Equivalence of System F and $\lambda 2$ in Coq based on Context Morphism Lemmas

Jonas Kaiser, Tobias Tebbi, and Gert Smolka

November 29, 2016

We give a machine-checked proof of the equivalence of the usual, two-sorted presentation of System F and its single-sorted pure type system variant λ2. This is established by reducing the typability problem of F to λ2 and vice versa. The difficulty lies in aligning different binding-structures and different contexts (dependent vs. non-dependent). The use of de Bruijn syntax, parallel substitutions, and context morphism lemmas leads to an elegant proof. We use the Coq proof assistant and the substitution library Autosubst.

1 Introduction

There are different presentations of "System F" in the literature, and we often found that properties that have been proven for one presentation are assumed for another, without verifying that the presentations are in fact equivalent. A formalisation of such an equivalence proof is surprisingly intricate. The arising proof obligations become particularly involved when syntax has to be translated. Our goal is to demonstrate that a pure de Bruijn setup with parallel substitutions and the systematic use of context morphism lemmas lead to an elegant treatment of the hidden complexities. Parallel substitutions (i.e. multi-variable substitutions) enable clear and natural lemma statements [18], contrary to the usual criticism [3] of de Bruijn formalisations.

System F in its original form is due to Girard [12, 13], who introduced it in the context of proof theory. It was also independently discovered by Reynolds [16] as the polymorphic λ-calculus. Here we consider a two-sorted presentation, as given by Harper in [15], and a pure type system (PTS) variant λ2, closely related to the one found in Barendregt's λ-cube [5]. The respective type systems with de Bruijn binding are given in Figures 1 and 2. We provide a detailed discussion of these systems in Section 2.

In [11], Geuvers presents a proof sketch that valid typing judgements can be translated between these two presentations. Apart from that, we are not aware of other treatments of this correspondence. We improve upon Geuvers' result in two ways. First, we translate judgements

regardless of their derivability, thus obtaining full reductions of the decision problem of typability. Second, our proof is completely formalised with the Coq proof assistant[1].

Our main result (Theorem 31) essentially yields the following equivalences, where $\lfloor \cdot \rfloor$ and $\lceil \cdot \rceil$ are appropriate syntactic translations:

$$\vdash_{\mathsf{F}} s : A \iff \vdash_{2} \lfloor s \rfloor : \lfloor A \rfloor$$
$$\vdash_{2} a : b \iff \vdash_{\mathsf{F}} \lceil a \rceil : \lceil b \rceil$$

Proving this result turned out to be rather intricate. The two main difficulties are the use of different syntaxes, which necessitates syntactic translations, and the mismatch of the respective type systems. In a direct proof, these aspects interact, which significantly increases complexity. In order to avoid this, we introduce an auxiliary system P and decompose the proof into two steps separating the two concerns.

The intermediate system P is defined on the PTS syntax of $\lambda 2$, but equipped with a type system that closely mirrors F. In Section 5 we prove the equivalence of $\lambda 2$ and P, which is relatively straightforward due to their shared syntax. We then introduce syntactic translations in Section 6 and use these to state and prove the equivalence of P and F in Section 7. This part of the proof is involved as the translations $\lceil \cdot \rceil$ from the PTS syntax into the two sorts of F are necessarily partial.

We found that systematically phrasing all essential statements for both parts of the proof as context morphism lemmas [14, 2] leads to an elegant formalisation. In Sections 3 and 4 we introduce this proof technique.

We conclude with a discussion of related and future work in Sections 8 and 9.

In summary, our contributions are:

- A proof of the equivalence, via the reduction of the decision problem of typability, of two common presentations of F that is fully machine-checked with the proof assistant Coq.
- A case-study illustrating that working with pure de Bruijn syntax and parallel substitutions is not only possible but desirable as it leads to elegant theorem statements and proofs.
- The observation that context morphism lemmas are a powerful tool for relating syntaxes and type systems with different binding structures.

2 Preliminaries

Throughout this work we are going to deal with three variants of System F, all of which are given in a pure de Bruijn presentation. The first is a two-sorted presentation, as given by Harper [15] and shown in Figure 1. We are going to refer to this system simply as F. The second variant is the single-sorted PTS $\lambda 2$, shown in Figure 2, which is a close approximation of the one found in Barendregt's λ-cube [5]. The third variant, which we call P and present in

[1] The accompanying Coq development can be found at
`https://www.ps.uni-saarland.de/extras/cpp17-sysf/`

$$A, B, C \ ::= \ x_{\mathsf{ty}} \mid A \to B \mid \forall. A \qquad\qquad x : \mathbb{N}$$
$$s, t \ ::= \ x_{\mathsf{ter}} \mid s\ t \mid \lambda A. s \mid s\ A \mid \Lambda. s \qquad\qquad x : \mathbb{N}$$

$$\frac{x < N}{N \vdash^{\mathsf{ty}}_{\mathsf{F}} x_{\mathsf{ty}}} \qquad \frac{N \vdash^{\mathsf{ty}}_{\mathsf{F}} A \qquad N \vdash^{\mathsf{ty}}_{\mathsf{F}} B}{N \vdash^{\mathsf{ty}}_{\mathsf{F}} A \to B} \qquad \frac{N + 1 \vdash^{\mathsf{ty}}_{\mathsf{F}} A}{N \vdash^{\mathsf{ty}}_{\mathsf{F}} \forall. A}$$

$$\frac{A_x = A \qquad N \vdash^{\mathsf{ty}}_{\mathsf{F}} A}{N; A_n, \ldots, A_0 \vdash^{\mathsf{ter}}_{\mathsf{F}} x_{\mathsf{ter}} : A} \qquad \frac{N; \Gamma, A \vdash^{\mathsf{ter}}_{\mathsf{F}} s : B \qquad N \vdash^{\mathsf{ty}}_{\mathsf{F}} A}{N; \Gamma \vdash^{\mathsf{ter}}_{\mathsf{F}} \lambda A. s : A \to B} \qquad \frac{N; \Gamma \vdash^{\mathsf{ter}}_{\mathsf{F}} s : A \to B \qquad N; \Gamma \vdash^{\mathsf{ter}}_{\mathsf{F}} t : A}{N; \Gamma \vdash^{\mathsf{ter}}_{\mathsf{F}} s\ t : B}$$

$$\frac{N; \Gamma \vdash^{\mathsf{ter}}_{\mathsf{F}} s : \forall. A \qquad N \vdash^{\mathsf{ty}}_{\mathsf{F}} B}{N; \Gamma \vdash^{\mathsf{ter}}_{\mathsf{F}} s\ B : A[B \cdot \mathsf{id}]} \qquad \frac{N + 1; \Gamma[+1] \vdash^{\mathsf{ter}}_{\mathsf{F}} s : A}{N; \Gamma \vdash^{\mathsf{ter}}_{\mathsf{F}} \Lambda. s : \forall. A}$$

Figure 1: System F – Two-sorted de Bruijn syntax, type formation and typing rules.

$$u \ ::= \ * \mid \Box$$
$$a, b, c, d \ ::= \ u \mid x \mid a\ b \mid \lambda a. b \mid \Pi a. b \qquad\qquad x : \mathbb{N}$$

$$\frac{}{0 : a[+1] \in \Gamma, a} \qquad \frac{x : a \in \Gamma}{(x + 1) : a[+1] \in \Gamma, b}$$

$$\frac{}{\Gamma \vdash_2 * : \Box} \qquad \frac{x : a \in \Gamma \qquad \Gamma \vdash_2 a : u}{\Gamma \vdash_2 x : a}$$

$$\frac{\Gamma \vdash_2 a : u \qquad \Gamma, a \vdash_2 b : *}{\Gamma \vdash_2 \Pi a. b : *} \qquad \frac{\Gamma \vdash_2 a : \Pi c. d \qquad \Gamma \vdash_2 b : c}{\Gamma \vdash_2 a\ b : d[b \cdot \mathsf{id}]} \qquad \frac{\Gamma \vdash_2 a : u \qquad \Gamma, a \vdash_2 b : c \qquad \Gamma, a \vdash_2 c : *}{\Gamma \vdash_2 \lambda a. b : \Pi a. c}$$

Figure 2: PTS – Single-sorted de Bruijn syntax, dependent context lookup, and typing rules of $\lambda 2$.

