x^{\lfloor s \rfloor} : A \vdash B : r \qquad \Gamma, x^{\lfloor s \rfloor} : A \vdash t : B}{\Gamma \vdash \lambda(x^{\lfloor s \rfloor} : A).t : \forall^{\lfloor r \rfloor}(x^{\lfloor s \rfloor} : A).B} \qquad \frac{\mathrm{md}(t) = \mathfrak{r} \qquad \mathrm{md}(u) = \mathfrak{s} \qquad \Gamma \vdash t : \forall^{\mathfrak{r}}(x^{\mathfrak{s}} : A).B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\ u : B[x^{\mathfrak{s}} := u]}$$

$$\frac{}{\Gamma \vdash \bot : \mathrm{Prop}} \qquad \frac{\mathrm{md}(A) = \mathbb{K} \qquad \mathrm{md}(p) = \mathbb{P} \qquad \Gamma \vdash A : s \qquad \Gamma \vdash p : \bot}{\Gamma \vdash \mathrm{exfalso}_{\lfloor s \rfloor}(A, p) : A}$$

$$\frac{\mathrm{md}(A) = \mathbb{K} \qquad \Gamma \vdash A : \mathrm{Type}_i}{\Gamma \vdash \mathrm{erased}\ A : \mathrm{Ghost}_i} \qquad \frac{\mathrm{md}(A) = \mathbb{K} \qquad \mathrm{md}(t) = \mathbb{T} \qquad \Gamma \vdash A : \mathrm{Type} \qquad \Gamma \vdash t : A}{\Gamma \vdash \mathrm{erase}\ t : \mathrm{erased}\ A}$$

$$\frac{\mathrm{md}(p) = \lfloor s \rfloor \qquad \mathrm{md}(t) = \mathbb{G} \qquad \mathrm{md}(P) = \mathbb{K} \qquad \Gamma \vdash t : \mathrm{erased}\ A \qquad \Gamma \vdash P : \mathrm{erased}\ A \to s \qquad \Gamma \vdash p : \forall^{\lfloor s \rfloor} x^{\mathbb{G}}.\ P\ (\mathrm{erase}\ x^{\mathbb{G}}) \qquad \lfloor s \rfloor \in \{\mathbb{P}, \mathbb{G}\}}{\Gamma \vdash \mathrm{reveal}(t, P, p) : P\ t}$$

$$\frac{\mathrm{md}(t) = \mathbb{G} \qquad \mathrm{md}(p) = \mathbb{K} \qquad \Gamma \vdash t : \mathrm{erased}\ A \qquad \Gamma \vdash p : A \to \mathrm{Prop}}{\Gamma \vdash \mathrm{reveal}_*(t, p) : \mathrm{Prop}}$$

$$\frac{\mathrm{md}(A) = \mathbb{K} \qquad \mathrm{md}(u) = \mathrm{md}(v) = \mathbb{G} \qquad \Gamma \vdash A : \mathrm{Ghost} \qquad \Gamma \vdash u : A \qquad \Gamma \vdash v : A}{\Gamma \vdash u \approx_A v : \mathrm{Prop}} \qquad \frac{\mathrm{md}(A) = \mathbb{K} \qquad \mathrm{md}(u) = \mathbb{G} \qquad \Gamma \vdash A : \mathrm{Ghost} \qquad \Gamma \vdash u : A}{\Gamma \vdash \mathrm{gh\text{-}refl}_A\ u : u \approx_A u}$$

$$\frac{\begin{array}{c}\mathrm{md}(A) = \mathrm{md}(P) = \mathbb{K} \qquad \mathrm{md}(u) = \mathrm{md}(v) = \mathbb{G} \qquad \mathrm{md}(t) = \lfloor s \rfloor \qquad \mathrm{md}(e) = \mathbb{P} \\ \Gamma \vdash A : \mathrm{Ghost} \qquad \Gamma \vdash u : A \qquad \Gamma \vdash v : A \\ \Gamma \vdash e : u \approx_A v \qquad \Gamma \vdash P : A \to s \qquad \Gamma \vdash t : P\ u \qquad \lfloor s \rfloor \neq \mathbb{K}\end{array}}{\Gamma \vdash \mathrm{cast}(e, P, t) : P\ v}$$

Fig. 1. Typing rules of GTT

$$\frac{\mathrm{md}(p) = \mathrm{md}(q) = \mathbb{P}}{p \equiv q} \qquad \frac{|u| =_\alpha |v|}{u \equiv v} \qquad \frac{\mathrm{md}(u) = \mathfrak{s}}{(\lambda(x^{\mathfrak{s}} : A).t)\ u \equiv t[x^{\mathfrak{s}} := u]}$$

$$\frac{\mathrm{md}(p) \in \{\mathbb{P}, \mathbb{G}\} \qquad \mathrm{md}(t) \in \{\mathbb{T}, \mathbb{K}\}}{\mathrm{reveal}(\mathrm{erase}\ \ t, P, p) \equiv p\ t} \qquad \frac{\mathrm{md}(p) = \mathbb{K} \qquad \mathrm{md}(t) = \mathbb{T}}{\mathrm{reveal}_*(\mathrm{erase}\ \ t, p) \equiv p\ t}$$

Fig. 2. Conversion for GTT (excerpt)

Proof. The proof is by induction on the derivation of $\Gamma \vdash t : A$ where we remark that the conversion rule trivially verifies the wanted property and where the $\vdash \Gamma$ hypothesis is used for the variable typing case. The trickiest case is perhaps the application rule. By induction hypothesis we have some r' such that $\Gamma \vdash \forall^{\mathfrak{s}}(x^{\mathfrak{s}} : A).B : r'$ and $\lfloor r' \rfloor = \mathrm{md}(t)$. By inversion of typing, we thus know that $\Gamma \vdash A : s$ and $\Gamma, x^{\mathfrak{s}} : A \vdash B : r$ for some s and r such that $\lfloor s \rfloor = \mathfrak{s}$ and $\lfloor r \rfloor = \mathfrak{r}$. Since we have $\Gamma \vdash u : A$ with $\mathrm{md}(u) = \mathfrak{s}$ we can use Lemma 2.5 to conclude $\Gamma \vdash B[x^{\mathfrak{s}} := u] : r$. And we indeed have $\mathrm{md}(t\ u) = \mathrm{md}(t) = \mathfrak{r}$.

□

3 MODEL OF GTT

We now give a syntactical model [Boulier et al. 2017] of GTT by adapting the proof of Pédrot and Tabareau [2018]. Their proof is about interpreting a type theory with exceptions so it may come as surprising that it applies to a very different topic. As we hinted at in the introduction, the connection comes from the fact that exfalso can be used to produce relevant terms. But for their erasure we cannot rely on the proof of $\bot$ any more since it will be erased. The corresponding erasure of exfalso is thus a form of raised exception.

Pédrot and Tabareau perform two translations: one that maps all types to pointed types (to interpret the exceptions); together with a parametricity translation that shows the first translation produces *reasonable* terms, *i.e.* terms that do not raise exceptions at top-level. We follow them, but introduce a third translation in-between to deal with ghost values.

Target. The target of these translations is a now standard variant of MLTT with a universe of definitional proof-irrelevant propositions and inductive types. We take the presentation of Section 4.5 of the theory presented by Gilbert et al. [2019], meaning one can think of the target as either Agda or Coq. Essentially, it is a variant of GTT presented in Section 2 with a few differences: (1) Kind and Type are collapsed and there is neither a Ghost universe nor the associated constructions; (2) annotations are limited to $\mathbb{P}$ and $\mathbb{T}$ and they correspond to the Relevant and Irrelevant annotations of Gilbert et al. [2019]; (3) extra inductive types are defined, we will detail them as we need them.

3.1 Erasure

We now define erasure which removes all proofs (of propositions) and ghost values from terms while preserving typing. Essentially a term $t : A$ is erased to $[t]_\varepsilon : [\![A]\!]_\varepsilon$ and we have a distinguished inhabitant for each translated type $[A]_\emptyset : [\![A]\!]_\varepsilon$ (the exception).

To perform the translation, we require three inductive types in the target: the true proposition $\top$: Prop with unique inhabitant $\star : \top$; the unit type with inhabitant () : unit; and a data type for representing universes ty_i for each i. ty has one constructor to embed pointed types $\mathrm{tyval} : \forall(A : \mathrm{Type}_i)\ (a : A).\ \mathrm{ty}_i$, and an error constructor $\mathrm{ty}_\emptyset : \mathrm{ty}_i$ to make ty_i itself a pointed type. Using its eliminator we can define two *projections* $\mathrm{El}_i : \mathrm{ty}_i \to \mathrm{Type}_i$ and $\mathrm{Err}_i : \forall(T : \mathrm{ty}_i).\ \mathrm{El}_i\ T$ such

$$
\begin{array}{lcl}
[x^{\mathbb{K}/\mathbb{T}}]_\varepsilon & := & x^{\mathbb{T}} \\
[\mathrm{Kind}_i]_\varepsilon & := & \mathrm{tyval}\ \mathrm{ty}_i\ \mathrm{ty}_\emptyset \\
[\mathrm{Type}_i]_\varepsilon & := & \mathrm{tyval}\ \mathrm{ty}_i\ \mathrm{ty}_\emptyset \\
[\mathrm{Ghost}_i]_\varepsilon & := & \mathrm{tyval}\ \mathrm{ty}_i\ \mathrm{ty}_\emptyset \\
[\mathrm{Prop}]_\varepsilon & := & \mathrm{tyval}\ \top\ \star \\
[\forall^{\mathbb{K}/\mathbb{T}}(x^{\mathbb{K}/\mathbb{T}} : A).B]_\varepsilon & := & \mathrm{tyval}\ (\forall(x : [\![A]\!]_\varepsilon).[\![B]\!]_\varepsilon)\ (\lambda x.[B]_\emptyset) \\
[\forall^{\mathbb{K}/\mathbb{T}}(x^{\mathbb{G}/\mathbb{P}} : A).B]_\varepsilon & := & [B]_\varepsilon \\
[\forall^{\mathbb{G}}(x^{\mathbb{K}/\mathbb{T}} : A).B]_\varepsilon & := & \mathrm{tyval}\ (\forall(x : [\![A]\!]_\varepsilon).[\![B]\!]_\varepsilon)\ (\lambda x.[B]_\emptyset) \\
[\forall^{\mathbb{G}}(x^{\mathbb{G}} : A).B]_\varepsilon & := & \mathrm{tyval}\ ([\![A]\!]_\varepsilon \to [\![B]\!]_\varepsilon)\ (\lambda\_.[B]_\emptyset) \\
[\forall^{\mathbb{P}}(x^{\mathbb{P}} : A).B]_\varepsilon & := & [B]_\varepsilon \\
[\forall^{\mathbb{P}}(x^{\mathfrak{s}} : A).B]_\varepsilon & := & \star \\
[\lambda(x^{\mathbb{K}/\mathbb{T}} : A).t^{\mathbb{K}/\mathbb{T}}]_\varepsilon & := & \lambda(x : [\![A]\!]_\varepsilon).[t]_\varepsilon \\
[\lambda(x^{\mathbb{G}/\mathbb{P}} : A).t^{\mathbb{K}/\mathbb{T}}]_\varepsilon & := & [t]_\varepsilon \\
[t^{\mathbb{K}/\mathbb{T}}\ u^{\mathbb{K}/\mathbb{T}}]_\varepsilon & := & [t]_\varepsilon\ [u]_\varepsilon \\
[t^{\mathbb{K}/\mathbb{T}}\ u^{\mathbb{G}/\mathbb{P}}]_\varepsilon & := & [t]_\varepsilon \\
[\mathrm{erased}\ A]_\varepsilon & := & [A]_\varepsilon \\
[\mathrm{reveal}_*(t,p)]_\varepsilon & := & \star \\
[u \approx_A v]_\varepsilon & := & \star \\
[\mathrm{cast}(e,P,t)]_\varepsilon & := & [t]_\varepsilon \\
[\bot]_\varepsilon & := & \star \\
[\mathrm{exfalso}_{\mathbb{K}/\mathbb{T}}(A,p)]_\varepsilon & := & [A]_\emptyset \\
[t^{\mathbb{G}/\mathbb{P}}]_\varepsilon & := & \blacksquare
\end{array}
$$

$$
\begin{array}{lcl}
[\![A]\!]_\varepsilon & := & \mathrm{El}\ [A]_\varepsilon \\
[A]_\emptyset & := & \mathrm{Err}\ [A]_\varepsilon \\
[\![\bullet]\!]_\varepsilon & := & \bullet \\
[\![\Gamma,\ x^{\mathbb{K}/\mathbb{T}} : A]\!]_\varepsilon & := & [\![\Gamma]\!]_\varepsilon,\ x^{\mathbb{T}} : [\![A]\!]_\varepsilon \\
[\![\Gamma,\ x^{\mathbb{G}/\mathbb{P}} : A]\!]_\varepsilon & := & [\![\Gamma]\!]_\varepsilon
\end{array}
$$

Fig. 3. Erasure translation

that we have the following computation rules (following the construction of Pédrot and Tabareau):

$$\mathrm{El}\ (\mathrm{tyval}\ A\ a) \equiv A \qquad \mathrm{Err}\ (\mathrm{tyval}\ A\ a) \equiv a \qquad \mathrm{El}\ \mathrm{ty}_\emptyset \equiv \mathrm{unit} \qquad \mathrm{Err}\ \mathrm{ty}_\emptyset \equiv ()$$

In Figure 3, we syntactically define erasure with the idea that it should only operate on relevant terms (whose mode is $\mathbb{T}$ or $\mathbb{K}$). In order to handle partiality of the function we assume the existence of a dummy term $\blacksquare$ in the target that is never well typed. Herein we write $\mathbb{K}/\mathbb{T}$ to mean $\mathbb{T}$ or $\mathbb{K}$, similarly for $\mathbb{G}/\mathbb{P}$ and so on. We also write $t^{\mathfrak{s}}$ to mean that we match on t such that $\mathrm{md}(t) = \mathfrak{s}$ to avoid cluttering the definition.

Note that the universe of propositions and ghost types as well as propositions and ghost types themselves are not irrelevant so they need to have an erasure as well. Since we don't care about the content of propositions, it is not necessary to erase them to types. We thus erase[3] Prop to $\top$ and propositions to $\star$. It is in fact needed in order to be able to produce propositions from ghost data using reveal_*. In contrast, we do preserve ghost types even if their values are erased. For them, erasure essentially recovers the associated relevant type, but removing dependencies on

[3]Note that we implicitly lift from Prop to Type in the target.

ghost values since these values are removed from the context. Also notice the translation of exfalso which may not use the proof of $\bot$ given it is erased so instead raises an exception, declaring that the branch is inaccessible.

