From Rewrite Rules to Axioms in the $\lambda\Pi$-Calculus Modulo Theory

Valentin Blot[1], Gilles Dowek[1], Thomas Traversié[1,2(✉)], and Théo Winterhalter[1]

[1] Université Paris-Saclay, Inria, ENS Paris-Saclay, CNRS, LMF, Gif-sur-Yvette, France
{valentin.blot,gilles.dowek,thomas.traversie,theo.winterhalter}@inria.fr
[2] Université Paris-Saclay, CentraleSupélec, MICS, Gif-sur-Yvette, France

Abstract. The $\lambda\Pi$-calculus modulo theory is an extension of simply typed λ-calculus with dependent types and user-defined rewrite rules. We show that it is possible to replace the rewrite rules of a theory of the $\lambda\Pi$-calculus modulo theory by equational axioms, when this theory features the notions of proposition and proof, while maintaining the same expressiveness. To do so, we introduce in the target theory a heterogeneous equality, and we build a translation that replaces each use of the conversion rule by the insertion of a transport. At the end, the theory with rewrite rules is a conservative extension of the theory with axioms.

Keywords: Rewrite rules · Equality · Logical Framework.

1 Introduction

For Poincaré, the reasoning by which we deduce that $2+2 = 4$ is not a meaningful proof, but a simple verification. He concludes that the goal of exact sciences is to "dispense with these direct verifications" [20]. Far from being solely a philosophical issue, this principle impacts the foundations of logical systems and in particular the choice between *axioms* and *rewrite rules*. For instance, in systems with axioms $x+\mathsf{succ}\ y = \mathsf{succ}\ (x+y)$ and $x+0 = x$, we can *prove* that $2+2 = 4$. On the other hand, in systems with rewrite rules $x+\mathsf{succ}\ y \hookrightarrow \mathsf{succ}\ (x+y)$ and $x+0 \hookrightarrow x$, we just need to prove $4 = 4$ as we can *compute* that $(2 + 2 = 4) \equiv (4 = 4)$. In that respect, logical systems with computation rules are convenient tools for making proofs. That is why rewrite rules have been added to systems such as Agda [5] or Coq [12] and why Dowek [9,10] developed Deduction modulo theory, an extension of first-order logic that mixes computation and proof. Since logical systems with rewrite rules are more user-friendly, one may ask whether or not the results are the same as in axiomatic logical systems.

Rewrite rules are at the core of the $\lambda\Pi$-calculus modulo theory, an extension of simply typed λ-calculus with dependent types and user-definable rewrite rules [6]. The combination of β-reduction and of the rewrite rules of a signature Σ forms the conversion $\equiv_{\beta\Sigma}$. If we know that $t : A$ with conversion $A \equiv_{\beta\Sigma} B$,

N. Kobayashi and J. Worrell (Eds.): FoSSaCS 2024, LNCS 14575, pp. 3–23, 2024.
https://doi.org/10.1007/978-3-031-57231-9_1

then we can derive that $t : B$. In this system, a theory is a set of rewrite rules, together with a set of axioms (that are typed constants). The $\lambda\Pi$-calculus modulo theory is a powerful logical framework in which many theories can be expressed, such as Predicate logic, Simple type theory or the Calculus of constructions [3]. It is the theory behind the Dedukti language [2,16] and the Lambdapi proof assistant.

In this paper, we choose to study the replacement of rewrite rules by axioms in the $\lambda\Pi$-calculus modulo theory. Since it is a logical framework, the result applies to many theories. Moreover, as Dedukti is geared towards the interoperability between proof systems, if we want to exchange proofs between a system with rewrite rules and a system without rewrite rules *via* Dedukti, we need to replace rewrite rules by axioms in the $\lambda\Pi$-calculus modulo theory. Working in this logical framework rather than in an extension of Martin-Löf type theory [17] is therefore relevant on both theoretical and practical levels, but complicates the task as the $\lambda\Pi$-calculus modulo theory does not feature identity types or an infinite hierarchy of sorts.

One method to replace rewrite rules by axioms is to mimic the behavior of the conversion rule using transports: if we have $t : A$ and $A \equiv_{\beta\Sigma} B$ with p an equality between A and B, then we can deduce that $\mathsf{transp}\ p\ t : B$, but we do not directly have $t : B$. However trivial this seems, we face several challenges when trying to demonstrate it fully: the insertion of transports in terms and types is difficult due to the presence of dependent types, and the building of transports is involved as we cannot have inside the $\lambda\Pi$-calculus modulo theory an equality between types.

A similar problem is the elimination of equality reflection from extensional systems. Equality reflection states that $\ell = r$ implies $\ell \equiv r$, just like $\ell \hookrightarrow r$ implies $\ell \equiv r$ in systems with rewrite rules. In extensional systems, typing is eased by a more powerful conversion. Hofmann [14,15] investigated categorically the problem. Oury [19] developed a translation of proofs from an extensional version of the Calculus of Constructions to the Calculus of Inductive Constructions with equality axioms. Winterhalter, Sozeau and Tabareau [23,24] built upon this result to reduce the number of axioms needed.

The replacement of rewrite rules by axioms paves the way for the interpretation of a theory into another inside the $\lambda\Pi$-calculus modulo theory. Indeed, when interpreting a theory into another, we represent each constant of the source theory by a term in the target theory, but we cannot generally do the same for rewrite rules. We can however pre-process the source theory to replace its rewrite rules by axioms, and then interpret it. The interpretation of theories allows to prove relative consistency and relative normalization theorems [8].

Contribution. The main contribution of this paper is the translation of a theory with rewrite rules to a theory with equational axioms. To do so, we restrict the theories considered to theories with an encoding of the notions of proposition and proof inside the $\lambda\Pi$-calculus modulo theory. So as to compare objects that possibly do not have the same type, we define a heterogeneous equality—following the one defined by McBride [18]. The restriction considered allows us to build an

equality between particular types—called small types. We define a type system with typed conversion for the $\lambda\Pi$-calculus modulo theory, so that the proofs are done by induction on the derivation trees more easily.

Outline of the paper. In Section 2, we present the $\lambda\Pi$-calculus modulo theory, we detail a prelude encoding of the notions of proposition and proof in it, and we identify the assumptions made on the considered theories. The heterogeneous equality and the equality between small types are presented in Section 3. The replacement of rewrite rules by axioms and the translation of terms, judgments and theories are presented in Section 4.

2 Theories in the $\lambda\Pi$-Calculus Modulo Theory

In this section, we give a more detailed overview of the $\lambda\Pi$-calculus modulo theory [6] and its type system. In particular, we present an encoding of the notions of proposition and proof in the $\lambda\Pi$-calculus modulo theory [3]. We characterize small types—a subclass of types for which we can define an equality.

2.1 The $\lambda\Pi$-Calculus Modulo Theory

The $\lambda\Pi$-calculus, also known as the Edinburgh Logical Framework [13], is an extension of simply typed λ-calculus with dependent types. The $\lambda\Pi$-calculus modulo theory ($\lambda\Pi/\equiv$) [6] is an extension of the $\lambda\Pi$-calculus, in which user-definable rewrite rules have been added [7]. Its syntax is given by:

$$
\begin{array}{ll}
\textit{Sorts} & s ::= \texttt{TYPE} \mid \texttt{KIND} \\
\textit{Terms} & t, u, A, B ::= c \mid x \mid s \mid \Pi x : A.\ B \mid \lambda x : A.\ t \mid t\ u \\
\textit{Contexts} & \Gamma ::= \langle\rangle \mid \Gamma, x : C \\
\textit{Signatures} & \Sigma ::= \langle\rangle \mid \Sigma, c : D \mid \Sigma, \ell \hookrightarrow r
\end{array}
$$

where c is a constant and x is a variable (ranging over disjoint sets), C and r are terms, D is a closed term (*i.e.* a term with no free variables) and ℓ is a term such that $\ell = c\ t_1 \dots t_k$ with c a constant. `TYPE` and `KIND` are two sorts: terms of type `TYPE` are called types, and terms of type `KIND` are called kinds. $\Pi x : A.\ B$ is a dependent product, $\lambda x : A.\ t$ is an abstraction and $t\ u$ is an application. $\Pi x : A.\ B$ is simply written $A \to B$ if x does not appear in B. Signatures and contexts are finite sequences, and are written $\langle\rangle$ when empty. Signatures contain both typed constants and rewrite rules (written $\ell \hookrightarrow r$). $\lambda\Pi/\equiv$ is a logical framework, in which Σ is fixed by the user depending on the logic they are working in.

The relation $\hookrightarrow_{\beta\Sigma}$ is generated by β-reduction and by the rules of Σ. More explicitly, $\hookrightarrow_{\beta\Sigma}$ is the smallest relation, closed by context, such that if t rewrites to u for some rule in Σ or by β-reduction then $t \hookrightarrow_{\beta\Sigma} u$. Conversion $\equiv_{\beta\Sigma}$ is the reflexive, symmetric, and transitive closure of $\hookrightarrow_{\beta\Sigma}$.

2.2 The Type System of the $\lambda\Pi$-Calculus Modulo Theory

We introduce in Figs. 1 and 2 typing rules for $\lambda\Pi/\equiv$. Fig. 1 presents the usual typing rules while Fig. 2 focuses on the conversion rules. We write $\vdash \Gamma$ when the context Γ is well formed and $\Gamma \vdash t : A$ when t is of type A in the context Γ. $\langle\rangle \vdash t : A$ is simply written $\vdash t : A$. The notation $(\vdash \Gamma_1) \equiv (\vdash \Gamma_2)$ means that Γ_1 and Γ_2 are both well formed, have the same length and have the same variables with convertible types. We write $(\Gamma_1 \vdash t_1 : A_1) \equiv (\Gamma_2 \vdash t_2 : A_2)$ when t_1 and t_2 are convertible with $\Gamma_1 \vdash t_1 : A_1$ and $\Gamma_2 \vdash t_2 : A_2$. In particular, convertible terms $t_1 \equiv t_2$ are authorized to have different types—provided that both types are convertible—and to be typed in different contexts—provided that both contexts are convertible. In ConvRule, $\boldsymbol{x}$ is a vector representing the free variables of ℓ. The standard weakening rule and substitution lemma can be derived from this type system.

$$\frac{}{\vdash \langle\rangle}\ [\textsc{Empty}] \qquad \frac{\vdash \Gamma \qquad \Gamma \vdash A : s}{\vdash \Gamma, x : A}\ [\textsc{Decl}]\ x \notin \Gamma \qquad \frac{\vdash \Gamma}{\Gamma \vdash \texttt{TYPE} : \texttt{KIND}}\ [\textsc{Sort}]$$

$$\frac{\vdash \Gamma \qquad \vdash A : s}{\Gamma \vdash c : A}\ [\textsc{Const}]\ c : A \in \Sigma \qquad \frac{\vdash \Gamma}{\Gamma \vdash x : A}\ [\textsc{Var}]\ x : A \in \Gamma$$

$$\frac{\Gamma \vdash A : \texttt{TYPE} \qquad \Gamma, x : A \vdash B : s}{\Gamma \vdash \Pi x : A.\ B : s}\ [\textsc{Prod}]$$

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

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

$$\frac{\Gamma \vdash t : A \qquad (\Gamma \vdash A : s) \equiv (\Gamma \vdash B : s)}{\Gamma \vdash t : B}\ [\textsc{Conv}]$$

Fig. 1. Typing rules of the $\lambda\Pi$-calculus modulo theory

Lemma 1 (Substitution).

