Trocq: Proof Transfer for Free, With or Without Univalence

Cyril Cohen, Enzo Crance, Assia Mahboubi

Trocq: Proof Transfer for Free, With or Without Univalence

CYRIL COHEN*, Université Côte d'Azur, Inria, France
ENZO CRANCE*, Mitsubishi Electric R&D Centre Europe, France and Nantes Université, École Centrale Nantes, CNRS, INRIA, LS2N, UMR 6004, France
ASSIA MAHBOUBI*, Nantes Université, École Centrale Nantes, CNRS, INRIA, LS2N, UMR 6004, France

Libraries of formalized mathematics use a possibly broad range of different representations for a same mathematical concept. Yet light to major manual input from users remains most often required for obtaining the corresponding variants of theorems, when such obvious replacements are typically left implicit on paper. This article presents Trocq, a new proof transfer framework for dependent type theory. Trocq is based on a novel formulation of type equivalence, used to generalize the univalent parametricity translation. This framework takes care of avoiding dependency on the axiom of univalence when possible, and may be used with more relations than just equivalences. We have implemented a corresponding plugin for the Coq proof assistant, in the Coq-Elpi meta-language. We use this plugin on a gallery of representative examples of proof transfer issues in interactive theorem proving, and illustrate how Trocq covers the spectrum of several existing tools, used in program verification as well as in formalized mathematics in the broad sense.

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

Additional Key Words and Phrases: Parametricity, Representation independence, Univalence, Proof assistants

1 INTRODUCTION

Formalized mathematics is the art of devising explicit data structures for every object and statement of the mathematical literature, in a certain choice of foundational formalism. As one would expect, several such explicit representations are most often needed for a same mathematical concept. Sometimes, these different choices are made explicit on paper: multivariate polynomials can for instance be represented as lists of coefficient-monomial pairs, *e.g.*, when computing Gröbner bases, but also as univariate polynomials with polynomial coefficients, *e.g.*, for the purpose of projecting algebraic varieties. The conversion between these equivalent data structures will however remain implicit on paper, as they code in fact for the same free commutative algebra. In some other cases, implementation details are just ignored on paper, *e.g.*, when a proof involves both reasoning with Peano arithmetic and computing with large integers.

Example 1.1 (Relating proof-oriented data-structures with computation-oriented ones). The standard library of the Coq proof assistant [The Coq Development Team 2022] actually proposes two data structures for representing natural numbers. Type $\mathbb{N}$ uses a unary representation, so that the associated elimination principle `N_ind` expresses the usual recurrence scheme:

```
Inductive ℕ : Type :=
  | O_ℕ : ℕ
  | S_ℕ (n : ℕ) : ℕ.

ℕ_ind : ∀ P : ℕ → □, P O_ℕ → (∀ n : ℕ, P n → P (S n)) → ∀ n : ℕ, P n.
```

*All authors contributed equally to this research.

Authors' addresses: Cyril Cohen, cyril.cohen@inria.fr, Université Côte d'Azur, Inria, 2004 route des Lucioles, Valbonne, France, 06902; Enzo Crance, Mitsubishi Electric R&D Centre Europe, Rennes, France, 35700 and Nantes Université, École Centrale Nantes, CNRS, INRIA, LS2N, UMR 6004, Nantes, France, 44300; Assia Mahboubi, AssiaMahboubi, Nantes Université, École Centrale Nantes, CNRS, INRIA, LS2N, UMR 6004, Nantes, France, 44000-F.

Type N uses a binary representation positive of non-negative integers, as sequences of bits with a head 1, and is thus better suited for coding efficient arithmetic operations. The successor function $S_N : N \to N$ is no longer a constructor of the type, but can be implemented as a program, via an auxiliary successor function S_{pos} for type positive.

```
Inductive positive : Type :=
  | xI : positive → positive (* p1 *)
  | x0 : positive → positive (* p0 *)
  | xH : positive.           (* 1  *)

Inductive N : Type :=
 | 0_N : N
 | Npos : positive → N.

Fixpoint S_pos (p : positive) : positive :=
  match p with xH ⇒ x0 xH | x0 p ⇒ xI p | xI p ⇒ x0 (S_pos p) end.

Definition S_N (n : N) := match n with Npos p ⇒ Npos (S_pos p) | _ ⇒ Npos xH end.
```

This successor is useful to implement conversions $\uparrow_{\mathbb{N}} : N \to \mathbb{N}$ and $\downarrow_{\mathbb{N}} : \mathbb{N} \to N$ between the unary and binary representations. These conversion functions are in fact inverses of each other. The natural recurrence scheme on natural numbers thus *transfers* to type N:

```
N_ind : ∀ P : N → □, P 0_N → (∀ n : N, P n → P (S_N n)) → ∀ n : N, P n.
```

Incidentally, N_ind can be proved from $\mathbb{N}$_ind by using only the fact that $\downarrow_{\mathbb{N}}$ is a left inverse of $\uparrow_{\mathbb{N}}$, and the following compatibility lemma:

$$\forall n : \mathbb{N}, \quad \downarrow_{\mathbb{N}} (S_{\mathbb{N}}\ n) = S_N\ (\downarrow_{\mathbb{N}}\ n)$$

Program verification supplies numerous examples of proof transfer use-cases, but this issue goes way beyond computational concerns. For instance, the formal study of summation and integration, in basic real analysis, provides a classic example of frustrating proof transfer bureaucracy.

Example 1.2 (Extended domains). Given a sequence $(u_n)_{n\in\mathbb{N}}$ of non-negative real numbers, *i.e.*, a function $u : \mathbb{N} \to [0, +\infty[$, u is said to be *summable* when the sequence $(\sum_{k=0}^{n} u_k)_{n\in\mathbb{N}}$ has a finite limit, denoted $\sum u$. Now for two summable sequences u and v, it is easy to see that $u + v$, the point-wise addition of u and v, is also a summable sequence, and that:

$$\sum(u + v) = \sum u + \sum v \tag{1}$$

Making the definition of the real number $\sum u$ depend on a summability witness does not scale, as every other algebraic operation "under the sum" then requires a new proof of summability. In a classical setting, the standard approach rather assigns a default value to the case of an infinite sum, for instance by introducing an extended domain $[0, +\infty]$, and extending the addition operation to the extra $+\infty$ case. Now for a sequence $u : \mathbb{N} \to [0, +\infty]$, the limit $\sum u$ is always defined, as increasing partial sums either converge to a finite limit, or diverge to $+\infty$. The road map is then to prove first that Equation 1 holds for *any* two sequences of *extended* non-negative numbers. The result is then *transferred* to the special case of summable sequences of non-negative numbers. Major libraries of formalized mathematics including Lean's mathlib [DBL 2020], Isabelle/HOL's Archive of Formal Proofs, coq-interval [Martin-Dorel and Melquiond 2016] or Coq's mathcomp-analysis [Affeldt and Cohen 2023], resort to such extended domains and transfer steps, notably for defining measure theory. Yet, as reported by expert users [Gouëzel 2021], the associated transfer bureaucracy is essentially done manually and thus significantly clutters formal developments in real and complex analysis, probabilities, etc.

While formalizing mathematics in practice, users of interactive theorem provers should be allowed to elude mundane arguments pertaining to proof transfer, as they would on paper, and

spare themselves the related, quickly overwhelming bureaucracy. Yet, they still need to convince the proof checker and thus have to provide explicit transfer proofs, albeit ideally automatically generated ones. The present work aims at providing a general method for implementing this nature of automation, for a diverse range of proof transfer problems.

In this paper, we focus on interactive theorem provers based on dependent type theory, such as Coq, Agda [Norell 2008] or Lean [de Moura and Ullrich 2021]. These proof management systems are genuine functional programming languages, with full-spectrum dependent types, a context in which representation independence meta-theorems can be turned into concrete instruments for achieving program and proof transfer.

Seminal results on the contextual equivalence of distinct implementations of a same abstract interface were obtained for system F, using logical relations [Mitchell 1986] and parametricity meta-theorems [Reynolds 1983; Wadler 1989]. In the context of type theory, such meta-theorems can be turned into syntactic translations of the type theory of interest into itself, automating this way the generation of the statement and the proof of parametricity properties for type families and for programs. Such syntactic relational models can accommodate dependent types [Bernardy and Lasson 2011], inductive types [Bernardy et al. 2012] and in fact the full Calculus of Inductive Constructions, including its impredicative sort [Keller and Lasson 2012].

In particular, the *univalent parametricity* translation [Tabareau et al. 2021] makes benefit of the univalence axiom [Univalent Foundations Program 2013] so as to transfer programs and theorems using established equivalences of types. This approach crucially removes the need for devising an explicit common interface for the types in relation. In presence of an internalized univalence axiom and of higher-inductive types, the *structure invariance principle* provides internal representation of independence results, for more general relational correspondences between types than equivalences [Angiuli et al. 2021b]. This last approach is thus particularly relevant in the frame of cubical type theory [Cohen et al. 2017; Vezzosi et al. 2019]. Indeed, a computational interpretation of the univalence axiom brings computational adequacy to otherwise possibly stuck terms, those resulting from a transfer involving an axiomatized univalence principle.

Unfortunately, a Swiss-army knife for automating the bureaucracy of proof transfer is still missing from the arsenal available to users of major proof assistants like Coq, Lean or Agda. Besides implementation concerns, the above examples actually illustrate fundamental limitations of the scope of existing approaches:

Univalence is overkill. Both univalent parametricity and the structure invariance principle can be used to derive the statement and the proof of the induction principle `N_ind` of Example 1.1, from the elimination scheme of type $\mathbb{N}$. But up to our knowledge, all the existing methods for automating this implication will pull in the univalence principle in the proof, although it can be obtained by hand by very elementary means. This limitation is all the more unsatisfactory that the univalence axiom is incompatible with proof irrelevance, a commonly assumed axiom in libraries formalizing classical mathematics, as Lean's mathlib.

Equivalences are not enough, neither are quotients. Univalent parametricity cannot help with our Example 1.2, as it is geared towards equivalences. But in this case, we are in fact not aware of an implemented method which would apply. In particular, the structure invariance principle [Angiuli et al. 2021b] would not apply as such in this case.

This leads us to the crux of our problem: existing techniques for transferring results from one type to another, *e.g.*, from $\mathbb{N}$ to `N` or from extended real numbers to real numbers, are either not suitable for dependent types, or too coarse to track the exact amount of data needed in a given proof, and not more.

Contributions. This paper presents three contributions:

- A parametricity framework *à la carte*, which generalizes [Tabareau et al. 2021]'s univalent parametricity translation, as well as refinements à la CoqEAL [Cohen et al. 2013] and generalized rewriting [Sozeau 2009]. Its pivotal ingredient is an appropriate, and up to our knowledge novel, phrasing of type equivalence, which allows for a finer-grained control of the data propagated by the translation.
- A conservative subtyping extension of CC_ω [Coquand and Huet 1988], used to formulate an inference algorithm for the synthesis of parametricity proofs.
- The implementation of a new parametricity plugin for the Coq proof assistant, using the Coq-Elpi [Tassi 2019] meta-language. This plugin rests on original formal proofs, conducted on top of the HoTT library [Bauer et al. 2017], and is distributed with a collection of application examples.

Outline. The rest of this paper is organized as follows. Section 2 introduces proof transfer and recalls the principle, strengths and weaknesses of the univalent parametricity translation. In Section 3, we present a new definition of type equivalence and we put this definition to good use in a hierarchy of structures for relations preserved by parametricity. Section 4 then presents variants of the raw and univalent parametricity translations, and the Trocq translation. In Section 5, we eventually discuss a few applications, including Examples 1.1 and 1.2, before concluding in Section 6.

2 STRENGTHS AND LIMITS OF UNIVALENT PARAMETRICITY

We first clarify the essence of proof transfer in dependent type theory (§ 2.1) and briefly recall a few concepts related to type equivalence and to the univalence principle (§ 2.2). We then review and discuss the limits of univalent parametricity (§ 2.3).

2.1 Proof transfer in type theory

Let us first recall the syntax of the Calculus of Constructions, CC_ω, a λ-calculus with dependent function types and a predicative hierarchy of universes, denoted $\square_i$:

$$A, B, M, N ::= \square_i \mid x \mid M\ N \mid \lambda x : A.\, M \mid \Pi x : A.\, B$$

We omit the typing rules of the calculus, available in Appendix A. We also use the standard equality type, called propositional equality, as well as dependent pairs, denoted $\Sigma x : A.\, B$. We write $t \equiv u$ the definitional equality between two terms t and u. Proof assistants Coq, Agda and Lean are based on various extensions of this core, notably with inductive types and with an impredicative sort. When the universe level does not matter, we casually remove the annotation and use notation $\square$.

In this context, proof transfer from type T_1 to type T_2 roughly amounts to *synthesizing* a new type former $W : T_2 \to \square$, *i.e.*, a type parametric in some type T_2, from an initial type former $V : T_1 \to \square$, *i.e.*, a type parametric in some type T_1, so as to ensure that for some given relations $R_T : T_1 \to T_2 \to \square$ and $R_\square : \square \to \square \to \square$, there is a proof w that:

$$\Gamma \vdash w : \forall (t_1 : T_1)(t_2 : T_2), R_T\ t_1\ t_2 \to R_\square (V\ t_1)(W\ t_2)$$

for a suitable context Γ. This setting generalizes as expected to k-ary type formers, and to more pairs of related types. In practice, relation $R_\square$ is often a right-to-left arrow, *i.e.*, $R_\square\ A\ B \triangleq B \to A$, as in this case the proof w substantiates a proof step turning goal clause $\Gamma \vdash V\ t_1$ into $\Gamma \vdash W\ t_2$.

