Sub-extensional type theory

Jérémy Ledent, supervised by Hugo Herbelin
Laboratoire PPS, équipe πr^2, Université Paris 7

March - July, 2015

General context

Martin-Löf's type theory is the basis of a variety of formal systems underlying many proof assistants. There are a lot of variants of type theory, among which two classes can be identified: *intensional* and *extensional* type theory. The difference between the two lies in the manner in which the equality is dealt with, but this results in huge differences. Intensional type theory requires the user to write explicit coercions, which can lead to a big overhead when programming with dependent types. It also lacks some desirable properties such as function extensionality. On the other hand, extensional type theory deals more smoothly with these issues, but as a result it suffers from undecidable type-checking. Moreover, it allows to type non-normalizing terms.

There have been many different attempts to reconcile these two views, either by reducing the overhead in intensional type theory, by trying to put up with the undecidability of extensional type theory, or by proposing new kinds of type theories which combine the good aspects of the two.

Problem Studied

The goal of this internship is to provide an alternative, *sub-extensional type theory*, which lies in-between intensional and extensional type theory. It should be decidable, strongly-normalizing, and offer some degree of extensionality. Such a theory could then be implemented in order to make programming with dependent types easier than what is currently possible with Coq or Agda. This would facilitate the conception of certified programs, and the formalization of mathematics inside proof assistants.

Several other theories with a similar purpose already exist: for example, Observational Type Theory, the Calculus of Algebraic Constructions, or Coq Modulo Theory. Some of them are still in a theoretical phase, and some have experimental implementations. It is difficult to judge which idea will prove to be more practical than the others until we are able to experiment with them in hindsight. What we propose is a new, different approach to experiment with.

Proposed contribution

Our solution hinges on the same idea as the Calculus of Algebraic Constructions: by adding rewriting rules to the conversion relation, we can obtain a spectrum of theories which range from intensional to extensional type theory. The problem with this approach is that the theory quickly becomes undecidable. To overcome this issue, we enrich the syntax of terms to allow explicit proofs of conversion that provide enough information for the system to type-check the terms.

In order to make sure that our theory is relevant, we compare its expressiveness to that of other related type-systems. We then study its metatheory.

Arguments supporting its validity

Our new approach seems to meet most of its ambitions: it is decidable by construction, offers some degree of extensionality, and even though some work still needs to be done to check strong normalization, it seems that it should hold provided we make some reasonable assumptions on which rewriting rules are allowed (e.g., provable equalities that live in small types). Compared to the other solutions, our theory is rather close to intensional type theory. Therefore, we might be able to implement it as a plug-in in an already existing proof assistant like Coq, in the form of a mechanism that allows the user to choose provable rewriting rules that are to be dealt with implicitly by the system. This is a good advantage compared to other solutions which require more drastic changes, and thus need to be implemented from scratch in a new proof assistant.

We have chosen to work in the framework of pure type systems for reasons of simplicity, generality and taste, but this choice is not restrictive. We expect that in the presence of inductive types, or of Coq's cumulative universes, our approach should work just as well, since these problems are orthogonal to the way the conversion relation is dealt with.

Summary and future work

Our Sub-extensional Type Theory offers a new alternative way to cope with the pros and cons of intensional and extensional type theories. It allows the user to control the degree of extensionality of the theory, while preserving decidability.

There is still some theoretical work to do: what kind of rewriting rules should or shouldn't be allowed to have strong normalization? Can we characterize precisely what extensional properties can be obtained in our framework? It would also be nice to formalize some of the metatheory in Coq, since the proofs are mostly long, boring inductions that are tedious to check on paper.

Finally, a concrete implementation, either as an experimental proof assistant or as a plug-in in an already existing one, would allow us to judge how useful this approach would be in practice.

Introduction

Martin-Löf's type theory is a kind of formalism based on the Curry-Howard correspondence which is both a dependently-typed programming language and a logical system. It is for instance at the core of a wide variety of proof assistants. Type theory usually comes in two flavors: *intensional* type theory (ITT) or *extensional* type theory (ETT). Examples of proof assistants based on ITT include Coq [10], Agda [13] or Isabelle/HOL [12]; whereas variants of ETT are implemented in NuPRL [9] or in Andrej Bauer's experimental proof-system Andromeda [4].

In ITT, there are two coexisting notions of equality between terms. The *conversion* relation is a small, decidable relation which covers the computational part of equality (usually, it is the closure of β or $\beta\eta$-reduction). We can think of convertible terms as being "equal by definition": for example, if we define a function f such that $f(x) = x^2$, then the expression $f\ 3$ is convertible to 3^2. The proof-system must be able to guess the convertibility between terms in order to type-check a proof. The second notion of equality is called *propositional equality*. It contains the conversion relation, but also more complicated equalities that are usually proved by induction, for example, $x + y = y + x$. Propositional equality is generally undecidable, but ITT requires the user to provide enough information inside the terms so that the type-checking of the system remains decidable. However, these explicit coercions asked from the user can quickly lead to a big overhead when programming with dependent types.

In ETT, these two notions of equality coincide thanks to the so-called *reflection rule*, which roughly says that whenever two terms are propositionally equal, then they are convertible. Thus, type-checking becomes undecidable, since the proof assistant would have to guess whether two terms are propositionally equal, which is undecidable. On the other hand, this makes the theory more expressive: some desirable properties such as *function extensionality* or *uniqueness of identity proofs* become provable, whereas they cannot be proved in ITT.

There has been a lot of work trying to accommodate the pros and the cons of both intensional and extensional type theories. The two main motivations are to facilitate the use of type theory as a dependently-typed programming language, and to simplify the formalization of mathematics in proof assistants.

In Coq, there are several libraries which aim at automatizing the treatment of equality when programming with dependent types. Chung-kil Hur's library *Heq* [8] provides a notion of heterogeneous equality which hides the explicit coercions, along with a set of tactics which help manipulate these coercions smoothly. Matthieu Sozeau's Russell [16] is a language for writing programs with dependent types in a very straightforward way, while giving them a rich specification. Some proof obligations are then generated in order to make sure that the program meets its specification. Once these obligations are proved, the program is translated to a Coq term where the coercions are inserted automatically where needed.

On the other hand, there has been some attempts at implementing a proof assistant

based on ETT. The main challenge is to put up with the undecidability of type-checking. NuPRL was the first such proof-system to be implemented. More recently, Andrej Bauer has started developing an experimental proof assistant, ANDROMEDA, which directly implements the reflection rule. It is an example of a *Homotopy Type System (HTS)*, a kind of type theory proposed by Vladimir Voevodsky which is equipped with *two* equality types, one with a reflection rule and the other without.

An alternative solution to the "intensional vs. extensional" question is to build a new kind of type theory which is neither intensional nor extensional, but combines the good aspects of the two. Martin Hofmann's PhD thesis [7] investigates how we can add extensional concepts to ITT, and how the resulting theory compares with ETT. Nicolas Oury's thesis focuses on the Calculus of Inductive Constructions (CIC), and shows how by making some reasonable additions to intensional CIC, we can embed extensional CIC. Thorsten Altenkirch and Conor McBride have developed the so-called *Observational Type Theory* (OTT) [2], which allows some degree of extensionality thanks to an alternative notion of propositional equality based on setoids, while keeping a decidable type-checking. OTT has been implemented in the experimental proof assistant EPIGRAM.

The *Calculus of Algebraic Constructions* (CAC) [5] allows the user to define some rewriting rules that are to be added to the conversion relation. The resulting theory lies somewhere in-between intensional and extensional type theory. However, adding too many rewriting rules will usually result in losing the decidability of type-checking. In the same vein, Pierre-Yves Strub has developed an extension of the proof assistant COQ, called *Coq Modulo Theory*, along with an experimental implementation COQMT [17]. COQMT embeds in its computational mechanism the validity of user-defined first-order equational theories, and ensures that decidability is preserved.

In this report, we define a new class of type systems based on CAC, which we call *sub-extensional type theory*. In order to keep the decidability of type-checking, we extend the syntax of terms to allow explicit proofs of conversion, adapting the work of Van Doorn, Geuvers and Wiedijk [18]. The second section compares our type system with other neighbouring systems. Finally, the third section studies some of its meta-theoretical properties.

1 Towards the syntax of $\mathrm{PTS}^{\mathcal{R}}_{\textbf{SUB-EXT}}$

The COQ proof assistant is based on the Calculus of Inductive Constructions (CIC) with a hierarchy of cumulative universes. In order to simplify our study, we will work in the more general setting of Pure Type Systems (PTS) with equality, even though the theory behind COQ is outside the framework of PTS*s*. Since we are going to talk about conversion and equality a lot, the word "equality" will always denote *propositional equality* (or "Leibniz equality"), written with the '=' symbol, whereas conversion will always be referred to as "conversion", written '≡'. (In the literature, when there is no Leibniz equality involved, the conversion relation is sometimes called "equality" and noted '='.)

In order to define so-called *sub-extensional* pure type systems ($\text{PTS}^{\mathcal{R}}_{\text{Sub-Ext}}$), we will rely on two different ideas.

Following the ideas of the Calculus of Algebraic Constructions (CAC) [5], we allow to extend the conversion relation through a set of user-defined rewrite rules. We can thus control the degree of extensionality of our type system: without rewrite rules, we get intensional type theory; with one rewrite rule for each provable equality, we get an extensional type theory. We are interested in having something in-between.

Adding too much extensionality (i.e., too many rewrite rules) could result in undecidability of type-checking. To prevent that from happening, we add explicit witnesses of conversion inside the syntax of the terms, reusing the work of Van Doorn, Wiedijk and Geuvers [18].

1.1 Pure type systems

There are two presentations of pure type systems: one with *untyped conversion* which we call PTS, and one with *typed conversion*, which we call PTS_e. The equivalence between the two was shown by Siles and Herbelin [15], using ideas introduced by Adams [1]. The syntax for the terms, types and sorts is the same for both systems, the difference lies in the typing judgements.

A specification of a PTS is given by the following data:

- A set S of *sorts*, whose elements are usually written s.
- A set $\mathsf{Ax} \subseteq \mathsf{S} \times \mathsf{S}$ of *axioms*.
- A set $\mathsf{Rel} \subseteq \mathsf{S} \times \mathsf{S} \times \mathsf{S}$ of *relations*.

We fix an infinite set $\mathcal{V}$ of *variables*, denoted by x, y, z. The syntax of pseudoterms is the following:

$$M, N ::= x \mid s \mid MN \mid \lambda x : M.N \mid \Pi x : M.N$$

We then define *one-step β-reduction*, written either $\rightsquigarrow_\beta$ or $\rightsquigarrow$, which is the congruent closure of the relation:

$$(\lambda x : A.M)N \rightsquigarrow M[x := N]$$

where $M[x := N]$ denotes the capture-free substitution of variable x by N in M, defined as usual. *β-reduction* is the reflexive-transitive closure of $\rightsquigarrow_\beta$, written $\rightsquigarrow^*_\beta$ or $\rightsquigarrow^*$. *β-conversion* is the reflexive-symmetric-transitive closure of $\rightsquigarrow_\beta$, written $\equiv_\beta$ or $\equiv$.

Pseudocontexts, denoted by Γ, Δ, are finite lists of the form $\Gamma = x_1 : A_1, \ldots, x_n : A_n$, where all variables (x_i) are distinct. We write $\Gamma(x_i) = A_i$ the type associated to x_i by Γ.

The typing rules of PTS rely on two judgements:

- $\Gamma \vdash \mathsf{wf}$, which says that Γ is a well-formed pseudocontext, or just *context*.

- $\Gamma \vdash M : A$, which says that M has type A in context Γ. When A is a sort, M is called a *type*. When A is a type, M is called a *term*.

The system PTS_e uses a third judgement for typed conversion:

- $\Gamma \vdash M \equiv N : A$, which says that M is convertible to N through a reduction path along which all terms have type A in context Γ.

The full sets of typing rules for PTS and PTS_e are given in appendix A and C.