We now show that erasure is sound through the following lemmas: two syntactical properties of erasure in relation with substitution and cast removal; and the fact that it preserves typing.

Lemma 3.1. *Erasure commutes with substitution: assuming* $\mathrm{md}(u) = \mathfrak{s}$, *we have*

- $[t[x^{\mathfrak{s}} := u]]_\varepsilon =_\alpha [t]_\varepsilon[x^{\mathbb{T}} := [u]_\varepsilon]$, *when* $\mathfrak{s} \in \{\mathbb{T}, \mathbb{K}\}$.
- $[t[x^{\mathfrak{s}} := u]]_\varepsilon =_\alpha [t]_\varepsilon$, *when* $\mathfrak{s} \in \{\mathbb{P}, \mathbb{G}\}$.

Proof. The key is to use Lemma 2.1, the proof is then by straightforward induction on t. □

Lemma 3.2. *Erasure ignores casts: if* $|u| =_\alpha |v|$ *then* $[u]_\varepsilon =_\alpha [v]_\varepsilon$.

Proof. The key is to use Lemma 2.4 and the fact that casts are transparent for erasure. □

Lemma 3.3. *Erasure preserves conversion and typing:*

- *If* $\vdash \Gamma$ *then* $\vdash [\![\Gamma]\!]_\varepsilon$.
- *If* $u \equiv v$ *then* $[u]_\varepsilon \equiv [v]_\varepsilon$.
- *If* $\Gamma \vdash t : A$ *and* $\mathrm{md}(t) \in \{\mathbb{T}, \mathbb{K}\}$ *then* $[\![\Gamma]\!]_\varepsilon \vdash [t]_\varepsilon : [\![A]\!]_\varepsilon$.

A direct corollary is that whenever $\Gamma \vdash A :$ Type/Kind and $\mathrm{md}(A) = \mathbb{K}$ then $[\![\Gamma]\!]_\varepsilon \vdash [A]_\emptyset : [\![A]\!]_\varepsilon$; another is that $A \equiv B$ implies $[A]_\emptyset \equiv [B]_\emptyset$. We use these facts in the proof.

Proof. The first item is a corollary of the two others that we both prove by induction on the derivation, using Lemmas 3.2 and 3.3. Note that we make use of the fact that conversion is untyped as well in the target so that $\blacksquare \equiv \blacksquare$ holds. For instance to show that $[\mathrm{reveal}(\mathrm{erase}\ t, P, p)]_\varepsilon \equiv [p\ t]_\varepsilon$ holds, we remark that both sides have the same mode, namely $\mathrm{md}(p)$ so by forcing it to be $\mathbb{P}$ or $\mathbb{G}$ we know both sides erase to $\blacksquare$. For the congruence rules of conversion, Lemma 2.4 is useful to make sure both sides are indeed erased in the same way so that the induction hypotheses are enough to conclude. Aside from that there are no difficulties given the mode assumptions in the derivations. □

3.2 Revival

Erasure preserves ghost types to some extent but not their inhabitants, to perform the parametricity translation we thus need to add an extra translation to recover them as they can appear in propositions. We call this translation *revival* to echo *resurrection* of irrelevant variables in a context as introduced by Pfenning [2001], whereas we *revive* values that have been erased. We define $[\![-]\!]_v$ in Figure 4 with the idea that whenever $\Gamma \vdash t : A$ with $\mathrm{md}(t) = \mathbb{G}$ then $[\![\Gamma]\!]_v \vdash [\![A]\!]_v : [\![A]\!]_\varepsilon$. Like for erasure, we treat partiality using the dummy value $\blacksquare$.

We now prove soundness of revival. We first make an observation about context that says that when $\Gamma \vdash t : A$ then $[t]_\varepsilon$ still makes sense in $[\![\Gamma]\!]_v$ (*i.e.* is implicitly lifted to it). The rest is similar to what we did for erasure.

Lemma 3.4. $[\![\Gamma]\!]_\varepsilon$ *is a sub-context of* $[\![\Gamma]\!]_v$

Lemma 3.5. *Revival commutes with substitution: assuming* $\mathrm{md}(u) = \mathfrak{s}$, *we have*

- $[\![t[x^{\mathfrak{s}} := u]]\!]_v =_\alpha [\![t]\!]_v[x^{\mathbb{T}} := [u]_\varepsilon]$, *when* $\mathfrak{s} \in \{\mathbb{T}, \mathbb{K}\}$.
- $[\![t[x^{\mathfrak{s}} := u]]\!]_v =_\alpha [\![t]\!]_v[x^{\mathbb{T}} := [\![u]\!]_v]$, *when* $\mathfrak{s} = \mathbb{G}$.
- $[\![t[x^{\mathfrak{s}} := u]]\!]_v =_\alpha [\![t]\!]_v$, *when* $\mathfrak{s} = \mathbb{P}$.

$$
\begin{array}{lcl}
[\![x^{\mathbb{G}}]\!]_v & := & x^{\mathbb{T}} \\
[\![\lambda(x^{\mathbb{K}/\mathbb{T}/\mathbb{G}} : A).t^{\mathbb{G}}]\!]_v & := & \lambda(x : [\![A]\!]_\varepsilon).[\![t]\!]_v \\
[\![\lambda(x^{\mathbb{P}} : A).t^{\mathbb{G}}]\!]_v & := & [\![t]\!]_v \\
[\![t^{\mathbb{G}}\ u^{\mathbb{K}/\mathbb{T}}]\!]_v & := & [\![t]\!]_v\ [u]_\varepsilon \\
[\![t^{\mathbb{G}}\ u^{\mathbb{G}}]\!]_v & := & [\![t]\!]_v\ [\![u]\!]_v \\
[\![t^{\mathbb{G}}\ u^{\mathbb{P}}]\!]_v & := & [\![t]\!]_v \\
[\![\mathsf{erase}\ \ t]\!]_v & := & [t]_\varepsilon \\
[\![\mathsf{reveal}(t, P, p^{\mathbb{G}})]\!]_v & := & [\![p]\!]_v\ [\![t]\!]_v \\
[\![\mathsf{cast}(e, P, t^{\mathbb{G}})]\!]_v & := & [\![t]\!]_v \\
[\![\mathsf{exfalso}_{\mathbb{G}}(A, p)]\!]_v & := & [A]_\emptyset \\
[\![t^{\mathbb{K}/\mathbb{T}/\mathbb{P}}]\!]_v & := & \blacksquare \\
\\
[\![\Gamma,\ x^{\mathbb{K}/\mathbb{T}/\mathbb{G}} : A]\!]_v & := & [\![\Gamma]\!]_v,\ x^{\mathbb{T}} : [\![A]\!]_\varepsilon \\
[\![\Gamma,\ x^{\mathbb{P}} : A]\!]_v & := & [\![\Gamma]\!]_v
\end{array}
$$

Fig. 4. Revival translation

$$
\frac{\Gamma \vdash A : \mathsf{Type}}{\Gamma \vdash \|A\| : \mathsf{Prop}} \qquad \frac{\Gamma \vdash t : A}{\Gamma \vdash \mathsf{sq}\ \ t : \|A\|}
$$

$$
\frac{\Gamma \vdash e : \|A\| \qquad \Gamma \vdash P : \|A\| \to \mathsf{Prop} \qquad \Gamma \vdash t : \forall x.\ P\ (\mathsf{sq}\ \ x)}{\Gamma \vdash \mathsf{sq\text{-}elim}(e, P, t) : P\ e}
$$

Fig. 5. Propositional truncation (squash) in the target

Proof. As before, we use Lemma 2.1. The only tricky cases are when revival involves erasure. Let us consider for instance the case of $\mathsf{erase}\ \ t$ and $\mathfrak{s} = \mathbb{G}$. We have to show $[\![(\mathsf{erase}\ \ t)[x^{\mathbb{G}} := u]]\!]_v =_\alpha [\![\mathsf{erase}\ \ t]\!]_v[x^{\mathbb{T}} := [\![u]\!]_v]$. The left-hand side is $[\![\mathsf{erase}\ \ t[x^{\mathbb{G}} := u]]\!]_v =_\alpha [t[x^{\mathbb{G}} := u]]_\varepsilon =_\alpha [t]_\varepsilon$ by Lemma 3.1 but the right-hand side is equal to $[t]_\varepsilon[x^{\mathbb{T}} := [\![u]\!]_v]$ and it might seem like it does not work. However, $[t]_\varepsilon$ lives in context $[\![\Gamma]\!]_\varepsilon$ and is implicitly lifted to $[\![\Gamma]\!]_v$ (Lemma 3.4) before the substitution is applied to it. As such, the substitution operates on a variable not in the scope of $[t]_\varepsilon$ so its action is trivial: $[t]_\varepsilon[x^{\mathbb{T}} := [\![u]\!]_v] =_\alpha [t]_\varepsilon$. □

Lemma 3.6. *Revival ignores casts: if* $|u| =_\alpha |v|$ *then* $[\![u]\!]_v =_\alpha [\![v]\!]_v$.

Lemma 3.7. *Revival preserves typing and conversion:*

- *If* $\vdash \Gamma$ *then* $\vdash [\![\Gamma]\!]_v$.
- *If* $u \equiv v$ *then* $[\![u]\!]_v \equiv [\![v]\!]_v$.
- *If* $\Gamma \vdash t : A$ *and* $\mathrm{md}(t) = \mathbb{G}$ *then* $[\![\Gamma]\!]_v \vdash [\![t]\!]_v : [\![A]\!]_\varepsilon$.

Proof. The proof is very similar to that of Lemma 3.3. As before we use Lemmas 2.1 and 2.4 as well as the lemmas established above (Lemmas 3.4 to 3.6). □

3.3 Parametricity

We are now ready to perform the parametricity translation. We follow Keller and Lasson [2012]—hence the use of Kind—and interpret types by propositional predicates. Propositions are also interpreted as propositions so in particular $u \approx v$ must be interpreted by a proposition. Having

$$\frac{\Gamma \vdash A : \mathrm{Type}_i \qquad \Gamma \vdash u : A \qquad \Gamma \vdash v : A}{\Gamma \vdash u =_A v : \mathrm{Type}_i} \qquad \frac{\Gamma \vdash u : A}{\Gamma \vdash \mathrm{refl}_A\ u : u =_A u}$$

$$\frac{\Gamma \vdash e : u =_A v \qquad \Gamma \vdash P : \forall (x : A).\ u =_A x \to s \qquad \Gamma \vdash t : P\ u\ (\mathrm{refl}_A\ u)}{\Gamma \vdash \mathrm{J}(e, P, t) : P\ v\ e}$$

$$\overline{\mathrm{J}(\mathrm{refl}_A\ u, P, t) \equiv t\ u\ (\mathrm{refl}_A\ u)}$$

Fig. 6. Propositional equality in the target

propositional equality in a definitional proof-irrelevant Prop is a notorious problem [Abel and Coquand 2020] and while there are options to circumvent the issue [Pujet and Tabareau 2022, 2023], we decide instead to simply squash an equality type that might live in Type. To that end we assume the target theory features propositional truncation (or squash) described in Figure 5, and (proof-relevant) equality, given in Figure 6.

The parametricity translation $[\![-]\!]_{\mathcal{P}}$ is given in Figure 7 except for the translation of casts which is a bit more involved and which we will now detail. $[\![\mathrm{cast}(e, P, t)]\!]_{\mathcal{P}}$ depends on the mode of t so we will detail only the case when it is $\mathbb{T}$, the other cases are similar and given in Appendix B. Let us start by computing the expected type for $[\![\mathrm{cast}(e, P, t)]\!]_{\mathcal{P}}$:

$$\begin{aligned}[\![P\ v]\!]_{\mathcal{P}}\ [\mathrm{cast}(e, P, t)]_{\varepsilon} &= [\![P]\!]_{\mathcal{P}}\ [\![v]\!]_{v}\ [\![v]\!]_{\mathcal{P}}\ [\mathrm{cast}(e, P, t)]_{\varepsilon} \\ &= [\![P]\!]_{\mathcal{P}}\ [\![v]\!]_{v}\ [\![v]\!]_{\mathcal{P}}\ [t]_{\varepsilon}\end{aligned}$$

We also compute the type of $[\![P]\!]_{\mathcal{P}}$:

$$\begin{aligned}[\![A \to \mathrm{Type}]\!]_{\mathcal{P}}\ [P]_{\varepsilon} &= \forall (x : [\![A]\!]_{\varepsilon})\ (\overline{x} : [\![A]\!]_{\mathcal{P}}\ x).\ [\![\mathrm{Type}]\!]_{\mathcal{P}}\ [P]_{\varepsilon} \\ &= \forall (x : [\![A]\!]_{\varepsilon})\ (\overline{x} : [\![A]\!]_{\mathcal{P}}\ x).\ [\![P]\!]_{\varepsilon} \to \mathrm{Prop}\end{aligned}$$

This confirms that $[\![P]\!]_{\mathcal{P}}\ [\![v]\!]_{v}\ [\![v]\!]_{\mathcal{P}}\ [t]_{\varepsilon}$ is a proposition so we can prove it by eliminating the squash from $[\![e]\!]_{\mathcal{P}}$ which is of type $\|[\![u]\!]_{v} = [\![v]\!]_{v}\|$. Hence we can define $[\![\mathrm{cast}(e, P, t^{\mathbb{T}})]\!]_{\mathcal{P}}$ as

$$\mathrm{sq\text{-}elim}([\![e]\!]_{\mathcal{P}},\ \lambda\_.\ [\![P]\!]_{\mathcal{P}}\ [\![v]\!]_{v}\ [\![v]\!]_{\mathcal{P}}\ [t]_{\varepsilon},\ \lambda(e' : [\![u]\!]_{v} = [\![v]\!]_{v}).\ w)$$

where $w : [\![P]\!]_{\mathcal{P}}\ [\![v]\!]_{v}\ [\![v]\!]_{\mathcal{P}}\ [t]_{\varepsilon}$ is defined as

$$\mathrm{J}(e',\ \lambda\ (a : [\![A]\!]_{\varepsilon})\ p.\ \forall (h : [\![A]\!]_{\mathcal{P}}\ a).\ [\![P]\!]_{\mathcal{P}}\ a\ h\ [t]_{\varepsilon},\ \lambda(h : [\![A]\!]_{\mathcal{P}}\ [\![u]\!]_{v}).\ [\![t]\!]_{\mathcal{P}})\ [\![v]\!]_{\mathcal{P}}.$$

Note that above we give $[\![t]\!]_{\mathcal{P}} : [\![P]\!]_{\mathcal{P}}\ [\![u]\!]_{v}\ h\ [t]_{\varepsilon}$ by exploiting the fact that $h \equiv [\![u]\!]_{\mathcal{P}}$ since they are both proofs of the same proposition:[4] $[\![A]\!]_{\mathcal{P}}\ [\![u]\!]_{v}$.