Phrased as such, this synthesis problem is arguably quite loosely specified. Consider for instance the transfer problem discussed in Example 1.1. A first possible formalization involves the design of an appropriate common interface structure for types $\mathbb{N}$ and N, for instance by setting both T_1 and T_2 as $\Sigma N : \square.N \times (N \to N)$, and both V and W as: $\lambda X : T_1.\, \Pi P : X.1 \to \square.\, P\ X.2 \to (\Pi n :$

$X.1.\, P\ n \to P\ (X.3\ n)) \to \Pi n : X.1.\, P\ n$, where $X.i$ denotes the i-th item in the dependent tuple X. In this case, relation R_T may characterize isomorphic instances of the structure. Such instances of proof transfer are elegantly addressed in cubical type theories via internal structure univalence principles, whose implementation [Angiuli et al. 2021b] is able to automatize the synthesis of the required structure. The hassle of devising explicit structures manually for concrete instances of proof transfer is sometimes referred to as the anticipation problem [Tabareau et al. 2021].

Another option is to consider two different types $T_1 \triangleq \mathbb{N} \times (\mathbb{N} \to \mathbb{N})$ and $T_2 \triangleq \mathrm{N} \times (\mathrm{N} \to \mathrm{N})$ and

$$V' \triangleq \lambda X : T_1.\ \forall P : \mathbb{N} \to \square.\ P\ X.1 \to (\forall n : \mathbb{N}, P\ n \to P(X.2\ n)) \to \forall n : \mathbb{N}, P\ n$$
$$W' \triangleq \lambda X : T_2.\ \forall P : \mathrm{N} \to \square.\ P\ X.1 \to (\forall n : \mathrm{N}, P\ n \to P(X.2\ n)) \to \forall n : \mathrm{N}, P\ n.$$

Here one would typically expect R_T to be a type equivalence between T_1 and T_2, so as to transport $(V'\ t_1)$ to $(W'\ t_2)$, along this equivalence.

Note that some solutions of given instances of proof transfer problems are in fact too trivial to be of interest. Consider for example the case of a *functional* relation between T_2 and T_1, with $R_T\ t_1\ t_2$ defined as $t_1 = \phi\ t_2$, for some $\phi : T_2 \to T_1$. In this case, the composition $V \circ \phi$ is an obvious candidate for W, but an often uninformative one. Indeed, this composition can only propagate structural arguments, blind to the additional mathematical proofs of program equivalences potentially available in the context. For instance, here is a constructible but rather useless variant of W':

$$W'' \triangleq \lambda X : T_2.\ \forall P : \mathbb{N} \to \square.\ P\ (\uparrow_{\mathbb{N}} X.1) \to (\forall n : \mathbb{N}, P\ n \to P\ (\uparrow_{\mathbb{N}} (X.2\ (\downarrow_{\mathbb{N}} n)))) \to \forall n : \mathbb{N}, P\ n.$$

Automation devices dedicated to proof transfer thus typically consist of a meta-program which attempts to compute type former W and proof w by induction on the structure of V, by composing registered canonical pairs of related terms, and the corresponding proofs. These tools differ by the nature of relations they can accommodate, and by the class of type formers they are able to synthesize. For instance, *generalized rewriting* [Sozeau 2009], which provides essential support to formalizations based on setoids [Barthe et al. 2003], addresses the case of homogeneous (and reflexive) relations, *i.e.*, when T_1 and T_2 coincide. The CoqEAL library [Cohen et al. 2013] provides another example of such transfer automation tool, geared towards *refinements*, typically from a proof-oriented data-structure to a computation-oriented one. It is thus specialized to heterogeneous, functional relations but restricted to closed, quantifier-free type formers. We now discuss the few transfer methods which can accommodate dependent types and heterogeneous relations.

2.2 Type equivalences, univalence

Let us first focus on the special case of types related by an *equivalence*, and start with a few standard definitions, notations and lemmas. Omitted details can be found in the usual references, like the Homotopy Type Theory book [Univalent Foundations Program 2013]. Two functions $f, g : A \to B$ are *point-wise equal*, denoted $f \doteq g$ when their values coincide on all arguments, that is $f \doteq g : \Pi a : A.\ f\ a = g\ a$. For any type A, id_A denotes $\lambda a : A.\ a$, the identity function on A, and we will write id when the implicit type A is not ambiguous.

Definition 2.1 (Type isomorphism, type equivalence). A function $f : A \to B$ is an *isomorphism*, denoted $\mathsf{IsIso}(f)$, if there exists a function $g : B \to A$ which satisfies the section and retraction properties, which respectively assert that g is both a point-wise left and right inverse of f. An isomorphism f is an *equivalence*, denoted $\mathsf{IsEquiv}(f)$, when it moreover enjoys a last *adjunction property*, relating the proofs of the section and retraction properties and ensuring that $\mathsf{IsEquiv}(f)$ is proof-irrelevant.

Two types A and B are *equivalent*, denoted $A \simeq B$, when there is an equivalence $f : A \to B$:

$$A \simeq B \quad \triangleq \quad \Sigma f : A \to B.\, \mathsf{IsEquiv}(f)$$

Lemma 2.2. *Any isomorphism $f : A \to B$ is also an equivalence.*

The data of an equivalence $e : A \simeq B$ thus include two *transport functions*, denoted respectively $\uparrow_e : A \to B$ and $\downarrow_e : B \to A$. They can be used for proof transfer from A to B, using $\uparrow_e$ at covariant occurrences, and $\downarrow_e$ at contravariant ones. The *univalence principle* asserts that equivalent types are indistinguishable.

Definition 2.3 (Univalent universe). A universe $\mathcal{U}$ is univalent if for any two types A and B in $\mathcal{U}$, the canonical map $A = B \to A \simeq B$ is an equivalence.

In variants of CC_ω, univalence can be postulated as an axiom for all universes $\square_i$, with no explicit computational content, as done for instance in the HoTT library for the Coq proof assistant [Bauer et al. 2017]. Some more recent variants of dependent type theory [Angiuli et al. 2021a; Cohen et al. 2017] feature a built-in computational univalence principle, and are used to implement experimental proof assistants, such as Cubical Agda [Vezzosi et al. 2019]. In both cases, the univalence principle provides a powerful proof transfer principle from $\square$ to $\square$, as for any two types A and B such that $A \simeq B$, and any $P : \square \to \square$, we can obtain that $P\ A \simeq P\ B$ as a direct corollary of univalence. Concretely, $P\ B$ is obtained from $P\ A$ by appropriately allocating the transfer functions provided by the equivalence data, a transfer process typically useful in the context of proof engineering [Ringer et al. 2021].

Going back to our example from § 2.1, transferring along an equivalence $\mathbb{N} \simeq \mathtt{N}$ will thus produce W'' from V'. In presence of univalence, even in its non-computational form, it is also possible to achieve the more informative transport from V' to W', using a method called *univalent parametricity* [Tabareau et al. 2021], which we shall discuss in the next section.

2.3 Parametricity translations

Univalent parametricity strengthens the transfer principle provided by the univalence axiom by combining it with parametricity. In CC_ω, the essence of parametricity, which is to devise a relational interpretation of types, can be turned into an actual syntactic translation, as relations can themselves be modeled as λ-terms in CC_ω. The seminal work of Bernardy *et al.*, Keller and Lasson combine in what we refer to as the *raw parametricity translation*, which essentially defines inductively a logical relation $[\![T]\!]$ for any type T, as described on Figure 1.

- Context translation:

$$[\![\langle\rangle]\!] = \langle\rangle \tag{2}$$

$$[\![\Gamma, x : A]\!] = [\![\Gamma]\!], x : A, x' : A', x_R : [\![A]\!]\ x\ x' \tag{3}$$

- Term translation:

$$[\![\square_i]\!] = \lambda A\, A'.\, A \to A' \to \square_i \tag{4}$$

$$[\![x]\!] = x_R \tag{5}$$

$$[\![A\ B]\!] = [\![A]\!]\ B\ B'\ [\![B]\!] \tag{6}$$

$$[\![\lambda x : A.\, t]\!] = \Pi(x : A)(x' : A')(x_R : [\![A]\!]\ x\ x').\, [\![t]\!] \tag{7}$$

$$[\![\Pi x : A.\, B]\!] = \lambda f\, f'.\, \Pi(x : A)(x' : A')(x_R : [\![A]\!]\ x\ x').\, [\![B]\!](f\ x)(f'\ x') \tag{8}$$

Fig. 1. Raw parametricity translation for CC_ω.

This presentation uses the standard convention that t' is the term obtained from a term t by replacing every variable x in t with a fresh variable x'. A variable x is translated into a variable

x_R, where x_R is a fresh name. The associated abstraction theorem ensures that this translation preserves typing, in the following sense:

Theorem 2.4. *If* $\Gamma \vdash t : T$ *then* $[\![\Gamma]\!] \vdash t : T$, $[\![\Gamma]\!] \vdash t' : T'$ *and* $[\![\Gamma]\!] \vdash [\![t]\!] : [\![T]\!]\ t\ t'$.

Proof. By structural induction on the typing judgment, see for instance [Keller and Lasson 2012]. □

A key, albeit mundane ingredient of Theorem 2.4 is the fact that the rules of Figure 1 ensure that:

$$\vdash [\![\Box_i]\!] \ : \ [\![\Box_{i+1}]\!]\ \Box_i\ \Box_i \tag{9}$$

This translation precisely generates the statements expected from a parametric type family or program. For instance, the translation of a Π-type, given by Equation 8, is a type of relations on functions, which relates those producing related outputs from related inputs. Concrete implementations of this translation are available [Keller and Lasson 2012; Tassi 2019], and useful to generate and prove parametricity properties for type families or for constants, improved induction schemes, etc.

The key observation of univalent parametricity is that, it is possible to preserve the abstraction theorem while restricting to relations that are in fact (heterogeneous) equivalences. This however requires a careful design in the translation of universes:

$$[\![\Box_i]\!]\ A\ B \ \triangleq \ \Sigma(R : A \to B \to \Box_i)(e : A \simeq B).\, \Pi(a : A)(b : B).\, R\ a\ b \simeq (a = \downarrow_e b)$$

where $[\![\cdot]\!]$ now refers to the *univalent* parametricity translation, replacing the notation introduced for the raw variant. For any two types A and B, $[\![\Box_i]\!]\ A\ B$ packages a relation R and an equivalence e such that R is equivalent to the functional relation associated with $\downarrow_e$. Crucially, one can show that assuming univalence, $[\![\Box_i]\!]$ is equivalent to equivalence, that is, for any two types A and B:

$$[\![\Box_i]\!]\ A\ B \simeq (A \simeq B).$$

This observation is actually an instance of a more general technique available for constructing syntactic models of type theory [Boulier et al. 2017], based on attaching extra intensional specification to negative type constructors. In these models, a standard way to recover the abstraction theorem consists in refining the translation into two variants, for any term $T : \Box_i$, that is also a type. Its translation as a *term*, denoted $[\ T\ ]$, should be a dependent pair, which equips a relation with the additional data prescribed by the interpretation $[\![\Box_i]\!]$ of the universe. The translation $[\![T]\!]$ of T as a *type* will be the relation itself, that is, the projection of the dependent pair $[\ T\ ]$ onto its first component, denoted $\mathsf{rel}([\ T\ ])$. We refer to the original publication [Tabareau et al. 2021, Figure 4] for a complete description of the translation.

We can now state the resulting abstraction theorem [Tabareau et al. 2021], where $\vdash_u$ refers to a typing judgment of CC_ω assuming the univalence axiom:

Theorem 2.5. *If* $\Gamma \vdash t : T$ *then* $[\![\Gamma]\!] \vdash_u [\ t\ ] \ : \ [\![T]\!]\ t\ t'$.

Note that proving the abstraction theorem 2.5 involves in particular proving that:

$$\vdash_u [\ \Box_i\ ] \ : \ [\![\Box_{i+1}]\!]\ \Box_i\ \Box_i \quad \text{and} \quad \mathsf{rel}([\ \Box_i\ ]) \equiv [\![\Box_i]\!]. \tag{10}$$

The definition of relation $[\ \Box_i\ ]$ uses the univalence principle in a crucial way, in order to prove that the relation in the universe is equivalent to equality on the universe, *i.e.*, to prove that:

$$\vdash_u \Pi A\, B : \Box_i.\, [\![\Box_i]\!]\ A\ B \simeq (A = B).$$

Importantly, this univalent parametricity translation can be seamlessly extended so as to also make use of a global context of user-defined equivalences. Now let us go back to our motivating Example 1.1. A closer look at [Tabareau et al. 2021, Figure 4] reveals why the univalent parametricity translation can only resort to the univalence axiom in transferring the recurrence principle from

type $\mathbb{N}$ to type N. Because of the role of univalence in Equation 10, univalence is actually necessary as soon as the translated term involves an essential occurrence of a universe $\square_i$.