Terminology and notations. We will try to use lowercase letters a, b, t, u for terms, and uppercase letters A, B for types. When it is unclear what objects we are manipulating, we will write $M : A$, in which case M is either a term or a type, and A is either a type or a sort.

Since the words "pseudoterm" and "pseudocontext" are a bit awkward to use, we will sometimes leave out the prefix "pseudo" if it doesn't hinder comprehension. For example, when we write $M : A$, we might call M a term even though it could actually be a type. The distinction between terms and types is usually either obvious from the context, or irrelevant. Similarly, we might call A the type of M, even though it could actually be a sort.

1.2 Adding rewriting rules in the conversion

The first step towards $\text{PTS}^{\mathcal{R}}_{\text{Sub-Ext}}$ is to enrich the conversion relation by allowing rewriting rules in addition to β-reduction. This idea has been studied for various systems, such as the so-called algebraic-λ-cube [3], or the calculus of algebraic constructions [5].

Definition 1.1. A *rewriting rule* is a pair of pseudoterms written $\ell \to r$, such that:

- $FV(r) \subseteq FV(\ell)$.
- There exist a context Γ_ℓ and a type A_ℓ such that $\Gamma_\ell \vdash \ell : A_\ell$, and A_ℓ is not a sort.
- For every Γ and A such that $\Gamma \vdash \ell : A$, $\Gamma \vdash r : A$ is derivable.

Such a rewriting rule induces a reduction relation on pseudoterms, which is the congruent closure of the relation $\ell[\overrightarrow{x} := \overrightarrow{N}] \to r[\overrightarrow{x} := \overrightarrow{N}]$, where $\overrightarrow{N}$ is any instantiation of the free variables $\overrightarrow{x}$ of ℓ.

Given a set $\mathcal{R}$ of rewriting rules, we write $\rightsquigarrow_{\mathcal{R}}$ the one-step reduction relation induced by the rules of $\mathcal{R}$, $\rightsquigarrow^*_{\mathcal{R}}$ its reflexive-transitive closure, and $\equiv_{\mathcal{R}}$ its reflexive-transitive-symmetric closure. Similarly, we write $\rightsquigarrow_{\beta\mathcal{R}}$, $\rightsquigarrow^*_{\beta\mathcal{R}}$ and $\equiv_{\beta\mathcal{R}}$ when both β-reduction and rewriting rules are allowed.

The question of what kind of rules should or shouldn't be allowed in order to preserve good properties of the type system are studied in [5], with a definition of a "General Schema"

for higher-order rules. We will not deal with this issue here, and will work with arbitrary rules instead. Here are some examples of rewrite rules:

$$x + 0 \to x \qquad x + S(y) \to S(x + y) \qquad 0 + x \to x \qquad S(x) + y \to S(x + y)$$

$$x + y \to y + x \qquad \mathtt{if}(\mathsf{true}, u, v) \to u \qquad \mathtt{if}(\mathsf{false}, u, v) \to v \qquad \mathtt{nat\_rect}(u, v, 0) \to u$$

$$\mathtt{nat\_rect}(u, v, \mathsf{S}(n)) \to v\ n\ (\mathtt{nat\_rec}(u, v, n)) \qquad \mathtt{eq\_rect}(u, \mathsf{refl}) \to u$$

Given a set $\mathcal{R}$ of rewriting rules and a specification $(\mathsf{S}, \mathsf{Ax}, \mathsf{Rel})$, we define the system $\mathrm{PTS}^{\mathcal{R}}$. It is very similar to PTS: the syntax of the pseudoterms and pseudocontexts is the same, the only difference lies in the conversion rule, where the side-condition $(A \equiv_\beta A')$ is replaced by $(A \rightsquigarrow^*_{\beta\mathcal{R}} A'$ or $A' \rightsquigarrow^*_{\beta\mathcal{R}} A)$.

The only difference with the more intuitive condition $(A \equiv_{\beta\mathcal{R}} A')$ is that it ensures that along the reduction path from A to A', all the types that appear at the "peaks" are well-typed. We use the abbreviation $A \mathrel{\widehat{\equiv}}_{\beta\mathcal{R}} A'$ to mean that there exist well-typed terms $A_1, \ldots, A_n$ such that $A \rightsquigarrow^*_{\beta\mathcal{R}} A_1 \ {}_{\beta\mathcal{R}}^{*}\!\!\leftsquigarrow A_2 \rightsquigarrow^*_{\beta\mathcal{R}} \cdots {}_{\beta\mathcal{R}}^{*}\!\!\leftsquigarrow A_n \rightsquigarrow^*_{\beta\mathcal{R}} A'$.

Using this notation, the conversion rule of $\mathrm{PTS}^{\mathcal{R}}$ is written:

$$\textsc{Conv}\ \frac{\Gamma \vdash a : A \qquad \Gamma \vdash A' : s \qquad A \mathrel{\widehat{\equiv}}_{\beta\mathcal{R}} A'}{\Gamma \vdash a : A'}$$

This technical point is only there to make sure that the system enjoys the *subject reduction* property. Indeed, the usual proof of subject reduction for PTS relies on the fact that the β-reduction relation is *confluent*. Here, since we added arbitrary rewriting rules, the relation $\rightsquigarrow^*_{\beta\mathcal{R}}$ is not necessarily confluent anymore. No proof of subject reduction for $\mathrm{PTS}^{\mathcal{R}}$ using the usual conversion rule $(A \equiv_{\beta\mathcal{R}} A')$ is currently known, hence we use the more restrictive conversion rule with "typed peaks". The proof of subject reduction for $\mathrm{PTS}^{\mathcal{R}}$ can be found in [3].

The full set of rules for the system $\mathrm{PTS}^{\mathcal{R}}$ can be found in appendix B.

1.3 Adding explicit conversion

When the conversion relation becomes richer, knowing whether two given terms are convertible might become undecidable: for example, by losing the confluence property, or by introducing loops in our rewriting system. Our next step is thus to add explicit witnesses of conversion in the syntax of our terms. These proofs of conversion help up recover decidability by giving additional information to the type-checker. In this section, we follow the work done in [18] to define the system PTS_f, and add some modifications to get the system PTS_{fe} that we're interested in.

Explicit conversion ($\mathbf{PTS}_f$). We extend the syntax of pseudoterms with the following construction:

$$M, N ::= \ldots \mid M^H$$

where H is a proof of conversion, whose syntax is given by the grammar:

$$H ::= \underline{H} \mid \overleftarrow{H} \mid H \cdot H' \mid \{H, [x : A]H'\} \mid \langle H, [x : A]H'\rangle \mid HH' \mid \beta(M) \mid \iota(M^H)$$

Then, the deduction rules are those of PTS, where the conversion rule is replaced by the following:

$$\textsc{Conv}\ \frac{\Gamma \vdash a : A \qquad \Gamma \vdash A' : s \qquad \Gamma \vdash H : A \equiv A'}{\Gamma \vdash a^H : A'}$$

This rule introduces a new judgement $\Gamma \vdash H : A \equiv A'$, whose intended meaning is that H is a proof of convertibility between A and A'. Thus, we need some additional rules to properly define this judgement. This gives us the system PTS_f, whose full set of typing rules is given in appendix D.

We also define the *erasure map* $M \mapsto |M|$ which recursively deletes all proofs of conversion inside a term, and we extend it to contexts in the obvious way. If M is a PTS term (i.e., without conversion proofs), a *lift* of M is a PTS_f term M' such that its erasure is M.

We can now make precise the meaning of the conversion judgement: $M \equiv_\beta N$ and both M and N have a type under Γ *iff* there are lifts M', N', Γ' of M, N, Γ such that $\Gamma' \vdash H : M' \equiv N'$ is derivable [18].

Typed explicit conversion ($\mathbf{PTS}_{fe}$). Since we are interested in combining the idea of explicit conversion with the presence of rewriting rules of $\mathrm{PTS}^{\mathcal{R}}$, the system PTS_f is not entirely satisfactory. Indeed, the conversion rule of $\mathrm{PTS}^{\mathcal{R}}$ has the side-condition $A \mathrel{\widehat{\equiv}} A'$, i.e., it requires that A and A' be convertible through a path where the peaks are well-typed. On the other hand, the judgement $\Gamma \vdash H : A \equiv A'$ only implies that $|A| \equiv |A'|$, with no guarantee on what happens along the conversion path. Thus, we need another judgement for our proofs of conversion which contains more typing information, similarly to how the PTS_e judgement $\Gamma \vdash M \equiv M' : A$ guarantees that every term in the conversion path from M to M' is of type A in context Γ.

We define the system PTS_{fe} of pure type systems with typed explicit conversion (PTS_{fe} doesn't have rewriting rules yet, this paragraph is just about defining a system where the conversion is both explicit and typed). The syntax of the pseudoterms is the same as that of PTS_f. We extend the syntax of the proofs of conversion by the following construction:

$$H ::= \ldots \mid H^{H'}$$

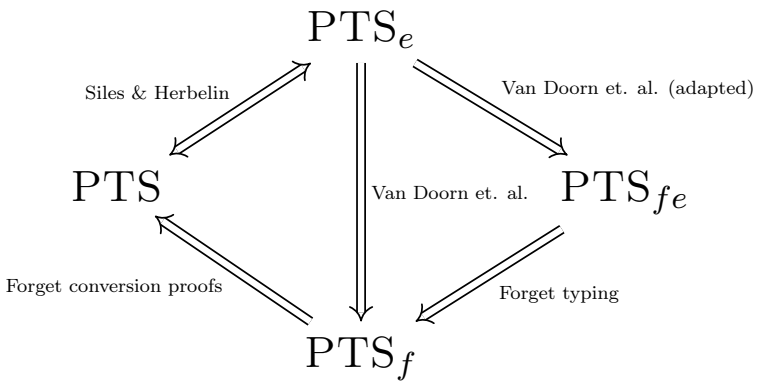
The full set of rules for this system is given in appendix E. The notation $|A|$ that appears in the rules Refl, Iota and Conv-eq refers to the erasure map defined previously.

PTS_{fe} uses a new judgement for proofs of conversion, $\Gamma \vdash H : M \equiv M' \bullet A$. It is important to note that M and M' are both PTS_{fe} terms, but A is just a PTS term: A doesn't contain any proof of conversion. The intended meaning of this judgement is the following: H is a proof that M and M' are convertible, and moreover, along the conversion path from M to M', all terms have an erasure which has type A in context $|\Gamma|$.

Equivalence. In their paper, Van Doorn, Geuvers and Wiedijk showed that PTS_f is equivalent to PTS and PTS_e [18] (the equivalence between PTS and PTS_e was previously established by Siles and Herbelin [15]). The direction from PTS_f to PTS is straightforward (we just have to forget the explicit proofs of conversion in the terms). The other direction is a little more involved since we have to rebuild the proofs of conversion, the proof given in [18] goes from PTS_e to PTS_f.

When going from PTS_e to PTS_f, two things are accomplished: proofs of conversion are reconstructed, and the typing information of the conversion is forgotten. The same proof can actually be adapted to go from PTS_e to PTS_{fe}, by simply *not* forgetting the typing information which is already present in PTS_e. We do not work out the details of this proof here, since it is a very straightforward adaptation of Van Doorn's proof.

Then, going from PTS_{fe} to PTS_f simply requires to forget the typing information. This is proven by an easy induction.



1.4 The syntax of $\text{PTS}^{\mathcal{R}}_{\text{Sub-Ext}}$

We now define the system that we're interested in, which we call $\text{PTS}^{\mathcal{R}}_{\text{Sub-Ext}}$. This system is parameterized by the following data:

- A PTS specification $(\mathsf{S}, \mathsf{Ax}, \mathsf{Rel})$.
- Two sets of rewriting rules $\mathcal{R}_{\text{dec}}$ and $\mathcal{R}_{\text{ndec}}$, such that the relation $\equiv_{\beta\mathcal{R}_{\text{dec}}}$ is decidable. We write $\mathcal{R} = \mathcal{R}_{\text{dec}} \cup \mathcal{R}_{\text{ndec}}$ the set of all rewriting rules.