- *If we have* $\vdash \Gamma, x : A, \Delta$ *and* $\Gamma \vdash u : A$, *then* $\vdash \Gamma, \Delta[x \mapsto u]$.
- *If we have* $\Gamma, x : A, \Delta \vdash t : B$ *and* $\Gamma \vdash u : A$, *then* $\Gamma, \Delta[x \mapsto u] \vdash t[x \mapsto u] : B[x \mapsto u]$.
- *If we have* $(\vdash \Gamma_1, x : A_1, \Delta_1) \equiv (\vdash \Gamma_2, x : A_2, \Delta_2)$ *and* $\Gamma_1 \vdash u : A_1$, *then* $(\vdash \Gamma_1, \Delta_1[x \mapsto u]) \equiv (\vdash \Gamma_2, \Delta_2[x \mapsto u])$.

$$\frac{\Gamma \vdash u : A}{(\Gamma \vdash u : A) \equiv (\Gamma \vdash u : A)}\ [\textsc{ConvRefl}] \qquad \frac{(\Gamma \vdash u : A) \equiv (\Gamma \vdash v : B)}{(\Gamma \vdash v : B) \equiv (\Gamma \vdash u : A)}\ [\textsc{ConvSym}]$$

$$\frac{(\Gamma \vdash u : A) \equiv (\Gamma \vdash v : B) \qquad (\Gamma \vdash v : B) \equiv (\Gamma \vdash w : C)}{(\Gamma \vdash u : A) \equiv (\Gamma \vdash w : C)}\ [\textsc{ConvTrans}]$$

$$\frac{(\vdash \Gamma_1) \equiv (\vdash \Gamma_2) \qquad (\Gamma_1 \vdash A_1 : s) \equiv (\Gamma_2 \vdash A_2 : s)}{(\vdash \Gamma_1, x : A_1) \equiv (\vdash \Gamma_2, x : A_2)}\ [\textsc{ConvDecl}]\ x \notin \Gamma_1, \Gamma_2$$

$$\frac{(\vdash \Gamma_1) \equiv (\vdash \Gamma_2) \qquad \vdash A : s}{(\Gamma_1 \vdash c : A) \equiv (\Gamma_2 \vdash c : A)}\ [\textsc{ConvConst}]\ c : A \in \Sigma$$

$$\frac{(\vdash \Gamma_1) \equiv (\vdash \Gamma_2)}{(\Gamma_1 \vdash x : A_1) \equiv (\Gamma_2 \vdash x : A_2)}\ [\textsc{ConvVar}]\ x : A_1 \in \Gamma_1, x : A_2 \in \Gamma_2$$

$$\frac{\begin{array}{c}(\Gamma_1 \vdash A_1 : \texttt{TYPE}) \equiv (\Gamma_2 \vdash A_2 : \texttt{TYPE}) \\ (\Gamma_1, x : A_1 \vdash B_1 : s) \equiv (\Gamma_2, x : A_2 \vdash B_2 : s)\end{array}}{(\Gamma_1 \vdash \Pi x : A_1.\ B_1 : s) \equiv (\Gamma_2 \vdash \Pi x : A_2.\ B_2 : s)}\ [\textsc{ConvProd}]$$

$$\frac{\begin{array}{c}(\Gamma_1 \vdash A_1 : \texttt{TYPE}) \equiv (\Gamma_2 \vdash A_2 : \texttt{TYPE}) \\ (\Gamma_1, x : A_1 \vdash B_1 : s) \equiv (\Gamma_2, x : A_2 \vdash B_2 : s) \\ (\Gamma_1, x : A_1 \vdash t_1 : B_1) \equiv (\Gamma_2, x : A_2 \vdash t_2 : B_2)\end{array}}{(\Gamma_1 \vdash \lambda x : A_1.\ t_1 : \Pi x : A_1.\ B_1) \equiv (\Gamma_2 \vdash \lambda x : A_2.\ t_2 : \Pi x : A_2.\ B_2)}\ [\textsc{ConvAbs}]$$

$$\frac{\begin{array}{c}(\Gamma_1 \vdash t_1 : \Pi x : A_1.\ B_1) \equiv (\Gamma_2 \vdash t_2 : \Pi x : A_2.\ B_2) \\ (\Gamma_1 \vdash u_1 : A_1) \equiv (\Gamma_2 \vdash u_2 : A_2)\end{array}}{(\Gamma_1 \vdash t_1\ u_1 : B_1[x \mapsto u_1]) \equiv (\Gamma_2 \vdash t_2\ u_2 : B_2[x \mapsto u_2])}\ [\textsc{ConvApp}]$$

$$\frac{\Gamma \vdash A : \texttt{TYPE} \qquad \Gamma, x : A \vdash t : B \qquad \Gamma, x : A \vdash B : s \qquad \Gamma \vdash u : A}{(\Gamma \vdash (\lambda x : A.\ t)\ u : B[x \mapsto u]) \equiv (\Gamma \vdash t[x \mapsto u] : B[x \mapsto u])}\ [\textsc{ConvBeta}]$$

$$\frac{\boldsymbol{x} : \boldsymbol{B} \vdash \ell : A \qquad \boldsymbol{x} : \boldsymbol{B} \vdash r : A \qquad \Gamma \vdash \boldsymbol{t} : \boldsymbol{B}}{(\Gamma \vdash \ell[\boldsymbol{x} \mapsto \boldsymbol{t}] : A[\boldsymbol{x} \mapsto \boldsymbol{t}]) \equiv (\Gamma \vdash r[\boldsymbol{x} \mapsto \boldsymbol{t}] : A[\boldsymbol{x} \mapsto \boldsymbol{t}])}\ [\textsc{ConvRule}]\ \ell \hookrightarrow r \in \Sigma$$

$$\frac{\Gamma \vdash u : A \qquad (\Gamma \vdash A : s) \equiv (\Gamma \vdash B : s)}{(\Gamma \vdash u : A) \equiv (\Gamma \vdash u : B)}\ [\textsc{ConvConv}]$$

Fig. 2. Convertibility rules of the $\lambda\Pi$-calculus modulo theory

– *If we have* $(\Gamma_1, x : A_1, \Delta_1 \vdash t_1 : B_1) \equiv (\Gamma_2, x : A_2, \Delta_2 \vdash t_2 : B_2)$ *and* $\Gamma_1 \vdash u : A_1$*, then* $(\Gamma_1, \Delta_1[x \mapsto u] \vdash t_1[x \mapsto u] : B_1[x \mapsto u]) \equiv (\Gamma_2, \Delta_2[x \mapsto u] \vdash t_2[x \mapsto u] : B_2[x \mapsto u])$.

Proof. We proceed by induction on the typing derivation.

We chose to present a type system with *typed* conversion (written $\equiv$)—so as to easily do proofs on the derivations—while the usual type system for $\lambda\Pi/\equiv$ features *untyped* conversion (written $\equiv_{\beta\Sigma}$). The equivalence between type systems with typed conversion and type systems with untyped conversion has been a longstanding question: Geuvers and Werner [11] investigated the case of Pure Type Systems with $\beta\eta$-convertibility, Adams [1] proved the equivalence in the case of functional Pure Type Systems, and Siles [21,22] later proved the equivalence in the general case of the Pure Type Systems. The case of $\lambda\Pi/\equiv$, in which we have β-convertibility but also user-defined rewrite rules, remains to be investigated.

We write $|\Sigma|$ for the set of constants of Σ, and $\Lambda(\Sigma)$ for the set of terms t whose constants belong to $|\Sigma|$. We say that $\mathcal{T} = \Sigma$ is a theory when for each rule $\ell \hookrightarrow r \in \Sigma$ we have ℓ and r in $\Lambda(\Sigma)$, when $\hookrightarrow_{\beta\Sigma}$ is confluent on $\Lambda(\Sigma)$, and when every rule of Σ preserves typing in Σ (that is when for all context Γ and for all term $A \in \Lambda(\Sigma)$, if $\Gamma \vdash \ell : A$ then $\Gamma \vdash r : A$).

Example 1 (Natural numbers and lists). We can define in $\lambda\Pi/\equiv$ a partial theory of natural numbers and indexed lists of natural numbers. nat represents the type of natural numbers and list represents the dependent type of indexed lists of natural numbers. cons adds a new element to a list, concat concatenates two lists, and isRev checks if the first given list is the reverse of the second.

$$\mathsf{nat} : \mathtt{TYPE} \qquad 0 : \mathsf{nat} \qquad \mathsf{succ} : \mathsf{nat} \to \mathsf{nat} \qquad + : \mathsf{nat} \to \mathsf{nat} \to \mathsf{nat}$$

$$x + 0 \hookrightarrow x \qquad x + \mathsf{succ}\ y \hookrightarrow \mathsf{succ}\ (x + y) \qquad \mathsf{list} : \mathsf{nat} \to \mathtt{TYPE} \qquad \mathsf{nil} : \mathsf{list}\ 0$$

$$\mathsf{cons} : \Pi x : \mathsf{nat}.\ \mathsf{list}\ x \to \mathsf{nat} \to \mathsf{list}\ (\mathsf{succ}\ x)$$

$$\mathsf{isRev} : \Pi x : \mathsf{nat}.\ \mathsf{list}\ x \to \mathsf{list}\ x \to \mathtt{TYPE}$$

$$\mathsf{concat} : \Pi x, y : \mathsf{nat}.\ \mathsf{list}\ x \to \mathsf{list}\ y \to \mathsf{list}\ (x + y)$$

In the context $\ell : \mathsf{list}\ (\mathsf{succ}\ 0)$, we have $\mathsf{concat}\ (\mathsf{succ}\ 0)\ 0\ \ell\ \mathsf{nil}$ of type $\mathsf{list}\ (\mathsf{succ}\ 0 + 0)$. If we want to compare ℓ and this new list with isRev, we cannot directly do it because they do not have the same type. However, we can use the conversion rule with $\mathsf{list}\ (\mathsf{succ}\ 0 + 0) \equiv_{\beta\Sigma} \mathsf{list}\ (\mathsf{succ}\ 0)$. This conversion derives from the rewrite rule $x + 0 \hookrightarrow x$ instantiated with $x \coloneqq \mathsf{succ}\ 0$.