3 TYPE EQUIVALENCE IN KIT

This section describes the first step towards overcoming the limitations of univalent parametricity, as identified in Section 2.3. We thus propose (§ 3.1) an equivalent, modular presentation of type equivalence, phrased as a nested sigma type. Then (§ 3.2), we carve a hierarchy of structures on relations out of this dependent tuple, selectively picking pieces. Last, we revisit (§ 3.3) parametricity translations through the lens of this finer grained analysis of the relational interpretations of types.

3.1 Disassembling type equivalence

Let us first observe that the Definition 2.1, of type equivalence, is quite asymmetrical, although this fact is somehow put under the rug by the infix $A \simeq B$ notation. First, the data of an equivalence $e : A \simeq B$ privileges the left-to-right direction, as $\uparrow_e$ is directly accessible from e as its first projection, while accessing the right-to-left transport requires an additional projection. Second, the statement of the adjunction property, which we eluded in Definition 2.1, is actually:

$$\Pi a : A.\, \mathsf{ap}_{\uparrow_e}(s\ a) = r \circ \downarrow_e$$

where $\mathsf{ap}_f(t)$ is the term $f\ u = f\ v$, for any identity proof $t : u = v$. This statement uses proofs s and r, respectively of the section and retraction properties of e, but not in a symmetrical way, although swapping them leads to an equivalent definition. This entanglement prevents tracing the respective roles of each direction of transport, left-to-right or right-to-left, during the course of a given univalent parametricity translation. Exercise 4.2 in the HoTT book [Univalent Foundations Program 2013] however suggests a symmetrical wording of the definition of type equivalence, in terms of functional relations.

Definition 3.1. A relation $R : A \to B \to \square_i$, is *functional*, denoted $\mathsf{IsFun}(R)$, when:

$$\Pi a : A.\, \mathsf{IsContr}(\Sigma b : B.\, R\ a\ b)$$

where for any type T, $\mathsf{IsContr}(T)$ is the standard contractibility predicate $\Sigma t : T.\, \Pi t' : T.\, t = t'$.

We can now obtain an equivalent but symmetrical characterization of type equivalence, as a functional relation whose symmetrization is also functional.

Lemma 3.2. *For any types $A, B : \square$, type $A \simeq B$ is equivalent to:*

$$\Sigma R : A \to B \to \square.\ \mathit{IsFun}(R) \times \mathit{IsFun}(R^{-1})$$

where relation $R^{-1} : B \to A \to \square$ just swaps the arguments of an arbitrary $R : A \to B \to \square$.

We sketch below a proof of this result, left as an exercise in [Univalent Foundations Program 2013]. The essential argument is the following characterization of functional relations:

Lemma 3.3. *For any types $A, B : \square$, we have $(A \to B) \simeq \Sigma R : A \to B \to \square.\ \mathit{IsFun}(R)$.*

Proof. The proof goes by chaining the following equivalences:

$$(\Sigma R : A \to B \to \square.\, \mathsf{IsFun}(R)) \simeq (A \to \Sigma P : B \to \square.\, \mathsf{IsContr}(\Sigma b : B.\, P\ b)) \simeq (A \to B)$$

□

Proof of Lemma 3.2. The proof goes by chaining the following equivalences:

$$\begin{aligned}
(A \simeq B) &\simeq \Sigma f : A \to B.\, \mathsf{IsEquiv}(f) && \text{by definition of } (A \simeq B)\\
&\simeq \Sigma f : A \to B.\, \Pi b : B.\, \mathsf{IsContr}(\Sigma a. f\ a = b) && \text{standard result in HoTT}\\
&\simeq \Sigma f : A \to B.\, \mathsf{IsFun}(\lambda (b : B)(a : A).\, f\ a = b) && \text{by definition of } \mathsf{IsFun}(\cdot)\\
&\simeq \Sigma \left(f : \Sigma R : A \to B \to \square.\, \mathsf{IsFun}(R)\right).\, \mathsf{IsFun}(\pi_1(f)^{-1}) && \text{by Lemma 3.3}\\
&\simeq \Sigma R : A \to B \to \square.\, \mathsf{IsFun}(R) \times \mathsf{IsFun}(R^{-1}) && \text{by associativity of } \Sigma
\end{aligned}$$

□

However, the symmetrical version of type equivalence provided by Lemma 3.2 does not expose explicitly the two transfer functions in its data, although this computational content can be extracted via first projections of contractibility proofs. In fact, it is possible to devise a definition of type equivalence which directly provides the two transport functions in its data, while remaining symmetrical. The essential ingredient of this rewording is an alternative characterization of functional relations.

Definition 3.4. For any types $A, B : \square$, a relation $R : A \to B \to \square$, is a *univalent map*, denoted $\mathsf{IsUmap}(R)$ when there exists a function $m : A \to B$ together with proofs:

$$g_1 : \Pi (a : A)(b : B).\, m\ a = b \to R\ a\ b \quad \text{and} \quad g_2 : \Pi (a : A)(b : B).\, R\ a\ b \to m\ a = b$$

such that:

$$\Pi (a : A)(b : B).\, (g_1\ a\ b) \circ (g_2\ a\ b) \doteqdot id.$$

Now comes the crux lemma of this section, formally proved in the companion code[1].

Lemma 3.5. *For any types $A, B : \square$ and any relation $R : A \to B \to \square$*

$$\mathit{IsFun}(R) \simeq \mathit{IsUmap}(R).$$

Proof. The proof goes by rewording the left hand side, in the following way:

$$\begin{aligned}
\Pi x.\, \mathsf{IsContr}(R\ x) &\simeq \Pi x.\, \Sigma (r : \Sigma y.\, R\ x\ y).\, \Pi (p : \Sigma y.\, R\ x\ y).\, r = p\\
&\simeq \Pi x.\, \Sigma y.\, \Sigma (r : R\ x\ y).\, \Pi (p : \Sigma y.\, R\ x\ y).\, (y, r) = p\\
&\simeq \Sigma f.\, \Pi x.\, \Sigma (r : R\ x\ (f\ x)).\, \Pi (p : \Sigma y.\, R\ x\ y).\, (f\ x, r) = p\\
&\simeq \Sigma f.\, \Sigma (r : \Pi x.\, R\ x\ (f\ x)).\, \Pi x.\, \Pi (p : \Sigma y.\, R\ x\ y).\, (f\ x, r\ x) = p\\
&\simeq \Sigma f.\, \Sigma r.\, \Pi x.\, \Pi y.\, \Pi (p : R\ x\ y).\, (f\ x, r\ x) = (y, p)\\
&\simeq \Sigma f.\, \Sigma r.\, \Pi x.\, \Pi y.\, \Pi (p : R\ x\ y).\, \Sigma (e : f\ x = y).\, r\ x =_e p\\
&\simeq \Sigma f.\, \Sigma r.\, \Sigma (e : \Pi x.\, \Pi y.\, R\ x\ y \to f\ x = y).\, \Pi x.\, \Pi y.\, \Pi p.\, (r\ x) =_{e\ x\ y\ p} p
\end{aligned}$$

After a suitable reorganization of the sigma types we are left to show that

$$\Sigma (r : \Pi x.\, \Pi y.\, f\ x = y \to R\ x\ y).\, (e\ x\ y) \circ (r\ x\ y) \doteqdot id \simeq \Sigma (r : \Pi x.\, R\ x\ (f\ x)).\, \Pi x.\, \Pi y.\, \Pi p.\, r\ x =_{e\ x\ y\ p} p$$

which proof we do not detail, referring the reader to the companion code. □

As a direct corollary, we obtain a novel characterization of type equivalence:

[1] See lemma `map_graph_equiv_isfun` in file `Uparam.v`.

Theorem 3.6. *For any types $A, B : \square_i$, we have:*

$$(A \simeq B) \simeq \boxdot^{\top}\ A\ B$$

where the relation $\boxdot^{\top}\ A\ B$ is defined as:

$$\Sigma R : A \to B \to \square_i.\ \mathit{IsUmap}(R) \times \mathit{IsUmap}(R^{-1})$$

The collection of data packed in a term of type $\boxdot^{\top}\ A\ B$ is now symmetrical, as the right-to-left direction of the equivalence based on univalent maps can be obtained from the left-to-right by flipping the relation and swapping the two functionality proofs. If the η-rule for records is verified, symmetry is even *definitionally* involutive.

3.2 Reassembling type equivalence

Definition 3.4 of univalent maps and the resulting rephrasing of type equivalence suggest introducing a hierarchy of structures for heterogeneous relations, which explains how close a given relation is to type equivalence. In turn, this distance is described in terms of structure available respectively on the left-to-right and right-to-left transport functions.

Definition 3.7. For $n, k \in \{0, 1, 2_a, 2_b, 3, 4\}$, and $\alpha = (n, k)$, relation $\boxdot^{\alpha} : \square \to \square \to \square$, is defined as:

$$\boxdot^{\alpha} \triangleq \lambda(A\ B : \square).\Sigma(R : A \to B \to \square).\mathsf{Class}_{\alpha}\ R$$

where the *map class* $\mathsf{Class}_{\alpha}\ R$ itself unfolds to a pair type $(\mathsf{M}_n\ R) \times (\mathsf{M}_k\ R^{-1})$, with M_i defined as[2]:

$$\begin{aligned}
&\mathsf{M}_0\ R \triangleq .\\
&\mathsf{M}_1\ R \triangleq (A \to B)\\
&\mathsf{M}_{2_a}\ R \triangleq \Sigma m : A \to B.\ G_{2_a}\ m\ R \quad \text{with } G_{2_a}\ m\ R \triangleq \Pi a\ b.\ m\ a = b \to R\ a\ b\\
&\mathsf{M}_{2_b}\ R \triangleq \Sigma m : A \to B.\ G_{2_b}\ m\ R \quad \text{with } G_{2_b}\ m\ R \triangleq \Pi a\ b.\ R\ a\ b \to m\ a = b\\
&\mathsf{M}_3\ R \triangleq \Sigma m : A \to B.\ (G_{2_a}\ m\ R) \times (G_{2_b}\ m\ R)\\
&\mathsf{M}_4\ R \triangleq \Sigma m : A \to B.\ \Sigma(g_1 : G_{2_a}\ m\ R).\ \Sigma(g_2 : G_{2_b}\ m\ R).\ \Pi a\ b.\ (g_1\ a\ b) \circ (g_1\ a\ b) \doteqdot id
\end{aligned}$$

For any types A and B, and any $r : \boxdot^{\alpha}\ A\ B$ we will use notations $\mathsf{rel}(r)$, $\mathsf{map}(r)$ and $\mathsf{comap}(r)$ to refer respectively to the relation, map of type $A \to B$, map of type $B \to A$, included in the data of r, for a suitable α.

Definition 3.8. We denote $\mathcal{A}$ the set $\{0, 1, 2_a, 2_b, 3, 4\}^2$, used to index map classes in Definition 3.7. This set is partially ordered for the product order on $\{0, 1, 2_a, 2_b, 3, 4\}$ defined from the partial order $0 < 1 < 2_* < 3 < 4$ for 2_* either 2_a or 2_b, and with 2_a and 2_b being incomparable.

Remark 3.9. Relation $\boxdot^{(4,4)}$ of Definition 3.7 coincides with the relation $\boxdot^{\top}$ introduced in Theorem 3.6. Similarly, we denote $\boxdot^{\bot}$ the relation $\boxdot^{(0,0)}$. A relation equipped with structure $\boxdot^{(4,0)}\ A\ B$ (resp. $\boxdot^{(3,3)}\ A\ B$) is the graph of a univalent map from A to B (resp. isomorphism between A and B).

In the supplementary material, the corresponding lattice to the collection of M_n is implemented as a hierarchy of dependent tuples, more precisely, of record types. Each arrow of Figure 2 represents an inclusion of the data packed in the source structure into the data packed in the target one. Moreover, nodes are labeled with the names of the corresponding record fields introduced by the richer structure.

[2]For the sake of readability, we omit implicit arguments, e.g., although M_i has type $\lambda(T_1\ T_2 : \square).(T_1 \to T_2 \to \square) \to \square$, we write $\mathsf{M}_n\ R$ for $(\mathsf{M}_n\ A\ B\ R)$.

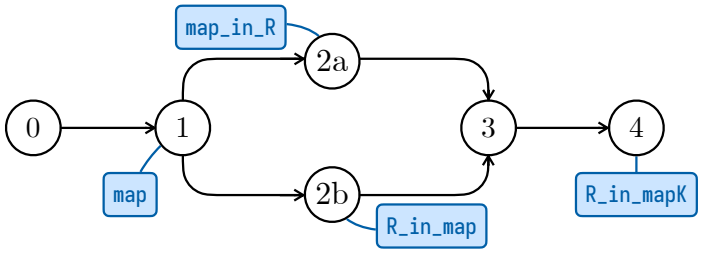

Fig. 2. Implementation of the hierarchy of Definition 3.7

3.3 Populating the hierarchy of relations

We shall now revisit the parametricity translations of Section 2.3. In particular, combining Theorem 3.6 with Equation 10, crux of the abstraction theorem for univalent parametricity, ensures the existence of a term $p_{\square_i}$ such that:

$$\vdash_u p_{\square_i} \; : \; \circledcirc_{i+1}^{\top} \, \square_i \; \square_i \quad \text{and} \quad \mathsf{rel}(p_{\square_i}) \simeq \circledcirc_i^{\top}.$$

Otherwise said, relation $\circledcirc^{\top} : \square \to \square \to \square$ can be endowed with a $\circledcirc^{\top}$ structure, assuming univalence. Similarly, Equation 9, for the raw parametricity translation, can be read as the fact that relation $\circledcirc^{\bot}$ on universes can be endowed with a $\circledcirc^{\bot}$ $\square$ $\square$ structure.