The idea is to have two distinct conversion rules in the type system: one of them deals with $\equiv_{\beta\mathcal{R}_{\text{dec}}}$ transparently, as in $\text{PTS}^{\mathcal{R}}$, whereas the other one deals with $\equiv_{\mathcal{R}_{\text{ndec}}}$ and asks the user to provide explicit proofs of conversion, as in PTS_{fe}.

$$\textsc{Conv-dec}\ \frac{\Gamma \vdash a : A \qquad \Gamma \vdash A' : s \qquad A \mathrel{\widehat{\equiv}}_{\beta\mathcal{R}_{\text{dec}}} A'}{\Gamma \vdash a : A'} \qquad \textsc{Conv-ndec}\ \frac{\Gamma \vdash a : A \qquad \Gamma \vdash H : A \equiv A' \bullet s}{\Gamma \vdash a^H : A'}$$

The judgement $\Gamma \vdash H : A \equiv A' \bullet s$ that appears in the Conv-exp rule is defined like in PTS_{fe}, but the Beta rule is removed (since β-reduction is dealt with transparently in Conv-dec), and a new rule is added to deal with the rewriting rules:

$$\textsc{Rewrite}\ \frac{x_1 : A_1, \ldots, x_k : A_k \vdash \ell : A \qquad \forall i,\ \Gamma \vdash N_i : A_i \qquad \ell \to r \in \mathcal{R}_{\text{ndec}}}{\Gamma \vdash \mathcal{R}(\ell \to r, \overrightarrow{N}) : \ell[\overrightarrow{x} := \overrightarrow{N}] \equiv r[\overrightarrow{x} := \overrightarrow{N}] \bullet A[\overrightarrow{x} := \overrightarrow{N}]}$$

In the above rule, $\ell \to r$ is a rewriting rule in $\mathcal{R}_{\text{ndec}}$, and $FV(r) \subseteq FV(\ell) = \{x_1, \ldots, x_k\}$ (see definition 1.1). The full set of typing rules for $\text{PTS}^{\mathcal{R}}_{\text{Sub-Ext}}$ is given in appendix F.

Ideally, the set $\mathcal{R}_{\text{dec}}$ should be as big as possible, in order to minimize the number of explicit proofs of conversion required from the user. The problem of knowing how to optimally decompose a set $\mathcal{R}$ of rewriting rules into two sets $\mathcal{R}_{\text{dec}} \cup \mathcal{R}_{\text{ndec}}$ where $\mathcal{R}_{\text{dec}} \cup \{\beta\}$ is decidable will not be studied in this report.

In the end, since the only thing that must be guessed by the type-checker is $\mathcal{R}_{\text{dec}} \cup \{\beta\}$ and we defined it to be decidable, we obtain decidability of type-checking:

Theorem 1.2 (Decidability)**.** *Type-checking is decidable in* $\text{PTS}^{\mathcal{R}}_{\text{Sub-Ext}}$.

2 The ecosystem of $\text{PTS}^{\mathcal{R}}_{\text{Sub-Ext}}$

The system $\text{PTS}^{\mathcal{R}}_{\text{Sub-Ext}}$ that we have previously defined is actually just a particular case among a large class of type systems. Indeed, we can generalize the idea of splitting the set of rewriting rules $\mathcal{R}$ into parts and having several conversion rules in our type system in order to deal with them separately.

2.1 Pure type systems with hybrid conversion

Fix a PTS specification $(\mathsf{S}, \mathsf{Ax}, \mathsf{Rel})$ and a set $\mathcal{R}$ of rewriting rules.

In the presence of rewriting rules, there are three ways we can deal with the conversion relation:

(1) Implicit conversion with "typed peaks", as in $\mathrm{PTS}^{\mathcal{R}}$ (with the side-condition $A \mathrel{\widehat{\equiv}} A'$).

(2) Typed implicit conversion, as in PTS_e (using the judgement $\Gamma \vdash A \equiv A' : s$)

(3) Typed explicit conversion, as in PTS_{fe} (using the judgement $\Gamma \vdash H : A \equiv A' \bullet s$).

We leave out the untyped explicit conversion (as in PTS_f), since there is currently no known proof of subject reduction for pure type systems with rewriting rules and untyped conversion.

An *hybrid pure type system* on $\mathcal{R}$ is specified by dividing the set $\{\beta\} \cup \mathcal{R}$ into three sets $\mathcal{R}_1$, $\mathcal{R}_2$, $\mathcal{R}_3$, each of which is dealt with independently in one of the three ways mentioned above. Note that the β-reduction relation can also be placed in any of the three groups.

For example, $\mathrm{PTS}^{\mathcal{R}}_{\text{Sub-Ext}}$ and $\mathrm{PTS}^{\mathcal{R}}$ are two particular cases of hybrid PTS*s*:

- $\mathrm{PTS}^{\mathcal{R}}_{\text{Sub-Ext}}$ is obtained by having $\mathcal{R}_1 = \{\beta\} \cup \mathcal{R}_{\text{dec}}$, $\mathcal{R}_2 = \varnothing$ and $\mathcal{R}_3 = \mathcal{R}_{\text{ndec}}$.
- $\mathrm{PTS}^{\mathcal{R}}$ is obtained by having $\mathcal{R}_1 = \{\beta\} \cup \mathcal{R}$ and $\mathcal{R}_2 = \mathcal{R}_3 = \varnothing$.

We define two other such systems:

- $\mathrm{PTS}^{\mathcal{R}}_e$, where we put everything in $\mathcal{R}_2$. It is the analog of PTS_e, where we added rewriting rules. See appendix G for the full type system.
- $\mathrm{PTS}^{\mathcal{R}}_{fe}$, where we put everything in $\mathcal{R}_3$, so that all conversions are explicit. See appendix H for the full type system. $\mathrm{PTS}^{\mathcal{R}}_{\text{Sub-Ext}}$ lies somewhere in-between $\mathrm{PTS}^{\mathcal{R}}$ and $\mathrm{PTS}^{\mathcal{R}}_{fe}$.

2.2 Equivalence

We conjecture that given $(\mathsf{S}, \mathsf{Ax}, \mathsf{Rel})$ and $\mathcal{R}$, all hybrid pure type systems built on $\mathcal{R}$ should be equivalent. This would mean that the three ways to deal with conversion mentioned in section 2.1 are equivalent, and moreover, that they can be combined arbitrarily without interfering with each other (i.e., without changing the expressiveness of the theory).

Let us take a look at the equivalence between $\mathrm{PTS}^{\mathcal{R}}$, $\mathrm{PTS}^{\mathcal{R}}_e$ and $\mathrm{PTS}^{\mathcal{R}}_{fe}$. This is very similar to the equivalence between PTS, PTS_e and PTS_{fe}, which was proved by Siles and Herbelin [15] (for the direction $\mathrm{PTS} \to \mathrm{PTS}_e$), and by Van Doorn, Geuvers and Wiedijk [18] (for the direction $\mathrm{PTS}_e \to \mathrm{PTS}_f$, easily adapted for PTS_{fe}). We just need to check whether these proofs still work in the presence of rewriting rules.

The easy directions. As usual, the directions which only involve forgetting information are proved by a rather straightforward induction.

- $\mathrm{PTS}_e^{\mathcal{R}} \to \mathrm{PTS}^{\mathcal{R}}$: Forget typing information, that is, prove that if $\Gamma \vdash M \equiv M' : A$ is derivable in $\mathrm{PTS}_e^{\mathcal{R}}$, then $M \mathrel{\widehat{\equiv}}_{\beta\mathcal{R}} M'$.
- $\mathrm{PTS}_{fe}^{\mathcal{R}} \to \mathrm{PTS}_e^{\mathcal{R}}$: Forget the conversion proofs, that is, prove that if $\Gamma \vdash H : M \equiv M' \bullet A$ is derivable in $\mathrm{PTS}_{fe}^{\mathcal{R}}$, then $|\Gamma| \vdash |M| \equiv |M'| : A$ is derivable in $\mathrm{PTS}_e^{\mathcal{R}}$.
- $\mathrm{PTS}_{fe}^{\mathcal{R}} \to \mathrm{PTS}^{\mathcal{R}}$: Forget both conversion proofs and typing. This can be done either by putting together the two previous proofs, or by directly proving that if $\Gamma \vdash H : M \equiv M' \bullet A$ is derivable in $\mathrm{PTS}_{fe}^{\mathcal{R}}$, then $|M| \mathrel{\widehat{\equiv}}_{\beta\mathcal{R}} |M'|$.

Note that since the judgements $\Gamma \vdash M \equiv M' : A$ and $\Gamma \vdash H : M \equiv M' \bullet A$ both rely at some point on the typing judgement $\Gamma \vdash M : A$, each of these three properties must be proved alongside the fact that if $\Gamma \vdash M : A$ is derivable in the first system, then it is also derivable in the second one (possibly with some erasures, when we forget conversion proofs).

From $\mathrm{PTS}_e^{\mathcal{R}}$ to $\mathrm{PTS}_{fe}^{\mathcal{R}}$. We need to reconstruct the conversion proofs. Once again, the proof by Van Doorn et. al. [18] can be adapted rather easily to account for the presence of rewriting rules.

Let us state precisely the property that we need to prove:

Theorem 2.1. *For all Γ, M, N, A, the following statements hold:*

1. *if $\Gamma \vdash_e^{\mathcal{R}} \mathsf{wf}$, then there is a legal lift Γ' of Γ, that is, such that $\Gamma' \vdash_{fe}^{\mathcal{R}} \mathsf{wf}$ and $|\Gamma'| = \Gamma$.*
2. *if $\Gamma \vdash_e^{\mathcal{R}} M : A$, then there is a legal lift Γ' of Γ, and for every such legal lift Γ', there are lifts M', A' of M, A, such that $\Gamma' \vdash_{fe}^{\mathcal{R}} M' : A'$.*
3. *if $\Gamma \vdash_e^{\mathcal{R}} M \equiv N : A$, then there is a legal lift Γ' of Γ, and for every such legal lift Γ', there are lifts M', N' of M, N, and a conversion proof H, such that $\Gamma' \vdash_{fe}^{\mathcal{R}} H : M' \equiv N' \bullet A$.*

Proof. This is shown by mutual induction on the three judgements. The case of the rule App-eq involves some technical details that are worked out in [18]. □

From $\mathrm{PTS}^{\mathcal{R}}$ to $\mathrm{PTS}_e^{\mathcal{R}}$. This direction is much more difficult.

Without considering the rewriting rules, the question of whether we can go from PTS to PTS_e remained open for a few decades, with some first results proving the implication on some subclasses of PTS (Functional PTS by Adams in 2006 [1], and Semi-Full PTS by Siles in 2010 [14]), and the general result by Siles in 2012 [15].

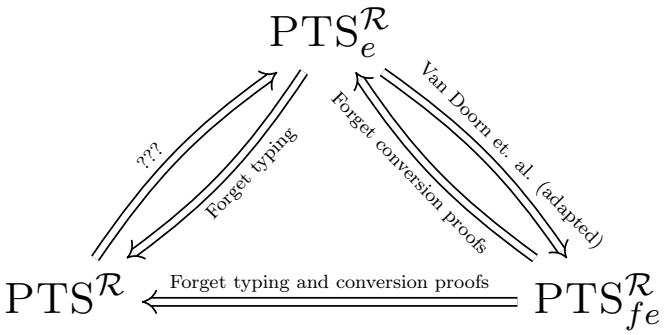
It was early noticed by Geuvers [6] that proving this implication, i.e., turning a untyped equality into a typed one, requires to prove the subject reduction property in the latter

system. Indeed, this is roughly what Siles's proof achieves: he first defines an intermediary system with a typed parallel β-reduction and some useful type annotations in order to embed the proof of confluence in his system. Then, he builds on this to prove subject reduction for PTS_e.

When we add rewriting rules to our systems, the picture becomes even more complicated.