2.3 A Prelude Encoding for the $\lambda\Pi$-Calculus Modulo Theory

It is possible to introduce in $\lambda\Pi/\equiv$ the notions of proposition and proof [3]. In particular, this encoding—called prelude encoding—gives the possibility to quantify on certain propositions through codes, which is not possible inside the standard $\lambda\Pi/\equiv$. This encoding is defined by following signature.

Definition 1. *The signature Σ_{pre} contains the following constants and rewrite rules:*

$$
\begin{array}{ll}
Set : \texttt{TYPE} & o : Set \\
El : Set \to \texttt{TYPE} & Prf : El\ o \to \texttt{TYPE} \\
\rightsquigarrow_d : \Pi x : Set.\ (El\ x \to Set) \to Set & \Rightarrow_d : \Pi x : El\ o.\ (Prf\ x \to El\ o) \to El\ o \\
\pi : \Pi x : El\ o.\ (Prf\ x \to Set) \to Set & \forall : \Pi x : Set.\ (El\ x \to El\ o) \to El\ o \\
El\ (x \rightsquigarrow_d y) \hookrightarrow \Pi z : El\ x.\ El\ (y\ z) & Prf\ (x \Rightarrow_d y) \hookrightarrow \Pi z : Prf\ x.\ Prf\ (y\ z) \\
El\ (\pi\ x\ y) \hookrightarrow \Pi z : Prf\ x.\ El\ (y\ z) & Prf\ (\forall\ x\ y) \hookrightarrow \Pi z : El\ x.\ Prf\ (y\ z)
\end{array}
$$

We declare the constant *Set*, which represents the universe of types, along with the injection *El* that maps terms of type *Set* into TYPE. *o* is a term of type *Set* such that *El o* defines the universe of propositions. The injection *Prf* maps propositions into TYPE. $\rightsquigarrow_d$ (respectively $\Rightarrow_d$) is written infix and is used to represent dependent function types between terms of type *Set* (respectively *El o*). The symbol π (respectively $\forall$) is used to represent dependent function types between elements of type *El o* and *Set* (respectively *Set* and *El o*).

The main advantage of this encoding is that it allows us to quantify on propositions. Indeed, in $\lambda\Pi/\equiv$, we cannot quantify on TYPE. Instead, we can quantify on objects of type *El o*, and then inject them into TYPE using *Prf*.

2.4 Small Types and Small Derivations

As we work in $\lambda\Pi/\equiv$ rather than in an extension of Martin-Löf type theory, we do not have a pre-defined equality. Moreover, we cannot define an equality between types since such object would have type $\texttt{TYPE} \to \texttt{TYPE} \to \texttt{TYPE}$, which is not allowed in $\lambda\Pi/\equiv$.

If we want to compare types *Prf a* and *Prf b*, we cannot do it directly, but we can compare *a* and *b* (that are of type *El o*). We can proceed similarly to compare types *El a* and *El b* (with *a* and *b* of type *Set*). In that respect, we want types to be into a special form—called small type—that takes advantages of the prelude encoding, so as to compare them if necessary. To put types of the prelude encoding into this special form, we use the reverse of the rewrite rules of Σ_{pre} to represent dependent types with the symbols $\rightsquigarrow_d$, $\Rightarrow_d$, π and $\forall$ whenever it is possible. This is achieved by the partial function ν, defined by:

$$\nu(Set) = Set \qquad \nu(Prf\ a) = Prf\ a \qquad \nu(El\ a) = El\ a$$

$$
\nu(\Pi x : A.\ B) = \begin{cases}
Prf\ (a \Rightarrow_d (\lambda x : Prf\ a.\ b)) & \text{if } \nu(A) = Prf\ a \text{ and } \nu(B) = Prf\ b \\
El\ (a \rightsquigarrow_d (\lambda x : El\ a.\ b)) & \text{if } \nu(A) = El\ a \text{ and } \nu(B) = El\ b \\
Prf\ (\forall\ a\ (\lambda x : El\ a.\ b)) & \text{if } \nu(A) = El\ a \text{ and } \nu(B) = Prf\ b \\
El\ (\pi\ a\ (\lambda x : Prf\ a.\ b)) & \text{if } \nu(A) = Prf\ a \text{ and } \nu(B) = El\ b \\
\Pi x : \nu(A).\ \nu(B) & \text{otherwise}
\end{cases}
$$

Therefore, when $\nu(A)$ is defined, we have $A \equiv_{\beta\Sigma_{pre}} \nu(A)$. Note that ν is partial because we do not handle the case where a type is a β-reducible expression, as in practice we will not have types under λ-abstraction form.

To continue to characterize a particular form of types, we define the three following grammars:

$$\mathcal{S} ::= \mathit{Set} \mid \mathcal{S} \to \mathcal{S} \qquad \mathcal{P} ::= \mathit{Prf}\ a \mid \mathcal{P} \to \mathcal{S} \mid \Pi z : \mathcal{S}.\ \mathcal{P}$$

$$\mathcal{E} ::= \mathit{El}\ b \mid \mathcal{E} \to \mathcal{S} \mid \Pi z : \mathcal{S}.\ \mathcal{E}$$

with $a : \mathit{El}\ o$ and $b : \mathit{Set}$. The notation $A \in \mathcal{S}$ means that A is generated by the grammar $\mathcal{S}$. The grammar $\mathcal{S}$ generates types that only contain *Set*. Therefore, if $\nu(A) \in \mathcal{S}$ then $\nu(A) = A$. The grammars $\mathcal{P}$ and $\mathcal{E}$ generate types that contain a central symbol *Prf* or *El*.

Definition 2 (Small type, Small context). *A type A is small when $\nu(A)$ is defined and $\nu(A) \in \mathcal{S} \cup \mathcal{P} \cup \mathcal{E}$. In that case, $\nu(A)$ is called the small form of A. A context Γ is small when for every $x : A \in \Gamma$ we have that A is a small type.*

Example 2. $\mathit{Prf}\ a \to \mathit{Prf}\ b$, with $a, b : \mathit{El}\ o$, is a small type since its small form $\mathit{Prf}\ (a \Rightarrow_d (\lambda z.\ b))$ is generated by the grammar $\mathcal{P}$. The type $\Pi x : \mathit{Prf}\ b.\ \mathit{El}\ c$, with $c : \mathit{Set}$ depending on x, is a small type since its small form $\mathit{El}\ (\pi\ b\ (\lambda x : \mathit{Prf}\ b.\ c))$ is generated by the grammar $\mathcal{E}$. The type $\mathit{Prf}\ a \to \mathit{Set} \to \mathit{Prf}\ b$ is not small, since $\nu(\mathit{Prf}\ a \to \mathit{Set} \to \mathit{Prf}\ b) = \mathit{Prf}\ a \to \mathit{Set} \to \mathit{Prf}\ b \notin \mathcal{S} \cup \mathcal{P} \cup \mathcal{E}$.

We would ideally like all the types to be small, so that we can compare them if necessary. Therefore, if $\Gamma \vdash t : A$, we want A to be a small type, or t to be a small type and $A = \texttt{TYPE}$. However, small types are built using the constants of Σ_{pre}. In particular, the type of the constants o, $\rightsquigarrow_d$, $\Rightarrow_d$ and $\forall$ are small, but the types of π, *Prf* and *El* are not. Note that the type of an application of π, *Prf* or *El* is small. We thus come up with the following notion.

Definition 3 (Small judgment). *$\vdash \Gamma$ is a small judgment when Γ is a small context. $\Gamma \vdash t : A$ is a small judgment when Γ is a small context and when*

- *$t : A \in \Sigma_{pre}$,*
- *or t is the type of a constant of Σ_{pre},*
- *or A is a small type,*
- *or t is a small type.*

$(\Gamma_1 \vdash t_1 : A_1) \equiv (\Gamma_2 \vdash t_2 : A_2)$ is a small judgment when $\Gamma_1 \vdash t_1 : A_1$ and $\Gamma_2 \vdash t_2 : A_2$ are small.

Definition 4 (Small derivation). *A small derivation is a derivation in which all the judgments are small.*

2.5 Theories with Prelude Encoding

We define the theories we will consider in the rest of the paper: theories that features the prelude encoding inside $\lambda\Pi/\equiv$.

Definition 5 (Theory with prelude encoding). *We say that a theory $\mathcal{T} = \Sigma$ in the $\lambda\Pi/\equiv$ is a theory with prelude encoding when:*

- *there exists* $\Sigma_{\mathcal{T}}$ *such that* $\Sigma = \Sigma_{pre} \cup \Sigma_{\mathcal{T}}$ *and* $\Sigma_{pre} \cap \Sigma_{\mathcal{T}} = \emptyset$,
- *for every* $c : A \in \Sigma_{\mathcal{T}}$, A *is small and admits a small derivation* $\vdash A : \mathtt{TYPE}$,
- *for every* $\ell \hookrightarrow r \in \Sigma_{\mathcal{T}}$, *we have small derivations* $\boldsymbol{x} : \boldsymbol{B} \vdash \ell : A$ *and* $\boldsymbol{x} : \boldsymbol{B} \vdash r : A$ *with* A *a small type, where* $\boldsymbol{x}$ *represents the free variables of* ℓ.

A theory with prelude encoding is a theory with the constants and rewrite rules Σ_{pre}, and additional user-defined constants and rewrite rules. To ensure that $\Sigma_{\mathcal{T}}$ is encoded *inside* the prelude encoding, we can only define new constants whose types are small. We do not allow the use of rewrite rules $\ell \hookrightarrow r$ when ℓ has TYPE in its type. In particular, we cannot define new rewrite rules on *Prf* or *El* and change the behavior of these constants. It follows that the three grammars $\mathcal{S}$, $\mathcal{P}$ and $\mathcal{E}$ generate disjoint types.

In the following examples, we present three theories with prelude encoding in $\lambda\Pi/\equiv$. The examples of predicate logic and set theory illustrate that the restrictions considered are generally respected, even for expressive theories.