$$\frac{x : * \in \Gamma}{\Gamma \vdash^{\mathsf{ty}}_{\mathsf{P}} x} \qquad \frac{\Gamma \vdash^{\mathsf{ty}}_{\mathsf{P}} a \qquad \Gamma, a \vdash^{\mathsf{ty}}_{\mathsf{P}} b}{\Gamma \vdash^{\mathsf{ty}}_{\mathsf{P}} \Pi a.\, b} \qquad \frac{\Gamma, * \vdash^{\mathsf{ty}}_{\mathsf{P}} a}{\Gamma \vdash^{\mathsf{ty}}_{\mathsf{P}} \Pi *.\, a}$$

$$\frac{x : a \in \Gamma \qquad \Gamma \vdash^{\mathsf{ty}}_{\mathsf{P}} a}{\Gamma \vdash^{\mathsf{ter}}_{\mathsf{P}} x : a}$$

$$\frac{\Gamma \vdash^{\mathsf{ty}}_{\mathsf{P}} a \qquad \Gamma, a \vdash^{\mathsf{ter}}_{\mathsf{P}} b : c}{\Gamma \vdash^{\mathsf{ter}}_{\mathsf{P}} \lambda a.\, b : \Pi a.\, c} \qquad \frac{\Gamma \vdash^{\mathsf{ter}}_{\mathsf{P}} a : \Pi c.\, d \qquad \Gamma \vdash^{\mathsf{ter}}_{\mathsf{P}} b : c}{\Gamma \vdash^{\mathsf{ter}}_{\mathsf{P}} a\, b : d[b \cdot \mathsf{id}]} \qquad \frac{\Gamma, * \vdash^{\mathsf{ter}}_{\mathsf{P}} a : b}{\Gamma \vdash^{\mathsf{ter}}_{\mathsf{P}} \lambda *.\, a : \Pi *.\, b}$$

$$\frac{\Gamma \vdash^{\mathsf{ter}}_{\mathsf{P}} a : \Pi *.\, b \qquad \Gamma \vdash^{\mathsf{ty}}_{\mathsf{P}} c}{\Gamma \vdash^{\mathsf{ter}}_{\mathsf{P}} a\, c : b[c \cdot \mathsf{id}]}$$

Figure 3: PTS – Auxiliary type system P.

Figure 3, acts as an intermediary for the equivalence proof of λ2 and F:

$$\lambda 2 \overset{(A)}{\iff} \mathrm{P} \overset{(B)}{\iff} \mathrm{F}$$

This proof structure governs the design of P, taking into account the two major differences of λ2 and F:

1. The syntax of F is two-sorted while λ2 is formulated with single-sorted PTS syntax.
2. The type system of F has separate type formation and typing judgements. λ2 only has a single typing judgement. Further structural differences of the type systems are a consequence of the uniformity of PTS syntax.

We thus define P on the PTS syntax of λ2, but its type system is aligned with that of F. Accordingly, part (A) of the proof (Section 5) handles the alignment of the type systems, while part (B) (Sections 6 and 7) deals with the translation of syntax.

We now introduce the three systems in more detail. We start with two-sorted F. Before we consider the particular aspects of de Bruijn syntax, let us first take a look at the named presentation. Its syntax is defined as

$$\begin{aligned} A, B, C &::= X \mid A \to B \mid \forall X.\, A \\ s, t &::= x \mid s\, t \mid \lambda x : A.\, s \mid s\, A \mid \Lambda X.\, s \end{aligned}$$

where the type variables X and the term variables x are taken from disjoint universes.

This language can express the polymorphic identity,

$$\Lambda X.\, \lambda x : X.\, x$$

and its universally quantified, or polymorphic, type:

$$\forall X.\, X \to X$$

$$\frac{X \in \Delta}{\Delta \vdash^{\mathsf{ty}}_{\mathsf{F}} X} \qquad \frac{\Delta \vdash^{\mathsf{ty}}_{\mathsf{F}} A \qquad \Delta \vdash^{\mathsf{ty}}_{\mathsf{F}} B}{\Delta \vdash^{\mathsf{ty}}_{\mathsf{F}} A \to B} \qquad \frac{\Delta, X \vdash^{\mathsf{ty}}_{\mathsf{F}} A}{\Delta \vdash^{\mathsf{ty}}_{\mathsf{F}} \forall X.\, A}\ X \notin \Delta \qquad \frac{\Gamma(x) = A \qquad \Delta \vdash^{\mathsf{ty}}_{\mathsf{F}} A}{\Delta;\Gamma \vdash^{\mathsf{ter}}_{\mathsf{F}} x : A}$$

$$\frac{\Delta;\Gamma \vdash^{\mathsf{ter}}_{\mathsf{F}} s : A \to B \qquad \Delta;\Gamma \vdash^{\mathsf{ter}}_{\mathsf{F}} t : A}{\Delta;\Gamma \vdash^{\mathsf{ter}}_{\mathsf{F}} s\ t : B} \qquad \frac{\Delta;\Gamma, x : A \vdash^{\mathsf{ter}}_{\mathsf{F}} s : B \qquad \Delta \vdash^{\mathsf{ty}}_{\mathsf{F}} A}{\Delta;\Gamma \vdash^{\mathsf{ter}}_{\mathsf{F}} \lambda x : A.\, s : A \to B}\ x \notin \mathrm{dom}(\Gamma)$$

$$\frac{\Delta;\Gamma \vdash^{\mathsf{ter}}_{\mathsf{F}} s : \forall X.\, A \qquad \Delta \vdash^{\mathsf{ty}}_{\mathsf{F}} B}{\Delta;\Gamma \vdash^{\mathsf{ter}}_{\mathsf{F}} s\ B : A[B/X]} \qquad \frac{\Delta, X;\Gamma \vdash^{\mathsf{ter}}_{\mathsf{F}} s : A}{\Delta;\Gamma \vdash^{\mathsf{ter}}_{\mathsf{F}} \Lambda X.\, s : \forall X.\, A}\ X \notin \Delta$$

Figure 4: System F – Named presentation.

Observe how implications are taken as function spaces for ordinary abstractions, while universal quantifications are used to type abstractions over type variables.

The type system consists of a **type formation** judgement that derives well-formed types, e.g.,

$$\vdash^{\mathsf{ty}}_{\mathsf{F}} \forall X.\, X \to X$$

and a **typing** judgement that assigns types to terms:

$$\vdash^{\mathsf{ter}}_{\mathsf{F}} \Lambda X.\, \lambda x : X.\, x : \forall X.\, X \to X$$

The full type system is given in Figure 4. We consider a presentation with explicit type variable contexts Δ. These are simply sets of type variables that are allowed to appear freely. Term variable contexts Γ are mappings from variables to types. If necessary, we are going to explicitly denote the empty context with $\langle\rangle$.

As an example, consider the following valid typing with non-empty contexts:

$$A; f : \forall Y.\, Y \to Y, z : A \vdash^{\mathsf{ter}}_{\mathsf{F}} f\ A\ z : A$$

In order to highlight certain aspects of the various discussed systems, we are going to revisit this example throughout this section.

Note how we have taken the liberty to express, in this example, the type of the polymorphic identity as

$$\forall Y.\, Y \to Y$$

That is, we have relabelled the bound identifier from X to Y. The two types are not equal but α-equivalent, which is sufficient for an informal treatment. For a formalisation though, equality is a lot nicer to work with. This is where de Bruijn syntax comes into play.

We observe that variables are essentially references, either to binders of the same expression or into the surrounding context. In the de Bruijn setting these references are implemented

as natural numbers, called **indices**. The index n refers to the n-th enclosing binder of the corresponding variable scope, counting from 0. In the following we write x, y, z for such de Bruijn indices. Note that a variable x is **free** in a term s if $x + k$ occurs in s under k binders.

The de Bruijn presentation of the two-sorted F introduced above is shown in Figure 1. All binders, that is λ, Λ and $\forall$, are nameless[2] and both type and term variables are given as natural numbers $x : \mathbb{N}$. Note that the A in $\lambda A.\, s$ is the type of the bound variable, not its name. We express the typing of the polymorphic identity as

$$\vdash^{\mathsf{ter}}_{\mathsf{F}} \Lambda.\, \lambda 0_{\mathsf{ty}}.\, 0_{\mathsf{ter}} : \forall.\, 0_{\mathsf{ty}} \to 0_{\mathsf{ty}}$$

and our running example becomes

$$1; (\forall.\, 0_{\mathsf{ty}} \to 0_{\mathsf{ty}}), 0_{\mathsf{ty}} \vdash^{\mathsf{ter}}_{\mathsf{F}} 1_{\mathsf{ter}}\, 0_{\mathsf{ty}}\, 0_{\mathsf{ter}} : 0_{\mathsf{ty}}$$

This demonstrates how the removal of names affects the contexts. The type variable context Δ degenerates to a plain natural number $N : \mathbb{N}$. It is a strict upper bound on the indices of freely occurring type variables. Thus $N = 0$ would yield the empty context. In our case we represent the free type variable A as 0_{ty} and have $N = 1$ to enforce that it is in fact the only one.

In the de Bruijn setting, term variable contexts Γ are reduced from maps to plain lists of types. Free term variables are taken as indices into these contexts. Note that, to conform to the usual orientation of contexts, we count positions from right to left. Thus the free occurrence of 0_{ter} in our example references the rightmost element of the context. That is, we have $0_{\mathsf{ter}} : 0_{\mathsf{ty}}$.