The usual proof of subject reduction for PTS hinges in the confluence property of β-reduction, and consequently, one of the main steps of Siles's proof is to prove the confluence property. In $\mathrm{PTS}^{\mathcal{R}}$, we cannot expect the conversion relation to be confluent in general: we want to allow less restrictive sets of rewriting rules. Therefore, we must find another way to prove subject reduction in $\mathrm{PTS}^{\mathcal{R}}_e$. It might be possible to adapt the proof of subject-reduction for $\mathrm{PTS}^{\mathcal{R}}$ given in [3] by embedding it in an intermediary type system (like in Siles's proof), but since it is significantly more involved than the usual proof of subject reduction for PTS, we expect the task to be quite technical. We do not discuss it in this report.

In the end, we have the following picture:



3 Metatheory

In this section, we study some metatheoretical properties of our systems.

First, we need to add some notion of propositional equality to our type systems. This will allow us to distinguish whether the rewriting rules that we use are provable equalities, or arbitrary. We will show that any provable equality can be safely added to the conversion relation as a rewriting rule.

In order to deal with the consistency and normalization properties of our systems, we will follow the work of Alexandre Miquel's PhD thesis [11], which was originally done in the

setting of the Implicit Calculus of Constructions, and we will adapt it for pure type systems with equality. The idea is to define a notion of *abstract consistency model* (resp., *abstract normalization model*), such that the existence of such a model implies the consistency (resp., normalization) of the system. This allows us to split the proof in two steps: first we prove that the existence of an abstract model implies the consistency (resp., normalization) of the system, and then we construct one particular such model.

Using these notions of abstract models, we will be able to prove that the consistency and normalization properties are preserved from a system to another, by showing that we can turn an abstract model of the first system into an abstract model for the second one.

3.1 Pure type systems with equality

We want to have some notion of equality in our type systems. We will use the usual *propositional equality* (also known as *homogeneous equality*, *Leibniz equality* or *identity type*). In proof assistants like Coq, it is usually defined as an inductive type:

```
Inductive eq (A : Type) (x : A) : A -> Type :=
 | refl : eq A x x.
```

Since we do not have the inductive types machinery, we will formally define it by requiring to have three constants eq, refl and $\mathsf{eq\_rect}$ with the right types and computational behaviour. The type $\mathsf{eq}\ A\ x\ y$ will be noted $x =_A y$.

Definition 3.1 (PTS with equality)**.** A *Pure type system with equality* (abbreviated $\mathrm{PTS}_=$) is a PTS equipped with three constants eq, refl and $\mathsf{eq\_rect}$ of the following types:

$$\begin{aligned}
\mathsf{eq}: &\ \Pi(A:s).\Pi(x:A).\Pi(y:A).\ s\\
\mathsf{refl}: &\ \Pi(A:s).\Pi(x:A).\ x =_A x\\
\mathsf{eq_rect}: &\ \Pi(A:s).\Pi(x:A).\Pi(P:\Pi(y:A).x =_A y \to s).\Pi(u:P\ x\ (\mathsf{refl}\ A\ x)).\\
&\ \Pi(y:A).\Pi(h:x =_A y).\ P\ y\ h
\end{aligned}$$

Moreover, the reduction relation is such that $\mathsf{eq\_rect}\ A\ x\ P\ u\ x\ (\mathsf{refl}\ A\ x) \rightsquigarrow u$.

Since the types in the definition above are not very easy to read, we can equivalently say that the constants eq, refl and $\mathsf{eq\_rect}$ must satisfy the three following typing rules:

$$\text{=-Form}\ \frac{\Gamma \vdash A : s \qquad \Gamma \vdash x : A \qquad \Gamma \vdash y : A}{\Gamma \vdash x =_A y : s} \qquad \text{=-Intro}\ \frac{\Gamma \vdash A : s \qquad \Gamma \vdash x : A}{\Gamma \vdash \mathsf{refl}\ A\ x : x =_A x}$$

$$\text{=-Elim}\ \frac{\begin{array}{c}\Gamma \vdash A : s \qquad \Gamma \vdash a : A\\ \Gamma, y : A, h : a =_A y \vdash P : s \qquad \Gamma \vdash u : P[y := a][h := \mathsf{refl}\ A\ a] \qquad \Gamma \vdash a' : A \qquad \Gamma \vdash h' : a =_A a'\end{array}}{\Gamma \vdash \mathsf{eq\_rect}\ A\ a\ (\lambda y h.P)\ u\ a'\ h' : P[y := a'][h := h']}$$

When there is no ambiguity, we will often omit some implicit arguments, i.e., we will write $x = y$ instead of $x =_A y$, refl or $\mathsf{refl}\ x$ instead of $\mathsf{refl}\ A\ x$, and $\mathsf{eq\_rect}\ P\ u\ h$ instead of $\mathsf{eq\_rect}\ A\ x\ P\ u\ y\ h$.

3.2 Consistency

Our goal is to show that if $\mathrm{PTS}_=$ is consistent, then we can safely pick any set $\mathcal{R}$ of *provable* rewriting rule and add them to the conversion relation to get a consistent system $\mathrm{PTS}_=^{\mathcal{R}}$.

This is not a completely trivial question: given a provable equality $h : \ell = r$, $\mathsf{eq\_rect}\ P\ u\ h$ allows us to rewrite ℓ by r in the type of u, provided it is of the form $P(\ell)$. This is more restrictive than having the rewriting rule $\ell \to r$ in $\mathrm{PTS}^{\mathcal{R}}$, because the conversion rule allows us to rewrite under λ's and π's, which is not possible with $\mathsf{eq\_rect}$. Thus, adding provable rewriting rules allows us to prove strictly more formulas. However, since our intuition is that we get a theory which lies somewhere in-between intensional and extensional type theory, and both are consistent, it is expected that the resulting theory should be consistent.

Let us first make precise what we mean by provable rewriting rule:

Definition 3.2. A rewriting rule $\ell \to r$ is said to be *provable* in some system if the equality $\ell = r$ is provable in the empty context, when we quantify universally on all the free variables of ℓ (recall that we already ask that $FV(r) \subseteq FV(\ell)$, cf. def. 1.1).

More formally, if we denote by $x_1, \dots, x_k$ the free variables of ℓ, then there should be some inhabited types $A_1, \dots, A_k$ and some term M such that the judgement $x_1 : A_1, \dots, x_k : A_k \vdash M : \ell = r$ is derivable.

Now, let us define our notion of abstract consistency model of PTS, inspired from [11].

Definition 3.3 (Consistency model for PTS)**.** Let $(\mathsf{S}, \mathsf{Ax}, \mathsf{Rel})$ be the specification of a PTS. We denote by $\mathcal{V}$ the set of variables and Λ the set of pure λ-terms (that is, terms without sorts or products). A *consistency model* is given by the following data:

- A set $\mathcal{M}$.
- A binary operation $\cdot : \mathcal{M} \times \mathcal{M} \to \mathcal{M}$, called the *application*. We will often omit the dot and write ab instead of $a \cdot b$.
- An interpretation function $[\![-]\!]_- : \Lambda \times \mathsf{Val}_{\mathcal{M}} \to \mathcal{M}$, where $\mathsf{Val}_{\mathcal{M}}$ denotes the set of *valuations* on $\mathcal{M}$, i.e., functions from $\mathcal{V}$ to $\mathcal{M}$.
- A set $\mathcal{T} \subseteq \mathcal{M}$, whose elements are called *types*.
- A function $\mathsf{El} : \mathcal{T} \to \mathscr{P}(\mathcal{M})$, which gives the set of *elements* of a type.
- A constant $\Pi \in \mathcal{M}$.
- For each $s \in \mathsf{S}$, a constant $u_s \in \mathcal{T}$.

Moreover, they should satisfy the following axioms:

1. $[\![x]\!]_\rho = \rho(x)$
2. $[\![M\ N]\!]_\rho = [\![M]\!]_\rho \cdot [\![N]\!]_\rho$
3. $[\![\lambda x : A.M]\!]_\rho \cdot a = [\![M]\!]_{\rho;x\leftarrow a}$
4. If $\forall a \in \mathcal{M}, [\![M]\!]_{\rho;x\leftarrow a} = [\![M]\!]_{\rho';x\leftarrow a}$, then $[\![\lambda x : A.M]\!]_\rho = [\![\lambda x : A.M]\!]_{\rho'}$
5. If $\Pi tu \in \mathcal{T}$, then $t \in \mathcal{T}$ and $\forall a \in \mathsf{El}(t), ua \in \mathcal{T}$
6. If $\Pi tu \in \mathcal{T}$, then $\mathsf{El}(\Pi tu) = \{f \in \mathcal{M} \mid \forall a \in \mathsf{El}(t), fa \in \mathsf{El}(ua)\}$
7. $\forall s \in \mathsf{S}, \mathsf{El}(u_s) \subseteq \mathcal{T}$
8. $\forall (s, s') \in \mathsf{Ax}, u_s \in \mathsf{El}(u_{s'})$
9. $\forall (s_1, s_2, s_3) \in \mathsf{Rel}$, if $t \in \mathsf{El}(u_{s_1})$ and $\forall a \in \mathsf{El}(t), ua \in \mathsf{El}(u_{s_2})$, then $\Pi tu \in \mathsf{El}(u_{s_3})$

Axioms $1-4$ describe the behaviour of the interpretation function on pure λ-terms; axioms $5-6$ define the well-formedness and elements of product types; and axioms $7-9$ ensure that the specification $(\mathsf{S}, \mathsf{Ax}, \mathsf{Rel})$ of the PTS is respected.

Note that the interpretation function is only defined on pure λ-terms. We can extend it to all PTS pseudoterms as follows:

- $[\![\Pi x : A.B]\!]_\rho := (\Pi \cdot [\![A]\!]_\rho) \cdot [\![\lambda x : A.B]\!]_\rho$
- For all $s \in \mathsf{S}$, $[\![s]\!]_\rho := u_s$.

We also define the interpretation of contexts, which is a subset $[\![\Gamma]\!] \subseteq \mathsf{Val}_\mathcal{M}$:

- $[\![\varnothing]\!] := \mathsf{Val}_\mathcal{M}$.
- $[\![\Gamma, x : A]\!] := \begin{cases} \{\rho \in [\![\Gamma]\!] \mid \rho(x) \in \mathsf{El}([\![A]\!]_\rho)\} & \text{if } [\![A]\!]_\rho \in \mathcal{T} \\ \varnothing & \text{otherwise} \end{cases}$

We can finally prove the validity of our notion of model:

Theorem 3.4 (Validity)**.** *If the judgement $\Gamma \vdash M : A$ is derivable, then for all $\rho \in [\![\Gamma]\!]$,*

$$[\![A]\!]_\rho \in \mathcal{T} \text{ and } [\![M]\!]_\rho \in \mathsf{El}([\![A]\!]_\rho)$$

Proof. This is proved by induction on the derivation of $\Gamma \vdash M : A$. The Conv case relies on the fact that whenever $M \equiv_\beta M'$, then $[\![M]\!]_\rho = [\![M']\!]_\rho$. □

The validity property will imply the consistency of the system, provided that there is at lease one type which is not inhabited in the model.

Theorem 3.5 (Consistency)**.** *Given some* PTS *and a sort s, if there is a consistency model $\mathcal{M}$ such that there exists $t \in \mathsf{El}(u_s)$ with $\mathsf{El}(t) = \varnothing$, then the type $\mathsf{False}_s := \Pi(A : s).A$ is not provable in the empty context.*

Proof. If False_s is a well-formed type in the empty context, then using validity we have $[\![\mathsf{False}_s]\!]_\rho \in \mathcal{T}$. By axiom 6, $\mathsf{El}([\![\mathsf{False}_s]\!]_\rho) = \{f \in \mathcal{M} \mid \forall t \in \mathsf{El}(u_s), ft \in \mathsf{El}(t)\} = \varnothing$.