Soundness of parametricity is very similar to that of erasure and revival.

Lemma 3.8. *$[\![\Gamma]\!]_{v}$ is a sub-context of $[\![\Gamma]\!]_{\mathcal{P}}$.*

Combined with Lemma 3.4, $[\![\Gamma]\!]_{\varepsilon}$ is thus also a sub-context of $[\![\Gamma]\!]_{\mathcal{P}}$.

Lemma 3.9. *Parametricity commutes with substitution: assuming $\mathrm{md}(u) = \mathfrak{s}$ we have*

- *If $\mathfrak{s} \in \{\mathbb{K}, \mathbb{T}\}$ then $[\![t[x^{\mathfrak{s}} := u]]\!]_{\mathcal{P}} =_{\alpha} [\![t]\!]_{\mathcal{P}}[x^{\mathbb{T}} := [u]_{\varepsilon}, \overline{x}^{\mathbb{P}} := [\![u]\!]_{\mathcal{P}}]$.*
- *If $\mathfrak{s} = \mathbb{G}$ then $[\![t[x^{\mathbb{G}} := u]]\!]_{\mathcal{P}} =_{\alpha} [\![t]\!]_{\mathcal{P}}[x^{\mathbb{T}} := [\![u]\!]_{v}, \overline{x}^{\mathbb{P}} := [\![u]\!]_{\mathcal{P}}]$.*
- *If $\mathfrak{s} = \mathbb{P}$ then $[\![t[x^{\mathbb{P}} := u]]\!]_{\mathcal{P}} =_{\alpha} [\![t]\!]_{\mathcal{P}}[x^{\mathbb{P}} := [\![u]\!]_{\mathcal{P}}]$.*

[4] This informal argument relies on modes to be made formal.

$$
\begin{array}{lcl}
[\![x^{\mathbb{K}}]\!]_{\mathcal{P}} & := & \overline{x}^{\mathbb{T}} \\
[\![x^{\mathbb{T}/\mathbb{G}}]\!]_{\mathcal{P}} & := & \overline{x}^{\mathbb{P}} \\
[\![x^{\mathbb{P}}]\!]_{\mathcal{P}} & := & x^{\mathbb{P}} \\
[\![\mathrm{Kind}_i]\!]_{\mathcal{P}} & := & \lambda A.\ \mathrm{El}\ A \to \mathrm{Type}_i \\
[\![\mathrm{Type}_i]\!]_{\mathcal{P}} & := & \lambda A.\ \mathrm{El}\ A \to \mathrm{Prop} \\
[\![\mathrm{Ghost}_i]\!]_{\mathcal{P}} & := & \lambda A.\ \mathrm{El}\ A \to \mathrm{Prop} \\
[\![\mathrm{Prop}]\!]_{\mathcal{P}} & := & \lambda \_.\ \mathrm{Prop} \\
[\![\forall^{\mathbb{K}/\mathbb{T}}(x^{\mathbb{K}/\mathbb{T}} : A).B]\!]_{\mathcal{P}} & := & \lambda f.\ \forall (x : [\![A]\!]_{\varepsilon})\ (\overline{x} : [\![A]\!]_{\mathcal{P}}\ x).\ [\![B]\!]_{\mathcal{P}}\ (f\ x) \\
[\![\forall^{\mathbb{K}/\mathbb{T}}(x^{\mathbb{G}} : A).B]\!]_{\mathcal{P}} & := & \lambda f.\ \forall (x : [\![A]\!]_{\varepsilon})\ (\overline{x} : [\![A]\!]_{\mathcal{P}}\ x).\ [\![B]\!]_{\mathcal{P}}\ f \\
[\![\forall^{\mathbb{K}/\mathbb{T}}(x^{\mathbb{P}} : A).B]\!]_{\mathcal{P}} & := & \lambda f.\ [\![A]\!]_{\mathcal{P}} \to [\![B]\!]_{\mathcal{P}}\ f \\
[\![\forall^{\mathbb{G}}(x^{\mathbb{K}/\mathbb{T}/\mathbb{G}} : A).B]\!]_{\mathcal{P}} & := & \lambda f.\ \forall (x : [\![A]\!]_{\varepsilon})\ (\overline{x} : [\![A]\!]_{\mathcal{P}}\ x).\ [\![B]\!]_{\mathcal{P}}\ (f\ x) \\
[\![\forall^{\mathbb{G}}(x^{\mathbb{P}} : A).B]\!]_{\mathcal{P}} & := & \lambda f.\ \forall (x : [\![A]\!]_{\mathcal{P}}).\ [\![B]\!]_{\mathcal{P}}\ f \\
[\![\forall^{\mathbb{P}}(x^{\mathbb{K}/\mathbb{T}/\mathbb{G}} : A).B]\!]_{\mathcal{P}} & := & \forall (x : [\![A]\!]_{\varepsilon})\ (\overline{x} : [\![A]\!]_{\mathcal{P}}\ x).\ [\![B]\!]_{\mathcal{P}} \\
[\![\forall^{\mathbb{P}}(x^{\mathbb{P}} : A).B]\!]_{\mathcal{P}} & := & \forall (x : [\![A]\!]_{\mathcal{P}}).\ [\![B]\!]_{\mathcal{P}} \\
[\![\lambda(x^{\mathbb{K}/\mathbb{T}/\mathbb{G}} : A).t]\!]_{\mathcal{P}} & := & \lambda (x : [\![A]\!]_{\varepsilon})\ (\overline{x} : [\![A]\!]_{\mathcal{P}}\ x).\ [\![t]\!]_{\mathcal{P}} \\
[\![\lambda(x^{\mathbb{P}} : A).t]\!]_{\mathcal{P}} & := & \lambda (x : [\![A]\!]_{\mathcal{P}}).\ [\![t]\!]_{\mathcal{P}} \\
[\![t\ u^{\mathbb{K}/\mathbb{T}}]\!]_{\mathcal{P}} & := & [\![t]\!]_{\mathcal{P}}\ [u]_{\varepsilon}\ [\![u]\!]_{\mathcal{P}} \\
[\![t\ u^{\mathbb{G}}]\!]_{\mathcal{P}} & := & [\![t]\!]_{\mathcal{P}}\ [\![u]\!]_{v}\ [\![u]\!]_{\mathcal{P}} \\
[\![t\ u^{\mathbb{P}}]\!]_{\mathcal{P}} & := & [\![t]\!]_{\mathcal{P}}\ [\![u]\!]_{\mathcal{P}} \\
[\![\mathrm{erased}\ A^{\mathbb{K}}]\!]_{\mathcal{P}} & := & [\![A]\!]_{\mathcal{P}} \\
[\![\mathrm{erase}\ t^{\mathbb{T}}]\!]_{\mathcal{P}} & := & [\![t]\!]_{\mathcal{P}} \\
[\![\mathrm{reveal}(t, P, p^{\mathbb{G}/\mathbb{P}})]\!]_{\mathcal{P}} & := & [\![p]\!]_{\mathcal{P}}\ [\![t]\!]_{v}\ [\![t]\!]_{\mathcal{P}} \\
[\![\mathrm{reveal}_*(t, p^{\mathbb{K}})]\!]_{\mathcal{P}} & := & [\![p]\!]_{\mathcal{P}}\ [\![t]\!]_{v}\ [\![t]\!]_{\mathcal{P}} \\
[\![u \approx_A v]\!]_{\mathcal{P}} & := & ||[\![u]\!]_{v} = [\![v]\!]_{v}|| \\
[\![\text{gh-refl}_A\ u]\!]_{\mathcal{P}} & := & \mathrm{sq}\ \mathrm{refl} \\
[\![\mathrm{cast}(e, P, t)]\!]_{\mathcal{P}} & := & \text{(given in the text)} \\
[\![\bot]\!]_{\mathcal{P}} & := & \bot \\
[\![\mathrm{exfalso}_{\mathbb{K}/\mathbb{T}/\mathbb{G}}(A, p)]\!]_{\mathcal{P}} & := & \mathrm{exfalso}([\![A]\!]_{\mathcal{P}}\ [A]_{\emptyset}, [\![p]\!]_{\mathcal{P}}) \\
[\![\mathrm{exfalso}_{\mathbb{P}}(A, p)]\!]_{\mathcal{P}} & := & \mathrm{exfalso}([\![A]\!]_{\mathcal{P}}, [\![p]\!]_{\mathcal{P}}) \\
[\![\_]\!]_{\mathcal{P}} & := & \blacksquare \\
 & & \\
[\![\bullet]\!]_{\mathcal{P}} & := & \bullet \\
[\![\Gamma,\ x^{\mathbb{K}} : A]\!]_{\mathcal{P}} & := & [\![\Gamma]\!]_{\mathcal{P}},\ x^{\mathbb{T}} : [\![A]\!]_{\varepsilon},\ \overline{x}^{\mathbb{T}} : [\![A]\!]_{\mathcal{P}}\ x^{\mathbb{T}} \\
[\![\Gamma,\ x^{\mathbb{T}/\mathbb{G}} : A]\!]_{\mathcal{P}} & := & [\![\Gamma]\!]_{\mathcal{P}},\ x^{\mathbb{T}} : [\![A]\!]_{\varepsilon},\ \overline{x}^{\mathbb{P}} : [\![A]\!]_{\mathcal{P}}\ x^{\mathbb{T}} \\
[\![\Gamma,\ x^{\mathbb{P}} : A]\!]_{\mathcal{P}} & := & [\![\Gamma]\!]_{\mathcal{P}},\ x^{\mathbb{P}} : [\![A]\!]_{\mathcal{P}}
\end{array}
$$

Fig. 7. Parametricity translation

Proof. We again use Lemma 2.1, and similarly to Lemma 3.5 we need to be careful when handling erasure or revival appearing in subterms and simply note that the substitution of $\overline{x}$ by $[\![u]\!]_{\mathcal{P}}$ in such terms has no effect. □

Similarly to erasure and revival, we show that when two terms are the same up to casts, then their parametricity translations are equal. Before, we proved the translations were syntactically equal, this time we only prove definitional equality as we need to exploit proof irrelevance. Before

we do that we thus need to establish that the parametricity translation produces terms of the right modes:

Lemma 3.10. *Parametricity produces consistent modes:*

- *If* $\mathrm{md}(t) \in \{\mathbb{P}, \mathbb{G}, \mathbb{T}\}$ *then* $\mathrm{md}(\llbracket t \rrbracket_{\mathcal{P}}) = \mathbb{P}$.
- *If* $\mathrm{md}(t) = \mathbb{K}$ *then* $\mathrm{md}(t) = \mathbb{T}$.

Proof. This proof works purely by case analysis. There are some caveats however. For instance, erased A is of mode $\mathbb{K}$ regardless of the mode of A so just taking $\llbracket A \rrbracket_{\mathcal{P}}$ as its translation is not guaranteed to yield anything of mode $\mathbb{T}$. Thankfully, this only ever happens when such an expression is ill typed. This means we are free to choose their translations however we want as long as they respect mode. So in this case we only define the translation when $\mathrm{md}(A) = \mathbb{K}$. The other cases are discarded using the dummy value, as indicated by the clause $\llbracket \_ \rrbracket_{\mathcal{P}} := \blacksquare$ in Figure 7. □

Lemma 3.11. *Parametricity ignores casts: if* $|u| =_{\alpha} |v|$ *then* $\llbracket u \rrbracket_{\mathcal{P}} \equiv \llbracket v \rrbracket_{\mathcal{P}}$.

Proof. The proof essentially makes repeated use of congruence rules of conversion and uses proof irrelevance to get rid of the translation of casts. To be able to do this, we have to argue that in those cases, both sides are in mode $\mathbb{P}$. Concretely, if $\mathrm{md}(t) \in \{\mathbb{P}, \mathbb{G}, \mathbb{T}\}$, then by Lemma 3.10 we know $\mathrm{md}(\llbracket t \rrbracket_{\mathcal{P}}) = \mathbb{P}$, and the same goes for $\llbracket \mathrm{cast}(e, P, t) \rrbracket_{\mathcal{P}}$, meaning they are indeed convertible. The case of $\mathrm{md}(t) = \mathbb{K}$ is slightly trickier, because by Lemma 3.10, $\mathrm{md}(\llbracket t \rrbracket_{\mathcal{P}}) = \mathbb{T}$. In order to make this work, we simply define $\llbracket \mathrm{cast}(e, P, t) \rrbracket_{\mathcal{P}}$ as $\llbracket t \rrbracket_{\mathcal{P}}$ in that case so that the equality trivially holds. As before, we are free to do so because casts are never well typed when $\mathrm{md}(t) = \mathbb{K}$. □

Lemma 3.12. *Parametricity preserves conversion and typing:*

- *If* $\vdash \Gamma$ *then* $\vdash \llbracket \Gamma \rrbracket_{\mathcal{P}}$.
- *If* $u \equiv v : A$ *then* $\llbracket u \rrbracket_{\mathcal{P}} \equiv \llbracket v \rrbracket_{\mathcal{P}}$.
- *If* $\mathrm{md}(t) \in \{\mathbb{K}, \mathbb{T}\}$ *and* $\Gamma \vdash t : A$ *then* $\llbracket \Gamma \rrbracket_{\mathcal{P}} \vdash \llbracket t \rrbracket_{\mathcal{P}} : \llbracket A \rrbracket_{\mathcal{P}}\ [t]_{\varepsilon}$.
- *If* $\mathrm{md}(t) = \mathbb{G}$ *and* $\Gamma \vdash t : A$ *then* $\llbracket \Gamma \rrbracket_{\mathcal{P}} \vdash \llbracket t \rrbracket_{\mathcal{P}} : \llbracket A \rrbracket_{\mathcal{P}}\ \llbracket t \rrbracket_{v}$.
- *If* $\mathrm{md}(t) = \mathbb{P}$ *and* $\Gamma \vdash t : A$ *then* $\llbracket \Gamma \rrbracket_{\mathcal{P}} \vdash \llbracket t \rrbracket_{\mathcal{P}} : \llbracket A \rrbracket_{\mathcal{P}}$.

Proof. We use Lemmas 3.3, 3.4, 3.7 and 3.8 to deal with erasure and revival in subterms and we prove the result by induction on the various derivations. We crucially use Lemma 3.11 for conversion and the various substitution lemmas (Lemmas 3.1, 3.5 and 3.9) for conversion and typing. Also note that thanks to Lemmas 2.4, 3.10 and 3.11 and proof irrelevance in the target, one only needs to consider terms of mode $\mathbb{K}$ in the proof of conversion preservation. □

4 META-THEORETICAL CONSEQUENCES OF THE MODEL

As we explained before, our target is standard enough to be a subset of Coq or Agda. As such, we argue it is safe to assume it is consistent and enjoys various properties such as type former discrimination (two different type formers are never convertible) or injectivity of constructors. Assuming this, we can lift certain properties back to GTT thanks to our translations. First and foremost, we obtain (relative) consistency of the type theory.