A core aspect of the de Bruijn setting is that terms are not invariant under context modifications. Whenever binders are added or removed, or when a new element is added to a context, then the indices of all free variables have to be adjusted. An elegant way to handle these adjustments is to use parallel substitutions, which act on all free variables at once. The idea dates back to de Bruijn's original presentation in [9] and a detailed explanation can be found in [1, 17].

We will introduce parallel substitutions for the types of F. The construction is identical for PTS terms. For the terms of F we have to take into account that they may also contain free type variables. This results in the definition of $s[\tau, \sigma]$, which simultaneously applies a type substitution τ to all free type variables of s and a term substitution σ to all free term variables of s.

All forms of substitution are automatically generated through the Autosubst framework, which also provides tools to efficiently work with the resulting definitions. We direct the reader to [18] for further details.

For the types of F, **parallel substitutions**, written σ, τ, are functions from variables to types. Two particular functions are considered as primitive substitutions, namely the **identity**

[2] We chose to preserve the dot in the notation as it is a uniform indicator for the presence of a binder. More importantly, it marks the precise position where de Bruijn indices change.

substitution id and the **shift substitution** $+1$, defined as:

$$\begin{aligned} \mathsf{id}(x_{\mathsf{ty}}) &:= x_{\mathsf{ty}} \\ +1(x_{\mathsf{ty}}) &:= (x+1)_{\mathsf{ty}} \end{aligned}$$

These also constitute examples of **renamings**[3], written ξ, ζ, that is, substitutions that map variables to variables. We further observe that a substitution σ can be viewed as the infinite stream $\sigma(0), \sigma(1), \ldots$ of types. This motivates the definition of a **cons** operation

$$A \cdot \sigma := A, \sigma(0), \sigma(1), \ldots$$

which allows us to, e.g., succinctly define $-1 := 0_{\mathsf{ty}} \cdot \mathsf{id}$ as a right inverse of $+1$. Note that $B \cdot \mathsf{id}$ is the correct de Bruijn form of β-substitution, mapping 0_{ty} to B and reducing all other indices by 1.

We then define the **application** of a substitution to a type, written $A[\sigma]$, mutually recursive with the **forward composition** of two substitutions, written $\sigma \circ \tau$, as follows

$$\begin{aligned} x_{\mathsf{ty}}[\sigma] &:= \sigma(x) & \qquad (\sigma \circ \tau) x_{\mathsf{ty}} &:= \sigma(x)[\tau] \\ (A \to B)[\sigma] &:= A[\sigma] \to B[\sigma] \\ (\forall . A)[\sigma] &:= \forall . A[\Uparrow \sigma] & \qquad \Uparrow \sigma &:= 0_{\mathsf{ty}} \cdot \sigma \circ +1 \end{aligned}$$

Note that we give composition higher precedence than cons, that is, $A \cdot \sigma \circ \tau$ should be read as $A \cdot (\sigma \circ \tau)$.

Let us now take a closer look at the PTS syntax[4] defined in Figure 2. Here we only have a single syntactic sort of terms, in contrast to the distinguished sorts of types and terms we have seen for F.

We summarise the basic structure. Abstractions, $\lambda a.\, b$, live in **dependent products**, $\Pi c.\, d$, which form the function spaces and live in the universe $*$. The hierarchy is capped with the top universe $\Box$ which has $*$ as its only inhabitant. In short, the stratification into terms and types[5] is internalised as

$$a : b : * : \Box$$

The PTS syntax is **uniform**. That is, implication and universal quantification are both expressed as dependent products. We can distinguish the two concepts based on whether the argument type is $*$ or lives in $*$. This design also extends to abstractions and applications.

Since we do not syntactically distinguish types from terms, we also only have a single class of variables. This heavily affects the definition of contexts. For F we have seen that

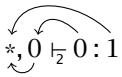
[3] Note that we do not require renamings to be injective or surjective. For example, $+1$ is not surjective.

[4] We opted for the somewhat non-standard syntactic term alphabet a, b, c, d to clearly distinguish it from both the types of F (A, B, C) as well as its terms (s, t). All of these entities will later appear in close proximity.

[5] Since the distinction of terms and types only arises through the typing relation, we refrain from partitioning the meta-variables $a, b, c, \ldots$ accordingly and simply assign them in alphabetical order.

a term variable context Γ is a list of types, which may have free type variables. These, in turn, referenced a type variable context Δ. For PTSs, these two context levels are merged to form a single self-referential, **dependent** context. Take for example the context Γ_1, a, Γ_2. Free variables in a index into Γ_1. As a consequence, free variables have to be adjusted when we extract them from a context. We achieve this by inductively defining the lookup operation $x : a \in \Gamma$ via the first two rules in Figure 2.

We illustrate the binding structure with respect to dependent contexts with a simple judgement (the named variant on the right is provided for reference):



$$*, 0 \vdash_2 0 : 1 \qquad\qquad X; x : X \vdash^{\mathsf{ter}}_{\mathsf{F}} x : X$$

Our primary type system for the PTS syntax and its dependent contexts is $\lambda 2$, shown in Figure 2. The rules of $\lambda 2$ are very close to the usual presentation of PTSs [5, 2]. That is, there is only a single judgement with one rule for each syntactic construct and type formation is internalised with judgements of the form $\Gamma \vdash_2 a : *$.

Since dependent de Bruijn contexts are constructed in such a way that cyclic dependencies or duplicate entries for a single variable are impossible we do not require a well-formedness judgement for contexts. In order to still obtain a reasonable typing judgement, we ensure well-formedness for every term added to, or extracted from, the context in a typing derivation.

A further difference of our presentation of $\lambda 2$ and the one found in [5] is the omission of the conversion rule. This is justified since for $\lambda 2$ it has no nontrivial applications: well-formed types do not contain β-redices.

In $\lambda 2$ we can express our running example as

$$*, (\Pi *. \Pi 0. 1), 1 \vdash_2 1\, 2\, 0 : 2$$

When we compare this to the F version above, it becomes clear that the correct mapping of the indices is going to be a major challenge of the syntax translation.

We conclude with a few remarks regarding P (Figure 3). In order to bridge the gap between $\lambda 2$ and F it utilises the PTS syntax of the former, but its type system mirrors the latter and is thus equipped with a separate type formation judgement. Further note that there are two abstraction rules and two application rules which correspond to the syntactically different abstraction and application mechanisms we have introduced for F. Our running example in P is identical to the $\lambda 2$ version, with $(\vdash_2)$ replaced by $(\vdash^{\mathsf{ter}}_{\mathsf{P}})$.

3 Context Morphism Lemmas

Pure de Bruijn syntax and parallel substitutions are a natural fit [9, 18, 17]. When it comes to reasoning about judgements of such de Bruijn style systems, context morphism lemmas (CMLs) elegantly complete the picture.

The purpose of a CML is to handle the situation where one valid judgement is transformed into another, either of the same system, or possibly another. The admissibility proofs of such

transformations usually involve an induction over the derivation of the initial judgement. For this induction to go through it is often necessary to devise a non-trivial generalisation of the original statement.

The intuition behind CMLs is to decompose these proofs into the base case that deals with typable variables and the inductive lifting to arbitrary typable terms. The benefit of this decomposition is twofold. Firstly, the variable case alone is often easy to handle. Secondly, when the variable condition is sufficiently general, the inductive lifting to arbitrary terms is usually rather mechanic. The CML itself can then be seen as a reduction of the original statement to one that only talks about variables.

To illustrate this idea, we are going to prove the admissibility of **weakening** for P:

$$\Gamma \vdash^{\mathsf{ter}}_{\mathsf{P}} a : b \quad \Longrightarrow \quad \Gamma, c \vdash^{\mathsf{ter}}_{\mathsf{P}} a[{+}1] : b[{+}1]$$

As it turns out, it is surprisingly tricky to prove this directly with an induction over the initial judgement. We have to be able to insert the new context element c at an arbitrary position of the context, not just at the head position. The substitution $+1$ needs to be modified to reflect this generalisation as well. Moreover, the context Γ is dependent. That is, we require a generalisation of the following form:

$$\Gamma_1, \Gamma_2 \vdash^{\mathsf{ter}}_{\mathsf{P}} a : b \quad \Longrightarrow \quad \Gamma_1, c, \Gamma_2' \vdash^{\mathsf{ter}}_{\mathsf{P}} a[\xi] : b[\xi]$$

where ξ is a suitable renaming that reflects the context rearrangement and Γ_2' is a suitably adjusted context. A concrete ξ can be given, and Γ_2' can be implemented as a certain fold over Γ_2, but the resulting proof is intricate. Taking into consideration that we still aim for a relatively simple property, these ad hoc technicalities are rather unsatisfactory.