Hence, if the judgement $\vdash M : \mathsf{False}_s$ was derivable, then by validity, we would have $[\![M]\!]_\rho \in \mathsf{El}([\![\mathsf{False}_s]\!]_\rho)$, which is impossible. □

We now need to extend the definition of a consistency model for pure type systems with equality and rewriting rules:

Definition 3.6 (Consistency model for $\mathrm{PTS}_=$)**.** In the presence of equality, a consistency model is given by the usual data $(\mathcal{M}, \cdot, [\![-]\!]_-, \mathcal{T}, \mathsf{El}, \Pi, (u_s)_{s\in\mathsf{S}})$ satisfying the axioms $1-9$ of definition 3.3, along with two new constants Eq and $\mathsf{Refl} \in \mathcal{M}$ which satisfy:

10. $(\mathsf{Eq}\ t\ x\ y) \in \mathcal{T}$ iff $t \in \mathcal{T}$ et $x, y \in \mathsf{El}(t)$.
11. $a \in \mathsf{El}(\mathsf{Eq}\ t\ x\ y)$ iff $a = \mathsf{Refl}$ and $x = y$ (this is the set-theoretic equality on $\mathcal{M}$).

We extend the interpretation naturally by setting $[\![M =_A M']\!]_\rho := \mathsf{Eq} \cdot [\![A]\!]_\rho \cdot [\![M]\!]_\rho \cdot [\![M']\!]_\rho$, $[\![\mathsf{refl}\ A\ a]\!]_\rho := \mathsf{Refl}$ and $[\![\mathsf{eq\_rect}\ A\ x\ P\ u\ y\ h]\!]_\rho := [\![u]\!]_\rho$.

Definition 3.7 (Consistency model for $\mathrm{PTS}^{\mathcal{R}}$)**.** If we are in the presence of rewriting rules, we just need to add one last axiom:

12. For every rewriting rule $\ell \to r \in \mathcal{R}$, for all Γ in which the equality $\ell = r$ is derivable, and for all $\rho \in [\![\Gamma]\!]$, $[\![\ell]\!]_\rho = [\![r]\!]_\rho$.

We can check that the two notions of consistency models from definition 3.6 and 3.7 both satisfy the validity property (cf. 3.4). This is shown by two easy inductions. In the second induction (for the $\mathrm{PTS}^{\mathcal{R}}$ model), the case of the Conv rule involves some technical details: we need to show that if $M \equiv_{\beta\mathcal{R}} M'$ where M and M' are well-typed in context Γ, then for all $\rho \in [\![\Gamma]\!]$, we have $[\![M]\!]_\rho = [\![M']\!]_\rho$. This follows from axiom 12, the fact that subterms of a well-typed term are well-typed, and the properties of the interpretation function with respect to substitution.

Finally, we can prove that consistency is preserved by adding provable rewriting rules to the conversion relation:

Theorem 3.8 (Preservation of consistency)**.** *Let* $(\mathsf{S}, \mathsf{Ax}, \mathsf{Rel})$ *be a* PTS *specification such that* $\mathrm{PTS}_=$ *has a consistency model $\mathcal{M}$, and let $\mathcal{R}$ be a set of provable rewriting rules. Then $\mathcal{M}$ is also a consistency model for* $\mathrm{PTS}^{\mathcal{R}}_=$.

Proof. We just need to show that $\mathcal{M}$ satisfies the axiom 12 of definition 3.7.

Let $\ell \to r$ be a rewriting rule in $\mathcal{R}$, and Γ a context such that $\Gamma \vdash M : \ell = r$ is derivable. Using the validity property, for all $\rho \in [\![\Gamma]\!]$, $[\![M]\!]_\rho \in \mathsf{El}([\![\ell = r]\!]_\rho)$. Thus, by axiom 11, $[\![\ell]\!]_\rho = [\![r]\!]_\rho$. □

Remark: We can even be a little more general and say that if $\mathrm{PTS}^{\mathcal{R}}_{=}$ has a consistency model (where $\mathcal{R}$ potentially contains arbitrary rules), then $\mathrm{PTS}^{\mathcal{R}'}_{=}$ (where $\mathcal{R}'$ is obtained by adding rewriting rules that are provable in $\mathrm{PTS}^{\mathcal{R}}_{=}$) also has a consistency model.

3.3 Normalization

The strong normalization property in the presence of rewriting rules have been studied in [3] and [5], although they work in more restrictive frameworks than $\mathrm{PTS}^{\mathcal{R}}$ (respectively, the algebraic λ-cube, and the calculus of algebraic constructions).

We will not try to extend these results to $\mathrm{PTS}^{\mathcal{R}}$ in this report. Instead, we are interested in looking at how the systems with explicit proofs of conversion behave with respect to the reduction rule $M^H \rightsquigarrow M$. More precisely, we would like to prove something like "if $\mathrm{PTS}^{\mathcal{R}}$ is strongly normalizing, then $\mathrm{PTS}^{\mathcal{R}}_{\text{Sub-Ext}}$ is strongly normalizing".

In fact, we will study a related sub-question concerning the computation rule of $\mathsf{eq\_rect}$. Indeed, the reduction rule $M^H \rightsquigarrow M$ is very similar to the reduction rule $\mathsf{eq\_rect}\ P\ M\ \mathsf{refl} \rightsquigarrow M$, except that in the first case, H is a proof of conversion, whereas in the second one, refl is a proof of equality. We will first make more precise this correspondence between M^H and $\mathsf{eq\_rect}\ P\ M\ h$, and then we will try to extend the computation rule $\mathsf{eq\_rect}\ P\ M\ h \rightsquigarrow M$ to a bigger class of equality proofs than just reflexivity, while preserving the strong normalization property.

Correspondence between $\mathsf{eq\_rect}$ and explicit conversion proofs. The two constructions M^H (when dealing with explicit conversion) and $\mathsf{eq\_rect}\ P\ M\ h$ (when dealing with propositional equality) have a very similar purpose: given some term of some type A (resp., $P(a)$), we want to force it to have another type A' (resp., $P(b)$), using a proof of conversion $H : A \equiv A'$ (resp., a proof of equality $h : a = b$). We could wonder whether it is truly necessary to have those two notions coexist.

However, there are two main differences between those two constructions:

- When we use $\mathsf{eq\_rect}$ to rewrite a by b in the type of M, this must happen in some type family P. On the contrary, explicit conversion can rewrite anywhere in the type of M, even under λ's and π's. This is the reason why the system becomes "more extensional" when we add provable equalities to the conversion relation. Thus, we cannot in general translate the term M^H to some expression which only uses $\mathsf{eq\_rect}$.
- The reduction rules associated to these two kinds of coercions are different: $M^H \rightsquigarrow M$ for explicit conversion, versus $\mathsf{eq\_rect}\ P\ M\ \mathsf{refl} \rightsquigarrow M$ for propositional equality. Once again, using $\mathsf{eq\_rect}$ is more restrictive.

Concerning the first point, it would be interesting to find a precise correspondence between the two worlds, for example by showing that M^H can be translated to some expression using $\mathsf{eq\_rect}$ and some extensionality axioms restricted to $\mathcal{R}$. We didn't investigate this direction.

We are more interested in the second point regarding computation: could we extend the computation rule of $\mathsf{eq\_rect}$ to behave like the rule $M^H \rightsquigarrow M$? More precisely, we would like to allow the reduction $\mathsf{eq\_rect}\ P\ M\ h \rightsquigarrow M$ to occur not only when h is reflexivity, but whenever it belongs to the class of equalities obtained as the congruent closure of the equalities of $\mathcal{R}$. If $\mathcal{R}$ has good properties (e.g., provable equalities that live in small types), does the system remain strongly normalizing? This would allow better computational properties of explicit coercions, and thus make programming with dependent types easier. Note that this question is not restricted to the type systems that we defined in this report: for the sake of generality, we examine it in the context of pure type systems with equality.

Strong normalization. Like in section 3.2, we are going to use a notion of abstract model to prove the preservation of normalization. Our definition of *normalization model* is once again adapted from [11]. We denote by SN the set of strongly normalizing $\mathrm{PTS}_=$ terms.

Definition 3.9 (Saturated set)**.** Let E denote a context generated by the following grammar: $E ::= [] \mid E\ N \mid \mathsf{eq\_rect}\ P\ N\ E$, where N is a term in SN. A *saturated set* is a set S of $\mathrm{PTS}_=$ terms such that:

1. $S \subseteq \mathsf{SN}$.
2. If M is neither a lambda nor refl, then $E(M) \in S$. This is analogous to the usual condition "saturated sets contain all neutral terms", taking into account $\mathsf{eq\_rect}$.
3. If $N \in \mathsf{SN}$ and $E(x[N := M]) \in S$, then $E((\lambda x.M)\ N) \in S$.
4. If $E(M) \in S$, then $E(\mathsf{eq\_rect}\ P\ M\ \mathsf{refl}) \in S$.

We denote by SAT the set of all saturated sets.

Definition 3.10. Let R and S be two sets of terms, we denote by $R \to S$ the set:

$$R \to S\ :=\ \{M \mid \forall N \in R, \quad M\ N \in S\}$$

We can check that whenever $R, S \in \mathsf{SAT}$, $R \to S \in \mathsf{SAT}$.

Definition 3.11 (Normalization model for $\mathrm{PTS}_=$)**.** A *normalization model* for $\mathrm{PTS}_=$ is given by $(\mathcal{M}, \cdot, [\![-]\!]_-, \mathcal{T}, \mathsf{El}, \mathsf{Red}, \Pi, (u_s)_{s\in\mathsf{S}}, \mathsf{Eq}, \mathsf{Refl}, \bot)$, such that:

- $(\mathcal{M}, \cdot, [\![-]\!]_-, \mathcal{T}, \mathsf{El}, \Pi, (u_s)_{s\in\mathsf{S}})$ is a consistency model in the sense of definition 3.3.
- $\mathsf{El} : \mathcal{T} \to \mathscr{P}(\mathcal{M})\backslash\{\varnothing\}$, i.e., $\mathsf{El}(t)$ is not allowed to be empty.
- $\mathsf{Red} : \mathcal{T} \to \mathsf{SAT}$ and $\mathsf{Eq}, \mathsf{Refl}, \bot \in \mathcal{M}$.

We need three axioms for equality and Red:

10. $(\mathsf{Eq}\ t\ x\ y) \in \mathcal{T}$ iff $t \in \mathcal{T}$ et $x, y \in \mathsf{El}(t)$.

11'. $\mathsf{El}(\mathsf{Eq}\ t\ x\ y) = \{\bot, \mathsf{Refl}\}$ if $x = y$, $\{\bot\}$ otherwise.

12'. If $\Pi tu \in \mathcal{T}$, then $\mathsf{Red}(\Pi tu) = \mathsf{Red}(t) \to \bigcap_{a \in \mathsf{El}(t)} \mathsf{Red}(ua)$.

Finally, we demand that $\mathsf{eq\_rect}$ satisfies $[\![\mathsf{eq\_rect}\ P\ M\ h]\!]_\rho = [\![M]\!]_\rho$ if $[\![h]\!]_\rho = \mathsf{Refl}$, $\bot$ otherwise.

This notion of model should satisfy the following validity property:

If $\Gamma \vdash M : A$, *then* $[\![\Gamma]\!] \neq \varnothing$, *and for all valuation* $\rho \in [\![\Gamma]\!]$,

$$[\![A]\!]_\rho \in \mathcal{T} \quad \textit{and} \quad [\![M]\!]_\rho \in \mathsf{El}([\![A]\!]_\rho) \quad \textit{and} \quad M \in \mathsf{Red}([\![A]\!]_\rho)$$

Thus, the validity property entails that $M \in \mathsf{SN}$ since all terms in a saturated set are normalizing. So if there is a normalization model, $\mathrm{PTS}_=$ is strongly normalizing.

We haven't fully checked that the validity property holds yet. It would also be nice to construct such a model for $\mathrm{PTS}_=$ in order to make sure that our definition of a normalization model is relevant.

In the end, we would like to prove the following property:

Conjecture 3.12 (Preservation of strong normalization)**.** Let $\mathcal{R}$ be a set of provable equalities (cf. def. 3.2) which live in small types. We call the *congruent closure* of $\mathcal{R}$ the set of equalities that can be proved by applying symmetry, transitivity, and $\mathtt{f\_equal} : x = y \to f\ x = f\ y$, to the equalities of $\mathcal{R}$. We denote by $\mathrm{PTS}'_=$ the system that is obtained by allowing the reduction $\mathsf{eq\_rect}\ P\ M\ h \rightsquigarrow M$ to occur whenever the equality h is in the congruent closure of $\mathcal{R}$.