Now the hierarchy of structures introduced by Definition 3.7 enables a finer grained analysis of the possible relational interpretations of universes. Not only would this put the raw and univalent parametricity translations under the same hood, but it would also allow for generalizing parametricity to a larger class of relations. For this purpose, we generalize the previous observation, on the key ingredient for translating universes: for each $\alpha \in \mathcal{A}$, relation $\circledcirc^{\alpha} : \square \to \square \to \square$ may be endowed with several structures from the lattice, and we need to study which ones, depending on α. Otherwise said, we need to identify the pairs $(\alpha, \beta) \in \mathcal{A}^2$ for which it is possible to construct a term $p_{\square}^{\alpha,\beta}$ such that:

$$\vdash_u p_{\square}^{\alpha,\beta} \; : \; \circledcirc^{\beta} \; \square \; \square \quad \text{and} \quad \mathsf{rel}(p_{\square}^{\alpha,\beta}) \equiv \circledcirc^{\alpha} \tag{11}$$

Note that we aim here at a definitional equality between $\mathsf{rel}(p_{\square}^{\alpha,\beta})$ and $\circledcirc^{\alpha}$, rather than at an equivalence. It is easy to see that a term $p_{\square}^{\alpha,\bot}$ exists for any $\alpha \in \mathcal{A}$, as $\circledcirc^{\bot}$ requires no structure on the relation. On the other hand, it is not possible to construct a term $p_{\square}^{\bot,\top}$, *i.e.*, to turn an arbitrary relation into a type equivalence.

Definition 3.10. We denote $\mathcal{D}_{\square}$ the following subset of $\mathcal{A}^2$:

$$\mathcal{D}_{\square} = \left(\mathcal{A} \times \{0, 1, 2_a\}^2\right) \cup \{(\top, \beta) \mid \beta \notin \{0, 1, 2_a\}^2\}$$

The supplementary material[3] constructs terms $p_{\square}^{\alpha,\beta}$ for every pair $(\alpha, \beta) \in \mathcal{D}_{\square}$, using a meta-program to generate them from a minimal collection of manual definitions. In particular, assuming univalence, it is possible to construct a term $p_{\square}^{\top,\top}$, which can be seen as an analogue of the translation $[\square]$ of univalent parametricity. More generally, the provided terms $p_{\square}^{\alpha,\beta}$ depend on univalence if and only if $\beta \notin \{0, 1, 2_a\}^2$.

The next natural question is the study of the possible structures $\circledcirc^{\gamma}$ that can equip a relation associated with a product type $\Pi x : A.\, B$, when the relations associated with types A and B are respectively equipped with structures $\circledcirc^{\alpha}$ and $\circledcirc^{\beta}$.

[3] File `Param_Type.v`

Otherwise said, we need to identify the triples $(\alpha, \beta, \gamma) \in \mathcal{A}^3$ for which it is possible to construct a term p_Π^γ such that:

$$\frac{\Gamma \vdash A_R \ : \ \boxdot^\alpha \ A \ A' \qquad \Gamma,\ x : A,\ x' : A',\ x_R : A_R \ x \ x' \ \vdash \ B_R \ : \ \boxdot^\beta \ B \ B'}{\Gamma \ \vdash \ p_\Pi^\gamma \ A_R \ B_R \ : \ \boxdot^\gamma \ (\Pi x : A.\, B) \ (\Pi x' : A'.\, B')} \quad \text{and}$$

$$\mathsf{rel}(p_\Pi^\gamma \ A_R \ B_R) \equiv \lambda f.\lambda f'.\Pi(x \ : \ A)(x' \ : \ A')(x_R : \mathsf{rel}(A_R) \ x \ x').\ \mathsf{rel}(B_R) \ (f \ x) \ (f \ x')$$

The corresponding collection of triples can actually be described as a function $\mathcal{D}_\Pi : \mathcal{A} \to \mathcal{A}^2$, such that $\mathcal{D}_\Pi(\gamma) = (\alpha, \beta)$ provides the *minimal* requirements on the structures associated with A and B, with respect to the partial order on $\mathcal{A}^2$. The supplementary material[4] provides a corresponding collection of terms p_Π^γ for each $\gamma \in \mathcal{A}$, as well as all the associated weakenings. Once again, these definitions are generated by a meta-program. Observe in particular that by symmetry, $p_\Pi^{(m,n)}$ can be obtained from $p_\Pi^{(m,0)}$ and $p_\Pi^{(n,0)}$ by swapping the latter and glueing it to the former. Therefore, the values of p_Π^γ and $\mathcal{D}_\Pi(\gamma)$ are completely determined by those of $p_\Pi^{(m,0)}$ and $\mathcal{D}_\Pi(m, 0)$. In particular, for any $m, n \in \mathcal{A}$:

$$\mathcal{D}_\Pi(m, n) = ((m_A, n_A), (m_B, n_B))$$

for $m_A, n_A, m_B, n_B \in \mathcal{A}$ defined as $\mathcal{D}_\Pi(m, 0) = ((0, n_A), (m_B, 0))$ and $\mathcal{D}_\Pi(n, 0) = ((0, m_A), (n_B, 0))$.

We sum up in Figure 3 the values of $\mathcal{D}_\Pi(m, 0)$.

m	$\mathcal{D}_\Pi(m,0)_1$	$\mathcal{D}_\Pi(m,0)_2$
0	$(0, 0)$	$(0, 0)$
1	$(0, 2_a)$	$(1, 0)$
2_a	$(0, 4)$	$(2_a, 0)$
2_b	$(0, 2_a)$	$(2_b, 0)$
3	$(0, 4)$	$(3, 0)$
4	$(0, 4)$	$(4, 0)$

m	$\mathcal{D}_\to(m,0)_1$	$\mathcal{D}_\to(m,0)_2$
0	$(0, 0)$	$(0, 0)$
1	$(0, 1)$	$(1, 0)$
2_a	$(0, 2_b)$	$(2_a, 0)$
2_b	$(0, 2_a)$	$(2_b, 0)$
3	$(0, 3)$	$(3, 0)$
4	$(0, 4)$	$(4, 0)$

Fig. 3. Minimal dependencies for dependent and non-dependent products at class $(m, 0)$

Note that in the case of a non-dependent product, constructing $p_\to^\gamma$ requires less structure on the domain A of an arrow type $A \to B$, which motivates the introduction of function $\mathcal{D}_\to(\gamma)$. Using the combinator for dependent products to interpret an arrow type, albeit correct, potentially pulls in unnecessary structure (and axiom) requirements. The supplementary material[5] includes a construction of terms $p_\to^\gamma$ for any $\gamma \in \mathcal{A}$.

4 A CALCULUS FOR PROOF TRANSFER

This section introduces Trocq, a framework for proof transfert designed as a generalization of parametricity translations, so as to allow for interpreting types as instances of the structures introduced in Section 3.2. We adopt a sequent style presentation, which fits closely the type system of CC_ω, while explaining in a consistent way the transformations of terms and contexts. This choice of presentation departs from the standard literature about parametricity in pure type systems. Yet, it brings the presentation closer to actual implementations, whose necessary management of parametricity contexts is put under the rug by notational conventions (*e.g.*, the primes of Section 2.3).

[4]File `Param_Forall.v`

[5]File `Param_Arrow.v`

For this purpose, we successively introduce four calculi, of increasing sophistication. We start (§ 4.1) with introducing this sequent style presentation by rephrasing the raw parametricity translation, and the univalent parametricity one (§ 4.2). We then introduce CC^{+}_{ω} (§ 4.3), a calculus of constructions with annotations on sorts and subtyping, before defining (§ 4.4) the Trocq calculus.

4.1 Raw parametricity sequents

We introduce *parametricity contexts*, under the form of a list of triples packaging pairs of variables together with a witness that they are related:

$$\Xi ::= \varepsilon \mid \Xi,\ x \sim x' \mathrel{\because} x_R$$

We write $(x, x', x_R) \in \Xi$ if there exists Ξ' and Ξ'' such that $\Xi = \Xi',\ x \sim x' \mathrel{\because} x_R,\ \Xi''$.

We denote $\mathrm{Var}(\Xi)$ the sequence of variables related in a parametricity context Ξ, with multiplicities:

$$\mathrm{Var}(\varepsilon) = \varepsilon \qquad \mathrm{Var}(\Xi,\ x \sim x' \mathrel{\because} x_R) = \mathrm{Var}(\Xi), x, x', x_R$$

A parametricity context Ξ is *well-formed*, written $\Xi \vdash$, if the sequence $\mathrm{Var}(\Xi)$ is duplicate-free . In this case, we use the notation $\Xi(x) = (x', x_R)$ as a synonym of $(x, x', x_R) \in \Xi$.

A *parametricity judgment* relates a parametricity context Ξ and three terms M, M', M_R of CC_ω. Parametricity judgments, denoted as:

$$\Xi \vdash M \sim M' \mathrel{\because} M_R,$$

are defined by rules of Figure 4 and read *in context Ξ, term M translates to the term M', because M_R.*

$$\frac{}{\Xi \vdash \square_i \sim \square_i \mathrel{\because} \lambda(A\,B : \square_i).\, A \to B \to \square_i}\ (\textsc{ParamSort}) \qquad \frac{(x, x', x_R) \in \Xi \qquad \Xi \vdash}{\Xi \vdash x \sim x' \mathrel{\because} x_R}\ (\textsc{ParamVar})$$

$$\frac{\Xi \vdash M \sim M' \mathrel{\because} M_R \quad \Xi \vdash N \sim N' \mathrel{\because} N_R}{\Xi \vdash M\,N \sim M'\,N' \mathrel{\because} M_R\,N\,N'\,N_R}\ (\textsc{ParamApp})$$

$$\frac{\Xi, x \sim x' \mathrel{\because} x_R \vdash M \sim M' \mathrel{\because} M_R}{\Xi \vdash \lambda x : A.\, M \sim \lambda x' : A'.\, M' \mathrel{\because} \lambda x\, x'\, x_R.\, M_R}\ (\textsc{ParamLam})$$

$$\frac{\Xi \vdash A \sim A' \mathrel{\because} A_R \qquad \Xi, x \sim x' \mathrel{\because} x_R \vdash B \sim B' \mathrel{\because} B_R \qquad x, x' \notin \mathrm{Var}(\Xi)}{\Xi \vdash \Pi x : A.\, B \sim \Pi x' : A'.\, B' \mathrel{\because} \lambda f\, g.\, \Pi x\, x'\, x_R.\, B_R\ (f\ x)\ (g\ x')}\ (\textsc{ParamPi})$$

Fig. 4. Param: sequent-style binary parametricity translation

Lemma 4.1. *The relation associating a term M with pairs (M', M_R) such that $\Xi \vdash M \sim M' \mathrel{\because} M_R$ holds, with Ξ a well-formed parametricity context is* functional*: for any term M and any well-formed Ξ:*

$$\forall M', N', M_R, N_R, \quad \Xi \vdash M \sim M' \mathrel{\because} M_R \ \wedge\ \Xi \vdash M \sim N' \mathrel{\because} N_R \implies (M', M_R) = (N', N_R)$$

Proof. Immediate by induction on the syntax of M. □

This presentation of parametricity thus provides an alternative definition of translation $[\![\cdot]\!]$, from Figure 1, and accounts for the prime-based notational convention used in the latter.