Example 3 (Predicate logic). Predicate logic can be encoded in a theory with prelude encoding. We declare constants for tautology and contradiction $\top, \bot : \mathit{El}\ o$, for negation $\neg : \mathit{El}\ o \to \mathit{El}\ o$, for conjunction and disjunction $\wedge, \vee : \mathit{El}\ o \to \mathit{El}\ o \to \mathit{El}\ o$, and for existential quantification $\exists : \Pi z : \mathit{Set}.\ (\mathit{El}\ z \to \mathit{El}\ o) \to \mathit{El}\ o$. The semantics of tautology is defined by the rewrite rule $\top \hookrightarrow \forall\ o\ (\lambda x : \mathit{El}\ o.\ x \Rightarrow x)$, which is equivalent to the more common form $\mathit{Prf}\ \top \hookrightarrow \Pi z : \mathit{El}\ o.\ \mathit{Prf}\ z \to \mathit{Prf}\ z$. The rewrite rule $\mathit{Prf}\ (A \wedge B) \hookrightarrow \Pi P : \mathit{El}\ o.\ (\mathit{Prf}\ A \to \mathit{Prf}\ B \to \mathit{Prf}\ P) \to \mathit{Prf}\ P$ can be encoded by $A \wedge B \hookrightarrow \forall\ o\ (\lambda P.\ (A \to B \to P) \to P)$. The rule $\mathit{Prf}\ (\neg A) \hookrightarrow \mathit{Prf}\ A \to \mathit{Prf}\ \bot$ is forbidden, but $\neg A \hookrightarrow A \Rightarrow \bot$ is allowed. We proceed similarly the other rewrite rules.

Example 4 (Natural numbers and lists). We can define our small theory of natural numbers and lists in the prelude encoding, by replacing TYPE by *Set* (in the universe of types) or $\mathit{El}\ o$ (in the universe of propositions), and by adding *El* and *Prf* at the necessary positions.

$$\mathsf{nat} : \mathit{Set} \qquad 0 : \mathit{El}\ \mathsf{nat} \qquad \mathsf{succ} : \mathit{El}\ \mathsf{nat} \to \mathit{El}\ \mathsf{nat} \qquad + : \mathit{El}\ \mathsf{nat} \to \mathit{El}\ \mathsf{nat} \to \mathit{El}\ \mathsf{nat}$$

$$\mathsf{list} : \mathit{El}\ \mathsf{nat} \to \mathit{Set} \qquad x + 0 \hookrightarrow x \qquad x + \mathsf{succ}\ y \hookrightarrow \mathsf{succ}\ (x + y)$$

$$\mathsf{nil} : \mathit{El}\ (\mathsf{list}\ 0) \qquad \mathsf{cons} : \Pi x : \mathit{El}\ \mathsf{nat}.\ \mathit{El}\ \mathsf{list}\ x \to \mathit{El}\ \mathsf{nat} \to \mathit{El}\ (\mathsf{list}\ (\mathsf{succ}\ x))$$

$$\mathsf{isRev} : \Pi x : \mathit{El}\ \mathsf{nat}.\ \mathit{El}\ (\mathsf{list}\ x) \to \mathit{El}\ (\mathsf{list}\ x) \to \mathit{El}\ o$$

$$\mathsf{concat} : \Pi x, y : \mathit{El}\ \mathsf{nat}.\ \mathit{El}\ (\mathsf{list}\ x) \to \mathit{El}\ (\mathsf{list}\ y) \to \mathit{El}\ (\mathsf{list}\ (x + y))$$

Example 5 (Set theory). The implementation in Dedukti of set theory [4] is a theory with prelude encoding. In this implementation, sets are represented by a more primitive notion of pointed graphs: we have graph and node of type *Set*. The predicate $\eta : \mathit{El}\ \mathsf{graph} \to \mathit{El}\ \mathsf{node} \to \mathit{El}\ \mathsf{node} \to \mathit{El}\ o$ is such that $\eta\ a\ x\ y$ is the proposition asserting that there is an edge in a from y to x. The operator $\mathsf{root} : \mathit{El}\ \mathsf{graph} \to \mathit{El}\ \mathsf{node}$ returns the root of a, which is a node.

In practice, the derivations of small judgments are small derivations. As we consider theories with prelude encoding, the only way of introducing a judgment that is not small is through λ-abstractions. For instance in Example 4 the judgment $\vdash El\ (\mathsf{list}\ ((\lambda x : El\ \mathsf{nat}.\ \lambda y : Set.\ x)\ 0\ \mathsf{nat})) : \mathtt{TYPE}$ is small, but in its derivation we have $\vdash \lambda x : El\ \mathsf{nat}.\ \lambda y : Set.\ x : El\ \mathsf{nat} \to Set \to El\ \mathsf{nat}$ which is not a small judgment. However, $\vdash El\ (\mathsf{list}\ 0) : \mathtt{TYPE}$ admits a small derivation. If the derivation is not small, we can in practice apply β-reduction on the fragments of the derivation that are not small to obtain a small derivation.

3 Equalities

Since we want to replace rewrite rules $\ell \hookrightarrow r$ by equational axioms $\ell = r$, we need to define an equality in the target theory. In this section, we present a heterogeneous equality and a method to compare small types. The heterogeneous equality is necessary to compare objects that do not have the same type. Although we cannot define an equality between types in $\lambda\Pi/\equiv$, it is possible to develop an equality between small types, taking advantage of their structure.

3.1 Heterogeneous Equality

In our development, we need to have an equality between two translations of the same term. However, the two translations do not necessarily have the same type, as we may have introduced transports over the course of the translation. To that end, we define a heterogeneous equality inspired by the one of McBride [18]. Our heterogeneous equality is defined by the constant schemas $\mathsf{heq}_{A,B} : A \to B \to El\ o$ where A and B are of type $\mathtt{TYPE}$. We write $u\ {}_A{\approx}_B\ v$ for $Prf\ (\mathsf{heq}_{A,B}\ u\ v)$. Heterogeneous equality is reflexive, symmetric, and transitive.

$$\mathsf{refl}_A : \Pi u : A.\ u\ {}_A{\approx}_A\ u$$
$$\mathsf{sym}_{A,B} : \Pi u : A.\ \Pi v : B.\ u\ {}_A{\approx}_B\ v \to v\ {}_B{\approx}_A\ u$$
$$\mathsf{trans}_{A,B,C} : \Pi u : A.\ \Pi v : B.\ \Pi w : C.\ u\ {}_A{\approx}_B\ v \to v\ {}_B{\approx}_C\ w \to u\ {}_A{\approx}_C\ w$$

When two objects have the same type, heterogeneous equality acts as Leibniz equality. In particular, we can replace u by v in the universes of propositions and types. The result of a Leibniz substitution on t remains equal to t.

$$\mathsf{leib}^{\mathsf{Prf}}_A : \Pi u, v : A.\ \Pi p : u\ {}_A{\approx}_A\ v.\ \Pi P : A \to El\ o.\ Prf\ (P\ u) \to Prf\ (P\ v)$$
$$\mathsf{eqLeib}^{\mathsf{Prf}}_A : \Pi u, v : A.\ \Pi p : u\ {}_A{\approx}_A\ v.\ \Pi P : A \to El\ o.\ \Pi t : Prf\ (P\ u).$$
$$\mathsf{leib}^{\mathsf{Prf}}_A\ u\ v\ p\ P\ t\ {}_{Prf\ (P\ v)}{\approx}_{Prf\ (P\ u)}\ t$$

The same axiom schemas exist for the universe of types, with superscript El instead of Prf, El instead of Prf, and Set instead of $El\ o$.

Finally, we add axioms for the congruence of each constructor of $\lambda\Pi/\equiv$.

Application constructor. For the application, we take:

$$\begin{aligned}\mathsf{app}_{A_1,A_2,B_1,B_2} : \Pi t_1 : (\Pi x : A_1.\ B_1).\ \Pi t_2 : (\Pi x : A_2.\ B_2).\\ \Pi u_1 : A_1.\ \Pi u_2 : A_2.\ t_1 \approx t_2 \to u_1 \approx u_2\\ \to t_1\ u_1\ {}_{B_1[x\mapsto u_1]}{\approx}_{B_2[x\mapsto u_2]}\ t_2\ u_2\end{aligned}$$

For the λ-abstraction and Π-type constructors, we cannot directly build equality axioms. Indeed, if we want to define an equality between functional terms t_1 of type $\Pi x : A_1.\ B_1$ and t_2 of type $\Pi x : A_2.\ B_2$, we need to ensure that types A_1 and A_2 are equal. Therefore, we would like to have

$$\begin{aligned}\mathsf{fun}_{A_1,A_2,B_1,B_2} : \Pi t_1 : (\Pi x : A_1.\ B_1).\ \Pi t_2 : (\Pi y : A_2.\ B_2).\ A_1 \approx A_2\\ \to (\Pi x : A_1.\ \Pi y : A_2.\ x \approx y \to t_1\ x \approx t_2\ y)\\ \to t_1 \approx t_2\end{aligned}$$

but we cannot take such an axiom, since the heterogeneous equality is not defined to compare objects that have type $\mathtt{TYPE}$, and $A_1 \approx A_2$ is therefore ill typed. This shortcoming is addressed by developing an equality between small types.

3.2 Equality between Small Types

We cannot build an equality between types, since such an equality would have type $\mathtt{TYPE} \to \mathtt{TYPE} \to \mathtt{TYPE}$, which is impossible in $\lambda\Pi/{\equiv}$. An option would be to take axiom schemas $A \approx B$ for every equality between types A and B. Such an equality would be too far from standard and would require additional axioms to build transports. An alternative is to define an equality between small types. By construction, if $\nu(A) \in \mathcal{P}$, then $\nu(A)$ is generated from $Prf\ a$ for some $a : El\ o$, and if $\nu(A) \in \mathcal{E}$, then $\nu(A)$ is generated from $El\ a$ for some $a : Set$. If the small form of A contains $Prf\ a$ and the small form of B contains $Prf\ b$, then we want an equality between a and b. We define the partial function κ on small forms by

$$\kappa(Prf\ a_1, Prf\ a_2) = a_1 \approx a_2 \qquad \kappa(El\ a_1, El\ a_2) = a_1 \approx a_2$$

$$\kappa(S, S) = \mathsf{True} \text{ if } S \in \mathcal{S} \qquad \kappa(T_1 \to S, T_2 \to S) = \kappa(T_1, T_2) \text{ if } S \in \mathcal{S}$$

$$\kappa(\Pi z : S.\ T_1, \Pi z : S.\ T_2) = \Pi z : S.\ \kappa(T_1, T_2) \text{ if } S \in \mathcal{S}$$

where $\mathsf{True} \coloneqq \Pi P : El\ o.\ Prf\ P \to Prf\ P$, so we can always give a witness of $\kappa(S, S)$ if $S \in \mathcal{S}$. By convention, we simply write $\kappa(A, B)$ for the result of $\kappa(\nu(A), \nu(B))$.

Example 6. $\kappa(\Pi x : Set.\ Prf\ P \to Prf\ Q, \Pi x : Set.\ Prf\ R) = \Pi x : Set.\ (P \Rightarrow_d \lambda z : P.\ Q) \approx R$ since $\nu(\Pi x : Set.\ Prf\ P \to Prf\ Q) = \Pi x : Set.\ Prf\ (P \Rightarrow_d (\lambda z : P.\ Q))$.

We can now go back to the definition of equality axioms for the constructors of $\lambda\Pi/{\equiv}$.