Theorem 4.1 (Consistency). *GTT is consistent.*

Proof. Assuming there is a proof of $\bot$ in the empty context GTT, we then translate $\vdash p : \bot$ to $\vdash \llbracket p \rrbracket_{\mathcal{P}} : \llbracket \bot \rrbracket_{\mathcal{P}}$ and since $\llbracket \bot \rrbracket_{\mathcal{P}}$ is $\bot$ we get a contradiction in the target. □

Another feature which we advocated for in the introduction was the ability to discriminate type formers, and we can use the model to lift this property back to GTT.

Theorem 4.2 (Discrimination of type formers). *GTT can discriminate type formers. Concretely we can disprove conversions involving different type formers at the head, such as:*

- $\forall^{\mathsf{r}}(x^{\mathsf{s}} : A).B \not\equiv \mathrm{Type}$
- $\mathrm{Kind}_i \not\equiv \mathrm{Type}_i$
- $\mathrm{Type}_i \not\equiv \mathrm{Ghost}_i$
- $\mathrm{Type}_i \not\equiv \mathrm{Prop}$
- $X \not\equiv \mathrm{Type}_i$
- ...

Proof. The idea is to remark that such type formers are distinguished in the model, either by using the erasure (Lemma 3.3) or the parametricity (Lemma 3.12) translations, depending on the case. Let us look precisely at some representative cases.

Let us start with $\forall^{\mathbb{K}}(x^{\mathbb{T}} : A).B$. Assuming $\forall^{\mathbb{K}}(x^{\mathbb{T}} : A).B \equiv \mathrm{Type}$, by applying erasure, we get

$$\mathrm{tyval}\ (\forall(x : [\![A]\!]_\varepsilon).[\![B]\!]_\varepsilon)\ (\lambda x.[B]_\emptyset) \equiv \mathrm{tyval}\ \mathrm{ty}_i\ \mathrm{ty}_\emptyset$$

then by congruence of El and its computation rules we get

$$\forall(x : [\![A]\!]_\varepsilon).[\![B]\!]_\varepsilon \equiv \mathrm{ty}_i$$

which is not possible because the target discriminates type formers.

Now, for $\forall^{\mathbb{K}}(x^{\mathbb{G}} : A).B$, the above does not work since it is not erased to a dependent function type. There we rely instead on parametricity. So assuming $\forall^{\mathbb{K}}(x^{\mathbb{G}} : A).B \equiv \mathrm{Type}$, we get

$$\lambda f.\ \forall(x : [\![A]\!]_\varepsilon)\ (\overline{x} : [\![A]\!]_{\mathcal{P}}\ x).\ [\![B]\!]_{\mathcal{P}}\ f \equiv \lambda A.\ \mathrm{El}\ A \to \mathrm{Prop}$$

By weakening, we can add some variable $f : [\![B]\!]_\varepsilon$, and use congruence of application and β-reduction to obtain

$$\forall(x : [\![A]\!]_\varepsilon)\ (\overline{x} : [\![A]\!]_{\mathcal{P}}\ x).\ [\![B]\!]_{\mathcal{P}}\ f \equiv \mathrm{El}\ f \to \mathrm{Prop}$$

which the target also discriminates (two function types against a function type with codomain Prop). Notice how the two sides are not even of the same type to begin with so the proof also exploits the fact that conversion is untyped. □

We argued already that this property is useful in practice: it allows for early returns in conversion algorithms and it helps avoiding cluttering user feedback with nonsensical error messages.

Finally, thanks to the above, we are also able to get rid of the greyed premises in the typing rules of GTT. The main ingredient for that is the following lemma that states that modes are consistent with sorts.

Lemma 4.3. *If* $\vdash \Gamma$ *and* $\Gamma \vdash t : A$ *and* $\Gamma \vdash A : s$ *then* $\mathrm{md}(t) = \lfloor s \rfloor$.

Proof. Given $\Gamma \vdash t : A$ and $\Gamma \vdash A : s$, we apply Lemma 2.7 to obtain some r such that $\Gamma \vdash A : r$ and $\mathrm{md}(t) = \lfloor r \rfloor$. We now have two sorts s and r for A. By Lemma 2.6 we thus know they are convertible: $s \equiv r$. Finally, by Theorem 4.2 we know that they cannot have different classes, and thus that $\lfloor s \rfloor = \lfloor r \rfloor$ which is enough to conclude. Actually, like for validity, the application case is slightly trickier and it involves using inversion to make sure the domain and codomain of the $\forall$-type are in sorts of the right class. □

We can now formally let go of the greyed premises in GTT. We will thus omit them in the remainder of this paper. Note that we use validity which means that we must ensure the context is well formed to use the more economic rules.

Theorem 4.4. *The rules of GTT presented in Figure 1 are still admissible without the greyed premises.*

$$t, u, A, B ::= \cdots \mid \mathrm{nat} \mid 0 \mid \mathrm{S}\ n \mid \mathrm{nat\text{-}elim}_{\mathfrak{r}}(n, P, z, s)$$

$$\mathrm{md}(\mathrm{nat}) = \mathbb{K} \qquad \mathrm{md}(0) = \mathrm{md}(\mathrm{S}\ n) = \mathbb{T} \qquad \mathrm{md}(\mathrm{nat\text{-}elim}_{\mathfrak{r}}(n, P, z, s)) = \mathfrak{r}$$

$$\frac{}{\Gamma \vdash \mathrm{nat} : \mathrm{Type}_i} \qquad \frac{}{\Gamma \vdash 0 : \mathrm{nat}} \qquad \frac{\Gamma \vdash n : \mathrm{nat}}{\Gamma \vdash \mathrm{S}\ n : \mathrm{nat}}$$

$$\frac{\Gamma \vdash P : \mathrm{nat} \to r \qquad \Gamma \vdash z : P\ 0 \qquad \Gamma \vdash s : \forall^{\lfloor r \rfloor}(x^{\mathbb{T}} : \mathrm{nat}).\ P\ x \to P\ (\mathrm{S}\ x) \qquad \lfloor r \rfloor \neq \mathbb{K} \qquad \Gamma \vdash n : \mathrm{nat}}{\Gamma \vdash \mathrm{nat\text{-}elim}_{\lfloor r \rfloor}(n, P, z, s) : P\ n}$$

$$\frac{}{\mathrm{nat\text{-}elim}_{\mathfrak{r}}(0, P, z, s) \equiv z} \qquad \frac{}{\mathrm{nat\text{-}elim}_{\mathfrak{r}}(\mathrm{S}\ n, P, z, s) \equiv s\ n\ (\mathrm{nat\text{-}elim}_{\mathfrak{r}}(n, P, z, s))}$$

Fig. 8. Rules for natural numbers

Proof. In the typing rules, two cases of redundant premises are present: (1) those where we have $\Gamma \vdash A : s$ and we want to get $\mathrm{md}(A) = \mathbb{K}$; and (2) those where we have $\Gamma \vdash t : A$ and $\Gamma \vdash A : s$ and we want $\mathrm{md}(t) = \lfloor s \rfloor$. Since sorts are always of type Kind, we only need to treat the second case, which is exactly Lemma 4.3. □

5 EXTENDING GTT WITH INDUCTIVE TYPES

As we advertised in the introduction, ghost types really shine when used as indices to inductive types. We will thus show how to extend GTT and the translations of Section 3 to natural numbers (Section 5.1) and vectors (Section 5.2). We leave a more general treatment of inductive types for future work, but hopefully this should prove convincing enough that the model can scale to all of them, the same way the model of Pédrot and Tabareau [2018] scales to inductive types. Our translation is basically the same as theirs, except we use eliminators rather than fixed points and pattern matching.

Before we delve in, a few remarks are in order. First, in the typing rules we are about to show, we ignore redundant premises carrying mode information for simplicity. Technically, we have to do the same thing we did: first define the typing rules with these premises and then show they are admissible. The second and more important remark is that the inductive types we consider in GTT cannot be eliminated to sort Kind. This is a limitation coming from the fact that the parametricity translation produces propositions and that proofs cannot in general be eliminated to produce relevant information which is expected from the translation of Kind. Keller and Lasson [2012, Section 4.4] show how to circumvent this problem for certain inductive types but we consider it out of scope for the current paper.

5.1 Natural numbers

We begin with natural numbers as they are a very simple example of inductive type and because we will need them for vectors later. We give the syntax and rules of nat in Figure 8.

In order to give the erasure of natural numbers we need to enrich the target with a new inductive type $\mathrm{nat}^{\bullet}$ of natural numbers with exceptions whose rules are given in Figure 9. They can be understood as a copy of natural numbers with a distinguished exception inhabitant called $\mathrm{nat}_{\emptyset}$. Erasure is then straightforward, the only thing is that the eliminator must be able to react to potential

$$\frac{}{\Gamma \vdash \mathrm{nat}^\bullet : \mathrm{Type}_i} \qquad \frac{}{\Gamma \vdash 0^\bullet : \mathrm{nat}^\bullet} \qquad \frac{\Gamma \vdash n : \mathrm{nat}^\bullet}{\Gamma \vdash \mathrm{S}^\bullet\ n : \mathrm{nat}^\bullet} \qquad \frac{}{\Gamma \vdash \mathrm{nat}_\emptyset : \mathrm{nat}^\bullet}$$

$$\frac{\Gamma \vdash P : \mathrm{nat} \to r \qquad \Gamma \vdash z : P\ 0^\bullet \qquad \Gamma \vdash n : \mathrm{nat}^\bullet \qquad \Gamma \vdash s : \forall (x : \mathrm{nat}^\bullet).\ P\ x \to P\ (\mathrm{S}^\bullet\ x) \qquad \Gamma \vdash e : P\ \mathrm{nat}_\emptyset}{\Gamma \vdash \mathrm{nat}^\bullet\text{-elim}(n, P, z, s, e) : P\ n}$$

$$\frac{}{\mathrm{nat}^\bullet\text{-elim}(0^\bullet, P, z, s, e) \equiv z} \qquad \frac{}{\mathrm{nat}^\bullet\text{-elim}(\mathrm{S}^\bullet\ n, P, z, s, e) \equiv s\ n\ (\mathrm{nat}^\bullet\text{-elim}(n, P, z, s, e))}$$

$$\frac{}{\mathrm{nat}^\bullet\text{-elim}(\mathrm{nat}_\emptyset, P, z, s, e) \equiv e}$$

Fig. 9. Exceptional natural numbers (target)

$$\frac{}{\Gamma \vdash \mathrm{nat}^{\mathcal{P}} : \mathrm{nat}^\bullet \to \mathrm{Prop}} \qquad \frac{}{\Gamma \vdash 0^{\mathcal{P}} : \mathrm{nat}^{\mathcal{P}}\ 0^\bullet} \qquad \frac{\Gamma \vdash n : \mathrm{nat}^\bullet \qquad \Gamma \vdash n^{\mathcal{P}} : \mathrm{nat}^{\mathcal{P}}\ n}{\Gamma \vdash \mathrm{S}^{\mathcal{P}}\ n\ n^{\mathcal{P}} : \mathrm{nat}^{\mathcal{P}}\ (\mathrm{S}^\bullet\ n)}$$

$$\frac{\Gamma \vdash n : \mathrm{nat}^\bullet \qquad \Gamma \vdash n^{\mathcal{P}} : \mathrm{nat}^{\mathcal{P}}\ n \qquad \Gamma \vdash P : \forall (n : \mathrm{nat}^\bullet).\ \mathrm{nat}^{\mathcal{P}}\ n \to \mathrm{Prop} \qquad \Gamma \vdash z : P\ 0^\bullet\ 0^{\mathcal{P}} \qquad \Gamma \vdash s : \forall (x : \mathrm{nat}^\bullet)\ (x^{\mathcal{P}} : \mathrm{nat}^{\mathcal{P}}\ x).\ P\ x\ x^{\mathcal{P}} \to P\ (\mathrm{S}^\bullet\ x)\ (\mathrm{S}^{\mathcal{P}}\ x\ x^{\mathcal{P}})}{\Gamma \vdash \mathrm{nat}^{\mathcal{P}}\text{-elim}(n^{\mathcal{P}}, P, z, s) : P\ n\ n^{\mathcal{P}}}$$

Fig. 10. Parametricity predicate of natural numbers (target)

exceptions ($\mathrm{nat}_\emptyset$) and so it uses the exception of the motive $\mathrm{Err}\ ([P]_\varepsilon\ [n]_\varepsilon)$ in that particular case.

$$\begin{array}{lcl}
[\mathrm{nat}]_\varepsilon & := & \mathrm{tyval}\ \mathrm{nat}^\bullet\ \mathrm{nat}_\emptyset \\
[0]_\varepsilon & := & 0^\bullet \\
[\mathrm{S}\ n]_\varepsilon & := & \mathrm{S}^\bullet\ [n]_\varepsilon \\
[\mathrm{nat\text{-}elim}_{\mathbb{T}}(n, P, z, s)]_\varepsilon & := & \mathrm{nat}^\bullet\text{-elim}([n]_\varepsilon, \lambda m.\ \mathrm{El}\ ([P]_\varepsilon\ m), [z]_\varepsilon, [s]_\varepsilon, \mathrm{Err}\ ([P]_\varepsilon\ [n]_\varepsilon))
\end{array}$$

We also need to extend the revival translation for the ghost values introduced by natural number elimination. It is essentially the same as erasure but using revival in the branches instead.

$$[\![\mathrm{nat\text{-}elim}_{\mathbb{G}}(n, P, z, s)]\!]_v := \mathrm{nat}^\bullet\text{-elim}([n]_\varepsilon, \lambda m.\ \mathrm{El}\ ([P]_\varepsilon\ m), [\![z]\!]_v, [\![s]\!]_v, \mathrm{Err}\ ([P]_\varepsilon\ [n]_\varepsilon))$$

For the parametricity translation we also need to extend the target with a parametricity predicate $\mathrm{nat}^{\mathcal{P}}$ that selects exceptional natural numbers that are free of $\mathrm{nat}_\emptyset$. It is given in Figure 10. Parametricity for natural numbers except their eliminator is the following.

$$\begin{array}{lcl}
[\![\mathrm{nat}]\!]_{\mathcal{P}} & := & \mathrm{nat}^{\mathcal{P}} \\
[\![0]\!]_{\mathcal{P}} & := & 0^{\mathcal{P}} \\
[\![\mathrm{S}\ n]\!]_{\mathcal{P}} & := & \mathrm{S}^{\mathcal{P}}\ [n]_\varepsilon\ [\![n]\!]_{\mathcal{P}}
\end{array}$$