Definition 4.2. A parametricity context Ξ is *admissible* for a well-formed typing context Γ, denoted $\Gamma \triangleright \Xi$, when Ξ is well-formed as a parametricity context and Γ provides coherent type annotations for all terms in Ξ, that is, for any variables x, x', x_R such that $\Xi(x) = (x', x_R)$, and for any terms A' and A_R:

$$\Xi \vdash \Gamma(x) \sim A' \mathrel{\because} A_R \quad \Longrightarrow \quad \Gamma(x') = A' \wedge \Gamma(x_R) \equiv A_R\ x\ x'$$

We can now state and prove an abstraction theorem:

Theorem 4.3 (Abstraction theorem).

$$\frac{\Gamma \vdash \qquad \Gamma \vdash M : A \qquad \Gamma \triangleright \Xi \qquad \Xi \vdash M \sim M' \mathrel{\because} M_R \qquad \Xi \vdash A \sim A' \mathrel{\because} A_R}{\Gamma \vdash M' : A' \qquad \textit{and} \qquad \Gamma \vdash M_R : A_R\ M\ M'}$$

Proof. By induction on the derivation of $\Xi \vdash M \sim M' \mathrel{\because} M_R$. □

4.2 Univalent parametricity sequents

We now propose in Figure 5 a rephrased version of the univalent parametricity translation [Tabareau et al. 2021], using the same sequent style and replacing the translation of universes with the equivalent relation $\boxdot^{\top}$. In this variant, parametricity judgments are denoted:

$$\Xi \vdash_u M \sim M' \mathrel{\because} M_R$$

where Ξ is a parametricity context and M, M', and M_R are terms of CC_ω. The u index is a reminder that typing judgments $\Gamma \vdash_u M : A$ involved in the associated abstraction theorem are typing judgments of CC_ω augmented with the univalence axiom.

$$\frac{}{\Xi \vdash_u \square_i \sim \square_i \mathrel{\because} p_{\square_i}^{\top,\top}}\ (\textsc{UParamSort}) \qquad \frac{(x, x', x_R) \in \Xi \qquad \Xi \vdash}{\Xi \vdash_u x \sim x' \mathrel{\because} x_R}\ (\textsc{UParamVar})$$

$$\frac{\Xi \vdash_u M \sim M' \mathrel{\because} M_R \qquad \Xi \vdash_u N \sim N' \mathrel{\because} N_R}{\Xi \vdash_u M\ N \sim M'\ N' \mathrel{\because} M_R\ N\ N'\ N_R}\ (\textsc{UParamApp})$$

$$\frac{\Xi \vdash_u A \sim A' \mathrel{\because} A_R \qquad \Xi, x \sim x' \mathrel{\because} x_R \vdash_u M \sim M' \mathrel{\because} M_R}{\Xi \vdash_u \lambda x : A.\, M \sim \lambda x' : A'.\, M' \mathrel{\because} \lambda x\, x'\, x_R.\, M_R}\ (\textsc{UParamLam})$$

$$\frac{\Xi \vdash_u A \sim A' \mathrel{\because} A_R \qquad \Xi, x \sim x' \mathrel{\because} x_R \vdash_u B \sim B' \mathrel{\because} B_R}{\Xi \vdash_u \Pi x : A.\, B \sim \Pi x' : A'.\, B' \mathrel{\because} p_{\Pi}^{\top}\ A_R\ B_R}\ (\textsc{UParamPi})$$

Fig. 5. UParam: univalent parametricity rules

We can now rephrase the abstraction theorem for univalent parametricity.

Theorem 4.4 (Univalent abstraction theorem).

$$\frac{\Gamma \vdash \qquad \Gamma \vdash M : A \qquad \Gamma \triangleright \Xi \qquad \Xi \vdash_u M \sim M' \mathrel{\because} M_R \qquad \Xi \vdash_u A \sim A' \mathrel{\because} A_R}{\Gamma \vdash M' : A' \qquad \textit{and} \qquad \Xi \vdash_u M_R : \mathit{rel}(A_R)\ M\ M'}$$

Proof. By induction on the derivation of $\Xi \vdash_u M \sim M' \mathrel{\because} M_R$. □

Remark 4.5. In Theorem 4.4, term $\mathsf{rel}(A_R)$ is indeed a relation of type $A \to A' \to \square$. Indeed:

$$\frac{\Gamma \vdash A : \square_i \qquad \Xi \vdash_u A \sim A' \mathrel{\because} A_R \qquad \Gamma \triangleright \Xi}{\Gamma \vdash_u A_R : \mathsf{rel}(p_{\square_i}^{\top,\top})\ A\ A'}$$

entails A_R has type $\mathsf{rel}(p_{\square_i}^{\top,\top})\ A\ A' \equiv \boxdot^{\top}\ A\ A' \equiv \big(\Sigma R : A \to A' \to \square.\ \mathsf{IsUmap}(R) \times \mathsf{IsUmap}(R^{-1})\big)$.

4.3 Annotated type theory

We are now ready to generalize the relational interpretation of types provided by the univalent parametricity translation, so as to allow for interpreting sorts with instances of weaker structures than equivalence. For this purpose, we introduce a variant CC_ω^+ of CC_ω where each universe is annotated with a label indicating the structure available on its relational interpretation. Recall from Section 3.2 that we have used pairs $\alpha \in \mathcal{A}^2$ to identify the different structures of the lattice disassembling type equivalence: these are the labels annotating sorts of CC_ω^+, so that if A has type $\square^\alpha$, then the associated relation A_R has type $\boxdot^\alpha\ A\ A'$. The syntax of CC_ω^+ is thus:

$$M, N, A, B \in \mathcal{T}_{CC_\omega^+} ::= \square_i^\alpha \mid x \mid M\ N \mid \lambda x : A.\ M \mid \Pi x : A.\ B$$
$$\alpha \in \mathcal{A} = \{0, 1, 2_\mathrm{a}, 2_\mathrm{b}, 3, 4\}^2 \qquad i \in \mathbb{N}$$

Before completing the actual formal definition of the Trocq proof transfer framework, let us informally illustrate how these annotations shall drive the interpretation of terms, and in particular, of a dependent product $\Pi x : A.\ B$. In this case, before translating B, three terms representing the bound variable x, its translation x', and the parametricity witness x_R are added to the context. The type of x_R is $\mathsf{rel}(A_R)\ x\ x'$ where A_R is the parametricity witness relating A to its translation A'. The role of annotation α on the sort typing type A is thus to to govern the amount of information available in witness x_R, by determining the type of A_R. This intent is reflected in the typing rules of CC_ω^+, which rely on the definition of the loci $\mathcal{D}_\square$, $\mathcal{D}\to$ and $\mathcal{D}\Pi$, introduced in §3.3.

Typing terms in CC_ω^+ requires defining a *subtyping* relation $\preccurlyeq$, defined by the rules of Figure 6. The typing rules of CC_ω^+ are available in Figure 7 and follow standard presentations [Aspinall and Compagnoni 2001]. The $\equiv$ relation in the (SubConv) rule is the *conversion* relation, defined as the closure of α-equivalence and β-reduction on this variant of λ-calculus. We hence have two types of judgment in this calculus:

$$\Gamma \vdash_+ A \preccurlyeq B \quad \text{and} \quad \Gamma \vdash_+ M : A$$

Where M, A and B are terms in CC_ω^+ and Γ is a context in CC_ω^+ ($\Gamma ::= \varepsilon \mid \Gamma,\ x : A$)

$$\frac{\Gamma \vdash_+ A : K \qquad \Gamma \vdash_+ B : K \qquad A \equiv B}{\Gamma \vdash_+ A \preccurlyeq B}\ (\textsc{SubConv}) \qquad \frac{\alpha \geq \beta \qquad i \leq j}{\Gamma \vdash_+ \square_i^\alpha \preccurlyeq \square_j^\beta}\ (\textsc{SubSort})$$

$$\frac{\Gamma \vdash_+ M' N : K \qquad \Gamma \vdash_+ M \preccurlyeq M'}{\Gamma \vdash_+ M\ N \preccurlyeq M'\ N}\ (\textsc{SubApp}) \qquad \frac{\Gamma, x : A \vdash_+ M \preccurlyeq M'}{\Gamma \vdash_+ \lambda x : A.\ M \preccurlyeq \lambda x : A.\ M'}\ (\textsc{SubLam})$$

$$\frac{\Gamma \vdash_+ \Pi x : A.\ B : \square_i \qquad \Gamma \vdash_+ A' \preccurlyeq A \qquad \Gamma, x : A' \vdash_+ B \preccurlyeq B'}{\Gamma \vdash_+ \Pi x : A.\ B \preccurlyeq \Pi x : A'.\ B'}\ (\textsc{SubPi}) \qquad K ::= \square_i \mid \Pi x : A.\ K$$

Fig. 6. Subtyping rules for CC_ω^+

$$\frac{\Gamma \vdash_+ M : A \qquad \Gamma \vdash_+ A \lesssim B}{\Gamma \vdash_+ M : B}\,(\textsc{Conv}^+) \qquad \frac{(\alpha,\beta) \in \mathcal{D}_\square}{\Gamma \vdash_+ \square_i^\alpha : \square_{i+1}^\beta}\,(\textsc{Sort}^+)$$

$$\frac{(x,A) \in \Gamma \qquad \Gamma \vdash_+}{\Gamma \vdash_+ x : A}\,(\textsc{Var}^+) \qquad \frac{\Gamma \vdash_+ A : \square_i \qquad x \notin \mathrm{Var}(\Gamma)}{\Gamma,\ x : A \vdash_+}\,(\textsc{Context}^+)$$

$$\frac{\Gamma \vdash_+ M : \Pi x : A.\, B \qquad \Gamma \vdash_+ N : A}{\Gamma \vdash_+ M\ N : B[x := N]}\,(\textsc{App}^+) \qquad \frac{\Gamma, x : A \vdash_+ M : B}{\Gamma \vdash_+ \lambda x : A.\, M : \Pi x : A.\, B}\,(\textsc{Lam}^+)$$

$$\frac{\Gamma \vdash_+ A : \square_i^\alpha \qquad \Gamma \vdash_+ B : \square_i^\beta \qquad \mathcal{D}_\to(\gamma) = (\alpha,\beta)}{\Gamma \vdash_+ A \to B : \square_i^\gamma}\,(\textsc{Arrow}^+)$$

$$\frac{\Gamma \vdash_+ A : \square_i^\alpha \qquad \Gamma, x : A \vdash_+ B : \square_i^\beta \qquad \mathcal{D}_\Pi(\gamma) = (\alpha,\beta)}{\Gamma \vdash_+ \Pi x : A.\, B : \square_i^\gamma}\,(\textsc{Pi}^+)$$

Fig. 7. Typing rules for CC_ω^+

We show that CC_ω^+ is a conservative extension over CC_ω, by defining an erasure function for terms $|\cdot|^- : \mathcal{T}_{CC_\omega^+} \to \mathcal{T}_{CC_\omega}$ and the associated erasure function for contexts, see Appendix B.

4.4 The Trocq calculus

The final stage of the announced generalization consists in building an analogue to the parametricity translations available in pure type systems, but for the annotated type theory of § 4.3. This analogue is geared towards proof transfer, as discussed in § 2.1, and therefore designed to *synthesize* the output of the translation from its input, rather than to *check* that certain pairs of terms are in relation. However, splitting up the interpretation of universes into a lattice of possible relation structures means that the source term of the translation is not enough to characterize the desired output: the translation needs to be informed with some extra information about the expected outcome of the translation. In the Trocq calculus, this extra information is a type of CC_ω^+.

We thus define Trocq *contexts* as lists of quadruples:

$$\Delta ::= \varepsilon \mid \Delta,\ x\ @\ A\ \sim\ x'\ \because\ x_R \quad \text{where } A \in \mathcal{T}_{CC_\omega^+}.$$

We also introduce a conversion function γ from Trocq contexts to CC_ω^+ contexts:

$$\begin{aligned} \gamma(\varepsilon) &= \varepsilon \\ \gamma(\Delta,\ x\ @\ A\ \sim\ x'\ \because\ x_R) &= \gamma(\Delta),\ x : A \end{aligned}$$

Now, a Trocq judgment is a 4-ary relation of the form $\Delta \vdash_t M\ @\ A\ \sim\ M'\ \because\ M_R$, which is read *in context* Δ*, term* M *of annotated type* A *translate to term* M'*, because* M_R and M_R is called a parametricity witness. Trocq judgments are defined by the rules of Figure 8. This definition involves a weakening function for parametricity witnesses, defined as follows.

Definition 4.6. For all $p, q \in \{0, 1, 2_\mathrm{a}, 2_\mathrm{b}, 3, 4\}$, such that $p \geq q$, we define the map $\downarrow_q^p : \mathsf{M}_p \to \mathsf{M}_q$ to be the function forgetting the fields from M_p that are not in M_q.