Function constructor. If A_1 and A_2 are small types, we can take $\kappa(A_1, A_2)$. We do not compare objects of type `TYPE` anymore, but objects that have either type *El o* or type *Set*. The axiom schema for the function constructor is thus:

$$\begin{aligned}\mathsf{fun}_{A_1,A_2,B_1,B_2} : \Pi t_1 : (\Pi x : A_1.\ B_1).\ \Pi t_2 : (\Pi y : A_2.\ B_2).\ \kappa(A_1, A_2)\\ \to (\Pi x : A_1.\ \Pi y : A_2.\ x \approx y \to t_1\ x \approx t_2\ y)\\ \to t_1 \approx t_2\end{aligned}$$

This axiom schema is a generalization of the *functional extensionality* principle with distinct domains A_1 and A_2 in the case of heterogeneous equality. Functional extensionality states that two pointwise-equal functions are equal. If the domains A_1 and A_2 are generated by $\mathcal{S}$, then they are syntactically equal and we can derive a simpler axiom schema:

$$\begin{aligned}\mathsf{fun}_{A,B_1,B_2} : \Pi t_1 : (\Pi x : A.\ B_1).\ \Pi t_2 : (\Pi x : A.\ B_2).\ (\Pi x : A.\ t_1\ x \approx t_2\ x)\\ \to t_1 \approx t_2\end{aligned}$$

Π-type constructor. The congruence axiom for dependent types aims at building $\kappa(\Pi x : A_1.\ B_1, \Pi x : A_2\ B_2)$. There are different cases depending on the grammars generating $\nu(A_1)$, $\nu(A_2)$, $\nu(B_1)$ and $\nu(B_2)$. If $\nu(A_1)$, $\nu(A_2)$, $\nu(B_1)$, $\nu(B_2) \in \mathcal{S}$, then $\Pi x : A_1.\ B_1$ and $\Pi x : A_2.\ B_2$ are syntactically equal and we can build an object of type True. If $\nu(A_1)$, $\nu(A_2) \in \mathcal{S}$ and $\nu(B_1)$, $\nu(B_2) \in \mathcal{P} \cup \mathcal{E}$, then $A_1 = A_2$ and $\kappa(\Pi x : A_1.\ B_1, \Pi x : A_2\ B_2) = \Pi x : A_1.\ \kappa(B_1, B_2)$. If $\nu(A_1)$, $\nu(A_2) \in \mathcal{P} \cup \mathcal{E}$ and $\nu(B_1)$, $\nu(B_2) \in \mathcal{S}$, then $B_1 = B_2$ and $\kappa(\Pi x : A_1.\ B_1, \Pi x : A_2\ B_2) = \kappa(A_1, A_2)$. If $\nu(A_1)$, $\nu(A_2)$, $\nu(B_1)$, $\nu(B_2) \in \mathcal{P} \cup \mathcal{E}$, then there are four cases, corresponding to $\rightsquigarrow_d$, $\Rightarrow_d$, π and $\forall$. For instance, if $\nu(A_1)$, $\nu(A_2)$, $\nu(B_1)$ and $\nu(B_2)$ are all generated by $\mathcal{E}$, then necessarily we have $\nu(A_1) = El\ a_1$, $\nu(A_2) = El\ a_2$, $\nu(B_1) = El\ b_1$ and $\nu(B_2) = El\ b_2$. Therefore $\kappa(\Pi x : A_1.\ B_1, \Pi x : A_2.\ B_2) \coloneqq (a_1 \rightsquigarrow_d (\lambda x : El\ a_1.\ b_1)) \approx (a_2 \rightsquigarrow_d (\lambda y : El\ a_2.\ b_2))$. The axiom is:

$$\begin{aligned}\mathsf{prod}_{\rightsquigarrow_d} : \Pi a_1, a_2 : Set.\ \Pi b_1 : (El\ a_1 \to Set).\ \Pi b_2 : (El\ a_2 \to Set).\ a_1 \approx a_2\\ \to (\Pi x : El\ a_1.\ \Pi y : El\ a_2.\ x \approx y \to b_1\ x \approx b_2\ y)\\ \to (a_1 \rightsquigarrow_d b_1) \approx (a_2 \rightsquigarrow_d b_2)\end{aligned}$$

Note that this axiom is derivable from the previous axioms. We proceed similarly for the cases $\Rightarrow_d$, π and $\forall$.

We write Σ_{eq} for the signature formed by the axiom schemas defining the heterogeneous equality. Reflexivity, symmetry, and transitivity are standard axioms of equality. We have also added axioms stating that a heterogeneous equality comparing two objects of the same type acts like Leibniz equality. Finally, we have an axiom for the application constructor and one axiom for the abstraction constructor—that is functional extensionality. Both axioms are used by Oury [19], who also assumes the uniqueness of identity proofs principle that entails the Leibniz principle we use.

4 Replacing Rewrite Rules

When working in theories with prelude encoding, rewriting originates from the rewrite rules of Σ_{pre} (which are generic rewrite rules), from the rewrite rules $\Sigma_{\mathcal{T}}$ (which are defined by the user) and from β-reduction. The goal of this work is to replace the user-defined rewrite rules $\Sigma_{\mathcal{T}}$ by equational axioms. In the rest of the paper, we write $\vdash_{\mathcal{R}}$ for a derivation inside the source theory—the theory with user-defined rewrite rules—and $\vdash$ for a derivation inside the target theory—the theory with axioms instead of user-defined rewrite rules.

We now have all the tools to replace rewrite rules by equational axioms. To do so, we build suitable transports, such that if $\Gamma \vdash t : A$ and $\Gamma \vdash p : \kappa(A, B)$, then $\Gamma \vdash \mathsf{transp}\ p\ t : B$. The goal is to insert such transports into the terms instead of using conversion with the rules of $\Sigma_{\mathcal{T}}$. In the signature, each rewrite rule $\ell \hookrightarrow r$ is replaced by the equational axiom $\bar{\ell} \approx \bar{r}$.

4.1 Transports

If we have $\Gamma \vdash t : A$ and $\Gamma \vdash p : \kappa(A, B)$, we want to transport t from A to B, that is to build a term $\mathsf{transp}\ p\ t$ such that $\Gamma \vdash \mathsf{transp}\ p\ t : B$. A paramount result is that t and $\mathsf{transp}\ p\ t$ are heterogeneously equal.

Lemma 2 (Transport). *Given $\Gamma \vdash t : A$ and $\Gamma \vdash p : \kappa(A, B)$ with A and B small types, there exists* $\mathsf{transp}\ p\ t$*, called transport of t along p, such that:*

- $\Gamma \vdash \mathsf{transp}\ p\ t : B$,
- *there exists* $\mathsf{eqTransp}$ *such that* $\Gamma \vdash \mathsf{eqTransp}\ p\ t : \mathsf{transp}\ p\ t\ {}_B{\approx}_A\ t$.

Proof. A and B are small types and we have an equality $\kappa(A, B)$. If $A, B \in \mathcal{S}$ then $\nu(A) = \nu(B) = A = B$ and we take $\mathsf{transp}\ p\ t \coloneqq t$ and $\mathsf{eqTransp}\ p\ t \coloneqq \mathsf{refl}_A\ t$. Otherwise, by construction of κ, we know that $\nu(A), \nu(B) \in \mathcal{P}$, or $\nu(A), \nu(B) \in \mathcal{E}$, and that $\nu(A)$ and $\nu(B)$ have the same structure. Moreover, using $A \equiv_{\beta\Sigma_{pre}} \nu(A)$, we have $\Gamma \vdash t : \nu(A)$. We proceed by induction on the grammar $\mathcal{P}$ (we proceed similarly for the grammar $\mathcal{E}$).

- If $\nu(A) = Prf\ a$ and $\nu(B) = Prf\ b$, then we have $\Gamma \vdash p : a \approx b$. We take $\mathsf{transp}\ p\ t \coloneqq \mathsf{leib}^{\mathsf{Prf}}_{El\ o}\ a\ b\ p\ (\lambda w : El\ o.\ w)\ t$. We conclude using $\mathsf{eqLeib}^{\mathsf{Prf}}_{El\ o}$.
- If $\nu(A) = A' \to S$ and $\nu(B) = B' \to S$, with $A', B' \in \mathcal{P}$ and $S \in \mathcal{S}$, then we have $\kappa(A', B') = \kappa(A, B)$. From $\Gamma \vdash p : \kappa(A', B')$ we can build some p' such that $\Gamma \vdash p' : \kappa(B', A')$ (using sym). By weakening, we also have $p' : \kappa(B', A')$ in the context $\Gamma, m_b : B'$. By induction, we have $\mathsf{transp}\ p'\ m_b : A'$ and $\mathsf{eqTransp}\ p'\ m_b : \mathsf{transp}\ p'\ m_b \approx m_b$ in the context $\Gamma, m_b : B'$. We take $\mathsf{transp}\ p\ t \coloneqq \lambda m_b : B'.\ t\ (\mathsf{transp}\ p'\ m_b)$. Using trans and app we obtain an equality $t\ (\mathsf{transp}\ p'\ m_b) \approx t\ m_a$ in the context $\Gamma, m_a : A', m_b : B', p_m : m_a \approx m_b$. Using fun and $\equiv_{\beta\Sigma_{pre}}$, we have $\lambda m_b : B'.\ t\ (\mathsf{transp}\ p'\ m_b) \approx t$ in the context Γ.

– If $\nu(A) = \Pi z : S.\ A'$ and $\nu(B) = \Pi z : S.\ B'$ with $A', B' \in \mathcal{P}$ and $S \in \mathcal{S}$, then we have $\kappa(A, B) = \Pi z : S.\ \kappa(A', B')$. By weakening and application, we have $\Gamma, z : S \vdash p\ z : \kappa(A', B')$. By induction we have $\mathsf{transp}\ (p\ z)\ (t\ z) : B'$ and $\mathsf{eqTransp}\ (p\ z)\ (t\ z) : \mathsf{transp}\ (p\ z)\ (t\ z) \approx t\ z$ in the context $\Gamma, z : S$. We take $\mathsf{transp}\ p\ t \coloneqq \lambda z : S.\ \mathsf{transp}\ (p\ z)\ (t\ z)$. We obtain $\lambda z : S.\ \mathsf{transp}\ (p\ z)\ (t\ z) \approx t$ using fun and $\equiv_{\beta\Sigma_{pre}}$. □

The transport of t from A to B depends on the small form of A and B. In that respect, there exists a different transport for each possible family of small form, and such transport is indexed over an equality of a small type.

4.2 Translation of Terms

To translate a theory with rewrite rules into a theory with equational axioms, we add transports at the proper locations in the terms and types. If we have $\Gamma \vdash_{\mathcal{R}} t : A$ in the source theory, we want to find $\overline{\Gamma}$, $\bar{t}$ and $\overline{A}$ that are translations of Γ, t and A, and such that $\overline{\Gamma} \vdash \bar{t} : \overline{A}$ in the target theory.