We now look at the eliminator. The Type case $[\![\mathrm{nat\text{-}elim}_{\mathbb{T}}(n, P, z, s)]\!]_{\mathcal{P}}$ is obtained by using the eliminator of $\mathrm{nat}^{\mathcal{P}}$ on $[\![n]\!]_{\mathcal{P}}$. It is mostly straightforward but we need to η-expand the successor branch so that it type-checks:

$$\mathrm{nat}^{\mathcal{P}}\text{-elim}([\![n]\!]_{\mathcal{P}}, \lambda m\ \overline{m}.\ [\![P]\!]_{\mathcal{P}}\ m\ \overline{m}\ (Q\ m), [\![z]\!]_{\mathcal{P}}, \lambda m\ \overline{m}\ h.\ [\![s]\!]_{\mathcal{P}}\ m\ \overline{m}\ (Q\ m)\ h)$$

where Q is the erasure of the eliminator with its argument generalised:

$$\lambda m.\ \mathrm{nat}^{\bullet}\text{-elim}(m, \lambda x.\, \mathrm{El}\ ([P]_{\varepsilon}\ x), [z]_{\varepsilon}, [s]_{\varepsilon}, \mathrm{Err}\ ([P]_{\varepsilon}\ [n]_{\varepsilon})).$$

The Ghost case is essentially the same with some revival instead of erasure. $[\![\mathrm{nat\text{-}elim}_{\mathbb{G}}(n, P, z, s)]\!]_{\mathcal{P}}$ is given by

$$\mathrm{nat}^{\mathcal{P}}\text{-elim}([\![n]\!]_{\mathcal{P}}, \lambda m\ \overline{m}.\ [\![P]\!]_{\mathcal{P}}\ m\ \overline{m}\ (Q\ m), [\![z]\!]_{\mathcal{P}}, \lambda m\ \overline{m}\ h.\ [\![s]\!]_{\mathcal{P}}\ m\ \overline{m}\ (Q\ m)\ h)$$

where $Q := \lambda m.\ \mathrm{nat}^{\bullet}\text{-elim}(m, \lambda x.\, \mathrm{El}\ ([P]_{\varepsilon}\ x), [\![z]\!]_{v}, [\![s]\!]_{v}, \mathrm{Err}\ ([P]_{\varepsilon}\ [n]_{\varepsilon}))$. The Prop case is much simpler as there is no need to mention erasure or revival:

$$[\![\mathrm{nat\text{-}elim}_{\mathbb{P}}(n, P, z, s)]\!]_{\mathcal{P}} := \mathrm{nat}^{\mathcal{P}}\text{-elim}([\![n]\!]_{\mathcal{P}}, [\![P]\!]_{\mathcal{P}}, [\![z]\!]_{\mathcal{P}}, [\![s]\!]_{\mathcal{P}})$$

The lemmas of Section 3 are preserved, and we can thus enjoy the properties derived in Section 4. Before we move on to vectors, we will show a few examples that will prove useful later, and that already illustrate what can be achieved with erased natural numbers.

Erased successor. We can define a wrapper around the successor function for erased natural numbers by using reveal.

$$\begin{array}{lcl} \mathrm{gS} & : & \mathrm{erased\ nat} \to \mathrm{erased\ nat} \\ \mathrm{gS} & := & \lambda n.\, \mathrm{reveal}(n, \lambda\_.\, \mathrm{erased\ nat}, \lambda m.\, \mathrm{erase}\ (\mathrm{S}\ m)) \end{array}$$

In the same fashion we can define addition for erased natural numbers which we will write $\oplus$ and for which we can prove the same properties as for regular addition.

Discriminating erased natural numbers. This example shows the power of ghost types: even though they cannot be used for computation, one can still distinguish values, something which cannot be done in Prop. The key is to use reveal_{*} to define a discrimination proposition discrP such that discrP (erase 0) $\equiv \bot \to \bot$ (the true proposition) and discrP (gS n) $\equiv \bot$.

$$\begin{array}{lcl} \mathrm{discrP} & : & \forall^{\mathbb{P}}(n^{\mathbb{G}} : \mathrm{erased\ nat}).\ \mathrm{Prop} \\ \mathrm{discrP} & := & \lambda n.\ \mathrm{reveal}_{*}(n, \lambda x.\ \mathrm{nat\text{-}elim}(x, \lambda\_.\, \mathrm{Prop}, \bot \to \bot, \lambda\_.\, \bot)) \end{array}$$

We can then use it to define the following discriminator:

$$\begin{array}{lcl} \mathrm{discr} & : & \forall^{\mathbb{P}}(n^{\mathbb{G}} : \mathrm{erased\ nat}).\ \mathrm{erase}\ 0 \approx \mathrm{gS}\ n \to \bot \\ \mathrm{discr} & := & \lambda n\ e.\ \mathrm{cast}(e, \mathrm{discrP}, \lambda x.x) \end{array}$$

5.2 Vectors

We now extend GTT with vectors where the length index is an erased natural number. Their syntax and typing rules are given in Figure 11. As before we start by extending the model to support vectors before giving examples of what we can do with these vectors.

5.2.1 Modelling vectors. Erasure, revival and parametricity for vectors are given below, using the type of lists with exceptions and the associated parametricity predicate, as given in Figures 12

$$t, u, A, B ::= \cdots \mid \mathrm{vec}\ A\ n \mid \mathrm{vnil}_A \mid \mathrm{vcons}\ a\ n\ v \mid \mathrm{vec\text{-}elim}_{\mathfrak{s}}(v, P, z, c)$$

$$\mathrm{md}(\mathrm{vec}\ A\ n) = \mathbb{K} \qquad \mathrm{md}(\mathrm{vnil}_A) = \mathrm{md}(\mathrm{vcons}\ a\ n\ v) = \mathbb{T} \qquad \mathrm{md}(\mathrm{vec\text{-}elim}_{\mathfrak{s}}(v, P, z, c)) = \mathfrak{s}$$

$$\frac{\Gamma \vdash A : \mathrm{Type}_i \qquad \Gamma \vdash n : \mathrm{erased\ nat}}{\Gamma \vdash \mathrm{vec}\ A\ n : \mathrm{Type}_i} \qquad \frac{\Gamma \vdash A : \mathrm{Type}}{\Gamma \vdash \mathrm{vnil}_A : \mathrm{vec}\ A\ (\mathrm{erase}\ 0)}$$

$$\frac{\Gamma \vdash a : A \qquad \Gamma \vdash v : \mathrm{vec}\ A\ n}{\Gamma \vdash \mathrm{vcons}\ a\ n\ v : \mathrm{vec}\ A\ (\mathrm{gS}\ n)}$$

$$\frac{\begin{array}{c}\Gamma \vdash v : \mathrm{vec}\ A\ n \qquad \Gamma \vdash P : \forall^{\mathbb{K}}(m^{\mathbb{G}} : \mathrm{erased\ nat}).\ \mathrm{vec}\ A\ m \to s \\ \Gamma \vdash z : P\ (\mathrm{erase}\ 0)\ \mathrm{vnil} \qquad \Gamma \vdash c : \forall^{\lfloor s \rfloor} a\ m\ w.\ P\ m\ w \to P\ (\mathrm{gS}\ m)\ (\mathrm{vcons}\ a\ m\ w) \qquad \lfloor s \rfloor \neq \mathbb{K}\end{array}}{\Gamma \vdash \mathrm{vec\text{-}elim}_{\mathfrak{s}}(v, P, z, c) : P\ n\ v}$$

$$\frac{}{\mathrm{vec\text{-}elim}_{\mathfrak{s}}(\mathrm{vnil}_A, P, z, c) \equiv z} \qquad \frac{}{\mathrm{vec\text{-}elim}_{\mathfrak{s}}(\mathrm{vcons}\ a\ n\ v, P, z, c) \equiv c\ a\ n\ v\ \ \mathrm{vec\text{-}elim}_{\mathfrak{s}}(v, P, z, c)}$$

Fig. 11. Vectors

and 13. We also need conjunction of propositions in the target; they are described in Figure 14.

$$\begin{array}{lcl}
[\mathrm{vec}\ A\ n]_\varepsilon & := & \mathrm{tyval}\ (\mathrm{list}^\bullet\ [\![A]\!]_\varepsilon)\ \mathrm{list}_\emptyset \\
[\mathrm{vnil}_A]_\varepsilon & := & \mathrm{nil}^\bullet_{[\![A]\!]_\varepsilon} \\
[\mathrm{vcons}\ a\ n\ v]_\varepsilon & := & \mathrm{cons}^\bullet\ [a]_\varepsilon\ [v]_\varepsilon \\
[\mathrm{vec\text{-}elim}_{\mathbb{T}}(v, P, z, c)]_\varepsilon & := & \mathrm{list\text{-}elim}^\bullet([v]_\varepsilon, \lambda x.\, \mathrm{El}\ ([P]_\varepsilon\ x), [z]_\varepsilon, [c]_\varepsilon, \mathrm{Err}\ ([P]_\varepsilon\ [v]_\varepsilon)) \\
\\
[\![\mathrm{vec\text{-}elim}_{\mathbb{G}}(v, P, z, c)]\!]_v & := & \mathrm{list\text{-}elim}^\bullet([v]_\varepsilon, \lambda x.\, \mathrm{El}\ ([P]_\varepsilon\ x), [\![z]\!]_v, [\![c]\!]_v, \mathrm{Err}\ ([P]_\varepsilon\ [v]_\varepsilon)) \\
\\
[\![\mathrm{vec}\ A\ n]\!]_{\mathcal{P}} & := & \lambda l.\ \|\mathrm{length}^\bullet\ l = [\![n]\!]_v\| \wedge \mathrm{list}^{\mathcal{P}}\ [\![A]\!]_\varepsilon\ [\![A]\!]_{\mathcal{P}}\ l \\
[\![\mathrm{vnil}]\!]_\varepsilon & := & \langle \mathrm{sq}\ \mathrm{refl}, \mathrm{nil}^{\mathcal{P}} \rangle \\
[\![\mathrm{vcons}\ a\ n\ v]\!]_\varepsilon & := & \langle p, \mathrm{cons}^{\mathcal{P}}\ [\![a]\!]_{\mathcal{P}}\ (\mathrm{snd}\ [\![v]\!]_{\mathcal{P}}) \rangle
\end{array}$$

where $p : \|\mathrm{length}^\bullet\ (\mathrm{cons}^\bullet\ [a]_\varepsilon\ [v]_\varepsilon) = \mathrm{S}^\bullet\ [\![n]\!]_v\|$ is a proof obtained by eliminating $[\![v]\!]_{\mathcal{P}}$ to get the first component of type $\|\mathrm{length}^\bullet\ [v]_\varepsilon = [\![n]\!]_v\|$ and where $\mathrm{length}^\bullet$ is the length function on exceptional lists, returning an exceptional natural number (defined in Appendix B).

For the definition of $[\![\mathrm{vec\text{-}elim}_{\mathbb{T}}(v, P, z, c)]\!]_\varepsilon$ we need to inhabit the type

$$[\![P]\!]_{\mathcal{P}}\ [\![n]\!]_v\ [\![n]\!]_{\mathcal{P}}\ [v]_\varepsilon\ [\![v]\!]_{\mathcal{P}}\ (\mathrm{list\text{-}elim}^\bullet([v]_\varepsilon, \lambda x.\, \mathrm{El}\ ([P]_\varepsilon\ x), [z]_\varepsilon, [c]_\varepsilon, \mathrm{Err}\ ([P]_\varepsilon\ [v]_\varepsilon)))$$

knowing we have $[\![v]\!]_{\mathcal{P}} : \|\mathrm{length}^\bullet\ [v]_\varepsilon = [\![n]\!]_v\| \wedge \mathrm{list}^{\mathcal{P}}\ [\![A]\!]_\varepsilon\ [\![A]\!]_{\mathcal{P}}\ [v]_\varepsilon$. Since the goal is a proposition we can eliminate $[\![v]\!]_{\mathcal{P}}$ to split it into the equality on one side and the parametricity predicate on the other. We need both because the induction principle for $\mathrm{list}^{\mathcal{P}}$ does not talk about the length. Instead, what we prove by induction on the $\mathrm{list}^{\mathcal{P}}$ proof is the following:

$$\forall l\ l^{\mathcal{P}}.\ [\![P]\!]_{\mathcal{P}}\ (\mathrm{length}^\bullet\ l)\ (\mathrm{length}^{\mathcal{P}}\ l\ l^{\mathcal{P}})\ l\ l^{\mathcal{P}}\ (\mathrm{list\text{-}elim}^\bullet(l, \lambda x.\, \mathrm{El}\ ([P]_\varepsilon\ x), [z]_\varepsilon, [c]_\varepsilon, \mathrm{Err}\ ([P]_\varepsilon\ [v]_\varepsilon)))$$

where $\mathrm{length}^{\mathcal{P}}$ (defined in Appendix B) is the predicate that shows $\mathrm{length}^\bullet$ of a proper list does not raise exceptions itself. The base case corresponds exactly to what $[\![z]\!]_{\mathcal{P}}$ gives us, while the $\mathrm{cons}^{\mathcal{P}}$ case is a consequence of $[\![c]\!]_{\mathcal{P}}$ which we instantiate by giving it $\mathrm{length}^\bullet$ for the natural number and refl for the equality proof. Note that we exploit the fact that $[\![\mathrm{gS}\ n]\!]_v = \mathrm{S}^\bullet\ [\![n]\!]_v$ and

$$\frac{\Gamma \vdash A : \mathrm{Type}_i}{\Gamma \vdash \mathrm{list}^\bullet\ A : \mathrm{Type}_i} \qquad \frac{\Gamma \vdash A : \mathrm{Type}}{\Gamma \vdash \mathrm{nil}^\bullet_A : \mathrm{list}^\bullet\ A} \qquad \frac{\Gamma \vdash a : A \qquad \Gamma \vdash l : \mathrm{list}^\bullet\ A}{\Gamma \vdash \mathrm{cons}^\bullet\ a\ l : \mathrm{list}^\bullet\ A} \qquad \frac{\Gamma \vdash A : \mathrm{Type}}{\Gamma \vdash \mathrm{list}_\emptyset : \mathrm{list}^\bullet\ A}$$

$$\frac{\begin{array}{c}\Gamma \vdash l : \mathrm{list}^\bullet\ A \qquad \Gamma \vdash P : \mathrm{list}^\bullet\ A \to s \\ \Gamma \vdash n : P\ \mathrm{nil}^\bullet \qquad \Gamma \vdash c : \forall (x : A)\ (l : \mathrm{list}^\bullet\ A).\ P\ l \to P\ (\mathrm{cons}^\bullet\ x\ l) \qquad \Gamma \vdash e : P\ \mathrm{list}_\emptyset\end{array}}{\Gamma \vdash \mathrm{list\text{-}elim}^\bullet(l, P, n, c, e) : P\ l}$$

$$\frac{}{\mathrm{list\text{-}elim}^\bullet(\mathrm{nil}^\bullet, P, n, c, e) \equiv n} \qquad \frac{}{\mathrm{list\text{-}elim}^\bullet(\mathrm{cons}^\bullet\ a\ l, P, n, c, e) \equiv c\ a\ l\ \ \mathrm{list\text{-}elim}^\bullet(l, P, n, c, e)}$$

$$\frac{}{\mathrm{list\text{-}elim}^\bullet(\mathrm{list}_\emptyset, P, n, c, e) \equiv e}$$