For all $\alpha, \beta \in \mathcal{A}$, such that $\alpha \geq \beta$, function $\Downarrow_\beta^\alpha : \boxdot^\alpha\ A\ B \to \boxdot^\beta\ A\ B$ is defined by:

$$\Downarrow_{(p,q)}^{(m,n)}\ \langle R, M^\to, M^\leftarrow \rangle := \langle R, \downarrow_p^m\ M^\to, \downarrow_q^n\ M^\leftarrow \rangle.$$

$$\frac{(\alpha,\beta) \in \mathcal{D}_{\square}}{\Delta \vdash_t \square_i^{\alpha} \ @\ \square_{i+1}^{\beta} \ \sim\ \square_i^{\alpha} \ \because\ p_{\square_i}^{\alpha,\beta}}\ (\textsc{TrocqSort}) \qquad \frac{(x, A, x', x_R) \in \Delta \qquad \gamma(\Delta) \vdash_+}{\Delta \vdash_t x \ @\ A \ \sim\ x' \ \because\ x_R}\ (\textsc{TrocqVar})$$

$$\frac{\Delta \vdash_t M \ @\ \Pi x : A.\, B \ \sim\ M' \ \because\ M_R \qquad \Delta \vdash_t N \ @\ A \ \sim\ N' \ \because\ N_R}{\Delta \vdash_t M\ N \ @\ B[x := N] \ \sim\ M'\ N' \ \because\ M_R\ N\ N'\ N_R}\ (\textsc{TrocqApp})$$

$$\frac{\Delta \vdash_t A \ @\ \square_i^{\alpha} \ \sim\ A' \ \because\ A_R \qquad \Delta, x \ @\ A \ \sim\ x' \ \because\ x_R \vdash_t M \ @\ B \ \sim\ M' \ \because\ M_R}{\Delta \vdash_t \lambda x : A.\, M \ @\ \Pi x : A.\, B \ \sim\ \lambda x' : A'.\, M' \ \because\ \lambda x\, x'\, x_R.\, M_R}\ (\textsc{TrocqLam})$$

$$\frac{\Delta \vdash_t A \ @\ \square_i^{\alpha} \ \sim\ A' \ \because\ A_R \qquad \Delta \vdash_t B \ @\ \square_i^{\beta} \ \sim\ B' \ \because\ B_R \qquad (\alpha,\beta) = \mathcal{D}_{\to}(\delta)}{\Delta \vdash_t A \to B \ @\ \square_i^{\delta} \ \sim\ A' \to B' \ \because\ p_{\to}^{\delta}\ A_R\ B_R}\ (\textsc{TrocqArrow})$$

$$\frac{\Delta \vdash_t A \ @\ \square_i^{\alpha} \ \sim\ A' \ \because\ A_R \qquad \Delta, x \ @\ A \ \sim\ x' \ \because\ x_R \vdash_t B \ @\ \square_i^{\beta} \ \sim\ B' \ \because\ B_R \qquad (\alpha,\beta) = \mathcal{D}_{\Pi}(\delta)}{\Delta \vdash_t \Pi x : A.\, B \ @\ \square_i^{\delta} \ \sim\ \Pi x' : A'.\, B' \ \because\ p_{\Pi}^{\delta}\ A_R\ B_R}\ (\textsc{TrocqPi})$$

$$\frac{\Delta \vdash_t M \ @\ A \ \sim\ M' \ \because\ M_R \qquad \gamma(\Delta) \vdash_+ A \lesssim B}{\Delta \vdash_t M \ @\ B \ \sim\ M' \ \because\ \Downarrow_B^A\ M_R}\ (\textsc{TrocqConv})$$

Fig. 8. Trocq rules

$$\Big\Downarrow_{\square_i^{\alpha'}}^{\square_i^{\alpha}} t_R := \Downarrow_{\alpha'}^{\alpha} t_R \qquad \Big\Downarrow_{A'\ M'}^{A\ M} N_R := \Big\Downarrow_{A'}^{A} M\ M'\ N_R \qquad \Big\Downarrow_{\lambda x:A'.\,B'}^{\lambda x:A.\,B} M\ M'\ N_R := \Big\Downarrow_{B'[x:=M']}^{B[x:=M]} N_R$$

$$\Big\Downarrow_{\Pi x:A'.\,B'}^{\Pi x:A.\,B} M_R := \lambda x\ x'\ x_R.\ \Big\Downarrow_{B'}^{B}\ \Big(M_R\ x\ x'\ \Big(\Big\Downarrow_{A}^{A'}\ x_R\Big)\Big) \qquad \Big\Downarrow_{A'}^{A} M_R := M_R$$

Fig. 9. Weakening of parametricity witnesses

The weakening function on parametricity witnesses is defined on Figure 9 by extending function $\Downarrow_{\beta}^{\alpha}$ to all relevant pairs of types of CC_{ω}^{+} , *i.e.*, $\Downarrow_U^T$ is defined for $T, U \in \mathcal{T}_{CC_{\omega}^{+}}$ as soon as $T \lesssim U$.

An abstraction theorem relates well-formed Trocq judgments and typing in CC_{ω}^{+} .

Theorem 4.7 (Trocq abstraction theorem).

$$\frac{\gamma(\Delta) \vdash_+ \qquad \gamma(\Delta) \vdash_+ M : A \qquad \Delta \vdash_t M \ @\ A \ \sim\ M' \ \because\ M_R \qquad \Delta \vdash_t A \ @\ \square_i^{\alpha} \ \sim\ A' \ \because\ A_R}{\gamma(\Delta) \vdash_+ M' : A' \qquad \textit{and} \qquad \gamma(\Delta) \vdash_+ M_R : \mathit{rel}(A_R)\ M\ M'}$$

Proof. By induction on derivation $\Delta \vdash_t M \ @\ A \ \sim\ M' \ \because\ M_R$. □

Note that type A in the typing hypothesis $\gamma(\Delta) \vdash_+ M : A$ of the abstraction theorem is exactly the extra information passed to the translation. The latter can thus also be seen as an inference algorithm, which infers annotations for the output of the translation from that of the input.

Remark 4.8. Since by definition of $p_{\square}^{\alpha,\beta}$ (Equation 11), we have $\vdash_t \square^\alpha \mathbin{@} \square^\beta \sim \square^\alpha \mathrel{\because} p_{\square}^{\alpha,\beta}$, by applying Theorem 4.7 with $\gamma(\Delta) \vdash_+ A : \square^\alpha$, we get:

$$\frac{\gamma(\Delta) \vdash_+ A : \square^\alpha \qquad \Delta \vdash_t A \mathbin{@} \square^\alpha \sim A' \mathrel{\because} A_R}{\gamma(\Delta) \vdash_+ A_R : \mathsf{rel}(p_{\square}^{\alpha,\beta})\ A\ A'}.$$

Now by the same definition, for any $\beta \in \mathcal{A}$, $\mathsf{rel}(p_{\square}^{\alpha,\beta}) = \boxdot^\alpha$, hence $\gamma(\Delta) \vdash A_R : \boxdot^\alpha\ A\ A'$, as expected by the type annotation $A : \square^\alpha$ in the input of the translation.

Remark 4.9. By applying the Remark 4.8 with $\vdash_+ \square^\alpha : \square^\beta$ we get $\vdash_+ p_{\square}^{\alpha,\beta} : \boxdot^\beta\ \square^\alpha\ \square^\alpha$ as expected, provided that $(\alpha, \beta) \in \mathcal{D}_\square$.

4.5 Constants

Concrete applications require extending Trocq with constants. Constants are similar to variables, except that they are stored in a global context instead of a typing context. A crucial difference though is that a constant may be assigned several different annotated types in CC_ω^+ .

Consider for example, a constant `list`, standing for the type of polymorphic lists. As `list` A is the type of lists with elements of type A, it can be annotated with type $\square^\alpha \to \square^\alpha$ for any $\alpha \in \mathcal{A}$.

Every constant declared in global environment has an associated collection of possible annotated types $T_c \subset \mathcal{T}_{CC_\omega^+}$. We require that all the possible annotated types of a same constant share the same erasure in CC_ω, *i.e.*, $\forall c, \forall A, \forall B,\ A, B \in T_c \Rightarrow |A|^- = |B|^-$. For example, $T_{\texttt{list}} = \{\square^\alpha \to \square^\alpha \mid \alpha \in \mathcal{A}\}$.

In addition, we provide translations $\mathcal{D}_c(A)$ for each possible annotated type A of each constant c in the global context. For example $\mathcal{D}_{\texttt{list}}(\square^{(1,0)} \to \square^{(1,0)})$ is well defined and equal to the translation

$$\big(\texttt{list}, \qquad \lambda A\, A'\, A_R.\ (\texttt{List.All2}\ A_R,\ \texttt{List.map}\ (\mathsf{map}(A_R))),\big)$$

where $\texttt{List.All2}\ A_R$ relates lists that are related by A_R element-wise, `List.map` is the standard map function on lists and $\mathsf{map}(A_R) : A \to A'$ extracts the *map* projection of the record A_R of type $\boxdot^{(1,0)}\ A\ A' \equiv \Sigma R.\, A \to A'$. Part of these translations can be generated automatically by weakening.

We describe in Figure 10 the additional rules for constants in CC_ω^+ and Trocq. Note that for an input term featuring constants, an unfortunate choice of annotation may lead a stuck translation.

$$\frac{c \in \mathcal{C} \qquad A \in T_c}{\Gamma \vdash c : A}\ (\textsc{Const}^+) \qquad\qquad \frac{\mathcal{D}_c(A) = (c', c_R)}{\Delta \vdash c \mathbin{@} A \sim c' \mathrel{\because} c_R}\ (\textsc{TrocqConst})$$

Fig. 10. Additional constant rules for CC_ω^+ and Trocq

5 IMPLEMENTATION AND APPLICATIONS

The supplementary material includes the source code of a plugin for Coq which provides a prototype implementation for Trocq, written in Elpi [Dunchev et al. 2015], a dialect of λProlog. We use Elpi as a meta-language for Coq, through the Coq-Elpi [Tassi 2019] plugin, which encodes Coq terms in higher-order abstract syntax, and provides a comprehensive API (typechecking, elaboration, interacting with the global environment, etc). The logic programming style of Elpi, as well as its approach to binder management proved particularly effective for the implementation of parametricity translations. Yet the implementation of Trocq also takes benefit of other features of Elpi, such as databases, constraint handling rules [Frühwirth and Raiser 2011], etc.

The core of the plugin consists in implementing each rule of the Trocq calculus, on top of Coq libraries formalizing the contents of Section 3. In the logic programming paradigm of Elpi,

each rule of Figure 8 translates gracefully into a corresponding λProlog predicate, making the corresponding source code very close to the presentation of §4.4. However, the Trocq plugin must also implement a much less trivial annotation inference algorithm, so as to hide the management of sort annotations to Coq users.

In this section we illustrate how the Trocq plugin covers the motivating examples given in Section 1. The supplementary material contains more examples, including an example of transfer from $\mathbb{Z}$ to a quotient $\mathbb{Z}/p\mathbb{Z}$[6], as well as examples showing that Trocq can be used to perform setoid rewriting[7] and generalized rewriting[8].

5.1 Example 1.1: transferring induction principles

The corresponding code to this example is available in the supplementary material[9]. Now recall that the problem here is obtain by proof transfer the following elimination scheme:

```
N_ind : ∀ P : N → □, P O_N → (∀ n : N, P n → P (S_N n)) → ∀ n : N, P n.
```

We first need to inform Trocq of three facts: that there is a split injection from `N` to $\mathbb{N}$, that zeros are related, and that successors are related:

```
N_R : Param2a3.Rel N ℕ
O_R : rel N_R O_N O_ℕ
S_R : ∀ m n, rel N_R m n → rel N_R (S_N m) (S_ℕ n).
```

Trocq is now able to generate and apply the implication:

```
(∀ P : ℕ → □, P O_ℕ → (∀ n : ℕ, P n → P (S_ℕ n)) → ∀ n : ℕ, P n)
  → ∀ P : N → □, P O_N → (∀ n : N, P n → P (S_N n)) → ∀ n : N, P n
```

5.2 Example 1.2: transferring results to a subtype

We setup an axiomatic context in Appendix E so as to state the goal on the type $\mathbb{R}_{\geq 0}$ of positive reals.[10] We prove the relation between this type and its extension $\overline{\mathbb{R}}_{\geq 0}$, their respective binary additions, infinite sums, and we declare them to Trocq. We can then prove:

```
Lemma Σ_{ℝ≥0}_add : forall u v : summable, Σ_{ℝ≥0} (u + v) = Σ_{ℝ≥0} u + Σ_{ℝ≥0} v.
Proof. trocq; exact: Σ_{ℝ̄≥0}_add. Qed.
```

6 RELATED WORK AND PERSPECTIVES

The functionalities of the prototype plugin presented in § 5 can be extended in several directions. It would be particularly fruitful to connect it with tools able to automate the generation of equivalence proofs, such as Pumpkin Pi [Ringer et al. 2021]. Other improvements, *e.g.*, addressing the case of Coq's impredicative sort, involve non-trivial implementation issues, related to Coq's management of universe polymorphism. We now discuss how the current state of this prototype compares with other implemented approaches to proof transfer in interactive theorem proving, listed in chronological order in the summary Table 1. For each such tool, the table indicates whether a given feature is available (✓), not available (✗) or only partially available (🩹 and ➡).

[6] File `int_to_Zp.v`
[7] File `trocq_setoid_rewrite.v`
[8] File `trocq_gen_rewrite.v`
[9] See file `peano_bin_nat.v`.
[10] Also see file `summable.v`

In the context of type theory, the idea that the computational content of type isomorphisms can be used for proof transfer already appears in [Barthe and Pons 2001]. The first implementation report of a tool based on this idea appeared soon after [Magaud 2003]. Implemented in a meta-language and based on proof rewriting, this heuristic translation was producing a candidate proof term from a given proof term, with no formal guarantee, not even that of being well-typed. As mentioned in § 2.1, generalized rewriting [Sozeau 2009], which generalizes setoid rewriting to preorders, is also a variant of proof transfer, albeit within the same type. As such, it allows in particular rewriting under binders. The restriction to homogeneous relations however excludes applications to quasi partial equivalence relations (QPER) [Krishnaswami and Dreyer 2013], or to datatype representation change.