So let us now backtrack and imagine that we instead had the following CML at our disposal:

$$\frac{\forall x b.\ \ \Gamma \vdash^{\mathsf{ter}}_{\mathsf{P}} x : b \quad \Longrightarrow \quad \Delta \vdash^{\mathsf{ter}}_{\mathsf{P}} x[\sigma] : b[\sigma]}{\Gamma \vdash^{\mathsf{ter}}_{\mathsf{P}} a : b \quad \Longrightarrow \quad \Delta \vdash^{\mathsf{ter}}_{\mathsf{P}} a[\sigma] : b[\sigma]}$$

Note how the proof obligation to show the result for all terms a typable under Γ has been reduced to only those variables x typable under Γ. Now weakening follows from

$$\Gamma \vdash^{\mathsf{ter}}_{\mathsf{P}} x : b \quad \Longrightarrow \quad \Gamma, c \vdash^{\mathsf{ter}}_{\mathsf{P}} x[{+}1] : b[{+}1]$$

which is straightforward to prove.

4 CMLs for P

In the previous section we introduced, without proof, a CML for the typing relation of P. We glossed over several technicalities and we completely ignored the fact that we require a CML for type formation before we can properly deal with typing. We are going to rectify this here and provide formal proofs of the CMLs for both type formation and typing.

The CML for type formation in P looks like this:

$$\frac{\forall x.\ \ \Gamma \vdash^{\mathsf{ty}}_{\mathsf{P}} x \ \ \Longrightarrow \ \ \Delta \vdash^{\mathsf{ty}}_{\mathsf{P}} x[\sigma]}{\Gamma \vdash^{\mathsf{ty}}_{\mathsf{P}} a \ \ \Longrightarrow \ \ \Delta \vdash^{\mathsf{ty}}_{\mathsf{P}} a[\sigma]}$$

To organise its proof, let us first introduce a notation for the premise, that is, the variable condition:

$$\vdash^{\mathsf{ty}}_{\mathsf{P}} \sigma : \Gamma \to \Delta \ \ := \ \ \forall x.\ \ \Gamma \vdash^{\mathsf{ty}}_{\mathsf{P}} x \ \ \Longrightarrow \ \ \Delta \vdash^{\mathsf{ty}}_{\mathsf{P}} \sigma(x)$$

The notation alludes to the fact that this assigns a type to the substitution σ. Concretely, σ can be seen as a type formation preserving morphism from initial context Γ to final context Δ. For this reason σ is called a **context morphism**[6] from Γ to Δ. Observe, how the appropriate notion of context morphism is determined by the CML.

The proof of the CML will proceed by induction on $\Gamma \vdash^{\mathsf{ty}}_{\mathsf{P}} a$. This raises two issues. Firstly, $\vdash^{\mathsf{ty}}_{\mathsf{P}} \sigma : \Gamma \to \Delta$ acts as a proof invariant. Since the binder rules extend the context we have to ensure that $\vdash^{\mathsf{ty}}_{\mathsf{P}} \sigma : \Gamma \to \Delta$ is preserved under such context extensions.

The bigger problem, though, is that the induction on the derivation follows the termination order of the type structure, rather than that for the application of substitutions to types. The latter is the one we need as the former is going to produce a circularity in the proof. We achieve this with an idea borrowed from [2] and [18]: before proving the CML for arbitrary substitutions σ, we will first prove it for the special case of renamings ξ. This fixes the issue of the termination order at the cost of duplicating proof structures.

As we will see shortly, though, interesting properties can already be obtained from the renaming version. Moreover, it is sometimes not even necessary to generalise to arbitrary substitutions. We will exploit this in the later parts of our work.

So we begin with renamings. We are going to add a second renaming ρ to the initial context, in order to later be able to also obtain a strengthening result. Our definition of a context morphism thus is

$$\vdash^{\mathsf{ty}}_{\mathsf{P}} \rho/\xi : \Gamma \to \Delta \ \ := \ \ \forall x.\ \ \Gamma \vdash^{\mathsf{ty}}_{\mathsf{P}} \rho(x) \ \ \Longrightarrow \ \ \Delta \vdash^{\mathsf{ty}}_{\mathsf{P}} \xi(x)$$

Before we continue it is worth noting that the addition of ρ leads to a non-standard form of context morphism. That is, the ρ cannot be generalised to arbitrary substitutions. For arbitrary substitutions we would lose the crucial property that the quantification is over variable typings. Fortunately, we only need strengthening for P type formation. Thus the present CML is the only one that has to deal with the added complexity of a second renaming. All other CMLs presented throughout this work will use standard context morphism definitions.

The first step of our present CML proof, then, is to establish that $\vdash^{\mathsf{ty}}_{\mathsf{P}} \rho/\xi : \Gamma \to \Delta$ is closed under context extensions.

[6] Context morphisms are written as $\sigma : \Delta \to \Gamma$ (note the inverted order of contexts) in [14] and correspond to Adams' [2] satisfaction relation $\Delta \vDash \sigma :: \Gamma$.

Lemma 1

$$\frac{\vdash^{\mathsf{tv}}_{\mathsf{P}} \rho / \xi : \Gamma \to \Delta}{\vdash^{\mathsf{tv}}_{\mathsf{P}} \Uparrow \rho / \Uparrow \xi : \Gamma, c[\rho] \to \Delta, c[\xi]}$$

PROOF Assume $\vdash^{\mathsf{tv}}_{\mathsf{P}} \rho / \xi : \Gamma \to \Delta$ and also $\Gamma, c[\rho] \vdash^{\mathsf{tv}}_{\mathsf{P}} \Uparrow \rho(x)$ for some variable x. We have to show $\Delta, c[\xi] \vdash^{\mathsf{tv}}_{\mathsf{P}} \Uparrow \xi(x)$. Now discriminate on whether $x = 0$ or $x = 1 + y$ for some y. Let $x = 0$, then we have $c[\rho][+1] = *$, that is $c = *$. Our proof obligation becomes $\Delta, * \vdash^{\mathsf{tv}}_{\mathsf{P}} 0$, which trivially holds. Let now $x = 1 + y$ for some y. Then we know $\Gamma \vdash^{\mathsf{tv}}_{\mathsf{P}} \rho(y)$ and from the morphism assumption it follows that $\Delta \vdash^{\mathsf{tv}}_{\mathsf{P}} \xi(y)$. Now since ξ is a renaming, $\xi(y)$ is a variable. We can thus use the variable rule and the definition of variable lookup to obtain $\Delta, c[\xi] \vdash^{\mathsf{tv}}_{\mathsf{P}} \xi(y)[+1]$ without resorting to any notion of weakening. Since $\xi \circ +1 = +1 \circ \Uparrow \xi$, this finishes the proof. ∎

With this we can now prove the renaming variant of our CML and obtain both weakening and strengthening as special cases. Note that we have slightly rearranged the premises to more closely reflect that the CML can be seen as an admissible rule for P.

Lemma 2 (Renaming CML for P Type Formation)

$$\frac{\Gamma \vdash^{\mathsf{tv}}_{\mathsf{P}} a[\rho] \qquad \vdash^{\mathsf{tv}}_{\mathsf{P}} \rho / \xi : \Gamma \to \Delta}{\Delta \vdash^{\mathsf{tv}}_{\mathsf{P}} a[\xi]}$$

PROOF By induction on $\Gamma \vdash^{\mathsf{tv}}_{\mathsf{P}} a[\rho]$, using Lemma 1 for the binder cases. ∎

Theorem 3 (Weakening for P Type Formation)

$$\frac{\Gamma \vdash^{\mathsf{tv}}_{\mathsf{P}} a}{\Gamma, c \vdash^{\mathsf{tv}}_{\mathsf{P}} a[+1]}$$

PROOF Instantiate Lemma 2. It is straightforward to show that $\vdash^{\mathsf{tv}}_{\mathsf{P}} \mathsf{id} / +1 : \Gamma \to \Gamma, c$. ∎

Theorem 4 (Strengthening for P Type Formation)

$$\frac{\Gamma, c \vdash^{\mathsf{tv}}_{\mathsf{P}} a[+1]}{\Gamma \vdash^{\mathsf{tv}}_{\mathsf{P}} a}$$

PROOF Instantiate Lemma 2. It is straightforward to show that $\vdash^{\mathsf{tv}}_{\mathsf{P}} +1 / \mathsf{id} : \Gamma, c \to \Gamma$. ∎

While it is possible to obtain weakening and strengthening from the renaming CML, we will later require β-substitutivity for type formation in P. This, in turn, is a consequence of the full CML, which we are going to prove next. Note that we can, at this point, switch back to the standard context morphism definition.

The first step is again the closure of $\vdash^{\mathsf{tv}}_{\mathsf{P}} \sigma : \Gamma \to \Delta$ under context extensions which then allows us to prove the CML, as well as its specific instances for β-substitutivity.