Fig. 12. Exceptional lists (target)

$$\frac{\Gamma \vdash A : \mathrm{Type} \qquad \Gamma \vdash A^{\mathcal{P}} : A \to \mathrm{Prop}}{\Gamma \vdash \mathrm{list}^{\mathcal{P}}\ A\ A^{\mathcal{P}} : \mathrm{list}^\bullet\ A \to \mathrm{Prop}} \qquad \frac{\Gamma \vdash A : \mathrm{Type} \qquad \Gamma \vdash A^{\mathcal{P}} : A \to \mathrm{Prop}}{\Gamma \vdash \mathrm{nil}^{\mathcal{P}} : \mathrm{list}^{\mathcal{P}}\ A\ A^{\mathcal{P}}\ \mathrm{nil}^\bullet}$$

$$\frac{\Gamma \vdash a : A \qquad \Gamma \vdash a^{\mathcal{P}} : A^{\mathcal{P}}\ a \qquad \Gamma \vdash l : \mathrm{list}^\bullet\ A \qquad \Gamma \vdash l^{\mathcal{P}} : \mathrm{list}^{\mathcal{P}}\ A\ A^{\mathcal{P}}\ l}{\Gamma \vdash \mathrm{cons}^{\mathcal{P}}\ a\ a^{\mathcal{P}}\ l\ l^{\mathcal{P}} : \mathrm{list}^{\mathcal{P}}\ A\ A^{\mathcal{P}}\ (\mathrm{cons}^\bullet\ a\ l)}$$

$$\frac{\begin{array}{c}\Gamma \vdash l : \mathrm{list}^\bullet\ A\ A^{\mathcal{P}} \\ \Gamma \vdash l^{\mathcal{P}} : \mathrm{list}^{\mathcal{P}}\ A\ A^{\mathcal{P}}\ l \qquad \Gamma \vdash P : \forall (l : \mathrm{list}^\bullet\ A).\ \mathrm{list}^{\mathcal{P}}\ A\ A^{\mathcal{P}}\ l \to \mathrm{Prop} \qquad \Gamma \vdash n : P\ \mathrm{nil}^\bullet\ \mathrm{nil}^{\mathcal{P}} \\ \Gamma \vdash c : \forall x\ (x^{\mathcal{P}} : A^{\mathcal{P}})\ l\ (l^{\mathcal{P}} : \mathrm{list}^{\mathcal{P}}\ A\ A^{\mathcal{P}}\ l).\ P\ l\ l^{\mathcal{P}} \to P\ (\mathrm{cons}^\bullet\ x\ l)\ (\mathrm{cons}^{\mathcal{P}}\ x\ x^{\mathcal{P}}\ l\ l^{\mathcal{P}})\end{array}}{\Gamma \vdash \mathrm{list\text{-}elim}^{\mathcal{P}}(l^{\mathcal{P}}, P, n, c) : P\ l\ l^{\mathcal{P}}}$$

Fig. 13. Parametricity predicate of lists (target)

$[\![\mathrm{gS}\ n]\!]_{\mathcal{P}} = \mathrm{S}^{\mathcal{P}}\ [\![n]\!]_v\ [\![n]\!]_{\mathcal{P}}$ in the translation of c. We finally obtain the desired result by using the equality contained in $[\![v]\!]_{\mathcal{P}}$.

The cases for $\mathbb{G}$ and $\mathbb{P}$ are similar and the proof terms not particularly enlightening (and they do not carry any computational information since they are proofs) so we omit them.

Remark that the parametricity translation of vectors is the first time we really see the use of the revival translation: without it there would be no way to recover the length to put inside the predicate.

This translation echoes the alternative representation of vectors as a refinement type with a condition on the length:

$$\mathrm{vec}\ A\ n \approx \{l : \mathrm{list}\ A \mid \mathrm{length}\, l = n\}.$$

There is a trade-off between the two versions: the inductive one makes it easier in theory to produce correct-by-construction results, but relevant terms end up being polluted by proofs, which can be tedious to write and manage (for instance, they might get in the way of equality or computation). We argue that our setting allows one to get the best of both worlds, as we will illustrate in Section 5.2.2 and this will become even clearer in Section 6.

$$\frac{\Gamma \vdash P : \mathsf{Prop} \qquad \Gamma \vdash Q : \mathsf{Prop}}{\Gamma \vdash P \wedge Q : \mathsf{Prop}} \qquad \frac{\Gamma \vdash p : P \qquad \Gamma \vdash q : Q}{\Gamma \vdash \langle p, q \rangle : P \wedge Q}$$

$$\frac{\Gamma \vdash h : P \wedge Q \qquad \Gamma \vdash R : P \to Q \to \mathsf{Prop} \qquad \Gamma \vdash c : \forall (p : P)\ (q : Q).\ R\ (\langle p, q \rangle)}{\Gamma \vdash \wedge\text{-elim}(h, R, c) : R\ h}$$

Fig. 14. Conjunction (target)

5.2.2 Examples. We start by showing that even though the length index is erased, we can still take advantage of it to define the usual head and tail total functions. To do it, we have to rely on exfalso but also on the discriminator discr introduced in Section 5.1.

$$\begin{array}{l} \mathrm{head} : \mathrm{vec}\ A\ (\mathrm{gS}\ n) \to A \\ \mathrm{tail} : \mathrm{vec}\ A\ (\mathrm{gS}\ n) \to \mathrm{vec}\ A\ n \end{array}$$

head is defined below:

$$\begin{array}{lcl} P & := & \lambda m\ w.\ \forall (k : \mathrm{erased\ nat}).\ m \approx \mathrm{gS}\ k \to A \\ p & := & \lambda k\ (h : \mathrm{erase}\ 0 \approx \mathrm{gS}\ k).\ \mathrm{exfalso}_{\mathbb{T}}(A, \mathrm{discr}\ k\ h) \\ q & := & \lambda a\ n\ v\ \_\ k\ h.\ a \\ \mathrm{head} & := & \lambda (v : \mathrm{vec}\ A\ (\mathrm{gS}\ n)).\ \mathrm{vec\text{-}elim}_{\mathbb{T}}(v, P, p, q)\ n\ (\mathrm{gh\text{-}refl}\ (\mathrm{gS}\ n)) \end{array}$$

tail has a similar definition but we also use a cast to rewrite in the length index of the vector.

$$\begin{array}{lcl} P & := & \lambda m\ w.\ \forall (k : \mathrm{erased\ nat}).\ m \approx \mathrm{gS}\ k \to \mathrm{vec}\ A\ k \\ p & := & \lambda k\ h.\ \mathrm{exfalso}_{\mathbb{T}}(\mathrm{vec}\ A\ k, \mathrm{discr}\ k\ h) \\ q & := & \lambda a\ n\ v\ \_\ k\ h.\ \mathrm{cast}(e, \lambda x.\ \mathrm{vec}\ A\ x, v) \\ \mathrm{tail} & := & \lambda (v : \mathrm{vec}\ A\ (\mathrm{gS}\ n)).\ \mathrm{vec\text{-}elim}_{\mathbb{T}}(v, P, p, q)\ n\ (\mathrm{gh\text{-}refl}\ (\mathrm{gS}\ n)) \end{array}$$

where $e : n \approx k$ is obtained from $h : \mathrm{gS}\ n \approx \mathrm{gS}\ k$.

Notice that the two functions erase as one would expect by raising an exception in the nil case:

$$\begin{array}{lcl} [\mathrm{head}\ v]_\varepsilon & \equiv & \mathrm{list\text{-}elim}^\bullet([v]_\varepsilon, \lambda\_.[\![A]\!]_\varepsilon, [A]_\emptyset, \lambda a\ v\ \_.\ a, [A]_\emptyset) \\ [\mathrm{tail}\ v]_\varepsilon & \equiv & \mathrm{list\text{-}elim}^\bullet([v]_\varepsilon, \lambda\_.\mathrm{list}^\bullet\ [\![A]\!]_\varepsilon, \mathrm{list}_\emptyset, \lambda a\ v\ \_.\ v, \mathrm{list}_\emptyset) \end{array}$$

Finally, we bring closure to our initial example of reversal of vectors and show how we define it in GTT:[5]

$$\begin{array}{l} \mathrm{rev} : \forall^{\mathbb{T}}\ n\ m.\ \mathrm{vec}\ A\ n \to \mathrm{vec}\ A\ m \to \mathrm{vec}\ A\ (n \oplus m) \\ \mathrm{rev}\ n\ m\ \mathrm{vnil}\ acc := acc \\ \mathrm{rev}\ (\mathrm{gS}\ k)\ m\ (\mathrm{vcons}\ a\ k\ w)\ acc := \mathrm{cast}(e, \lambda x.\ \mathrm{vec}\ A\ x, rec\ (\mathrm{gS}\ m)\ (\mathrm{vcons}\ a\ m\ acc)) \end{array}$$

where $e : k \oplus \mathrm{gS}\ m \approx \mathrm{gS}\ k \oplus m$ is standard, modulo reveal. In the next section we will show how the ideal version without cast can also be supported. However, we can already argue that the cast is no more than a crutch for the type checker and that it can safely be ignored for any subsequent proofs about rev. Equational reasoning is highly simplified when compared to the version with a transport over proof-relevant equality.

[5]The version using eliminators explicitly is given in Figure 16 in Appendix B.

6 GHOST REFLECTION

As stated in the introduction, one of the main motivations for this work is to be able to obtain a nicer notion of extensional type theory using ghost types. We now introduce GRTT (ghost reflection type theory) as a variant of GTT presented in Section 2. We will show how GRTT is conservative over GTT, allowing us to lift the properties we established for GTT in Section 4 to GRTT. We can do it because GTT has been set up *just right* so that it can support extensionality. In a sense, this section is going to be a justification for the rules of GTT.

6.1 Definition of GRTT

GRTT is a variant of GTT where we remove cast from syntax as well as the rules highlighted in blue from Figures 1 and 2. We annotate the turnstile of GRTT judgements with an x to differentiate them from GTT judgments.

Instead of casts, ghost equality is eliminated through a special *ghost equality reflection* rule that let us convert between types which have equal ghost values inside.

$$\frac{\Gamma \vdash_x e : u \approx_A v \qquad \Gamma \vdash_x P : A \to s \qquad \Gamma \vdash_x t : P\ u \qquad \lfloor s \rfloor \neq \mathbb{K}}{\Gamma \vdash_x t : P\ v}$$

This rule mimics the cast typing rule and has the same limitation that it cannot be used in Kind. This way, conservativity of GRTT over GTT essentially states that the recasting rule is indeed sufficient to ensure that casts do not get in the way of equality or computation. The reader may find it surprising that we limit reflection to applicative contexts instead of the usual generality of ETT; in other words we cannot use reflection of an equality that is only well formed under binders. The main reason is that it corresponds exactly to the GTT model we are about to give. If we wanted to allow for reflection of ghost equalities under binders we would need to extend GTT with extra extensionality principles that we conjecture would in turn require at least function extensionality in the target of the parametricity translation. For now we chose to limit ourselves to a more agnostic notion of equality that we in any case believe is enough for most applications and leave the investigation of other approaches to future work.

GRTT can thus be seen as a restriction of the regular ETT where reflection is limited only to certain positions such as the length index of vectors. Our running example of rev can be expressed in GRTT just as it was in Section 1.

6.2 Potential translations

We adapt—and mostly simplify—the proof of Oury [2005]; Winterhalter et al. [2019] to translate *derivations* in GRTT to derivations in GTT. The main idea being replacing the use of reflection with casts in the target. As Oury; Winterhalter et al. already discuss in their case, this means *a priori* that the same term (resp. context or type) can have several valid translations and typically vary in which casts appear in the term. Our own proof can become much simpler thanks to the fact that ghost casts are irrelevant for conversion. We can thus prove that all translations of the same term are definitionally equal.

We start making this fact precise by defining *potential translations* of a GRTT term t as GTT terms t' such that $|t'| =_\alpha t$. Thus, two potential translations of a same term are definitionally equal per the recasting rule of GTT.

We define potential translations of judgments as follows:

(1) $\vdash \Gamma' \in [\![\vdash_x \Gamma]\!]_x$ when $\vdash \Gamma'$ is derivable and $|\Gamma'| =_\alpha \Gamma$ (pointwise);
(2) $\Gamma' \vdash t' : A' \in [\![\Gamma \vdash_x t : A]\!]_x$ when it is derivable and $|\Gamma'| =_\alpha \Gamma$, and $|t'| =_\alpha t$, and $|A'| =_\alpha A$.

Note that we do not need to provide a translation to conversion since it is purely syntactic and ignores casts, meaning we can use them *as is*.

We will now define the two crucial lemmas for conducting the translation. They essentially state that the translation does not get in the way of typing.

Lemma 6.1 (Choice of type). *If* $\Gamma' \vdash t' : A' \in [\![\Gamma \vdash_x t : A]\!]_x$ *and* $\Gamma' \vdash A'' : s \in [\![\Gamma \vdash_x A : s]\!]_x$ *with* $\mathrm{md}(t) = \lfloor s \rfloor$ *then we also have* $\Gamma' \vdash t' : A'' \in [\![\Gamma \vdash_x t : A]\!]_x$.

Proof. We apply the conversion rule by remarking that $A' \equiv A''$ holds since $|A'| =_\alpha |A''|$. □

Lemma 6.2 (Preservation of type formers). *If* $\Gamma' \vdash t' : A' \in [\![\Gamma \vdash_x t : A]\!]_x$ *where* A *has a type former in its head then* A' *has the same head as* A.