The other proof transfer methods we are aware of all address the case of heterogeneous relations. Incidentally, they can thus also be used for the homogeneous case, as illustrated in § 2.1, although this special case is seldom emphasized. The Coq Effective Algebra Library (CoqEAL) [Cohen et al. 2013; Dénès et al. 2012] and the Isabelle/HOL transfer package [Haftmann et al. 2013; Huffman and Kunčar 2013; Lammich 2013; Lammich and Lochbihler 2019], pioneered the use of parametricity-based methods for proof transfer, motivated by the refinement of proof-oriented data-structures to computation-oriented counterparts. Together with a subsequent generalization of the CoqEAL approach [Zimmermann and Herbelin 2015], these tools address the case of a transfer between a subtype of a certain type A and a quotient of a certain type B, *i.e.*, the case of trivial QPER in which the zig-zag morphism is a partial surjection from A to B.

The next two columns of the table concern proof transfer in presence of the univalence principle, either axiomatic, in the case of univalent parametricity [Tabareau et al. 2021] or computational, in the case of [Angiuli et al. 2021b]. Key ingredients of the univalent parametricity were already present in earlier seemingly unpublished work [Anand and Morrisett 2017], implemented using an outdated ancestor of the MetaCoq library [Sozeau et al. 2020].

Table 1 indicates which tools can transfer along *heterogeneous relations*, as this is a prerequisite to changing type representation, and which ones operate by proving an *internal* implication lemma, as opposed to a monolithic translation of an input proof term. We borrow the terminology used in [Tabareau et al. 2021], in which *anticipation* refers to the need to define a dedicated structure for the signature to be transported. *Binders* can prevent transfer, as well as *dependent types*. The latter are recovered in presence of univalence. The first published publication [Tabareau et al. 2018] on the univalent parametricity translation suggested that the translation does not pull the axiom in when translating terms in the F^{ω} fragment. However, Trocq can get rid of it for a strictly larger class of terms. Finally, the table indicates which approaches can deal with *quasi-equivalence relations* (QER), and with (explicit) subtyping relations.

In its current state, the Trocq plugin can already address the proof transfer bureaucracy of state-of-the-art formal proofs, about abstract mathematics or program verification, or both [Allamigeon et al. 2023]. We expect that our work, once put in production, makes it possible to have the same lemma applicable to a wide variety of different types: isomorphic types, subtypes, and quotient types. This framework moreover opens the way to a broader range of extensions, *e.g.*, performing unification modulo both generalized rewriting and heterogeneous transfer relations, potentially solving problems sometimes referred to as *concept alignment*. We conclude with two concrete sticky issues in interactive theorem proving that such extensions could help addressing. The first one is the identification of canonical natural number objects in types, *e.g.*, $\{x : \mathbb{R} \mid \exists n : \mathbb{N}, x = \iota(n)\}$, etc. The last one is the identification of different parametric constructions, which happen to *coincide* for some specific classes of parameters, *e.g.*, the ring $\mathbb{Z}/q\mathbb{Z}$, defined for all integers $q > 0$, and the Galois field $\mathbb{F}_q$, defined when $q = p^k$, happen to be canonically isomorphic if and only if q is prime.

	[Magaud 2003]	Setoid rewrite [Sozeau 2009]	CoqEAL [Cohen et al. 2013]	Isabelle/HOL Transfer (2013)	[Zimmermann and Herbelin 2015]	[Tabareau et al. 2021]	[Angiuli et al. 2021b]	Trackt [Blot et al. 2023]	Trocq (2023)
Heterogeneous relations	✓	✗	✓	✓	✓	✓	✓	✓	✓
Internal	✗	✓	✓	✓	✓	✓	✓	✓	✓
No anticipation	✓	✓	✓	✓	✓	✓	✗	✓	✓
Substitution under ∀	✓	✓	✗	✓	✓	✓	✓	✓	✓
Substitution in dep. types	✓	✗	✗	✗	✗	✓	✓	✗	✓
No univalence for ?	✓	✓	✓	✓	✓	✗	✗	✓	✓
Preorder relations	✗	✓	?	?	?	✗	?	?	🖉
Subrelations	✗	✓	✗	✗	✗	✗	✗	✗	🖉
QERs	✗	🖉	➡	➡	➡	✗	✓	✗	➡
Subtyping relations	✗	✗	➡	➡	➡	✗	✗	➡	➡
System	Coq	Coq	Coq	Isabelle/HOL	Coq	Coq/HoTT	(Cubical) Agda	Coq	Coq or Coq/HoTT

Table 1. Comparison of proof transfer automation devices

REFERENCES

2020. The lean mathematical library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2020, New Orleans, LA, USA, January 20-21, 2020*, Jasmin Blanchette and Catalin Hritcu (Eds.). ACM, 367–381. https://doi.org/10.1145/3372885.3373824

Reynald Affeldt and Cyril Cohen. 2023. Measure Construction by Extension in Dependent Type Theory with Application to Integration. arXiv:2209.02345 [cs.LO] accepted for publication in JAR.

Xavier Allamigeon, Quentin Canu, and Pierre-Yves Strub. 2023. A Formal Disproof of Hirsch Conjecture. In *Proceedings of the 12th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2023, Boston, MA, USA, January 16-17, 2023*, Robbert Krebbers, Dmitriy Traytel, Brigitte Pientka, and Steve Zdancewic (Eds.). ACM, 17–29. https://doi.org/10.1145/3573105.3575678

Abhishek Anand and Greg Morrisett. 2017. Revisiting Parametricity: Inductives and Uniformity of Propositions. arXiv:1705.01163 [cs.LO]

Carlo Angiuli, Guillaume Brunerie, Thierry Coquand, Robert Harper, Kuen-Bang Hou (Favonia), and Daniel R. Licata. 2021a. Syntax and models of Cartesian cubical type theory. *Math. Struct. Comput. Sci.* 31, 4 (2021), 424–468. https://doi.org/10.1017/S0960129521000347

Carlo Angiuli, Evan Cavallo, Anders Mörtberg, and Max Zeuner. 2021b. Internalizing representation independence with univalence. *Proc. ACM Program. Lang.* 5, POPL (2021), 1–30. https://doi.org/10.1145/3434293

David Aspinall and Adriana B. Compagnoni. 2001. Subtyping dependent types. *Theor. Comput. Sci.* 266, 1-2 (2001), 273–309. https://doi.org/10.1016/S0304-3975(00)00175-4

Gilles Barthe, Venanzio Capretta, and Olivier Pons. 2003. Setoids in type theory. *J. Funct. Program.* 13, 2 (2003), 261–293. https://doi.org/10.1017/S0956796802004501

Gilles Barthe and Olivier Pons. 2001. Type Isomorphisms and Proof Reuse in Dependent Type Theory. In *Foundations of Software Science and Computation Structures*, Furio Honsell and Marino Miculan (Eds.). Springer Berlin Heidelberg, Berlin,

Heidelberg, 57–71.

Andrej Bauer, Jason Gross, Peter LeFanu Lumsdaine, Michael Shulman, Matthieu Sozeau, and Bas Spitters. 2017. The HoTT library: a formalization of homotopy type theory in Coq. In *Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs, CPP 2017, Paris, France, January 16-17, 2017*, Yves Bertot and Viktor Vafeiadis (Eds.). ACM, 164–172. https://doi.org/10.1145/3018610.3018615

Jean-Philippe Bernardy, Patrik Jansson, and Ross Paterson. 2012. Proofs for free - Parametricity for dependent types. *J. Funct. Program.* 22, 2 (2012), 107–152. https://doi.org/10.1017/S0956796812000056

Jean-Philippe Bernardy and Marc Lasson. 2011. Realizability and Parametricity in Pure Type Systems. In *Foundations of Software Science and Computational Structures - 14th International Conference, FOSSACS 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings (Lecture Notes in Computer Science, Vol. 6604)*, Martin Hofmann (Ed.). Springer, 108–122. https://doi.org/10.1007/978-3-642-19805-2_8

Valentin Blot, Denis Cousineau, Enzo Crance, Louise Dubois de Prisque, Chantal Keller, Assia Mahboubi, and Pierre Vial. 2023. Compositional Pre-processing for Automated Reasoning in Dependent Type Theory. In *Proceedings of the 12th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2023, Boston, MA, USA, January 16-17, 2023*, Robbert Krebbers, Dmitriy Traytel, Brigitte Pientka, and Steve Zdancewic (Eds.). ACM, 63–77. https://doi.org/10.1145/3573105.3575676

Simon Boulier, Pierre-Marie Pédrot, and Nicolas Tabareau. 2017. The next 700 syntactical models of type theory. In *Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs, CPP 2017, Paris, France, January 16-17, 2017*, Yves Bertot and Viktor Vafeiadis (Eds.). ACM, 182–194. https://doi.org/10.1145/3018610.3018620

Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. 2017. Cubical Type Theory: A Constructive Interpretation of the Univalence Axiom. *FLAP* 4, 10 (2017), 3127–3170. http://collegepublications.co.uk/ifcolog/?00019

Cyril Cohen, Maxime Dénès, and Anders Mörtberg. 2013. Refinements for Free!. In *Certified Programs and Proofs - Third International Conference, CPP 2013, Melbourne, VIC, Australia, December 11-13, 2013, Proceedings (Lecture Notes in Computer Science, Vol. 8307)*, Georges Gonthier and Michael Norrish (Eds.). Springer, 147–162. https://doi.org/10.1007/978-3-319-03545-1_10

Thierry Coquand and Gérard P. Huet. 1988. The Calculus of Constructions. *Inf. Comput.* 76, 2/3 (1988), 95–120. https://doi.org/10.1016/0890-5401(88)90005-3

Leonardo de Moura and Sebastian Ullrich. 2021. The Lean 4 Theorem Prover and Programming Language. In *Automated Deduction - CADE 28 - 28th International Conference on Automated Deduction, Virtual Event, July 12-15, 2021, Proceedings (Lecture Notes in Computer Science, Vol. 12699)*, André Platzer and Geoff Sutcliffe (Eds.). Springer, 625–635. https://doi.org/10.1007/978-3-030-79876-5_37

Maxime Dénès, Anders Mörtberg, and Vincent Siles. 2012. A Refinement-Based Approach to Computational Algebra in Coq. In *Interactive Theorem Proving*, Lennart Beringer and Amy Felty (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 83–98.

Cvetan Dunchev, Ferruccio Guidi, Claudio Sacerdoti Coen, and Enrico Tassi. 2015. ELPI: Fast, Embeddable, λProlog Interpreter. In *Logic for Programming, Artificial Intelligence, and Reasoning - 20th International Conference, LPAR-20 2015, Suva, Fiji, November 24-28, 2015, Proceedings (Lecture Notes in Computer Science, Vol. 9450)*, Martin Davis, Ansgar Fehnker, Annabelle McIver, and Andrei Voronkov (Eds.). Springer, 460–468. https://doi.org/10.1007/978-3-662-48899-7_32

Thom Frühwirth and Frank Raiser. 2011. Constraint Handling Rules: Compilation, Execution, and Analysis.

Sébastien Gouëzel. 2021. Vitali-Carathéodory theorem in mathlib. https://leanprover-community.github.io/mathlib_docs/measure_theory/integral/vitali_caratheodory.html.

Florian Haftmann, Alexander Krauss, Ondřej Kunčar, and Tobias Nipkow. 2013. Data Refinement in Isabelle/HOL. In *Interactive Theorem Proving*, Sandrine Blazy, Christine Paulin-Mohring, and David Pichardie (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 100–115.

Brian Huffman and Ondřej Kunčar. 2013. Lifting and Transfer: A Modular Design for Quotients in Isabelle/HOL. In *Certified Programs and Proofs*, Georges Gonthier and Michael Norrish (Eds.). Springer International Publishing, Cham, 131–146.

Chantal Keller and Marc Lasson. 2012. Parametricity in an Impredicative Sort. In *Computer Science Logic (CSL'12) - 26th International Workshop/21st Annual Conference of the EACSL, CSL 2012, September 3-6, 2012, Fontainebleau, France (LIPIcs, Vol. 16)*, Patrick Cégielski and Arnaud Durand (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 381–395. https://doi.org/10.4230/LIPIcs.CSL.2012.381

Neelakantan R. Krishnaswami and Derek Dreyer. 2013. Internalizing Relational Parametricity in the Extensional Calculus of Constructions. In *Computer Science Logic 2013 (CSL 2013) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 23)*, Simona Ronchi Della Rocca (Ed.). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 432–451. https://doi.org/10.4230/LIPIcs.CSL.2013.432

Peter Lammich. 2013. Automatic Data Refinement. In *Interactive Theorem Proving*, Sandrine Blazy, Christine Paulin-Mohring, and David Pichardie (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 84–99.

Peter Lammich and Andreas Lochbihler. 2019. Automatic Refinement to Efficient Data Structures: A Comparison of Two Approaches. *J. Autom. Reason.* 63, 1 (2019), 53–94. https://doi.org/10.1007/s10817-018-9461-9

Nicolas Magaud. 2003. Changing Data Representation within the Coq System. In *TPHOLs'2003*, Vol. 2758. LNCS, Springer-Verlag. http://www.springerlink.com/openurl.asp?genre=article&issn=0302-9743&volume=2758&spage=87 © Springer-Verlag.