Lemma 5

$$\frac{\vdash^{\mathsf{ty}}_{\mathsf{P}} \sigma : \Gamma \to \Delta}{\vdash^{\mathsf{ty}}_{\mathsf{P}} \Uparrow\sigma : \Gamma, c \to \Delta, c[\sigma]}$$

Proof The proof mostly follows that of Lemma 1, albeit without the added complexity of a second substitution. The interesting step is the one from $\Delta \vdash^{\mathsf{ty}}_{\mathsf{P}} \sigma(y)$ to $\Delta, c[\sigma] \vdash^{\mathsf{ty}}_{\mathsf{P}} \sigma(y)[+1]$, which now relies on weakening (Theorem 3), since $\sigma(y)$ is an arbitrary term. $+1 \circ \Uparrow\sigma = \sigma \circ +1$ holds for arbitrary substitutions. ∎

Lemma 6 (CML for P Type Formation)

$$\frac{\Gamma \vdash^{\mathsf{ty}}_{\mathsf{P}} a \qquad \vdash^{\mathsf{ty}}_{\mathsf{P}} \sigma : \Gamma \to \Delta}{\Delta \vdash^{\mathsf{ty}}_{\mathsf{P}} a[\sigma]}$$

Proof By induction on $\Gamma \vdash^{\mathsf{ty}}_{\mathsf{P}} a$, using Lemma 5 for the binder cases. ∎

Theorem 7 (β-Substitutivity for P Type Formation)

$$\frac{\Gamma, d \vdash^{\mathsf{ty}}_{\mathsf{P}} a \qquad \Gamma \vdash^{\mathsf{ter}}_{\mathsf{P}} c : d \qquad \Gamma \vdash^{\mathsf{ty}}_{\mathsf{P}} d}{\Gamma \vdash^{\mathsf{ty}}_{\mathsf{P}} a[c \cdot \mathsf{id}]} \qquad \frac{\Gamma, * \vdash^{\mathsf{ty}}_{\mathsf{P}} a \qquad \Gamma \vdash^{\mathsf{ty}}_{\mathsf{P}} c}{\Gamma \vdash^{\mathsf{ty}}_{\mathsf{P}} a[c \cdot \mathsf{id}]}$$

Proof Both follow from Lemma 6. It is straightforward to show that $\Gamma \vdash^{\mathsf{ty}}_{\mathsf{P}} d$ and $\Gamma \vdash^{\mathsf{ter}}_{\mathsf{P}} c : d$ entail $\vdash^{\mathsf{ty}}_{\mathsf{P}} c \cdot \mathsf{id} : \Gamma, d \to \Gamma$. Similarly, $\Gamma \vdash^{\mathsf{ty}}_{\mathsf{P}} c$ entails $\vdash^{\mathsf{ty}}_{\mathsf{P}} c \cdot \mathsf{id} : \Gamma, * \to \Gamma$. ∎

This completes the treatment of type formation for P. Let us next consider the CML for typing. The proof structure is very similar, so we will only present the renaming version to obtain weakening. The full CML can be found in the formalisation. It is, however, not relevant for the remainder of this work.

We again define a suitable notion of context morphism:

$$\vdash^{\mathsf{ter}}_{\mathsf{P}} \sigma : \Gamma \to \Delta \;:=\; \forall x b.\;\; \Gamma \vdash^{\mathsf{ter}}_{\mathsf{P}} x : b \implies \Delta \vdash^{\mathsf{ter}}_{\mathsf{P}} \sigma(x) : b[\sigma]$$

We show that this is closed under context extensions, then prove the CML and obtain weakening as a special case.

Lemma 8

$$\frac{\vdash^{\mathsf{ter}}_{\mathsf{P}} \xi : \Gamma \to \Delta \qquad \vdash^{\mathsf{ty}}_{\mathsf{P}} \mathsf{id}/\xi : \Gamma \to \Delta}{\vdash^{\mathsf{ter}}_{\mathsf{P}} \Uparrow\xi : \Gamma, c \to \Delta, c[\xi]}$$

Proof Assume $\vdash^{\mathsf{ter}}_{\mathsf{P}} \xi : \Gamma \to \Delta$ and also $\Gamma, c \vdash^{\mathsf{ter}}_{\mathsf{P}} x : d$ for some variable x. We have to show $\Delta, c[\xi] \vdash^{\mathsf{ter}}_{\mathsf{P}} \Uparrow\xi(x) : d[\Uparrow\xi]$. Now discriminate on whether $x = 0$ or $x = 1 + y$ for some y. Let $x = 0$, then we have $d = c[+1]$ and our proof obligation becomes $\Delta, c[\xi] \vdash^{\mathsf{ter}}_{\mathsf{P}} 0 : c[\xi][+1]$, which trivially holds. Let now $x = 1 + y$ for some y. Then we know $\Gamma \vdash^{\mathsf{ter}}_{\mathsf{P}} y : b$ with $d = b[+1]$. Since ξ is a context morphism, it follows that $\Delta \vdash^{\mathsf{ter}}_{\mathsf{P}} \xi(y) : b[\xi]$. Now since ξ is a renaming, $\xi(y)$ is a variable. We can thus use the variable rule and the definition of context lookup, as well as strengthening (Theorem 4) and weakening (Theorem 3) for type formation to obtain $\Delta, c[\xi] \vdash^{\mathsf{ter}}_{\mathsf{P}} \xi(y)[+1] : b[\xi][+1]$. Some trivial rearrangement of renamings yields our goal. ∎

Lemma 9 (Renaming CML for P Typing)

$$\frac{\Gamma \vdash^{\mathsf{ter}}_{\mathsf{P}} a : b \qquad \vdash^{\mathsf{ty}}_{\mathsf{P}} \mathsf{id}/\xi : \Gamma \to \Delta \qquad \vdash^{\mathsf{ter}}_{\mathsf{P}} \xi : \Gamma \to \Delta}{\Delta \vdash^{\mathsf{ter}}_{\mathsf{P}} a[\xi] : b[\xi]}$$

PROOF By induction on $\Gamma \vdash^{\mathsf{ter}}_{\mathsf{P}} a : b$, using Lemma 8 for the binder cases. ∎

Theorem 10 (Weakening for P)

$$\frac{\Gamma \vdash^{\mathsf{ter}}_{\mathsf{P}} a : b}{\Gamma, c \vdash^{\mathsf{ter}}_{\mathsf{P}} a[+1] : b[+1]}$$

PROOF Instantiate Lemma 9. The proof of $\vdash^{\mathsf{ter}}_{\mathsf{P}} +1 : \Gamma \to \Gamma, c$ is straightforward. ∎

The proof of each CML we are going to encounter throughout the remainder of this work will follow along the same lines.

5 Equivalence of $\lambda 2$ and P

We are now going to prove the first half of our equivalence result, that is, the equivalence of $\lambda 2$ and P. This is the easier part as both systems use the same PTS syntax. There are three main issues we have to consider:

1. We have to separate the single typing relation of $\lambda 2$ into the two relations, typing and type formation, of P.
2. The type system of $\lambda 2$ is aligned with the uniform PTS syntax. That is, there is only one abstraction and one application rule. P, on the other hand, distinguishes type and term level abstraction and application with separate rules.
3. Some of $\lambda 2$'s typing rules have more premises than their P counterparts.

For the first issue, it is sufficient to observe that, in $\lambda 2$, type formation is expressed as $\Gamma \vdash_2 a : *$.

The second is also not very hard, but mostly tedious, as it boils down to discriminating on the type of the abstracted variable to determine exactly what kind of product is being formed, or what kind of abstraction is being introduced or applied.

The hardest part is handling the third aspect. Consider, for example, the abstraction rule of $\lambda 2$ (Figure 2),

$$\frac{\Gamma \vdash_2 a : u \qquad \Gamma, a \vdash_2 b : c \qquad \Gamma, a \vdash_2 c : *}{\Gamma \vdash_2 \lambda a.\, b : \Pi a.\, c}$$

and observe in particular the third premise, which enforces that the type of the body of the abstraction, c, is a well-formed type. Contrast this to the two corresponding abstraction rules of P (Figure 3):

$$\frac{\Gamma \vdash^{\mathsf{ty}}_{\mathsf{P}} a \qquad \Gamma, a \vdash^{\mathsf{ter}}_{\mathsf{P}} b : c}{\Gamma \vdash^{\mathsf{ter}}_{\mathsf{P}} \lambda a.\, b : \Pi a.\, c} \qquad\qquad \frac{\Gamma, * \vdash^{\mathsf{ter}}_{\mathsf{P}} a : b}{\Gamma \vdash^{\mathsf{ter}}_{\mathsf{P}} \lambda *.\, a : \Pi *.\, b}$$

Neither of the rules has a counterpart to the third premise of the λ2 rule. We can bridge this gap with a property variously known as **propagation**, type correctness or validity of the type system:

Lemma 11 (Propagation for P) *If* $\Gamma \vdash^{\mathsf{ter}}_{\mathsf{P}} a : b$, *then* $\Gamma \vdash^{\mathsf{ty}}_{\mathsf{P}} b$.