Proof. Since $|A'| =_\alpha A$ we know that $A' =_\alpha \mathrm{cast}(e_1, P_1, \ldots \mathrm{cast}(e_n, P_n, A''))$ for some A'' with the same head as A. By Lemma 2.7 we also know that $\Gamma' \vdash A' : s$ for some sort s such that $\mathrm{md}(t) = \lfloor s \rfloor$. Using Lemma 4.3 we thus get that $\mathrm{md}(A') = \mathbb{K}$. By inversion of typing on the cast rule, we would however get that this mode cannot be $\mathbb{K}$. Hence, $n = 0$ (no casts were applied) and $A' =_\alpha A''$. □

6.3 Eliminating ghost reflection

We now define a translation from GRTT derivations to GTT derivations by showing that whenever a judgment $\mathcal{J}$ is derivable in GRTT, then $[\![\mathcal{J}]\!]_x$ is inhabited, *i.e.* $\mathcal{J}$ has a translation in GTT.

Theorem 6.3 (Translation from GRTT to GTT). *The following assertions hold.*

(1) If $\vdash_x \Gamma$, *then there exists* $\vdash \Gamma' \in [\![\vdash_x \Gamma]\!]_x$.

(2) If $\Gamma \vdash_x t : A$, *then for all* $\vdash \Gamma' \in [\![\vdash_x \Gamma]\!]_x$, *there exists* $\Gamma' \vdash t' : A' \in [\![\Gamma \vdash_x t : A]\!]_x$.

Proof. We prove the assertions above by induction on the derivations and freely use Lemmas 6.1 and 6.2 to align the various translations. We show a few representative cases.

The most interesting case is probably the ghost reflection rule. By induction hypothesis we have $\Gamma' \vdash e' : u' \approx_{A'} v' \in [\![\Gamma \vdash_x e : u \approx_A v]\!]_x$ and $\Gamma' \vdash P' : A' \to s \in [\![\Gamma \vdash_x P : A \to s]\!]_x$ and $\Gamma' \vdash t' : P'\ u' \in [\![\Gamma \vdash_x t : P\ u]\!]_x$ with $\lfloor s \rfloor \neq \mathbb{K}$. We can then easily conclude that $\Gamma' \vdash \mathrm{cast}(e', P', t') : P'\ v' \in [\![\Gamma \vdash_x t : P\ v]\!]_x$.

The conversion rule is also interesting because it shows how we do not need to translate definitional equality. By induction hypothesis we have $\Gamma' \vdash t' : A' \in [\![\Gamma \vdash_x t : A]\!]_x$ as well as $\Gamma' \vdash B' : s \in [\![\Gamma \vdash_x B : s]\!]_x$ with $\mathrm{md}(t) = \lfloor s \rfloor$. In particular we have $|A'| =_\alpha A \equiv B =_\alpha |B'|$ which gives us $A' \equiv B'$. We also have $\mathrm{md}(t') = \mathrm{md}(t) = \lfloor s \rfloor$, hence we can apply the conversion rule to conclude $\Gamma' \vdash t' : B' \in [\![\Gamma \vdash_x t : B]\!]_x$.

The remaining cases are straightforward, with the only obstacle being the rule for application where one has to use Lemma 2.3. □

6.4 Conservativity and meta-theoretical results

We now establish some meta-theoretical consequences of the GRTT to GTT translation. We show that GRTT is conservative over GTT, meaning that GRTT captures *exactly* what can be proven in GTT, they have the same logical power.

Theorem 6.4 (Conservativity). *Whenever we have valid type in GTT* $\vdash A : s$ *such that its cast-free version is inhabited in GRTT* $\vdash_x t : |A|$ *with* $\mathrm{md}(t) = \lfloor s \rfloor$ *then it is already inhabited in GTT,* i.e. *there is some* t' *such that* $|t'| =_\alpha t$ *and* $\vdash t' : A$.

Proof. Assume $\vdash A : s$ and $\vdash_x t : |A|$, then by applying Theorem 6.3 we have some $\vdash t' : A' \in [\![\vdash_x t : |A|]\!]_x$. Besides, we know that $\vdash A : s \in [\![\vdash_x |A| : s]\!]_x$ so by Lemma 6.1 we get $\vdash t' : A \in [\![\vdash_x t : |A|]\!]_x$. □

There is one caveat: the term t in GRTT needs to be in the right mode. This constraint is not a problem in practice because for concrete types like $\bot$ we will get if for free (using Lemma 4.3). Thanks to that we can prove consistency of GRTT.

Theorem 6.5 (Consistency of GRTT). *GRTT is consistent.*

Proof. Assuming $\vdash_x t : \bot$, then by Lemma 4.3 we also get $\mathrm{md}(t) = \mathbb{G}$ and thus we can apply Theorem 6.4 to get $\vdash t' : \bot$ which would disprove consistency of GTT (Theorem 4.1). □

7 RELATED WORK

Ghost types, erasure and shape irrelevance with dependent types. Shape irrelevance was introduced by Abel et al. [2017b] for sized types to be able to say that the size argument in a type does not affect its shape (*e.g.* one cannot match on a size to build a type that would be either nat or bool). Shape irrelevance was later refined by Nuyts and Devriese [2018] in the more general framework of modalities, from which one may extract a notion of heterogeneous equality. It shares similarities with the *erased modality* in that both are used to represent data that is irrelevant for computation but still holds meaning as a specification. Both are implemented in Agda, and in fact Agda's implementation of the erased modality is a special case of Quantitative type theory (QTT) [Atkey 2018] which is also implemented in Idris 2 [Brady 2021]. It tracks variable usage in judgements, separating occurrences in the term from the ones in the type. Compared to GTT, all of these cases are not type-based—although we believe that our proof should be adaptable to modalities—and to the best of our knowledge do not consider extensionality as we did. The ability to distinguish relevance in the term and in the type however is convenient to be able to consider *e.g.* the length arguments in vector *terms* to be irrelevant. For us, it is only achieved in GRTT for *propositionally equal* sizes, in other words for terms of the same type (which should not be a restriction in practice).

Ghost types in F* [Swamy et al. 2016] served as an inspiration for our Ghost universe, but there remains several differences. Indeed, in F*, *ghost* is an effect rather than a universe and it comes with implicit lifts to it so that ghost computations can be eliminated into any *non-informative* type. In contrast to GTT, it considers the universe to be non-informative, meaning that ghost computations can be examined not only to produce propositions but also any type. This is possible because the erasure they consider gets rid of all types, unlike what we do. However there is also an important similarity in how both F* and GRTT handle equality reflection: they both cannot freely go under binders.

Miquel [2001] introduced the Implicit Calculus of Constructions which features a special dependent product type whose arguments are not materialised and which behaves as an intersection type. There is no need to erased these arguments because they are not there in the first place. Cedille [Stump 2017; Stump and Jenkins 2018] also proposes intersection types, but this time it relies on a primitive notion of erasure for terms that discards types. Both systems offer a notion of erasure that is different to that of GTT, with different kinds of applications, perhaps more prominently for impredicative encodings. We instead focused on reasoning principles about erased data, which is what casts (or reflection) give us.

Extensionality for equality. Besides equality reflection, another related but mostly orthogonal line of work on extensionality is that of observational equality (OTT) [Altenkirch et al. 2007; Pujet and Tabareau 2022, 2023]. In a sense, ghost equality in GTT is very close to observational equality and it also uses casts as an eliminator. But contrarily to observational equality, ghost equality does not compute on type formers and ghost casts can be ignored whereas OTT casts need to compute on types and even contain non-linear rewrite rules to get rid of trivial casts. It would be interesting

to see how the two features would combine, in particular this might help dealing with binders in conversion.

8 CONCLUSION AND FUTURE WORK

The various translations we presented in the paper provide a model both for GTT and GRTT that we can view as two variants of the same system. GTT is better suited to serve as a kernel to a proof assistant while GRTT might be better as a practical system, despite the fact that type checking is obviously undecidable (as it is for type systems with equality reflection). Indeed, there are many ways to make this system practical, be it with definitional equality handlers *à la* Andromeda 1 [Bauer et al. 2016], or by delegating such proof obligations to an SMT solver *à la* F* [Swamy et al. 2016] or even via the use of (user-defined) rewrite rules. A general take-away from this is that automation confined to erased positions remains powerful and useful but crucially does not get in the way of equational reasoning or of extracted programs. The latter point is reinforced by the erasure translation we propose that gets rid of both proofs and ghost values and could as such serve as a first step towards extraction.

We thus believe that this and future work in that direction could help justify and inform the design and meta-theory of tools like F* or expand the capabilities of other proof assistants such as Coq or Agda. There are still several challenges to tackle before implementation. Perhaps the most prominent and achievable one would be to extend G(R)TT to support general inductive types. As these proofs are usually tricky, we would highly benefit from a formalisation such as that of MetaCoq [Sozeau et al. 2020a,b] which deals with most features of Coq. A general treatment of indices would be of particular interest, especially given the fact that some dependencies are erased even after revival. We could then hope to tackle other properties of GTT than the ones we establish with the current model. One of them is termination. We conjecture that GTT—and even GRTT (as opposed to ETT in general)—have a proper notion of computation and that they are in fact strongly normalising. We could prove this fact from our model by showing that it is in fact a simulation for reduction. In case this would prove insufficient to prove decidability of type checking—this time only for GTT—we could have a look at other methods such as logical relations [Abel et al. 2017a]. While decidability of type checking is out of reach for GRTT, we believe that what is decidable is to determine which proof obligations are required to make a term type-check. In other words we could decide where casts are necessary and type former discrimination would ensure that the obligations make sense and are typically about indices of inductive types, or constraints of refinement types.

Another point to sort out before implementation is the way we deal with binders for ghost reflection. As stated in the paper, the ghost reflection rule does not give rise directly to an implementation. Indeed, it should rather be merged with conversion to properly work. This means for one keeping track of the context, but also making a distinction between which variables were introduced by congruence so that they are not available in the context of the proof obligation. Alternatively, we could allow going under binder at the cost of—presumably—function extensionality.

Finally, it would be interesting to extend GTT to support accessibility predicates as ghost types. Indeed, in Coq they are currently considered as inductive types in Prop, but not in SProp. In other words, it does not enjoy definitional proof irrelevance, but it is still erased during extraction, which makes it a perfect candidate for our Ghost universe. Such an extension would require more investigation as the target cannot readily accommodate for potentially non-terminating definitions that we would only show terminating after the fact (typically through the parametricity translation).

REFERENCES

Andreas Abel and Thierry Coquand. 2020. Failure of normalization in impredicative type theory with proof-irrelevant propositional equality. *Logical Methods in Computer Science* 16 (2020).

Andreas Abel, Joakim Öhman, and Andrea Vezzosi. 2017a. Decidability of conversion for type theory in type theory. *Proceedings of the ACM on Programming Languages* 2, POPL (2017), 1–29.

Andreas Abel, Andrea Vezzosi, and Theo Winterhalter. 2017b. Normalization by evaluation for sized dependent types. *Proceedings of the ACM on Programming Languages* 1, ICFP (2017), 1–30.

Thorsten Altenkirch, Paolo Capriotti, and Nicolai Kraus. 2016. Extending homotopy type theory with strict equality. *arXiv preprint arXiv:1604.03799* (2016).

Thorsten Altenkirch, Conor McBride, and Wouter Swierstra. 2007. Observational equality, now!. In *Proceedings of the 2007 workshop on Programming languages meets program verification*. 57–68.

Robert Atkey. 2018. Syntax and semantics of quantitative type theory. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*. 56–65.

Andrej Bauer, Gaëtan Gilbert, Philipp G. Haselwarter, Matija Pretnar, and Chris Stone. 2016. The 'Andromeda' prover. http://www.andromeda-prover.org/

Simon Boulier, Pierre-Marie Pédrot, and Nicolas Tabareau. 2017. The Next 700 Syntactical Models of Type Theory. In *Certified Programs and Proofs – CPP 2017*. 182–194.

Edwin Brady. 2021. Idris 2: Quantitative type theory in practice. *arXiv preprint arXiv:2104.00480* (2021).

Jesper Cockx and Andreas Abel. 2016. Sprinkles of extensionality for your vanilla type theory. In *Types for Proofs and Programs (TYPES)*.

Jesper Cockx, Nicolas Tabareau, and Théo Winterhalter. 2021. The Taming of the Rew: A Type Theory with Computational Assumptions. *Proceedings of the ACM on Programming Languages* (2021). https://doi.org/10.1145/3434341

Robert L. Constable and Joseph L. Bates. 2014. The NuPrl system, PRL project. http://www.nuprl.org/

Coq development team. 2023. *The Coq proof assistant reference manual*. LogiCal Project. http://coq.inria.fr Version 8.17.

Gaëtan Gilbert, Jesper Cockx, Matthieu Sozeau, and Nicolas Tabareau. 2019. Definitional Proof-Irrelevance without K. *Proceedings of the ACM on Programming Languages* (Jan. 2019), 1–28. https://doi.org/10.1145/329031610.1145/3290316

Martin Hofmann. 1995. Conservativity of equality reflection over intensional type theory. In *International Workshop on Types for Proofs and Programs*. Springer, 153–164.

Chantal Keller and Marc Lasson. 2012. Parametricity in an impredicative sort. *arXiv preprint arXiv:1209.6336* (2012).

Alexandre Miquel. 2001. The implicit calculus of constructions extending pure type systems with an intersection type binder and subtyping. In *International Conference on Typed Lambda Calculi and Applications*. Springer, 344–359.

Ulf Norell. 2007. *Towards a practical programming language based on dependent type theory*. Vol. 32. Citeseer.

Andreas Nuyts and Dominique Devriese. 2018. Degrees of relatedness: A unified framework for parametricity, irrelevance, ad hoc polymorphism, intersections, unions and algebra in dependent type theory. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*. 779–788.

Nicolas Oury. 2005. Extensionality in the calculus of constructions. In *International Conference on Theorem Proving in Higher Order Logics*. Springer, 278–293.

Pierre-Marie Pédrot and Nicolas Tabareau. 2018. Failure is Not an Option An Exceptional Type Theory. In *ESOP 2018 - 27th European Symposium on Programming (LNCS, Vol. 10801)*. Springer, Thessaloniki, Greece, 245–271. https://doi.org/10.1007/978-3-319-89884-1_9

Frank Pfenning. 2001. Intensionality, extensionality, and proof irrelevance in modal type theory. In *Proceedings 16th Annual IEEE Symposium on Logic in Computer Science*. IEEE, 221–230.

Loïc Pujet and Nicolas Tabareau. 2022. Observational equality: now for good. *Proceedings of the ACM on Programming Languages* 6, POPL (2022), 1–27.

Loïc Pujet and Nicolas Tabareau. 2023. Impredicative Observational Equality. *Proceedings of the ACM on Programming Languages* 7, POPL (2023), 2171–2196.