Érik Martin-Dorel and Guillaume Melquiond. 2016. Proving Tight Bounds on Univariate Expressions with Elementary Functions in Coq. *J. Autom. Reason.* 57, 3 (2016), 187–217. https://doi.org/10.1007/s10817-015-9350-4

John C. Mitchell. 1986. Representation Independence and Data Abstraction. In *Conference Record of the Thirteenth Annual ACM Symposium on Principles of Programming Languages, St. Petersburg Beach, Florida, USA, January 1986*. ACM Press, 263–276. https://doi.org/10.1145/512644.512669

Rob Nederpelt and Herman Geuvers. 2014. *Type Theory and Formal Proof: An Introduction*. Cambridge University Press. https://doi.org/10.1017/CBO9781139567725

Ulf Norell. 2008. Dependently Typed Programming in Agda. In *Advanced Functional Programming, 6th International School, AFP 2008, Heijen, The Netherlands, May 2008, Revised Lectures (Lecture Notes in Computer Science, Vol. 5832)*, Pieter W. M. Koopman, Rinus Plasmeijer, and S. Doaitse Swierstra (Eds.). Springer, 230–266. https://doi.org/10.1007/978-3-642-04652-0_5

Christine Paulin-Mohring. 2015. Introduction to the Calculus of Inductive Constructions. In *All about Proofs, Proofs for All*, Bruno Woltzenlogel Paleo and David Delahaye (Eds.). Studies in Logic (Mathematical logic and foundations), Vol. 55. College Publications. https://inria.hal.science/hal-01094195

John C. Reynolds. 1983. Types, Abstraction and Parametric Polymorphism. In *Information Processing 83, Proceedings of the IFIP 9th World Computer Congress, Paris, France, September 19-23, 1983*, R. E. A. Mason (Ed.). North-Holland/IFIP, 513–523.

Talia Ringer, RanDair Porter, Nathaniel Yazdani, John Leo, and Dan Grossman. 2021. Proof repair across type equivalences. In *PLDI '21: 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, Virtual Event, Canada, June 20-25, 2021*, Stephen N. Freund and Eran Yahav (Eds.). ACM, 112–127. https://doi.org/10.1145/3453483.3454033

Matthieu Sozeau. 2009. A New Look at Generalized Rewriting in Type Theory. *J. Formaliz. Reason.* 2, 1 (2009), 41–62. https://doi.org/10.6092/issn.1972-5787/1574

Matthieu Sozeau, Abhishek Anand, Simon Boulier, Cyril Cohen, Yannick Forster, Fabian Kunze, Gregory Malecha, Nicolas Tabareau, and Théo Winterhalter. 2020. The MetaCoq Project. *J. Autom. Reason.* 64, 5 (2020), 947–999. https://doi.org/10.1007/s10817-019-09540-0

Nicolas Tabareau, Éric Tanter, and Matthieu Sozeau. 2018. Equivalences for free: univalent parametricity for effective transport. *Proceedings of the ACM on Programming Languages* 2, ICFP (2018), 1–29.

Nicolas Tabareau, Éric Tanter, and Matthieu Sozeau. 2021. The marriage of univalence and parametricity. *Journal of the ACM (JACM)* 68, 1 (2021), 1–44.

Enrico Tassi. 2019. Deriving proved equality tests in Coq-elpi: Stronger induction principles for containers in Coq. In *ITP 2019 - 10th International Conference on Interactive Theorem Proving*. Portland, United States. https://doi.org/10.4230/LIPIcs.CVIT.2016.23

The Coq Development Team. 2022. *The Coq Proof Assistant*. https://doi.org/10.5281/zenodo.7313584

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

Andrea Vezzosi, Anders Mörtberg, and Andreas Abel. 2019. Cubical agda: a dependently typed programming language with univalence and higher inductive types. *Proc. ACM Program. Lang.* 3, ICFP (2019), 87:1–87:29. https://doi.org/10.1145/3341691

Philip Wadler. 1989. Theorems for Free!. In *Proceedings of the fourth international conference on Functional programming languages and computer architecture, FPCA 1989, London, UK, September 11-13, 1989*, Joseph E. Stoy (Ed.). ACM, 347–359. https://doi.org/10.1145/99370.99404

Théo Zimmermann and Hugo Herbelin. 2015. Automatic and Transparent Transfer of Theorems along Isomorphisms in the Coq Proof Assistant. In *Conference on Intelligent Computer Mathematics*. Washington, D.C., United States. https://hal.science/hal-01152588

A THE CALCULUS OF CONSTRUCTIONS WITH UNIVERSES CC_ω

We recall the rules of the calculus of constructions (e.g. [Nederpelt and Geuvers 2014; Paulin-Mohring 2015]) in figure 11, and rely on folklore definitions of the relation $\preccurlyeq$.

$$\frac{\Gamma \vdash M : A \qquad \Gamma \vdash A \equiv B}{\Gamma \vdash M : B}\ (\textsc{Conv}) \qquad \frac{\Gamma \vdash}{\Gamma \vdash \square_i : \square_{i+1}}\ (\textsc{Sort}) \qquad \frac{(x, A) \in \Gamma \qquad \Gamma \vdash}{\Gamma \vdash x : A}\ (\textsc{Var})$$

$$\frac{\Gamma \vdash A : \square_i \qquad x \notin \mathrm{Var}(\Gamma)}{\Gamma,\ x : A \vdash}\ (\textsc{Context}) \qquad \frac{\Gamma \vdash M : \Pi x : A.\, B \qquad \Gamma \vdash N : A}{\Gamma \vdash M\ N : B[x := N]}\ (\textsc{App})$$

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x : A.\, M : \Pi x : A.\, B}\ (\textsc{Lam}) \qquad \frac{\Gamma \vdash A : \square_i \qquad \Gamma, x : A \vdash B : \square_i}{\Gamma \vdash \Pi x : A.\, B : \square_i}\ (\textsc{Pi})$$

Fig. 11. typing rules for CC_ω

B ERASURE OF ANNOTATIONS

We show that our extension is conservative over CC_ω, by defining an erasure function for terms $|\cdot|^- : \mathcal{T}_{CC^+_\omega} \to \mathcal{T}_{CC_\omega}$ and for contexts, defined in Figure 12.

$$\begin{aligned} \left|\square_i^C\right|^- &:= \square_i & |\varepsilon|^- &:= \varepsilon \\ |\Pi x : A.\, B|^- &:= \Pi x : |A|^-.\, |B|^- & |\Gamma, x : A|^- &:= \Gamma, x : |A|^- \\ |\lambda x : A.\, B|^- &:= \lambda x : |A|^-.\, |B|^- \\ |T\ U|^- &:= |T|^-\ |U|^- \\ |x|^- &:= x \end{aligned}$$

Fig. 12. Erasure function from CC^+_ω to CC_ω

We show that the erasure of subtyping is convertibility in CC_ω:

Lemma B.1 (Subtyping erasure).

$$\Gamma \vdash_{CC^+_\omega} A \preccurlyeq B \implies |\Gamma|^- \vdash_{CC_\omega} |A|^- \equiv |B|^-$$

Proof. By induction on the derivation. □

Finally we show that our extension is conservative

Theorem B.2 (Annotation erasure).

$$\Gamma \vdash_{CC^+_\omega} t : A \implies |\Gamma|^- \vdash_{CC_\omega} |t|^- \equiv |A|^-$$

Proof. By induction on the derivation. □

C ERASURE OF TROCQ TO PARAM

We show that Trocq entails raw parametricity after all annotations are erased.

Theorem C.1 (Erasure of Trocq).

$$\forall t, A, t', t_R \in \mathcal{T}_{CC^+_\omega}, \quad \Delta \vdash t \mathbin{@} A \sim t' \mathbin{\because} t_R \implies |\Delta|^- \vdash |t|^- \sim |t'|^- \mathbin{\because} rel^*(t_R)$$

where

$$\begin{aligned} rel^*(MN) &:= rel^*(M)\ rel^*(N) \\ rel^*(\lambda x.t) &:= \lambda x.rel^*(t) \\ rel^*(t) &:= rel(t) \end{aligned}$$

$$\begin{aligned} |\varepsilon|^- &:= \varepsilon \\ |\Delta,\ x \mathbin{@} A \sim x' \mathbin{\because} x_R|^- &:= |\Delta|^-,\ x \sim x' \mathbin{\because} x_R \end{aligned}$$

Proof. By induction on the derivation. □

D RECOVERING UPARAM FROM TROCQ

We show we can recover univalent parametricity by defining a maximal annotation function for terms $|\cdot|^+ : \mathcal{T}_{CC_\omega} \to \mathcal{T}_{CC^+_\omega}$ and for contexts, defined in Figure 13.

$$\begin{aligned} |\square_i|^+ &:= \square_i^\top \\ |\Pi x : A.\, B|^+ &:= \Pi x : |A|^+ .\, |B|^+ \\ |\lambda x : A.\, B|^+ &:= \lambda x : |A|^+ .\, |B|^+ \\ |T\ U|^+ &:= |T|^+\ |U|^+ \\ |x|^+ &:= x \end{aligned}$$

$$\begin{aligned} |\varepsilon|^+ &:= \varepsilon \\ |\Gamma, x : A|^+ &:= \Gamma, x : |A|^+ \end{aligned}$$

Fig. 13. Maximal annotation function from CC_ω to CC^+_ω

Indeed, we have the following theorem.

Theorem D.1 (Maximal Trocq).

$$\forall t, A, t', t_R \in \mathcal{T}_{CC_\omega}, \quad \Delta \vdash |t|^+ \mathbin{@} |A|^+ \sim |t'|^+ \mathbin{\because} |t_R|^+ \iff |\Delta|^- \vdash_u t \sim t' \mathbin{\because} t_R \quad \wedge \quad \gamma(\Delta) \vdash t : A$$

Proof. By induction on the derivation. □

E DETAILED EXAMPLE

```
(* We postulate the bare minimum about non-negative reals *)
Axioms (ℝ≥0 : Type) (O_ℝ≥0 : ℝ≥0) (+_ℝ≥0 : ℝ≥0 → ℝ≥0 → ℝ≥0).
(* Non-negative extended reals are a trivial extension *)
Inductive ℝ̄≥0 : Type := Fin : ℝ≥0 → ℝ̄≥0 | Inf : ℝ̄≥0.
(* We define the notions of sequences of numbers *)
```

```
Definition seq_{\overline{\mathbb{R}}_{≥0}} := nat → \overline{\mathbb{R}}_{≥0}.
Definition seq_{\mathbb{R}_{≥0}} := nat → \mathbb{R}_{≥0}.
(* Addition on the extended non-negative reals is definable *)
Definition a +_{\overline{\mathbb{R}}_{≥0}} b : \overline{\mathbb{R}}_{≥0} := match a, b with
  Fin x, Fin y ⇒ Fin (r1 +_{\mathbb{R}_{≥0}} r2) | _, _ ⇒ Inf end.
(* We can derive the addition on sequences *)
Definition u +_{seq_{\overline{\mathbb{R}}_{≥0}}} v : seq_{\overline{\mathbb{R}}_{≥0}} := fun n ⇒ u n +_{\overline{\mathbb{R}}_{≥0}} v n.

(* We now postulate the unconditional infinite summation
   on extended non-negative reals and its linearity *)
Axiom Σ_{\overline{\mathbb{R}}_{≥0}} : seq_{\overline{\mathbb{R}}_{≥0}} → \overline{\mathbb{R}}_{≥0}.
Axiom Σ_{\overline{\mathbb{R}}_{≥0}}_add : forall u v : seq_{\overline{\mathbb{R}}_{≥0}}, Σ_{\overline{\mathbb{R}}_{≥0}} (u + v) = Σ_{\overline{\mathbb{R}}_{≥0}} u + Σ_{\overline{\mathbb{R}}_{≥0}} v.

(* We define the notion of summable sequence *)
Definition isFin (a : \overline{\mathbb{R}}_{≥0}) := match a with Fin _ ⇒ true | _ ⇒ false end.
Definition truncate (a : \overline{\mathbb{R}}_{≥0}) := match a with Fin x ⇒ x | _ => 0_{\mathbb{R}_{≥0}} end.
Definition isSummable (u : seq_{\mathbb{R}_{≥0}}) := isFin (Σ_{\overline{\mathbb{R}}_{≥0}} (Fin ∘ u)).

(* We define the type of summable sequences *)
Record summable := {to_seq :> seq_{\mathbb{R}_{≥0}}; _ : isSummable to_seq}.

(* We postulate that summability is preserved by binary addition *)
Axiom summable_add :
  forall u v : summable, isSummable (fun n ⇒ u n +_{\mathbb{R}_{≥0}} v n) = true.
Definition u +_{summable} v : summable := Build_summable _ (Σ_{\overline{\mathbb{R}}_{≥0}}_add u v).

(* We define infinite sums on summable sequences *)
Definition Σ_{\mathbb{R}_{≥0}} (u : summable) : \mathbb{R}_{≥0} := truncate (Σ_{\overline{\mathbb{R}}_{≥0}} (Fin ∘ u)).
```

We then register various lemmas in Trocq, so that tactic `trocq` can achieve the desired transfer.

```
(* Finally, we transfer the proof *)
Lemma Σ_{\mathbb{R}_{≥0}}_add : forall u v : summable, Σ_{\mathbb{R}_{≥0}} (u + v) = Σ_{\mathbb{R}_{≥0}} u + Σ_{\mathbb{R}_{≥0}} v.
Proof. trocq; exact: Σ_{\overline{\mathbb{R}}_{≥0}}_add. Qed.
```