PROOF By induction on $\Gamma \vdash^{\mathsf{ter}}_{\mathsf{P}} a : b$. In the two application cases, β-substitutions are added to the types. We handle these with Theorem 7. ∎

Now that we have propagation, we can formulate the first major part of our equivalence proof:

Lemma 12 (Correspondence of λ2 and P)

(a) $$\Gamma \vdash^{\mathsf{ty}}_{\mathsf{P}} a \implies \Gamma \vdash_{2} a : *$$

(b) $$\Gamma \vdash^{\mathsf{ter}}_{\mathsf{P}} a : b \implies \Gamma \vdash_{2} a : b$$

(c) $$\Gamma \vdash_{2} a : b \implies \begin{cases} a = * & \text{if } b = \Box \\ \Gamma \vdash^{\mathsf{ty}}_{\mathsf{P}} a & \text{if } b = * \\ \Gamma \vdash^{\mathsf{ter}}_{\mathsf{P}} a : b & \text{otherwise} \end{cases}$$

PROOF Each by induction on the respective premise. The proof of (a) is routine. The proof of (b) requires propagation (Lemma 11) and (a), but is otherwise also routine. The proof of (c) repeatedly requires discrimination on the particular syntactic form of the argument type of an abstraction. ∎

Since Lemma 12 is somewhat technical, we summarise our understanding of the equivalence of λ2 and P in the following theorem.

Theorem 13 (Equivalence of λ2 and P)

$$\Gamma \vdash^{\mathsf{ter}}_{\mathsf{P}} a : b \iff \Gamma \vdash_{2} a : b \wedge \Gamma \vdash_{2} b : *$$

PROOF Repeated application of Lemma 12. Note that $\Gamma \vdash^{\mathsf{ty}}_{\mathsf{P}} b$ entails $b \neq *$ and $b \neq \Box$. ∎

6 Syntax Translations between F and PTS

The next part of our equivalence proof will establish the equivalence of F and P. In order to understand the difficulties of this step, let us recall the main example from Section 2. We have presented the F typing judgement

$$1; (\forall . 0_{\mathsf{ty}} \to 0_{\mathsf{ty}}), 0_{\mathsf{ty}} \vdash^{\mathsf{ter}}_{\mathsf{F}} 1_{\mathsf{ter}}\, 0_{\mathsf{ty}}\, 0_{\mathsf{ter}} : 0_{\mathsf{ty}}$$

and introduced

$$*, (\Pi * . \Pi 0 . 1), 1 \vdash^{\mathsf{ter}}_{\mathsf{P}} 1\ 2\ 0 : 2$$

$$
\begin{aligned}
\lfloor x_{\mathsf{ty}} \rfloor &:= x \\
\lfloor \forall . A \rfloor &:= \Pi *. \lfloor A \rfloor \\
\lfloor A \to B \rfloor &:= \Pi \lfloor A \rfloor . \lfloor B \rfloor [+1] \\
\lfloor x_{\mathsf{ter}} \rfloor &:= x \\
\lfloor s\, t \rfloor &:= \lfloor s \rfloor \lfloor t \rfloor \\
\lfloor s\, A \rfloor &:= \lfloor s \rfloor \lfloor A \rfloor \\
\lfloor \Lambda . s \rfloor &:= \lambda *. \lfloor s[\mathsf{id}, +1] \rfloor \\
\lfloor \lambda A . s \rfloor &:= \lambda \lfloor A \rfloor . \lfloor s[+1, \mathsf{id}] \rfloor
\end{aligned}
$$

Figure 5: Syntax translations from F to PTS.

$$
\begin{aligned}
\lceil x \rceil^{\Gamma}_{\mathsf{ty}} &:= x_{\mathsf{ty}} && \text{if } x : * \in \Gamma \\
\lceil \Pi *. a \rceil^{\Gamma}_{\mathsf{ty}} &:= \forall . \lceil a \rceil^{\Gamma,*}_{\mathsf{ty}} \\
\lceil \Pi a . b \rceil^{\Gamma}_{\mathsf{ty}} &:= \lceil a \rceil^{\Gamma}_{\mathsf{ty}} \to \lceil b \rceil^{\Gamma,a}_{\mathsf{ty}} [\text{-}1] \\
\lceil x \rceil^{\Gamma}_{\mathsf{ter}} &:= x_{\mathsf{ter}} && \text{if } x : a \in \Gamma, \quad a \neq * \\
\lceil a\, b \rceil^{\Gamma}_{\mathsf{ter}} &:= \lceil a \rceil^{\Gamma}_{\mathsf{ter}} \lceil b \rceil^{\Gamma}_{\mathsf{ty}} \\
\lceil a\, b \rceil^{\Gamma}_{\mathsf{ter}} &:= \lceil a \rceil^{\Gamma}_{\mathsf{ter}} \lceil b \rceil^{\Gamma}_{\mathsf{ter}} \\
\lceil \lambda *. a \rceil^{\Gamma}_{\mathsf{ter}} &:= \Lambda . \lceil a \rceil^{\Gamma,*}_{\mathsf{ter}} [\mathsf{id}, \text{-}1] \\
\lceil \lambda a . b \rceil^{\Gamma}_{\mathsf{ter}} &:= \lambda \lceil a \rceil^{\Gamma}_{\mathsf{ty}} . \lceil b \rceil^{\Gamma,a}_{\mathsf{ter}} [\text{-}1, \mathsf{id}]
\end{aligned}
$$

Figure 6: Syntax translations from PTS to F.

as a potential PTS counterpart. Equally valid, though, is

$$(\Pi *. \Pi 0. 1), *, 0 \vdash^{\mathsf{ter}}_{\mathsf{P}} 2\,1\,0 : 1$$

That is, the correspondence of F and P typing judgements is not a one-to-one mapping. We can also see that the correspondence of the indices is involved. Finally, applications are problematic. While the F term $1_{\mathsf{ter}}\, 0_{\mathsf{ty}}\, 0_{\mathsf{ter}}$ clearly distinguishes type and term application, it is impossible to make this distinction for a P term like $1\,2\,0$ unless the context is known.

To state the equivalence of these systems we have to introduce syntactic translations. Instead of lifting these to the level of contexts and attempting a direct proof, we will use CMLs to relate corresponding judgements. This will circumvent most of the obstacles outlined above.

The two (total) translation functions $\lfloor A \rfloor$ and $\lfloor s \rfloor$ from the types and terms of F to PTS terms are defined in Figure 5. Note that the presentation is somewhat simplified to enhance

readability. More precisely, the recursive calls for $\lfloor \Lambda.\, s \rfloor$ and $\lfloor \lambda A.\, s \rfloor$ are not structurally recursive. In the formalisation, this issue is handled via two additional arguments that act as accumulators for a type and, respectively, a term substitution. The accumulators provide a conduit when we wish to move a substitution through the translation in either direction.

For the inverse translation we introduce $\lceil a \rceil^{\Gamma}_{\mathsf{ty}}$ and $\lceil a \rceil^{\Gamma}_{\mathsf{ter}}$ in Figure 6. These two are necessarily partial as there are certain PTS terms that have no counterpart in F, neither as a type nor as a term. The context Γ is required to disambiguate type and term variables[7]. The translations $\lceil a \rceil^{\Gamma}_{\mathsf{ter}}$ and $\lceil a \rceil^{\Gamma}_{\mathsf{ty}}$ are given with the implicit assumption that a particular translation is defined iff all of its constituent components are defined. Note, in particular, that for a given Γ and a, at most one of $\lceil a \rceil^{\Gamma}_{\mathsf{ter}}$ and $\lceil a \rceil^{\Gamma}_{\mathsf{ty}}$ is defined. This justifies the two defining equations for $\lceil a\, b \rceil^{\Gamma}_{\mathsf{ter}}$.

The variable cases of the translations may also appear surprising, given that they transport de Bruijn indices directly from one side to the other. This only works under the assumption that all free variables are correctly mapped prior to the translation of a term. For closed terms this obviously holds and all the binder cases are carefully set up to preserve this property through the application of suitable renamings. For open terms, on the other hand, we will usually have a typing context at our disposal and the variable condition of the associated CML will ensure a correct mapping of variables.

We are going to exploit the fact that both F and P are able to internalise their contexts, and also that CMLs are very easy to instantiate when the context of the initial judgement happens to be empty. Hence we formulate our equivalence result as

$$\begin{aligned} \vdash^{\mathsf{ter}}_{\mathsf{F}} s : A \quad &\Longleftrightarrow \quad \vdash^{\mathsf{ter}}_{\mathsf{P}} \lfloor s \rfloor : \lfloor A \rfloor \\ \vdash^{\mathsf{ter}}_{\mathsf{P}} a : b \quad &\Longleftrightarrow \quad \vdash^{\mathsf{ter}}_{\mathsf{F}} \lceil a \rceil^{\langle\rangle}_{\mathsf{ter}} : \lceil b \rceil^{\langle\rangle}_{\mathsf{ty}} \end{aligned}$$

The final proof of this result will be given towards the end of Section 7 (Theorems 27 and 28). It relies on a number of carefully chosen steps. We first prove the two forward implications. To obtain each of the inverse implications, we will then suitably instantiate the respective other forward implication. The proof is then closed by establishing that accumulated translation steps cancel.