We conjecture that if $\mathrm{PTS}_=$ has a normalization model, then $\mathrm{PTS}'_=$ also has a normalization model.

Conclusion

We have defined a new class of type systems which draw a spectrum between intensional and extensional type theory. By enriching the syntax of terms with explicit conversion proofs, we ensure that the decidability of type-checking is preserved.

Looking at the normalization property in the presence of the rule $M^H \rightsquigarrow M$ has raised a more general sub-question: to what extent can we relax the computation rule for $\mathsf{eq\_rect}$? Although some details still need to be worked out, it looks like we can indeed make it less restrictive.

There is still a lot of future work to do in this direction. We have raised two conjectures throughout this report: the equivalence between Hybrid Type Systems, which would ensure that $\mathrm{PTS}^{\mathcal{R}}_{\text{Sub-Ext}}$ and all its neighbouring systems are equivalent; and a correspondence between the two constructions M^H and $\mathsf{eq\_rect}\ P\ M\ h$ would give us a more precise idea of what degree of extensionality our theory has.

Finally, it would be nice to have an implementation of our theory, either as an experimental proof assistant, or as a Coq plug-in.

References

[1] Robin Adams. Pure type systems with judgemental equality. *J. Funct. Program.*, 16(2):219–246, 2006.

[2] Thorsten Altenkirch and Conor McBride. Towards observational type theory. Manuscript, available online, February 2006.

[3] Franco Barbanera, Maribel Fernández, and Herman Geuvers. Modularity of strong normalization in the algebraic-lambda-cube. *J. Funct. Program.*, 7(6):613–660, 1997.

[4] Andrej Bauer. *Andromeda, an experimental implementation of type theory with a reflection rule*, 2014.

[5] Frédéric Blanqui, Jean-Pierre Jouannaud, and Mitsuhiro Okada. The calculus of algebraic constructions. In *Rewriting Techniques and Applications, 10th International Conference, RTA-99, Trento, Italy, July 2-4, 1999, Proceedings*, pages 301–316, 1999.

[6] Herman Geuvers. *Logics and Type Systems.* Phd thesis, Katholieke Universiteit Nijmegen, 1993.

[7] Martin Hofmann. *Extensional concepts in Intensional Type Theory.* Phd thesis, University of Edinburgh, 1995.

[8] Chung kil Hur. Heq: a coq library for heterogeneous equality.

[9] Christoph Kreitz. *The Nuprl Proof Development System, Version 5: Reference Manual and User's Guide.* Department of Computer Science, Cornell University, December 2002.

[10] The Coq development team. *The Coq proof assistant reference manual.* LogiCal Project, 2004.

[11] Alexandre Miquel. *Le calcul des constructions implicite : syntaxe et sémantique.* Phd thesis, Université Paris 7, 2001.

[12] Tobias Nipkow, Markus Wenzel, and Lawrence C. Paulson. *Isabelle/HOL: A Proof Assistant for Higher-order Logic.* Springer-Verlag, 2002.

[13] Ulf Norell. Dependently typed programming in agda. In *Proceedings of the 4th International Workshop on Types in Language Design and Implementation*, TLDI '09, pages 1–2. ACM, 2009.

[14] Vincent Siles and Hugo Herbelin. Equality is typable in semi-full pure type systems. In *25th annual IEEE symposium on Login in Computer Science (LICS '10), Edinburgh, UK*, pages 11–14, 2010.

[15] Vincent Siles and Hugo Herbelin. Pure type system conversion is always typable. *J. Funct. Program.*, 22(2):153–180, 2012.

[16] Matthieu Sozeau. Subset Coercions in Coq. In *TYPES'06*, volume 4502 of *Lecture Notes in Computer Science*, pages 237–252. Springer, 2007.

[17] Pierre-Yves Strub. Coq modulo theory. In Anuj Dawar and Helmut Veith, editors, *CSL*, volume 6247 of *Lecture Notes in Computer Science*, pages 529–543. Springer, 2010.

[18] Floris van Doorn, Herman Geuvers, and Freek Wiedijk. Explicit convertibility proofs in pure type systems. In *the Eighth ACM SIG-PLAN International Workshop on Logical Frameworks & Meta-languages: Theory & Practice, LFMTP '13, New York, USA*, pages 25–36, 2013.

A PTS

Inference rules for pure type systems (PTS).

$$\textsc{Nil}\ \frac{}{\vdash \mathsf{wf}} \qquad \textsc{Cons}\ \frac{\Gamma \vdash A : s \qquad x \notin \mathrm{dom}(\Gamma)}{\Gamma, x : A \vdash \mathsf{wf}} \qquad \textsc{Sort}\ \frac{\Gamma \vdash \mathsf{wf} \qquad (s_1, s_2) \in \mathsf{Ax}}{\Gamma \vdash s_1 : s_2} \qquad \textsc{Var}\ \frac{\Gamma \vdash \mathsf{wf} \qquad \Gamma(x) = A}{\Gamma \vdash x : A}$$

$$\textsc{Prod}\ \frac{\Gamma \vdash A : s_1 \qquad \Gamma, x : A \vdash B : s_2 \qquad (s_1, s_2, s_3) \in \mathsf{Rel}}{\Gamma \vdash \Pi x : A.B : s_3} \qquad \textsc{Lam}\ \frac{\Gamma \vdash A : s_1 \qquad \Gamma, x : A \vdash b : B \qquad \Gamma, x : A \vdash B : s_2 \qquad (s_1, s_2, s_3) \in \mathsf{Rel}}{\Gamma \vdash \lambda x : A.b : \Pi x : A.B}$$

$$\textsc{App}\ \frac{\Gamma \vdash F : \Pi x : A.B \qquad \Gamma \vdash a : A}{\Gamma \vdash Fa : B[x := a]} \qquad \textsc{Conv}\ \frac{\Gamma \vdash a : A \qquad \Gamma \vdash A' : s \qquad A \equiv_\beta A'}{\Gamma \vdash a : A'}$$

B PTS$^{\mathcal{R}}$

Inference rules for pure type systems with rewriting rules (PTS$^{\mathcal{R}}$). The only difference with PTS is the conversion rule (Conv).

$$\textsc{Nil}\ \frac{}{\vdash \mathsf{wf}} \qquad \textsc{Cons}\ \frac{\Gamma \vdash A : s \qquad x \notin \mathrm{dom}(\Gamma)}{\Gamma, x : A \vdash \mathsf{wf}} \qquad \textsc{Sort}\ \frac{\Gamma \vdash \mathsf{wf} \qquad (s_1, s_2) \in \mathsf{Ax}}{\Gamma \vdash s_1 : s_2} \qquad \textsc{Var}\ \frac{\Gamma \vdash \mathsf{wf} \qquad \Gamma(x) = A}{\Gamma \vdash x : A}$$

$$\textsc{Prod}\ \frac{\Gamma \vdash A : s_1 \qquad \Gamma, x : A \vdash B : s_2 \qquad (s_1, s_2, s_3) \in \mathsf{Rel}}{\Gamma \vdash \Pi x : A.B : s_3} \qquad \textsc{Lam}\ \frac{\Gamma \vdash A : s_1 \qquad \Gamma, x : A \vdash b : B \qquad \Gamma, x : A \vdash B : s_2 \qquad (s_1, s_2, s_3) \in \mathsf{Rel}}{\Gamma \vdash \lambda x : A.b : \Pi x : A.B}$$

$$\textsc{App}\ \frac{\Gamma \vdash F : \Pi x : A.B \qquad \Gamma \vdash a : A}{\Gamma \vdash Fa : B[x := a]} \qquad \textsc{Conv}\ \frac{\Gamma \vdash a : A \qquad \Gamma \vdash A' : s \qquad A \mathrel{\widehat{\equiv}}_{\beta\mathcal{R}} A'}{\Gamma \vdash a : A'}$$

C $\mathbf{PTS}_e$

Inference rules for pure type systems with typed conversion (PTS_e). Only the conversion rule (Conv) differs from PTS, with additional rules to define the typed conversion judgement ($\Gamma \vdash M \equiv M' : A$).

$$\textsc{Nil}\ \frac{}{\vdash \mathsf{wf}} \qquad \textsc{Cons}\ \frac{\Gamma \vdash A : s \qquad x \notin \mathrm{dom}(\Gamma)}{\Gamma, x : A \vdash \mathsf{wf}} \qquad \textsc{Sort}\ \frac{\Gamma \vdash \mathsf{wf} \qquad (s_1, s_2) \in \mathsf{Ax}}{\Gamma \vdash s_1 : s_2} \qquad \textsc{Var}\ \frac{\Gamma \vdash \mathsf{wf} \qquad \Gamma(x) = A}{\Gamma \vdash x : A}$$

$$\textsc{Prod}\ \frac{\Gamma \vdash A : s_1 \qquad \Gamma, x : A \vdash B : s_2 \qquad (s_1, s_2, s_3) \in \mathsf{Rel}}{\Gamma \vdash \Pi x : A.B : s_3} \qquad \textsc{Lam}\ \frac{\Gamma \vdash A : s_1 \qquad \Gamma, x : A \vdash b : B \qquad \Gamma, x : A \vdash B : s_2 \qquad (s_1, s_2, s_3) \in \mathsf{Rel}}{\Gamma \vdash \lambda x : A.b : \Pi x : A.B}$$

$$\textsc{App}\ \frac{\Gamma \vdash F : \Pi x : A.B \qquad \Gamma \vdash a : A}{\Gamma \vdash Fa : B[x := a]} \qquad \textsc{Conv}\ \frac{\Gamma \vdash a : A \qquad \Gamma \vdash A \equiv A' : s}{\Gamma \vdash a : A'} \qquad \textsc{Refl}\ \frac{\Gamma \vdash M : A}{\Gamma \vdash M \equiv M : A}$$

$$\textsc{Sym}\ \frac{\Gamma \vdash M \equiv M' : A}{\Gamma \vdash M' \equiv M : A} \qquad \textsc{Trans}\ \frac{\Gamma \vdash M \equiv M' : A \qquad \Gamma \vdash M' \equiv M'' : A}{\Gamma \vdash M \equiv M'' : A}$$

$$\textsc{Beta}\ \frac{\Gamma \vdash a : A \qquad \Gamma \vdash A : s_1 \qquad \Gamma, x : A \vdash b : B \qquad \Gamma, x : A \vdash B : s_2 \qquad (s_1, s_2, s_3) \in \mathsf{Rel}}{\Gamma \vdash (\lambda x : A.b)a \equiv b[x := a] : B[x := a]}$$

$$\textsc{Prod-eq}\ \frac{\Gamma \vdash A \equiv A' : s_1 \qquad \Gamma, x : A \vdash B \equiv B' : s_2 \qquad (s_1, s_2, s_3) \in \mathsf{Rel}}{\Gamma \vdash \Pi x : A.B \equiv \Pi x : A'.B' : s_3}$$

$$\textsc{Lam-eq}\ \frac{\Gamma \vdash A \equiv A' : s_1 \qquad \Gamma, x : A \vdash b \equiv b' : B \qquad \Gamma, x : A \vdash B : s_2 \qquad (s_1, s_2, s_3) \in \mathsf{Rel}}{\Gamma \vdash \lambda x : A.b \equiv \lambda x : A'.b' : \Pi x : A.B} \qquad \textsc{App-eq}\ \frac{\Gamma \vdash F \equiv F' : \Pi x : A.B \qquad \Gamma \vdash a \equiv a' : A}{\Gamma \vdash Fa \equiv F'a' : B[x := a]}$$

$$\textsc{Conv-eq}\ \frac{\Gamma \vdash a \equiv a' : A \qquad \Gamma \vdash A \equiv A' : s}{\Gamma \vdash a \equiv a' : A'}$$

D $\mathbf{PTS}_f$

Inference rules for pure type systems with explicit conversion (PTS_f). Only the conversion rule (Conv) differs from PTS, with additional rules to define the explicit conversion judgement ($\Gamma \vdash H : M \equiv M'$).