We add transports in a term by induction on a typing derivation—which is not unique—so we may have different translations for a same term. As such, we define a relation $\lhd$ where $\bar{t} \lhd t$ states that $\bar{t}$ is a translation of t. The relation is defined by induction on the terms of $\lambda\Pi/\equiv$. Variables, constants, `TYPE` and `KIND` are translations of themselves. The translations of λ-abstractions $\lambda x : A.\ t$, dependent types $\Pi x : A.\ B$ and applications $t\ u$ rely on the translations of t, u, A and B. The most important part of the definition is that the translation is stable by transports: if $\bar{t}$ is a translation of t, then $\mathsf{transp}\ p\ \bar{t}$ is also a translation of t, with p typically an equality. This relation captures all possible translations, but some are not correct as they may not be well typed. For instance, $\lambda x : \overline{A}.\ \bar{t}$ is not a valid translation of $\lambda x : A.\ t$ when the variable x used in $\bar{t}$ does not expect type $\overline{A}$ but another translation $\overline{A}'$.

Definition 6. *The translation relation $\lhd$ is defined by:*

$$\frac{}{x \lhd x} \qquad \frac{}{c \lhd c} \qquad \frac{}{\texttt{TYPE} \lhd \texttt{TYPE}} \qquad \frac{}{\texttt{KIND} \lhd \texttt{KIND}}$$

$$\frac{\overline{A} \lhd A \qquad \bar{t} \lhd t}{(\lambda x : \overline{A}.\ \bar{t}) \lhd (\lambda x : A.\ t)} \qquad \frac{\overline{A} \lhd A \qquad \overline{B} \lhd B}{(\Pi x : \overline{A}.\ \overline{B}) \lhd (\Pi x : A.\ B)}$$

$$\frac{\bar{t} \lhd t \qquad \bar{u} \lhd u}{(\bar{t}\ \bar{u}) \lhd (t\ u)} \qquad \frac{\bar{t} \lhd t}{(\mathsf{transp}\ p\ \bar{t}) \lhd t}$$

where p is an arbitrary term.

Due to the typing rules of $\lambda\Pi/\equiv$, transports for objects that have `TYPE` in their type do not exist. Therefore, the only well-typed translations of `TYPE`, `KIND`, *Set*, *Prf* and *El* are themselves, and the well-typed translations of $\Pi x : A.\ B$ are of the form $\Pi x : \overline{A}.\ \overline{B}$ with $\overline{A} \lhd A$ and $\overline{B} \lhd B$. It follows that a well-typed

translation of a small type is still a small type. In particular, if $A \in \mathcal{S}$ then for any $\overline{A}$ we have $\overline{A} \coloneqq A$; if $\nu(A) \in \mathcal{P}$ then $\nu(\overline{A}) \in \mathcal{P}$; and if $\nu(A) \in \mathcal{E}$ then $\nu(\overline{A}) \in \mathcal{E}$.

We extend the relation to contexts and signatures. For each rewrite rule $\ell \hookrightarrow r$ of a signature, we have $\boldsymbol{x} : \boldsymbol{B} \vdash_{\mathcal{R}} \ell : A$ and $\boldsymbol{x} : \boldsymbol{B} \vdash_{\mathcal{R}} r : A$, for some $\boldsymbol{B}$ and A, and some $\boldsymbol{x}$ representing the free variables of ℓ. The translation of the rewrite rule $\ell \hookrightarrow r$ is given by the equational axiom $\mathsf{eq}_{\ell r} : \Pi \boldsymbol{x} : \overline{\boldsymbol{B}}.\ \bar{\ell}\ {}_{\overline{A}}{\approx}_{\overline{A}}\ \bar{r}$. Since the type of a term is not unique in $\lambda\Pi/{\equiv}$, we have made a choice of $\boldsymbol{B}$ and A, which is not a problem as we will see in the proof of Theorem 1.

Definition 7. *$\triangleleft$ is defined on contexts and signatures by:*

$$\frac{}{\langle\rangle \ \triangleleft\ \langle\rangle} \qquad \frac{\overline{\Gamma} \ \triangleleft\ \Gamma \qquad \overline{A} \ \triangleleft\ A}{(\overline{\Gamma}, x : \overline{A}) \ \triangleleft\ (\Gamma, x : A)} \qquad \frac{\overline{\Sigma} \ \triangleleft\ \Sigma \qquad \overline{A} \ \triangleleft\ A}{(\overline{\Sigma}, c : \overline{A}) \ \triangleleft\ (\Sigma, c : A)}$$

$$\frac{\overline{\Sigma} \ \triangleleft\ \Sigma \qquad \bar{\ell} \ \triangleleft\ \ell \qquad \bar{r} \ \triangleleft\ r \qquad \overline{\boldsymbol{B}} \ \triangleleft\ \boldsymbol{B} \qquad \overline{A} \ \triangleleft\ A}{(\overline{\Sigma}, \mathsf{eq}_{\ell r} : \Pi \boldsymbol{x} : \overline{\boldsymbol{B}}.\ \bar{\ell}\ {}_{\overline{A}}{\approx}_{\overline{A}}\ \bar{r}) \ \triangleleft\ (\Sigma, \ell \hookrightarrow r)}$$

Lemma 3. *If $\bar{t} \ \triangleleft\ t$ and $\bar{u} \ \triangleleft\ u$ then $\bar{t}[x \mapsto \bar{u}] \ \triangleleft\ t[x \mapsto u]$.*

Proof. By induction on the derivation of $\bar{t} \ \triangleleft\ t$. For the case with the transport, we can prove that $(\mathsf{transp}\ p\ t)[x \mapsto u] = \mathsf{transp}\ p[x \mapsto u]\ t[x \mapsto u]$. □

Definition 8 (Relation $\sim$). *We say that $t_1 \sim t_2$ when there exists some t such that $t_1 \ \triangleleft\ t$ and $t_2 \ \triangleleft\ t$.*

Lemma 4. *$\sim$ is an equivalence relation.*

Proof. $\sim$ is reflexive, symmetric and transitive. When proving transitivity we exploit the fact that whenever $t \ \triangleleft\ u_1$ and $t \ \triangleleft\ u_2$, we have $u_1 = u_2$. Reflexivity is proved by induction on the term. □

An important result we need to prove is that two well-typed translations t_1 and t_2 of the same term t are heterogeneously equal. By construction, both terms do not necessarily have the same type or the same context. We will always consider $\Gamma_1 \vdash t_1 : A_1$ and $\Gamma_2 \vdash t_2 : A_2$, where Γ_1 and Γ_2 have the same length and the same variables (with possibly different types). The equality between t_1 and t_2 must be typed in some context, but Γ_1 and Γ_2 are not sufficient. That is why we define a common context $\Gamma_1 \star \Gamma_2$ (written $\mathsf{Pack}\ \Gamma_1\ \Gamma_2$ in the work of Winterhalter *et al.* [23]) by duplicating each variable and by assuming a witness of heterogeneous equality between these two duplicates. More precisely, we partially define $\star$ by induction on small contexts:

$$\langle\rangle \star \langle\rangle \coloneqq \langle\rangle$$

$$(\Gamma_1, x : A_1) \star (\Gamma_2, x : A_2) \coloneqq \Gamma_1 \star \Gamma_2, x_1 : A_1[\gamma_1], x_2 : A_2[\gamma_2], p_x : x_1 \approx x_2$$

where γ_1 substitutes variables z by z_1 and γ_2 substitutes variables z by z_2. We write γ_{12} for the substitution that replaces the variables z_1 and z_2 by z and the variable p_z by $\mathsf{refl}\ z$.

Lemma 5. *If $\Gamma \star \Gamma \vdash t : A$, then we can derive $\Gamma \vdash t[\gamma_{12}] : A[\gamma_{12}]$.*

Proof. We proceed by induction on the length of Γ. If we have $\langle\rangle \star \langle\rangle \vdash t : A$ then by definition we have $\langle\rangle \vdash t : A$. Suppose that we have $(\Gamma, x : B) \star (\Gamma, x : B) \vdash t : A$. We apply successively Lemma 1 to replace x_2 and x_1 by x and then p_x by $\mathsf{refl}\ x$. □

The following lemma states that two translations of a same term are heterogeneously equal.

Lemma 6 (Equal translations). *Let $t_1 \sim t_2$ such that $\Gamma_1 \vdash t_1 : A_1$ and $\Gamma_2 \vdash t_2 : A_2$ with Γ_1 and Γ_2 small contexts.*

1. *If $\Gamma_1 \vdash A_1 : \mathtt{TYPE}$ and $\Gamma_2 \vdash A_2 : \mathtt{TYPE}$, then there exists some p such that $\Gamma_1 \star \Gamma_2 \vdash p : t_1[\gamma_1]\ {}_{A_1[\gamma_1]}{\approx}_{A_2[\gamma_2]}\ t_2[\gamma_2]$.*
2. *If t_1 and t_2 are small types, then there exists some p such that $\Gamma_1 \star \Gamma_2 \vdash p : \kappa(t_1[\gamma_1], t_2[\gamma_2])$.*

Proof. We proceed by induction on the derivation of $t_1 \sim t_2$. We show two interesting cases.

- Transport $(\mathsf{transp}\ p\ t_1) \sim t_2$
  We have $\Gamma_1 \vdash \mathsf{transp}\ p\ t_1 : A_1$ and $\Gamma_2 \vdash t_2 : A_2$. By inversion of typing, we have $\Gamma_1 \vdash t_1 : A_1'$ and $\Gamma_1 \vdash p : \kappa(A_1', A_1)$. By induction there exists some p_t such that $\Gamma_1 \star \Gamma_2 \vdash p_t : t_1[\gamma_1] \approx t_2[\gamma_2]$. We also have $\Gamma_1 \vdash \mathsf{eqTransp}\ p\ t_1 : \mathsf{transp}\ p\ t_1 \approx t_1$. We derive that $\Gamma_1 \star \Gamma_2 \vdash (\mathsf{eqTransp}\ p\ t_1)[\gamma_1] : (\mathsf{transp}\ p\ t_1)[\gamma_1] \approx t_1[\gamma_1]$. We conclude using transitivity.