In this section we are going to establish the forward implications which state that the translations preserve typing. These implications can also be viewed as transforming valid judgements of F into valid judgements of P, and vice versa. This motivates our use of CMLs to handle the preservation statements. We will in fact only require the renaming variants of these CMLs.

We begin with the preservation of type formation from F to P. In order to formulate the correct CML, it is helpful to recall that we want a reduction from arbitrary judgements to

[7] Since all that is required is a partition of the free variables into term and type variables, we realise this context argument in the formalisation as a boolean function γ, which abstracts away the overhead of context lookups and syntactic comparisons.

variable judgements. That is, we would like to prove the following:

$$\frac{\forall x.\ \ N \vdash^{\mathsf{ty}}_{\mathsf{F}} x_{\mathsf{ty}} \ \implies\ \Gamma \vdash^{\mathsf{ty}}_{\mathsf{P}} \lfloor x_{\mathsf{ty}} \rfloor[\xi]}{N \vdash^{\mathsf{ty}}_{\mathsf{F}} A \ \implies\ \Gamma \vdash^{\mathsf{ty}}_{\mathsf{P}} \lfloor A \rfloor[\xi]}$$

which determines the required definition of context morphisms:

$$\xi : (N \vdash^{\mathsf{ty}}_{\mathsf{F}}) \to (\Gamma \vdash^{\mathsf{ty}}_{\mathsf{P}}) \ :=\ \forall x.\ \ N \vdash^{\mathsf{ty}}_{\mathsf{F}} x_{\mathsf{ty}} \ \implies\ \Gamma \vdash^{\mathsf{ty}}_{\mathsf{P}} \xi(x)$$

Note that we have simplified the translation step on the right from $\lfloor x_{\mathsf{ty}} \rfloor[\xi]$ to $\xi(x)$. We also adjusted the notation to reflect that we are now moving from one system to another.

It should come as no surprise that the proof of the CML will rely on the closure of $\xi : (N \vdash^{\mathsf{ty}}_{\mathsf{F}}) \to (\Gamma \vdash^{\mathsf{ty}}_{\mathsf{P}})$ under context extensions. What is new though, is that we obtain two separate closure rules. One for the addition of a new type variable and another for a new term variable.

Lemma 14

$$\frac{\xi : (N \vdash^{\mathsf{ty}}_{\mathsf{F}}) \to (\Gamma \vdash^{\mathsf{ty}}_{\mathsf{P}})}{\Uparrow \xi : (N + 1 \vdash^{\mathsf{ty}}_{\mathsf{F}}) \to (\Gamma, * \vdash^{\mathsf{ty}}_{\mathsf{P}})} \qquad \frac{\xi : (N \vdash^{\mathsf{ty}}_{\mathsf{F}}) \to (\Gamma \vdash^{\mathsf{ty}}_{\mathsf{P}})}{\xi \circ \mathrm{+1} : (N \vdash^{\mathsf{ty}}_{\mathsf{F}}) \to (\Gamma, \lfloor A \rfloor[\xi] \vdash^{\mathsf{ty}}_{\mathsf{P}})}$$

PROOF The first is by discrimination on the quantified variable (cf. proof of Lemma 8). The second is straightforward given weakening for type formation in P (Theorem 3). ∎

Lemma 15 (CML: F to P – Type Formation) *Type formation is preserved under $\lfloor \cdot \rfloor$, where ξ ensures a suitable renaming of the free type variables in A:*

$$\frac{N \vdash^{\mathsf{ty}}_{\mathsf{F}} A \qquad \xi : (N \vdash^{\mathsf{ty}}_{\mathsf{F}}) \to (\Gamma \vdash^{\mathsf{ty}}_{\mathsf{P}})}{\Gamma \vdash^{\mathsf{ty}}_{\mathsf{P}} \lfloor A \rfloor[\xi]}$$

PROOF By induction on $N \vdash^{\mathsf{ty}}_{\mathsf{F}} A$. Lemma 14 is used for the binder cases. ∎

Before we can play the same game for the typing level, we have to consider the issue of a variable clash. A term of F that contains the same index as a type and a term variable is potentially problematic. To avoid a clash we have to adjust the free type and term variables in a term with a pair of renamings with disjoint ranges. As mentioned above, we are going to build this idea of separating the free variables implicitly into our formulation of the CML. That is, when a pair $\langle \xi, \zeta \rangle$ constitutes a context morphism then it necessarily separates the free variables in a suitable manner.

Also note that since typing derivations contain type formation derivations, we are going to need Lemma 15, and thus also have to assume that the renaming ξ that is used on the type variables is a suitable context morphism in this regard. Thus our CML will take the following form:

$$\frac{\begin{array}{c}\xi : (N \vdash^{\mathsf{ty}}_{\mathsf{F}}) \to (\Gamma \vdash^{\mathsf{ty}}_{\mathsf{P}}) \\ \forall x A.\ \ N; \Delta \vdash^{\mathsf{ter}}_{\mathsf{F}} x_{\mathsf{ter}} : A \ \implies\ \Gamma \vdash^{\mathsf{ter}}_{\mathsf{P}} \lfloor x_{\mathsf{ter}}[\xi, \zeta] \rfloor : \lfloor A \rfloor[\xi]\end{array}}{N; \Delta \vdash^{\mathsf{ter}}_{\mathsf{F}} s : A \ \implies\ \Gamma \vdash^{\mathsf{ter}}_{\mathsf{P}} \lfloor s[\xi, \zeta] \rfloor : \lfloor A \rfloor[\xi]}$$

We define

$$\langle \xi, \zeta \rangle : (N; \Delta \vdash^{\mathsf{ter}}_{\mathsf{F}}) \to (\Gamma \vdash^{\mathsf{ter}}_{\mathsf{P}}) \;:=$$
$$\forall x A.\quad N; \Delta \vdash^{\mathsf{ter}}_{\mathsf{F}} x_{\mathsf{ter}} : A \implies \Gamma \vdash^{\mathsf{ter}}_{\mathsf{P}} \zeta(x) : \lfloor A \rfloor[\xi]$$

and proceed to first establish closure under context extensions and then prove the CML for typing under translations from F to P.

Lemma 16

$$\frac{\langle \xi, \zeta \rangle : (N; \Delta \vdash^{\mathsf{ter}}_{\mathsf{F}}) \to (\Gamma \vdash^{\mathsf{ter}}_{\mathsf{P}})}{\langle \Uparrow \xi, \zeta \circ +1 \rangle : (N+1; \Delta[+1] \vdash^{\mathsf{ter}}_{\mathsf{F}}) \to (\Gamma, * \vdash^{\mathsf{ter}}_{\mathsf{P}})} \qquad \frac{\langle \xi, \zeta \rangle : (N; \Delta \vdash^{\mathsf{ter}}_{\mathsf{F}}) \to (\Gamma \vdash^{\mathsf{ter}}_{\mathsf{P}}) \qquad \xi : (N \vdash^{\mathsf{ty}}_{\mathsf{F}}) \to (\Gamma \vdash^{\mathsf{ty}}_{\mathsf{P}})}{\langle \xi \circ +1, \Uparrow \zeta \rangle : (N; \Delta, A \vdash^{\mathsf{ter}}_{\mathsf{F}}) \to (\Gamma, \lfloor A \rfloor[\xi] \vdash^{\mathsf{ter}}_{\mathsf{P}})}$$

PROOF The first is relatively easy, with some minor technicalities to handle the operation $\Delta[+1]$. The second is the extension step that requires the case distinction on the quantified variable, while the side condition is needed to handle the type of that variable with Lemma 15. ∎

Lemma 17 (CML: F to P – Typing) *Typing is preserved under $\lfloor \cdot \rfloor$, where $\langle \xi, \zeta \rangle$ ensures a suitable renaming of the free type and term variables in s and A:*

$$\frac{N; \Delta \vdash^{\mathsf{ter}}_{\mathsf{F}} s : A \qquad \xi : (N \vdash^{\mathsf{ty}}_{\mathsf{F}}) \to (\Gamma \vdash^{\mathsf{ty}}_{\mathsf{P}}) \qquad \langle \xi, \zeta \rangle : (N; \Delta \vdash^{\mathsf{ter}}_{\mathsf{F}}) \to (\Gamma \vdash^{\mathsf{ter}}_{\mathsf{P}})}{\Gamma \vdash^{\mathsf{ter}}_{\mathsf{P}} \lfloor s[\xi, \zeta] \rfloor : \lfloor A \rfloor[\xi]}$$