$$\textsc{Nil}\ \frac{}{\vdash \mathsf{wf}} \qquad \textsc{Cons}\ \frac{\Gamma \vdash A : s \qquad x \notin \mathrm{dom}(\Gamma)}{\Gamma, x : A \vdash \mathsf{wf}} \qquad \textsc{Sort}\ \frac{\Gamma \vdash \mathsf{wf} \qquad (s_1, s_2) \in \mathsf{Ax}}{\Gamma \vdash s_1 : s_2} \qquad \textsc{Var}\ \frac{\Gamma \vdash \mathsf{wf} \qquad \Gamma(x) = A}{\Gamma \vdash x : A}$$

$$\textsc{Prod}\ \frac{\Gamma \vdash A : s_1 \qquad \Gamma, x : A \vdash B : s_2 \qquad (s_1, s_2, s_3) \in \mathsf{Rel}}{\Gamma \vdash \Pi x : A.B : s_3} \qquad \textsc{Lam}\ \frac{\Gamma \vdash A : s_1 \qquad \Gamma, x : A \vdash b : B \qquad \Gamma, x : A \vdash B : s_2 \qquad (s_1, s_2, s_3) \in \mathsf{Rel}}{\Gamma \vdash \lambda x : A.b : \Pi x : A.B}$$

$$\textsc{App}\ \frac{\Gamma \vdash F : \Pi x : A.B \qquad \Gamma \vdash a : A}{\Gamma \vdash Fa : B[x := a]} \qquad \textsc{Conv}\ \frac{\Gamma \vdash a : A \qquad \Gamma \vdash A' : s \qquad \Gamma \vdash H : A \equiv A'}{\Gamma \vdash a^H : A'} \qquad \textsc{Refl}\ \frac{\Gamma \vdash M : A}{\Gamma \vdash \underline{M} : M \equiv M}$$

$$\textsc{Sym}\ \frac{\Gamma \vdash H : M \equiv M'}{\Gamma \vdash \overleftarrow{H} : M' \equiv M} \qquad \textsc{Trans}\ \frac{\Gamma \vdash H : M \equiv M' \qquad \Gamma \vdash H' : M' \equiv M''}{\Gamma \vdash H \cdot H' : M \equiv M''}$$

$$\textsc{Beta}\ \frac{\Gamma \vdash a : A \qquad \Gamma \vdash A : s_1 \qquad \Gamma, x : A \vdash b : B \qquad \Gamma, x : A \vdash B : s_2 \qquad (s_1, s_2, s_3) \in \mathsf{Rel}}{\Gamma \vdash \beta((\lambda x : A.b)\ a) : (\lambda x : A.b)\ a \equiv b[x := a]}$$

$$\textsc{Prod-eq}\ \frac{\Gamma \vdash A : s_1 \qquad \Gamma, x : A \vdash B : s_2 \qquad (s_1, s_2, s_3) \in \mathsf{Rel} \qquad \Gamma \vdash A' : s_1' \qquad \Gamma, x' : A' \vdash B' : s_2' \qquad (s_1', s_2', s_3') \in \mathsf{Rel} \qquad \Gamma \vdash H : A \equiv A' \qquad \Gamma, x : A \vdash H' : B \equiv B'[x' := x^H]}{\Gamma \vdash \{H, [x : A]H'\} : \Pi x : A.B \equiv \Pi x : A'.B'}$$

$$\textsc{Lam-eq}\ \frac{\Gamma \vdash A : s_1 \qquad \Gamma, x : A \vdash b : B \qquad \Gamma, x : A \vdash B : s_2 \qquad (s_1, s_2, s_3) \in \mathsf{Rel} \qquad \Gamma \vdash A' : s_1' \qquad \Gamma, x' : A' \vdash b' : B' \qquad \Gamma, x' : A' \vdash B' : s_2' \qquad (s_1', s_2', s_3') \in \mathsf{Rel} \qquad \Gamma \vdash H : A \equiv A' \qquad \Gamma, x : A \vdash H' : b \equiv b'[x' := x^H]}{\Gamma \vdash \langle H, [x : A]H' \rangle : \lambda x : A.b \equiv \lambda x' : A'.b'}$$

$$\textsc{App-eq}\ \frac{\Gamma \vdash F : \Pi x : A.B \qquad \Gamma \vdash a : A \qquad \Gamma \vdash F' : \Pi x' : A'.B' \qquad \Gamma \vdash a' : A' \qquad \Gamma \vdash H : F \equiv F' \qquad \Gamma \vdash H' : a \equiv a'}{\Gamma \vdash HH' : Fa \equiv F'a'} \qquad \textsc{Iota}\ \frac{\Gamma \vdash a : A \qquad \Gamma \vdash A' : s \qquad \Gamma \vdash H : A \equiv A'}{\Gamma \vdash \iota(a^H) : a \equiv a^H}$$

E $\mathbf{PTS}_{fe}$

Inference rules for pure type systems with typed explicit conversion (PTS_{fe}). This system is similar to PTS_f, but we keep some typing information in the explicit conversion judgement $\Gamma \vdash H : M \equiv M' \bullet A$.

$$\textsc{Nil}\ \frac{}{\vdash \mathsf{wf}} \qquad \textsc{Cons}\ \frac{\Gamma \vdash A : s \qquad x \notin \mathrm{dom}(\Gamma)}{\Gamma, x : A \vdash \mathsf{wf}} \qquad \textsc{Sort}\ \frac{\Gamma \vdash \mathsf{wf} \qquad (s_1, s_2) \in \mathsf{Ax}}{\Gamma \vdash s_1 : s_2} \qquad \textsc{Var}\ \frac{\Gamma \vdash \mathsf{wf} \qquad \Gamma(x) = A}{\Gamma \vdash x : A}$$

$$\textsc{Prod}\ \frac{\Gamma \vdash A : s_1 \qquad \Gamma, x : A \vdash B : s_2 \qquad (s_1, s_2, s_3) \in \mathsf{Rel}}{\Gamma \vdash \Pi x : A.B : s_3} \qquad \textsc{Lam}\ \frac{\Gamma \vdash A : s_1 \qquad \Gamma, x : A \vdash b : B \qquad \Gamma, x : A \vdash B : s_2 \qquad (s_1, s_2, s_3) \in \mathsf{Rel}}{\Gamma \vdash \lambda x : A.b : \Pi x : A.B}$$

$$\textsc{App}\ \frac{\Gamma \vdash F : \Pi x : A.B \qquad \Gamma \vdash a : A}{\Gamma \vdash Fa : B[x := a]} \qquad \textsc{Conv}\ \frac{\Gamma \vdash a : A \qquad \Gamma \vdash H : A \equiv A' \bullet s}{\Gamma \vdash a^H : A'} \qquad \textsc{Refl}\ \frac{\Gamma \vdash M : A}{\Gamma \vdash \underline{M} : M \equiv M \bullet |A|}$$

$$\textsc{Sym}\ \frac{\Gamma \vdash H : M \equiv M' \bullet A}{\Gamma \vdash \overleftarrow{H} : M' \equiv M \bullet A} \qquad \textsc{Trans}\ \frac{\Gamma \vdash H : M \equiv M' \bullet A \qquad \Gamma \vdash H' : M' \equiv M'' \bullet A}{\Gamma \vdash H \cdot H' : M \equiv M'' \bullet A}$$

$$\textsc{Beta}\ \frac{\Gamma \vdash a : A \qquad \Gamma \vdash A : s_1 \qquad \Gamma, x : A \vdash b : B \qquad \Gamma, x : A \vdash B : s_2 \qquad (s_1, s_2, s_3) \in \mathsf{Rel}}{\Gamma \vdash \beta((\lambda x : A.b)\ a) : (\lambda x : A.b)\ a \equiv b[x := a] \bullet B[x := a]}$$

$$\textsc{Prod-eq}\ \frac{\Gamma \vdash H : A \equiv A' \bullet s_1 \qquad \Gamma, x : A \vdash H' : B \equiv B'[x' := x^H] \bullet s_2 \qquad (s_1, s_2, s_3) \in \mathsf{Rel}}{\Gamma \vdash \{H, [x : A]H'\} : \Pi x : A.B \equiv \Pi x : A'.B' \bullet s_3}$$

$$\textsc{Lam-eq}\ \frac{\Gamma \vdash H : A \equiv A' \bullet s_1 \qquad \Gamma, x : A \vdash H' : b \equiv b'[x' := x^H] \bullet B \qquad \Gamma, x : A \vdash B : s_2 \qquad (s_1, s_2, s_3) \in \mathsf{Rel}}{\Gamma \vdash \langle H, [x : A]H' \rangle : \lambda x : A.b \equiv \lambda x' : A'.b' \bullet \Pi x : A.B}$$

$$\textsc{App-eq}\ \frac{\Gamma \vdash H : F \equiv F' \bullet \Pi x : A.B \qquad \Gamma \vdash H' : a \equiv a' \bullet A}{\Gamma \vdash HH' : Fa \equiv F'a' \bullet B[x := a]} \qquad \textsc{Iota}\ \frac{\Gamma \vdash a : A \qquad \Gamma \vdash H : A \equiv A' \bullet s}{\Gamma \vdash \iota(a^H) : a \equiv a^H \bullet |A|}$$

$$\textsc{Conv-eq}\ \frac{\Gamma \vdash H : M \equiv M' \bullet A \qquad \Gamma \vdash H' : A \equiv A' \bullet s}{\Gamma \vdash H^{H'} : M \equiv M' \bullet |A'|}$$

F $\mathrm{PTS}^{\mathcal{R}}_{\textsc{Sub-Ext}}$

Inference rules for sub-extensional pure type systems ($\mathrm{PTS}^{\mathcal{R}}_{\textsc{Sub-Ext}}$).

$$\textsc{Nil}\ \frac{}{\vdash \mathsf{wf}} \qquad \textsc{Cons}\ \frac{\Gamma \vdash A : s \qquad x \notin \mathrm{dom}(\Gamma)}{\Gamma, x : A \vdash \mathsf{wf}} \qquad \textsc{Sort}\ \frac{\Gamma \vdash \mathsf{wf} \qquad (s_1, s_2) \in \mathsf{Ax}}{\Gamma \vdash s_1 : s_2} \qquad \textsc{Var}\ \frac{\Gamma \vdash \mathsf{wf} \qquad \Gamma(x) = A}{\Gamma \vdash x : A}$$

$$\textsc{Prod}\ \frac{\Gamma \vdash A : s_1 \qquad \Gamma, x : A \vdash B : s_2 \qquad (s_1, s_2, s_3) \in \mathsf{Rel}}{\Gamma \vdash \Pi x : A.B : s_3} \qquad \textsc{Lam}\ \frac{\Gamma \vdash A : s_1 \qquad \Gamma, x : A \vdash b : B \qquad \Gamma, x : A \vdash B : s_2 \qquad (s_1, s_2, s_3) \in \mathsf{Rel}}{\Gamma \vdash \lambda x : A.b : \Pi x : A.B}$$

$$\textsc{App}\ \frac{\Gamma \vdash F : \Pi x : A.B \qquad \Gamma \vdash a : A}{\Gamma \vdash Fa : B[x := a]} \qquad \textsc{Conv-dec}\ \frac{\Gamma \vdash a : A \qquad \Gamma \vdash A' : s \qquad A \mathrel{\widehat{\equiv}}_{\beta\mathcal{R}_{\mathrm{dec}}} A'}{\Gamma \vdash a : A'}$$

$$\textsc{Conv-ndec}\ \frac{\Gamma \vdash a : A \qquad \Gamma \vdash H : A \equiv A' \bullet s}{\Gamma \vdash a^H : A'} \qquad \textsc{Refl}\ \frac{\Gamma \vdash M : A}{\Gamma \vdash \underline{M} : M \equiv M \bullet |A|} \qquad \textsc{Sym}\ \frac{\Gamma \vdash H : M \equiv M' \bullet A}{\Gamma \vdash \overleftarrow{H} : M' \equiv M \bullet A}$$