- Application $(t_1\ u_1) \sim (t_2\ u_2)$
  Suppose that $t_1\ u_1$ and $t_2\ u_2$ are small types. Then the only possible cases are $t_1 = t_2 = \mathit{Prf}$ or $t_1 = t_2 = \mathit{El}$. If $t_1 = t_2 = \mathit{Prf}$, then we have $\Gamma_1 \vdash \mathit{Prf}\ u_1 : \mathtt{TYPE}$ and $\Gamma_2 \vdash \mathit{Prf}\ u_2 : \mathtt{TYPE}$. Since $\kappa(\mathit{Prf}\ u_1, \mathit{Prf}\ u_2) = u_1 \approx u_2$, the result is simply the induction hypothesis $\Gamma_1 \star \Gamma_2 \vdash p : u_1[\gamma_1] \approx u_2[\gamma_2]$. We proceed similarly for $\mathit{El}\ u_1 \sim \mathit{El}\ u_2$.
  Suppose that we have $\Gamma_1 \vdash t_1\ u_1 : T_1$ and $\Gamma_2 \vdash t_2\ u_2 : T_2$ with $\Gamma \vdash T_1 : \mathtt{TYPE}$ and $\Gamma \vdash T_2 : \mathtt{TYPE}$. Then by inversion of typing we have $\Gamma_1 \vdash u_1 : B_1$ and $\Gamma_2 \vdash u_2 : B_2$ and $\Gamma_1 \vdash t_1 : \Pi x : A_1.\ B_1$ and $\Gamma_2 \vdash t_2 : \Pi x : A_2.\ B_2$, with $T_1 \equiv_{\beta\Sigma_{pre}} B_1[x \mapsto u_1]$ and $T_2 \equiv_{\beta\Sigma_{pre}} B_2[x \mapsto u_2]$. By induction hypotheses, we have $\Gamma_1 \star \Gamma_2 \vdash p_t : t_1[\gamma_1] \approx t_2[\gamma_2]$ and $\Gamma_1 \star \Gamma_2 \vdash p_u : u_1[\gamma_1] \approx u_2[\gamma_2]$. We conclude using app. □

4.3 Translation of Judgments

In Section 4.2 we have seen all the possible translations for *terms*. However, the only translations that matter are the translations of *judgments*: context formation judgments and typing judgments.

Definition 9. *For any $\vdash_{\mathcal{R}} \Gamma$ we define a set $[\![\vdash_{\mathcal{R}} \Gamma]\!]$ of valid judgments such that $\vdash \overline{\Gamma} \in [\![\vdash_{\mathcal{R}} \Gamma]\!]$ if and only if $\overline{\Gamma} \lhd \Gamma$. For any $\Gamma \vdash_{\mathcal{R}} t : A$ we define a set $[\![\Gamma \vdash_{\mathcal{R}} t : A]\!]$ of valid judgments such that $\overline{\Gamma} \vdash \bar{t} : \overline{A} \in [\![\Gamma \vdash_{\mathcal{R}} t : A]\!]$ if and only if $\vdash \overline{\Gamma} \in [\![\vdash_{\mathcal{R}} \Gamma]\!]$, $\bar{t} \lhd t$ and $\overline{A} \lhd A$.*

We are now able to prove that it is possible to switch between two translations of a small type.

Lemma 7 (Switching translations). *Suppose that we have A a small type, $\overline{\Gamma} \vdash \bar{t} : \overline{A} \in [\![\Gamma \vdash_{\mathcal{R}} t : A]\!]$ and $\overline{\Gamma} \vdash \overline{A}' : \mathtt{TYPE} \in [\![\Gamma \vdash_{\mathcal{R}} A : \mathtt{TYPE}]\!]$ with $\overline{\Gamma}$ a small context. Then there exists $\bar{t}'$ such that $\overline{\Gamma} \vdash \bar{t}' : \overline{A}' \in [\![\Gamma \vdash_{\mathcal{R}} t : A]\!]$.*

Proof. If $\nu(A) \in \mathcal{S}$, then $\overline{A} \coloneqq A$ and $\overline{A}' \coloneqq A$, and we take $\bar{t}' \coloneqq \bar{t}$. If $\nu(A) \in \mathcal{P}$, then $\nu(\overline{A}), \nu(\overline{A}') \in \mathcal{P}$ (this is similar for $\mathcal{E}$). As $\overline{A}$ and $\overline{A}'$ are two translations of A, we have $\overline{A} \sim \overline{A}'$. From Lemma 6, we have $\overline{\Gamma} \star \overline{\Gamma} \vdash p : \kappa(\overline{A}[\gamma_1], \overline{A}'[\gamma_2])$. Using Lemma 5 we obtain $\overline{\Gamma} \vdash p[\gamma_{12}] : \kappa(\overline{A}, \overline{A}')$. Using Lemma 2, there exists some $\mathsf{transp}\ p[\gamma_{12}]\ \bar{t} \lhd t$ (since $\bar{t} \lhd t$) such that $\overline{\Gamma} \vdash \mathsf{transp}\ p[\gamma_{12}]\ \bar{t} : \overline{A}'$. □

4.4 Translation of Theories

Now that we have translated terms and judgments, we want to translate theories, so that the translation of every provable judgment in the source theory is provable in the target theory. The target theory $\mathcal{T}^{ax} = \Sigma_{pre} \cup \Sigma_{eq} \cup \overline{\Sigma}_{\mathcal{T}}$ is obtained by adding the axioms of equality to the signature, and by translating $\Sigma_{\mathcal{T}}$. To do so, we translate each typed constant and rewrite rule one by one. At the end, the rewrite rules of $\Sigma_{\mathcal{T}}$ have been replaced by equational axioms.

The paramount result of this paper is the following theorem. The first item concerns context formation. The second item is about the translation of typing judgments. The third item focuses on convertible contexts. The fourth and fifth items are about the conversion rules. It is worth noting that in the second item we use the universal quantifier on $\overline{\Gamma}$ instead of using the existential quantifier. We have opted for the universal quantifier so we can obtain the induction hypotheses for a common context.

Theorem 1 (Elimination of the rewrite rules). *Let a theory $\mathcal{T} = \Sigma$ in $\lambda\Pi/\equiv$ such that $\mathcal{T}$ is a theory with prelude encoding and such that all the derivations considered are small derivations. There exists a signature $\overline{\Sigma}_{\mathcal{T}} \lhd \Sigma_{\mathcal{T}}$ such that the theory $\mathcal{T}^{ax} = \Sigma_{pre} \cup \Sigma_{eq} \cup \overline{\Sigma}_{\mathcal{T}}$ satisfies:*

1. *If $\vdash_{\mathcal{R}} \Gamma$, then there exists $\vdash \overline{\Gamma} \in [\![\vdash_{\mathcal{R}} \Gamma]\!]$.*
2. *If $\Gamma \vdash_{\mathcal{R}} t : A$, then for every $\vdash \overline{\Gamma} \in [\![\vdash_{\mathcal{R}} \Gamma]\!]$ there exist $\bar{t}$ and $\overline{A}$ such that $\overline{\Gamma} \vdash \bar{t} : \overline{A} \in [\![\Gamma \vdash_{\mathcal{R}} t : A]\!]$.*
3. *If $(\vdash_{\mathcal{R}} \Gamma_1) \equiv (\vdash_{\mathcal{R}} \Gamma_2)$, then for every $\vdash \overline{\Gamma}_1 \in [\![\vdash_{\mathcal{R}} \Gamma_1]\!]$ and $\vdash \overline{\Gamma}_2 \in [\![\vdash_{\mathcal{R}} \Gamma_2]\!]$, we have $\vdash \overline{\Gamma}_1 \star \overline{\Gamma}_2$.*
4. *If $(\Gamma_1 \vdash_{\mathcal{R}} u_1 : A_1) \equiv (\Gamma_2 \vdash_{\mathcal{R}} u_2 : A_2)$ with $\Gamma_1 \vdash_{\mathcal{R}} A_1 : \mathtt{TYPE}$ and $\Gamma_2 \vdash_{\mathcal{R}} A_2 : \mathtt{TYPE}$, then for every $\vdash \overline{\Gamma}_1 \in [\![\vdash_{\mathcal{R}} \Gamma_1]\!]$ and $\vdash \overline{\Gamma}_2 \in [\![\vdash_{\mathcal{R}} \Gamma_2]\!]$, we have $\overline{\Gamma}_1 \vdash \bar{u}_1 : \overline{A}_1 \in [\![\Gamma_1 \vdash_{\mathcal{R}} u_1 : A_1]\!]$ and $\overline{\Gamma}_2 \vdash \bar{u}_2 : \overline{A}_2 \in [\![\Gamma_2 \vdash_{\mathcal{R}} u_2 : A_2]\!]$ and there exists some p such that $\overline{\Gamma}_1 \star \overline{\Gamma}_2 \vdash p : \bar{u}_1[\gamma_1]\ {}_{\overline{A}_1[\gamma_1]}{\approx}_{\overline{A}_2[\gamma_2]}\ \bar{u}_2[\gamma_2]$.*

5. *If* $(\Gamma_1 \vdash_{\mathcal{R}} u_1 : \texttt{TYPE}) \equiv (\Gamma_2 \vdash_{\mathcal{R}} u_2 : \texttt{TYPE})$*, then for every* $\vdash \overline{\Gamma}_1 \in [\![\vdash_{\mathcal{R}} \Gamma_1]\!]$ *and* $\vdash \overline{\Gamma}_2 \in [\![\vdash_{\mathcal{R}} \Gamma_2]\!]$*, we have* $\overline{\Gamma}_1 \vdash \bar{u}_1 : \texttt{TYPE} \in [\![\Gamma_1 \vdash_{\mathcal{R}} u_1 : \texttt{TYPE}]\!]$ *and* $\overline{\Gamma}_2 \vdash \bar{u}_2 : \texttt{TYPE} \in [\![\Gamma_2 \vdash_{\mathcal{R}} u_2 : \texttt{TYPE}]\!]$ *and there exists some* p *such that* $\overline{\Gamma}_1 \star \overline{\Gamma}_2 \vdash p : \kappa(\bar{u}_1[\gamma_1], \bar{u}_2[\gamma_2])$.

Proof. The proof of the five items is done by induction on the typing derivations, assuming the existence of $\overline{\Sigma}_{\mathcal{T}}$. We show three relevant cases.

– Prod:

$$\frac{\Gamma \vdash_{\mathcal{R}} A : \texttt{TYPE} \qquad \Gamma, x : A \vdash_{\mathcal{R}} B : s}{\Gamma \vdash_{\mathcal{R}} \Pi x : A.\ B : s}$$

Take $\vdash \overline{\Gamma} \in [\![\vdash_{\mathcal{R}} \Gamma]\!]$. By induction hypothesis, we have $\overline{\Gamma} \vdash \overline{A} : \texttt{TYPE} \in [\![\Gamma \vdash_{\mathcal{R}} A : \texttt{TYPE}]\!]$. We have $(\overline{\Gamma}, x : \overline{A}) \triangleleft (\Gamma, x : A)$ and we know that the only translation of sort s is itself, therefore by induction hypothesis we have $\overline{\Gamma}, x : \overline{A} \vdash \overline{B} : s \in [\![\Gamma, x : A \vdash_{\mathcal{R}} B : s]\!]$. We conclude that $\overline{\Gamma} \vdash \Pi x : \overline{A}.\ \overline{B} : s$ using the Prod rule.