Matthieu Sozeau, Abhishek Anand, Simon Boulier, Cyril Cohen, Yannick Forster, Fabian Kunze, Gregory Malecha, Nicolas Tabareau, and Théo Winterhalter. 2020a. The MetaCoq Project. *Journal of Automated Reasoning* (Feb. 2020). https://doi.org/10.1007/s10817-019-09540-0

Matthieu Sozeau, Simon Boulier, Yannick Forster, Nicolas Tabareau, and Théo Winterhalter. 2020b. Coq Coq Correct! Verification of Type Checking and Erasure for Coq, in Coq. *Proceedings of the ACM on Programming Languages* (Jan. 2020), 1–28. https://doi.org/10.1145/3371076

Pierre-Yves Strub. 2010. Coq modulo theory. In *Computer Science Logic: 24th International Workshop, CSL 2010, 19th Annual Conference of the EACSL, Brno, Czech Republic, August 23-27, 2010. Proceedings 24*. Springer, 529–543.

Aaron Stump. 2017. The calculus of dependent lambda eliminations. *Journal of Functional Programming* 27 (2017), e14.

Aaron Stump and Christopher Jenkins. 2018. Syntax and semantics of Cedille. *arXiv preprint arXiv:1806.04709* (2018).

Nikhil Swamy, Cătălin Hriţcu, Chantal Keller, Aseem Rastogi, Antoine Delignat-Lavaud, Simon Forest, Karthikeyan Bhargavan, Cédric Fournet, Pierre-Yves Strub, Markulf Kohlweiss, Jean-Karim Zinzindohoué, and Santiago Zanella-Béguelin. 2016. Dependent Types and Multi-Monadic Effects in F*. In *43rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*. ACM, 256–270. https://www.fstar-lang.org/papers/mumon/

Théo Winterhalter. 2020. *Formalisation and meta-theory of type theory.* Ph. D. Dissertation. Université de Nantes.

Théo Winterhalter, Matthieu Sozeau, and Nicolas Tabareau. 2019. Eliminating Reflection from Type Theory. In *CPP 2019 - 8th ACM SIGPLAN International Conference on Certified Programs and Proofs.* ACM, Lisbonne, Portugal, 91–103. https://doi.org/10.1145/3293880.3294095

A CONGRUENCE RULES OF GTT

Besides the rules of Figures 1 and 2, we have the following congruence rules for conversion. There is no congruence rule for cast because the recasting rule already deals with that.

$$\frac{}{u \equiv u} \qquad \frac{v \equiv u}{u \equiv v} \qquad \frac{u \equiv v \qquad v \equiv w}{u \equiv w} \qquad \frac{A \equiv A' \qquad B \equiv B'}{\forall^{\mathsf{r}}(x^{\mathsf{s}} : A).B \equiv \forall^{\mathsf{r}}(x^{\mathsf{s}} : A').B'}$$

$$\frac{A \equiv A' \qquad t \equiv t'}{\lambda(x^{\mathsf{s}} : A).t \equiv \lambda(x^{\mathsf{s}} : A').t'} \qquad \frac{t \equiv t' \qquad u \equiv u'}{t\ u \equiv t'\ u'} \qquad \frac{A \equiv A' \qquad p \equiv p'}{\mathrm{exfalso}_{\mathsf{s}}(A, p) \equiv \mathrm{exfalso}_{\mathsf{s}}(A', p')}$$

$$\frac{A \equiv A'}{\mathrm{erased}\ A \equiv \mathrm{erased}\ A'} \qquad \frac{t \equiv t'}{\mathrm{erase}\ t \equiv \mathrm{erase}\ t'} \qquad \frac{t \equiv t' \qquad P \equiv P' \qquad p \equiv p'}{\mathrm{reveal}(t, P, p) \equiv \mathrm{reveal}(t', P', p')}$$

$$\frac{t \equiv t' \qquad p \equiv p'}{\mathrm{reveal}_*(t, p) \equiv \mathrm{reveal}_*(t', p')} \qquad \frac{A \equiv A' \qquad u \equiv u' \qquad v \equiv v'}{u \approx_A v \equiv u' \approx_{A'} v'} \qquad \frac{A \equiv A' \qquad u \equiv u'}{\mathrm{gh\text{-}refl}_A\ u \equiv \mathrm{gh\text{-}refl}_{A'}\ u'}$$

B COMPLETE DEFINITIONS

We provide here the full definitions of cast removal (Figure 15) and of rev (Figure 16).

$$|x^{\mathsf{s}}| := x^{\mathsf{s}} \qquad |\mathrm{Kind}_i| := \mathrm{Kind}_i \qquad |\mathrm{Type}_i| := \mathrm{Type}_i \qquad |\mathrm{Ghost}_i| := \mathrm{Ghost}_i \qquad |\mathrm{Prop}| := \mathrm{Prop}$$

$$|\forall^{\mathsf{r}}(x^{\mathsf{s}} : A).B| := \forall^{\mathsf{r}}(x^{\mathsf{s}} : |A|).|B| \qquad |\lambda(x^{\mathsf{s}} : A).t| := \lambda(x^{\mathsf{s}} : |A|).|t| \qquad |t\ u| := |t|\ |u|$$

$$|\mathrm{erased}\ A| := \mathrm{erased}\ |A| \qquad |\mathrm{reveal}(t, P, p)| := \mathrm{reveal}(|t|, |P|, |p|)$$

$$|\mathrm{reveal}_*(t, p)| := \mathrm{reveal}(|t|, |p|) \qquad |u \approx_A v| := |u| \approx_{|A|} |v| \qquad |\mathrm{gh\text{-}refl}_A\ u| := \mathrm{gh\text{-}refl}_{|A|}\ |u|$$

$$|\mathrm{cast}(e, P, t)| := |t| \qquad |\bot| := \bot \qquad |\mathrm{exfalso}_{\mathsf{s}}(A, p)| := \mathrm{exfalso}_{\mathsf{s}}(|A|, |p|)$$

Fig. 15. Definition of cast removal from Section 2.2.

$$\begin{array}{lcl}
\mathrm{rev} & : & \forall^{\mathbb{T}}\ n\ m.\ \mathrm{vec}\ A\ n \to \mathrm{vec}\ A\ m \to \mathrm{vec}\ A\ (n \oplus m) \\
\mathrm{rev} & := & \lambda n\ m\ v.\ \mathrm{vec\text{-}elim}(v, \lambda\ k\ w.\ \forall m.\ \mathrm{vec}\ A\ m \to \mathrm{vec}\ A\ (k \oplus m), b_{\mathrm{vnil}}, b_{\mathrm{vcons}})\ m \\
b_{\mathrm{vnil}} & := & \lambda\ m\ acc.\ acc \\
b_{\mathrm{vcons}} & := & \lambda\ a\ k\ w\ rec\ m\ acc.\ \mathrm{cast}(e, \lambda x.\ \mathrm{vec}\ A\ x, rec\ (\mathrm{gS}\ m)\ (\mathrm{vcons}\ a\ m\ acc)) \\
e & : & k \oplus \mathrm{gS}\ m \approx \mathrm{gS}\ k \oplus m
\end{array}$$

Fig. 16. Definition of rev from Section 5.2.

Parametricity of casts. In Section 3.3 we only gave the parametricity translation of $\mathrm{cast}(e, P, t)$ when $\mathrm{md}(t) = \mathbb{T}$. This leaves us with two[6] other cases to handle: $\mathbb{G}$ and $\mathbb{P}$.

[6]Casts cannot be used in mode $\mathbb{K}$.

For $[\![\mathrm{cast}(e,P,t^{\mathbb{G}})]\!]_{\mathcal{P}}$, the expected type is:

$$\begin{aligned}[\![P\ v]\!]_{\mathcal{P}}\ [\![\mathrm{cast}(e,P,t)]\!]_v &= [\![P]\!]_{\mathcal{P}}\ [\![v]\!]_v\ [\![v]\!]_{\mathcal{P}}\ [\![\mathrm{cast}(e,P,t)]\!]_v \\ &= [\![P]\!]_{\mathcal{P}}\ [\![v]\!]_v\ [\![v]\!]_{\mathcal{P}}\ [\![t]\!]_v\end{aligned}$$

We also compute the type of $[\![P]\!]_{\mathcal{P}}$:

$$\begin{aligned}[\![A \to \mathrm{Ghost}]\!]_{\mathcal{P}}\ [P]_{\varepsilon} &= \forall (x : [\![A]\!]_{\varepsilon})\ (\overline{x} : [\![A]\!]_{\mathcal{P}}\ x).\ [\![\mathrm{Ghost}]\!]_{\mathcal{P}}\ [P]_{\varepsilon} \\ &= \forall (x : [\![A]\!]_{\varepsilon})\ (\overline{x} : [\![A]\!]_{\mathcal{P}}\ x).\ [\![P]\!]_{\varepsilon} \to \mathrm{Prop}\end{aligned}$$

Thus, $[\![P]\!]_{\mathcal{P}}\ [\![v]\!]_v\ [\![v]\!]_{\mathcal{P}}\ [\![t]\!]_v$ is a proposition so we can also eliminate the squash from $[\![e]\!]_{\mathcal{P}}$. Hence we define $[\![\mathrm{cast}(e,P,t^{\mathbb{G}})]\!]_{\mathcal{P}}$ as

$$\mathrm{sq\text{-}elim}([\![e]\!]_{\mathcal{P}},\ \lambda\_.\ [\![P]\!]_{\mathcal{P}}\ [\![v]\!]_v\ [\![v]\!]_{\mathcal{P}}\ [\![t]\!]_v,\ \lambda(e' : [\![u]\!]_v = [\![v]\!]_v).\ w)$$

where $w : [\![P]\!]_{\mathcal{P}}\ [\![v]\!]_v\ [\![v]\!]_{\mathcal{P}}\ [\![t]\!]_v$ is defined as

$$\mathrm{J}(e',\ \lambda\ (a : [\![A]\!]_{\varepsilon})\ p.\ \forall (h : [\![A]\!]_{\mathcal{P}}\ a).\ [\![P]\!]_{\mathcal{P}}\ a\ h\ [\![t]\!]_v,\ \lambda(h : [\![A]\!]_{\mathcal{P}}\ [\![u]\!]_v).\ [\![t]\!]_{\mathcal{P}})\ [\![v]\!]_{\mathcal{P}}.$$

We again use proof irrelevance.

For $[\![\mathrm{cast}(e,P,t^{\mathbb{P}})]\!]_{\mathcal{P}}$, the expected type is:

$$[\![P\ v]\!]_{\mathcal{P}} = [\![P]\!]_{\mathcal{P}}\ [\![v]\!]_v\ [\![v]\!]_{\mathcal{P}}$$

We also compute the type of $[\![P]\!]_{\mathcal{P}}$:

$$\begin{aligned}[\![A \to \mathrm{Prop}]\!]_{\mathcal{P}}\ [P]_{\varepsilon} &= \forall (x : [\![A]\!]_{\varepsilon})\ (\overline{x} : [\![A]\!]_{\mathcal{P}}\ x).\ [\![\mathrm{Prop}]\!]_{\mathcal{P}}\ [P]_{\varepsilon} \\ &= \forall (x : [\![A]\!]_{\varepsilon})\ (\overline{x} : [\![A]\!]_{\mathcal{P}}\ x).\ \mathrm{Prop}\end{aligned}$$

Thus, $[\![P]\!]_{\mathcal{P}}\ [\![v]\!]_v\ [\![v]\!]_{\mathcal{P}}$ is a proposition so we eliminate the squash from $[\![e]\!]_{\mathcal{P}}$ and define $[\![\mathrm{cast}(e,P,t^{\mathbb{P}})]\!]_{\mathcal{P}}$ as

$$\mathrm{sq\text{-}elim}([\![e]\!]_{\mathcal{P}},\ \lambda\_.\ [\![P]\!]_{\mathcal{P}}\ [\![v]\!]_v\ [\![v]\!]_{\mathcal{P}},\ \lambda(e' : [\![u]\!]_v = [\![v]\!]_v).\ w)$$

where $w : [\![P]\!]_{\mathcal{P}}\ [\![v]\!]_v\ [\![v]\!]_{\mathcal{P}}$ is defined as

$$\mathrm{J}(e',\ \lambda\ (a : [\![A]\!]_{\varepsilon})\ p.\ \forall (h : [\![A]\!]_{\mathcal{P}}\ a).\ [\![P]\!]_{\mathcal{P}}\ a\ h,\ \lambda(h : [\![A]\!]_{\mathcal{P}}\ [\![u]\!]_v).\ [\![t]\!]_{\mathcal{P}})\ [\![v]\!]_{\mathcal{P}}.$$

We again use proof irrelevance.

The case of $[\![\mathrm{cast}(e,P,t^{\mathbb{K}})]\!]_{\mathcal{P}}$ is slightly different. It will never be well typed, however defining it as ■ would cause problems in the proof of Lemma 3.11, so instead we define it as $[\![t]\!]_{\mathcal{P}}$. We are free to do so since it doesn't affect well-typed terms.

Definition of $\mathrm{length}^{\bullet}$. This length function operating on lists with exceptions is used in Section 5.2 for the parametricity translation of vectors. It is quite standard if one forgets about exceptions:

$$\begin{aligned}&\mathrm{length}^{\bullet} : \forall A.\ \mathrm{list}^{\bullet}\ A \to \mathrm{nat}^{\bullet} \\ &\mathrm{length}^{\bullet}\ A\ l := \mathrm{list\text{-}elim}^{\bullet}(l, \lambda\ \_.\ \mathrm{nat}^{\bullet}, 0^{\bullet}, \lambda\ \_\ \_\ n.\ \mathrm{S}^{\bullet}\ n, \mathrm{nat}_{\emptyset})\end{aligned}$$

Definition of $\mathrm{length}^{\mathcal{P}}$. We then show that $\mathrm{length}^{\bullet}$ is parametric by defining the following $\mathrm{length}^{\mathcal{P}}$ predicate. It is a simple proof by induction on the parametricity predicate of the list.

$$\begin{aligned}&\mathrm{length}^{\mathcal{P}} : \forall A\ A^{\mathcal{P}}\ (l : \mathrm{list}^{\bullet}\ A)\ (l^{\mathcal{P}} : \mathrm{list}^{\mathcal{P}}\ A\ A^{\mathcal{P}}).\ \mathrm{nat}^{\mathcal{P}}\ (\mathrm{length}^{\bullet}\ l) \\ &\mathrm{length}^{\bullet}\ A\ A^{\mathcal{P}}\ l\ l^{\mathcal{P}} := \mathrm{list\text{-}elim}^{\mathcal{P}}(l^{\mathcal{P}}, \lambda x\ h.\ \mathrm{nat}^{\mathcal{P}}\ (\mathrm{length}^{\bullet}\ x), 0^{\mathcal{P}}, \lambda a\ a^{\mathcal{P}}\ t\ t^{\mathcal{P}}\ h.\ \mathrm{S}^{\mathcal{P}}\ (\mathrm{length}^{\bullet}\ t)\ h)\end{aligned}$$