$$\textsc{Trans}\ \frac{\Gamma \vdash H : M \equiv M' \bullet A \qquad \Gamma \vdash H' : M' \equiv M'' \bullet A}{\Gamma \vdash H \cdot H' : M \equiv M'' \bullet A} \qquad \textsc{Rewrite}\ \frac{x_1 : A_1, \ldots, x_k : A_k \vdash \ell : A \qquad \forall i,\ \Gamma \vdash N_i : A_i \qquad \ell \to r \in \mathcal{R}_{\mathrm{ndec}}}{\Gamma \vdash \mathcal{R}(\ell \to r, \overrightarrow{N}) : \ell[\overrightarrow{x} := \overrightarrow{N}] \equiv r[\overrightarrow{x} := \overrightarrow{N}] \bullet A[\overrightarrow{x} := \overrightarrow{N}]}$$

$$\textsc{Prod-eq}\ \frac{\Gamma \vdash H : A \equiv A' \bullet s_1 \qquad \Gamma, x : A \vdash H' : B \equiv B'[x' := x^H] \bullet s_2 \qquad (s_1, s_2, s_3) \in \mathsf{Rel}}{\Gamma \vdash \{H, [x : A]H'\} : \Pi x : A.B \equiv \Pi x : A'.B' \bullet s_3}$$

$$\textsc{Lam-eq}\ \frac{\Gamma \vdash H : A \equiv A' \bullet s_1 \qquad \Gamma, x : A \vdash H' : b \equiv b'[x' := x^H] \bullet B \qquad \Gamma, x : A \vdash B : s_2 \qquad (s_1, s_2, s_3) \in \mathsf{Rel}}{\Gamma \vdash \langle H, [x : A]H' \rangle : \lambda x : A.b \equiv \lambda x' : A'.b' \bullet \Pi x : A.B}$$

$$\textsc{App-eq}\ \frac{\Gamma \vdash H : F \equiv F' \bullet \Pi x : A.B \qquad \Gamma \vdash H' : a \equiv a' \bullet A}{\Gamma \vdash HH' : Fa \equiv F'a' \bullet B[x := a]} \qquad \textsc{Iota}\ \frac{\Gamma \vdash a : A \qquad \Gamma \vdash H : A \equiv A' \bullet s}{\Gamma \vdash \iota(a^H) : a \equiv a^H \bullet |A|}$$

$$\textsc{Conv-eq}\ \frac{\Gamma \vdash H : M \equiv M' \bullet A \qquad \Gamma \vdash H' : A \equiv A' \bullet s}{\Gamma \vdash H^{H'} : M \equiv M' \bullet |A'|}$$

G $\mathbf{PTS}_e^{\mathcal{R}}$

Inference rules for $\mathrm{PTS}_e^{\mathcal{R}}$.

$$\textsc{Nil}\ \frac{}{\vdash \mathsf{wf}} \qquad \textsc{Cons}\ \frac{\Gamma \vdash A : s \qquad x \notin \mathrm{dom}(\Gamma)}{\Gamma, x : A \vdash \mathsf{wf}} \qquad \textsc{Sort}\ \frac{\Gamma \vdash \mathsf{wf} \qquad (s_1, s_2) \in \mathsf{Ax}}{\Gamma \vdash s_1 : s_2} \qquad \textsc{Var}\ \frac{\Gamma \vdash \mathsf{wf} \qquad \Gamma(x) = A}{\Gamma \vdash x : A}$$

$$\textsc{Prod}\ \frac{\Gamma \vdash A : s_1 \qquad \Gamma, x : A \vdash B : s_2 \qquad (s_1, s_2, s_3) \in \mathsf{Rel}}{\Gamma \vdash \Pi x : A.B : s_3} \qquad \textsc{Lam}\ \frac{\Gamma \vdash A : s_1 \qquad \Gamma, x : A \vdash b : B \qquad \Gamma, x : A \vdash B : s_2 \qquad (s_1, s_2, s_3) \in \mathsf{Rel}}{\Gamma \vdash \lambda x : A.b : \Pi x : A.B}$$

$$\textsc{App}\ \frac{\Gamma \vdash F : \Pi x : A.B \qquad \Gamma \vdash a : A}{\Gamma \vdash Fa : B[x := a]} \qquad \textsc{Conv}\ \frac{\Gamma \vdash a : A \qquad \Gamma \vdash A \equiv A' : s}{\Gamma \vdash a : A'} \qquad \textsc{Refl}\ \frac{\Gamma \vdash M : A}{\Gamma \vdash M \equiv M : A}$$

$$\textsc{Sym}\ \frac{\Gamma \vdash M \equiv M' : A}{\Gamma \vdash M' \equiv M : A} \qquad \textsc{Trans}\ \frac{\Gamma \vdash M \equiv M' : A \qquad \Gamma \vdash M' \equiv M'' : A}{\Gamma \vdash M \equiv M'' : A}$$

$$\textsc{Beta}\ \frac{\Gamma \vdash a : A \qquad \Gamma \vdash A : s_1 \qquad \Gamma, x : A \vdash b : B \qquad \Gamma, x : A \vdash B : s_2 \qquad (s_1, s_2, s_3) \in \mathsf{Rel}}{\Gamma \vdash (\lambda x : A.b)a \equiv b[x := a] : B[x := a]}$$

$$\textsc{Rewrite}\ \frac{x_1 : A_1, \ldots, x_k : A_k \vdash \ell : A \qquad \forall i,\ \Gamma \vdash N_i : A_i \qquad \ell \to r \in \mathcal{R}}{\Gamma \vdash \ell[\overrightarrow{x} := \overrightarrow{N}] \equiv r[\overrightarrow{x} := \overrightarrow{N}] : A[\overrightarrow{x} := \overrightarrow{N}]}$$

$$\textsc{Prod-eq}\ \frac{\Gamma \vdash A \equiv A' : s_1 \qquad \Gamma, x : A \vdash B \equiv B' : s_2 \qquad (s_1, s_2, s_3) \in \mathsf{Rel}}{\Gamma \vdash \Pi x : A.B \equiv \Pi x : A'.B' : s_3}$$

$$\textsc{Lam-eq}\ \frac{\Gamma \vdash A \equiv A' : s_1 \qquad \Gamma, x : A \vdash b \equiv b' : B \qquad \Gamma, x : A \vdash B : s_2 \qquad (s_1, s_2, s_3) \in \mathsf{Rel}}{\Gamma \vdash \lambda x : A.b \equiv \lambda x : A'.b' : \Pi x : A.B} \qquad \textsc{App-eq}\ \frac{\Gamma \vdash F \equiv F' : \Pi x : A.B \qquad \Gamma \vdash a \equiv a' : A}{\Gamma \vdash Fa \equiv F'a' : B[x := a]}$$

$$\textsc{Conv-eq}\ \frac{\Gamma \vdash a \equiv a' : A \qquad \Gamma \vdash A \equiv A' : s}{\Gamma \vdash a \equiv a' : A'}$$

H $\mathbf{PTS}^{\mathcal{R}}_{fe}$

Inference rules for $\mathrm{PTS}^{\mathcal{R}}_{fe}$.

$$\textsc{Nil}\ \frac{}{\vdash \mathsf{wf}} \qquad \textsc{Cons}\ \frac{\Gamma \vdash A : s \qquad x \notin \mathrm{dom}(\Gamma)}{\Gamma, x : A \vdash \mathsf{wf}} \qquad \textsc{Sort}\ \frac{\Gamma \vdash \mathsf{wf} \qquad (s_1, s_2) \in \mathsf{Ax}}{\Gamma \vdash s_1 : s_2} \qquad \textsc{Var}\ \frac{\Gamma \vdash \mathsf{wf} \qquad \Gamma(x) = A}{\Gamma \vdash x : A}$$

$$\textsc{Prod}\ \frac{\Gamma \vdash A : s_1 \qquad \Gamma, x : A \vdash B : s_2 \qquad (s_1, s_2, s_3) \in \mathsf{Rel}}{\Gamma \vdash \Pi x : A.B : s_3} \qquad \textsc{Lam}\ \frac{\Gamma \vdash A : s_1 \qquad \Gamma, x : A \vdash b : B \qquad \Gamma, x : A \vdash B : s_2 \qquad (s_1, s_2, s_3) \in \mathsf{Rel}}{\Gamma \vdash \lambda x : A.b : \Pi x : A.B}$$

$$\textsc{App}\ \frac{\Gamma \vdash F : \Pi x : A.B \qquad \Gamma \vdash a : A}{\Gamma \vdash Fa : B[x := a]} \qquad \textsc{Conv}\ \frac{\Gamma \vdash a : A \qquad \Gamma \vdash H : A \equiv A' \bullet s}{\Gamma \vdash a^H : A'} \qquad \textsc{Refl}\ \frac{\Gamma \vdash M : A}{\Gamma \vdash \underline{M} : M \equiv M \bullet |A|}$$

$$\textsc{Sym}\ \frac{\Gamma \vdash H : M \equiv M' \bullet A}{\Gamma \vdash \overleftarrow{H} : M' \equiv M \bullet A} \qquad \textsc{Trans}\ \frac{\Gamma \vdash H : M \equiv M' \bullet A \qquad \Gamma \vdash H' : M' \equiv M'' \bullet A}{\Gamma \vdash H \cdot H' : M \equiv M'' \bullet A}$$

$$\textsc{Beta}\ \frac{\Gamma \vdash a : A \qquad \Gamma \vdash A : s_1 \qquad \Gamma, x : A \vdash b : B \qquad \Gamma, x : A \vdash B : s_2 \qquad (s_1, s_2, s_3) \in \mathsf{Rel}}{\Gamma \vdash \beta((\lambda x : A.b)\ a) : (\lambda x : A.b)\ a \equiv b[x := a]}$$

$$\textsc{Rewrite}\ \frac{x_1 : A_1, \ldots, x_k : A_k \vdash \ell : A \qquad \forall i,\ \Gamma \vdash N_i : A_i \qquad \ell \to r \in \mathcal{R}}{\Gamma \vdash \mathcal{R}(\ell \to r, \overrightarrow{N}) : \ell[\overrightarrow{x} := \overrightarrow{N}] \equiv r[\overrightarrow{x} := \overrightarrow{N}] \bullet A[\overrightarrow{x} := \overrightarrow{N}]}$$

$$\textsc{Prod-eq}\ \frac{\Gamma \vdash H : A \equiv A' \bullet s_1 \qquad \Gamma, x : A \vdash H' : B \equiv B'[x' := x^H] \bullet s_2 \qquad (s_1, s_2, s_3) \in \mathsf{Rel}}{\Gamma \vdash \{H, [x : A]H'\} : \Pi x : A.B \equiv \Pi x : A'.B' \bullet s_3}$$

$$\textsc{Lam-eq}\ \frac{\Gamma \vdash H : A \equiv A' \bullet s_1 \qquad \Gamma, x : A \vdash H' : b \equiv b'[x' := x^H] \bullet B \qquad \Gamma, x : A \vdash B : s_2 \qquad (s_1, s_2, s_3) \in \mathsf{Rel}}{\Gamma \vdash \langle H, [x : A]H' \rangle : \lambda x : A.b \equiv \lambda x' : A'.b' \bullet \Pi x : A.B}$$

$$\textsc{App-eq}\ \frac{\Gamma \vdash H : F \equiv F' \bullet \Pi x : A.B \qquad \Gamma \vdash H' : a \equiv a' \bullet A}{\Gamma \vdash HH' : Fa \equiv F'a' \bullet B[x := a]} \qquad \textsc{Iota}\ \frac{\Gamma \vdash a : A \qquad \Gamma \vdash H : A \equiv A' \bullet s}{\Gamma \vdash \iota(a^H) : a \equiv a^H \bullet |A|}$$

$$\textsc{Conv-eq}\ \frac{\Gamma \vdash H : M \equiv M' \bullet A \qquad \Gamma \vdash H' : A \equiv A' \bullet s}{\Gamma \vdash H^{H'} : M \equiv M' \bullet |A'|}$$