– Conv:

$$\frac{\Gamma \vdash_{\mathcal{R}} t : A \qquad (\Gamma \vdash_{\mathcal{R}} A : s) \equiv (\Gamma \vdash_{\mathcal{R}} B : s)}{\Gamma \vdash_{\mathcal{R}} t : B}$$

Take $\vdash \overline{\Gamma} \in [\![\vdash_{\mathcal{R}} \Gamma]\!]$. As we consider small derivations, either A is a small type or A and B are the same type.
If A is a small type, then by induction hypothesis we have $\overline{\Gamma} \star \overline{\Gamma} \vdash p : \kappa(\overline{A}[\gamma_1], \overline{B}[\gamma_2])$. By Lemma 5 we obtain $\overline{\Gamma} \vdash p[\gamma_{12}] : \kappa(\overline{A}, \overline{B})$. By Lemma 7 and induction hypothesis we have $\overline{\Gamma} \vdash \bar{t} : \overline{A} \in [\![\Gamma \vdash_{\mathcal{R}} t : A]\!]$. Thanks to Lemma 2, there exists some $\bar{t}'$ such that $\overline{\Gamma} \vdash \bar{t}' : \overline{B} \in [\![\Gamma \vdash_{\mathcal{R}} t : B]\!]$.
If A and B are the same type, then no conversion is needed and the result is simply given the induction hypothesis $\overline{\Gamma} \vdash \bar{t} : \overline{A}$.

– ConvRefl:

$$\frac{\Gamma \vdash_{\mathcal{R}} u : A}{(\Gamma \vdash_{\mathcal{R}} u : A) \equiv (\Gamma \vdash_{\mathcal{R}} u : A)}$$

Take $\vdash \overline{\Gamma} \in [\![\vdash_{\mathcal{R}} \Gamma]\!]$. By induction hypothesis, we have $\overline{\Gamma} \vdash \bar{u} : \overline{A} \in [\![\Gamma \vdash_{\mathcal{R}} u : A]\!]$.
If $\Gamma \vdash_{\mathcal{R}} A : \texttt{TYPE}$, then we build $\overline{\Gamma} \star \overline{\Gamma} \vdash p : \bar{u}[\gamma_1] \approx \bar{u}[\gamma_2]$ using all the congruence rules of $\approx$.
We proceed similarly for the case $A = \texttt{TYPE}$.

The existence of $\overline{\Sigma}_{\mathcal{T}}$ is proved by induction on the length of $\Sigma_{\mathcal{T}}$, using the previous five items and $\langle\rangle \triangleleft \langle\rangle$. □

Corollary 1 (Preservation). *If* $\vdash_{\mathcal{R}} t : A$ *and* $\vdash A : s \in [\![\vdash_{\mathcal{R}} A : s]\!]$*, then there exists* $\bar{t}$ *such that* $\vdash \bar{t} : A$.

Proof. By Theorem 1 we have $\vdash \bar{t}' : \overline{A}' \in [\![\vdash_{\mathcal{R}} t : A]\!]$. Using Lemma 7 with $\overline{A} \coloneqq A$, we have some $\bar{t}$ such that $\vdash \bar{t} : A \in [\![\vdash_{\mathcal{R}} t : A]\!]$. □

We directly derive the two following conservativity and consistency results. We say that a theory $\mathcal{T}_2$ is conservative over a theory $\mathcal{T}_1$ when every formula in the common language of $\mathcal{T}_1$ and $\mathcal{T}_2$ that is provable in $\mathcal{T}_2$ is also provable in $\mathcal{T}_1$.

Corollary 2 (Conservativity). *$\mathcal{T}$ is a conservative extension of $\mathcal{T}^{ax}$.*

Corollary 3 (Relative consistency). *If $\mathcal{T}^{ax}$ is consistent then $\mathcal{T}$ is also consistent.*

5 Conclusion

Discussion. In this paper, we showed that it is possible to replace user-defined rewrite rules by equational axioms, in the case of the $\lambda\Pi$-calculus modulo theory. This result works for theories with prelude encoding—which is satisfied by expressive theories such as predicate logic and set theory—and for small derivations—which is in practice the case. So as to replace rewrite rules by equational axioms, we have defined a heterogeneous equality with standard axioms—reflexivity, symmetry, transitivity, Leibniz principle—and congruences for each constructor. At the end, the theory with rewrite rules is a conservative extension of the theory with axioms.

Related work. The similar problem of the translation from an extensional system to an intensional system has been investigated by Oury [19]. He proposed a translation from the Extensional Calculus of Constructions to the Calculus of Inductive Constructions with additional axioms that define a heterogeneous equality. Winterhalter, Sozeau and Tabareau provided a translation from extensional type theory to intensional type theory [23,24]. They took advantage of the presence of dependent pairs to encode a heterogeneous equality, unlike Oury who defined it with axioms.

In this paper, we have shown the existence of a translation from a theory with rewrite rules to a theory with equational axioms. Technical challenges appear as we are not in an extensional type system. In particular, Oury and Winterhalter *et al.* had a homogeneous equality in their source theory and introduce a heterogeneous equality in the target theory. In this work, the source theory does not contain a homogeneous equality, and the target theory only contains a heterogeneous equality.

The major difference with previous works is that we are in a logical framework without an infinite hierarchy of sorts $s_i : s_{i+1}$ for $i \in \mathbb{N}$. In $\lambda\Pi/\equiv$, we only have `TYPE` : `KIND`, which is the reason why we cannot define an equality between types. As such an equality is of paramount importance in the transports, we have considered a subclass of types—called small types—for which we can define an equality. However, it is worth noting that the sorts of $\lambda\Pi/\equiv$ allowed a simplification: by construction, there is no transports on types, so the translation of a dependent function type is directly a dependent function type.

References

1. Adams, R.: Pure type systems with judgemental equality. Journal of Functional Programming **16**(2), 219–246 (2006). https://doi.org/10.1017/S0956796805005770
2. Assaf, A., Burel, G., Cauderlier, R., Delahaye, D., Dowek, G., Dubois, C., Gilbert, F., Halmagrand, P., Hermant, O., Saillard, R.: Dedukti: a Logical Framework based on the $\lambda\Pi$-Calculus Modulo Theory (2016), manuscript
3. Blanqui, F., Dowek, G., Grienenberger, E., Hondet, G., Thiré, F.: A modular construction of type theories. Logical Methods in Computer Science **Volume 19, Issue 1** (Feb 2023). https://doi.org/10.46298/lmcs-19(1:12)2023, https://lmcs.episciences.org/10959
4. Blot, V., Dowek, G., Traversié, T.: An Implementation of Set Theory with Pointed Graphs in Dedukti. In: LFMTP 2022 - International Workshop on Logical Frameworks and Meta-Languages : Theory and Practice. Haïfa, Israel (Aug 2022), https://inria.hal.science/hal-03740004
5. Cockx, J., Abel, A.: Sprinkles of extensionality for your vanilla type theory (2016)
6. Cousineau, D., Dowek, G.: Embedding Pure Type Systems in the Lambda-Pi-Calculus Modulo. In: Della Rocca, S.R. (ed.) Typed Lambda Calculi and Applications. pp. 102–117. Springer Berlin Heidelberg, Berlin, Heidelberg (2007)
7. Dershowitz, N., Jouannaud, J.P.: Rewrite Systems. In: Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (1991)
8. Dowek, G., Miquel, A.: Relative normalization (2007), manuscript
9. Dowek, G.: La part du calcul. Habilitation à diriger des recherches, Université de Paris 7 (Jun 1999), https://inria.hal.science/tel-04114581
10. Dowek, G., Werner, B.: Proof Normalization Modulo. Research Report RR-3542, INRIA (1998), https://inria.hal.science/inria-00073143, projet COQ
11. Geuvers, H., Werner, B.: On the Church-Rosser property for expressive type systems and its consequences for their metatheoretic study. In: Proceedings Ninth Annual IEEE Symposium on Logic in Computer Science. pp. 320–329 (1994). https://doi.org/10.1109/LICS.1994.316057
12. Gilbert, G., Leray, Y., Tabareau, N., Winterhalter, T.: The Rewster: The Coq Proof Assistant with Rewrite Rules (2023), https://types2023.webs.upv.es/TYPES2023.pdf
13. Harper, R., Honsell, F., Plotkin, G.: A Framework for Defining Logics. Journal of the ACM **40**(1), 143–184 (January 1993). https://doi.org/10.1145/138027.138060, https://doi.org/10.1145/138027.138060
14. Hofmann, M.: Conservativity of equality reflection over intensional type theory. In: Berardi, S., Coppo, M. (eds.) Types for Proofs and Programs. pp. 153–164. Springer Berlin Heidelberg, Berlin, Heidelberg (1996)
15. Hofmann, M.: Extensional Constructs in Intensional Type Theory. Springer London (1997). https://doi.org/10.1007/978-1-4471-0963-1
16. Hondet, G., Blanqui, F.: The New Rewriting Engine of Dedukti. In: FSCD 2020 - 5th International Conference on Formal Structures for Computation and Deduction. p. 16. No. 167, Paris, France (Jun 2020). https://doi.org/10.4230/LIPIcs.FSCD.2020.35, https://inria.hal.science/hal-02981561
17. Martin-Löf, P.: Constructive mathematics and computer programming. Studies in logic and the foundations of mathematics **104**, 167–184 (1984), https://api.semanticscholar.org/CorpusID:61930968
18. McBride, C.: Dependently Typed Functional Programs and their Proofs. Ph.D. thesis, University of Edinburgh (1999)

19. Oury, N.: Extensionality in the Calculus of Constructions. In: Hurd, J., Melham, T. (eds.) Theorem Proving in Higher Order Logics. pp. 278–293. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)
20. Poincaré, H.: La Science et l'Hypothèse. Flammarion (1902)
21. Siles, V.: Investigation on the typing of equality in type systems. Ph.D. thesis, Ecole Polytechnique (Nov 2010), https://pastel.archives-ouvertes.fr/pastel-00556578
22. Siles, V., Herbelin, H.: Pure Type System conversion is always typable. Journal of Functional Programming **22**(2), 153 – 180 (May 2012). https://doi.org/10.1017/S0956796812000044, https://inria.hal.science/inria-00497177
23. Winterhalter, T., Sozeau, M., Tabareau, N.: Eliminating Reflection from Type Theory. In: CPP 2019 - 8th ACM SIGPLAN International Conference on Certified Programs and Proofs. pp. 91–103. ACM, Lisbonne, Portugal (Jan 2019). https://doi.org/10.1145/3293880.3294095, https://hal.science/hal-01849166
24. Winterhalter, T.: Formalisation and meta-theory of type theory. Ph.D. thesis, Université de Nantes (2020)