PROOF By induction on $N; \Delta \vdash^{\mathsf{ter}}_{\mathsf{F}} s : A$. Lemmas 14 and 16 are used for the binder cases. Nested type formation derivations are handled with Lemma 15. ∎

Lemma 18 (Preservation of Typing from F to P)

$$\vdash^{\mathsf{ter}}_{\mathsf{F}} s : A \implies \vdash^{\mathsf{ter}}_{\mathsf{P}} \lfloor s \rfloor : \lfloor A \rfloor$$

PROOF Instantiate Lemma 17 with the following:

$$\mathsf{id} : (0 \vdash^{\mathsf{ty}}_{\mathsf{F}}) \to (\langle\rangle \vdash^{\mathsf{ty}}_{\mathsf{P}}) \qquad\qquad \langle \mathsf{id}, \mathsf{id} \rangle : (0; \langle\rangle \vdash^{\mathsf{ter}}_{\mathsf{F}}) \to (\langle\rangle \vdash^{\mathsf{ter}}_{\mathsf{P}})$$

Both hold vacuously. ∎

For the preservation law from P to F we use the same setup and will only highlight a few points of interest. Note that since the translations from P to F are partial, all of the following results have to establish that all relevant instances of translation steps are in fact defined. In each of the proofs enough information is available to ensure this. Hence we are not going to further discuss the treatment of partiality.

We define context morphisms as

$$\xi : (\Gamma \vdash^{\mathsf{ty}}_{\mathsf{P}}) \to (N \vdash^{\mathsf{ty}}_{\mathsf{F}}) \;:=\; \forall x.\quad \Gamma \vdash^{\mathsf{ty}}_{\mathsf{P}} x \implies N \vdash^{\mathsf{ty}}_{\mathsf{F}} \xi(x)$$

and then prove the closure step and the CML.

Lemma 19

$$\frac{\xi : (\Gamma \vdash^{\mathsf{ty}}_{\mathsf{P}}) \to (N \vdash^{\mathsf{ty}}_{\mathsf{F}})}{\Uparrow \xi : (\Gamma, * \vdash^{\mathsf{ty}}_{\mathsf{P}}) \to (N+1 \vdash^{\mathsf{ty}}_{\mathsf{F}})} \qquad \frac{\xi : (\Gamma \vdash^{\mathsf{ty}}_{\mathsf{P}}) \to (N \vdash^{\mathsf{ty}}_{\mathsf{F}}) \qquad \Gamma \vdash^{\mathsf{ty}}_{\mathsf{P}} a}{-1 \circ \xi : (\Gamma, a \vdash^{\mathsf{ty}}_{\mathsf{P}}) \to (N \vdash^{\mathsf{ty}}_{\mathsf{F}})}$$

PROOF Both are by discrimination on the quantified variable. The side condition on the second rule is necessary to ensure that the index 0 is a term variable. This in turn entails that all type variables are non-zero and hence the application of the renaming -1 to a type variable can be inverted. ∎

Lemma 20 (CML: P to F – Type Formation) *Type formation is preserved under* $\lceil \cdot \rceil^{\Gamma}_{\mathsf{ty}}$*, where* ξ *ensures a suitable renaming of the free variables in* a*:*

$$\frac{\Gamma \vdash^{\mathsf{ty}}_{\mathsf{P}} a \qquad \xi : (\Gamma \vdash^{\mathsf{ty}}_{\mathsf{P}}) \to (N \vdash^{\mathsf{ty}}_{\mathsf{F}})}{N \vdash^{\mathsf{ty}}_{\mathsf{F}} \lceil a \rceil^{\Gamma}_{\mathsf{ty}}[\xi]}$$

PROOF By induction on $\Gamma \vdash^{\mathsf{ty}}_{\mathsf{P}} a$. Lemma 19 is used for the binder cases. ∎

For the preservation of typing from P to F we define context morphisms as

$$\begin{aligned}&\langle \xi, \zeta \rangle : (\Gamma \vdash^{\mathsf{ter}}_{\mathsf{P}}) \to (N; \Delta \vdash^{\mathsf{ter}}_{\mathsf{F}}) \;:=\\ &\quad \forall x b.\;\; \Gamma \vdash^{\mathsf{ter}}_{\mathsf{P}} x : b \implies N; \Delta \vdash^{\mathsf{ter}}_{\mathsf{F}} \zeta(x) : \lceil b \rceil^{\Gamma}_{\mathsf{ty}}[\xi]\end{aligned}$$

and again prove the extension lemma as well as the CML.

Lemma 21

$$\frac{\langle \xi, \zeta \rangle : (\Gamma \vdash^{\mathsf{ter}}_{\mathsf{P}}) \to (N; \Delta \vdash^{\mathsf{ter}}_{\mathsf{F}}) \qquad \xi : (\Gamma \vdash^{\mathsf{ty}}_{\mathsf{P}}) \to (N \vdash^{\mathsf{ty}}_{\mathsf{F}})}{\langle \Uparrow \xi, -1 \circ \zeta \rangle : (\Gamma, * \vdash^{\mathsf{ter}}_{\mathsf{P}}) \to (N+1; \Delta[\uparrow 1] \vdash^{\mathsf{ter}}_{\mathsf{F}})}$$

$$\frac{\langle \xi, \zeta \rangle : (\Gamma \vdash^{\mathsf{ter}}_{\mathsf{P}}) \to (N; \Delta \vdash^{\mathsf{ter}}_{\mathsf{F}}) \qquad \xi : (\Gamma \vdash^{\mathsf{ty}}_{\mathsf{P}}) \to (N \vdash^{\mathsf{ty}}_{\mathsf{F}})}{\langle -1 \circ \xi, \Uparrow \zeta \rangle : (\Gamma, a \vdash^{\mathsf{ter}}_{\mathsf{P}}) \to (N; \Delta, \lceil a \rceil^{\Gamma}_{\mathsf{ty}}[\xi] \vdash^{\mathsf{ter}}_{\mathsf{F}})}$$

PROOF Both are by discrimination on the quantified variable. ∎

Lemma 22 (CML: P to F – Typing) *Typing is preserved under* $\lceil \cdot \rceil^{\Gamma}_{\mathsf{ter}}$*, where* $\langle \xi, \zeta \rangle$ *ensures a suitable renaming of the free variables in* a *and* b*:*

$$\frac{\Gamma \vdash^{\mathsf{ter}}_{\mathsf{P}} a : b \qquad \xi : (\Gamma \vdash^{\mathsf{ty}}_{\mathsf{P}}) \to (N \vdash^{\mathsf{ty}}_{\mathsf{F}}) \qquad \langle \xi, \zeta \rangle : (\Gamma \vdash^{\mathsf{ter}}_{\mathsf{P}}) \to (N; \Delta \vdash^{\mathsf{ter}}_{\mathsf{F}})}{N; \Delta \vdash^{\mathsf{ter}}_{\mathsf{F}} \lceil a \rceil^{\Gamma}_{\mathsf{ter}}[\xi, \zeta] : \lceil b \rceil^{\Gamma}_{\mathsf{ty}}[\xi]}$$

PROOF By induction on $\Gamma \vdash^{\mathsf{ter}}_{\mathsf{P}} a : b$. Lemmas 19 and 21 are used for the binder cases. Nested type formation derivations are handled with Lemma 20. ∎

Lemma 23 (Preservation of Typing from P to F)

$$\vdash^{\mathsf{ter}}_{\mathsf{P}} a : b \implies \vdash^{\mathsf{ter}}_{\mathsf{F}} \lceil a \rceil^{\langle\rangle}_{\mathsf{ter}} : \lceil b \rceil^{\langle\rangle}_{\mathsf{ty}}$$

PROOF Instantiate Lemma 22 with the following:

$$\mathsf{id} : (\langle\rangle \vdash^{\mathsf{ty}}_{\mathsf{P}}) \to (0 \vdash^{\mathsf{ty}}_{\mathsf{F}}) \qquad\qquad \langle \mathsf{id}, \mathsf{id} \rangle : (\langle\rangle \vdash^{\mathsf{ter}}_{\mathsf{P}}) \to (0; \langle\rangle \vdash^{\mathsf{ter}}_{\mathsf{F}})$$

Both hold vacuously. ∎

7 Cancellation Laws and Equivalence Theorems

At this point we know that the introduced translations preserve judgements. To obtain a full reduction result we require the inverse implications as well. Proving these is the objective of this section.

The basic idea is relatively straightforward: we simply instantiate the preservation laws with the result of a first translation step. Thus to obtain the inverse of Lemma 18, we take Lemma 23 and instantiate it as follows:

$$\vdash^{\mathsf{ter}}_{\mathsf{P}} \lfloor s \rfloor : \lfloor A \rfloor \quad \Longrightarrow \quad \vdash^{\mathsf{ter}}_{\mathsf{F}} \lceil \lfloor s \rfloor \rceil^{\langle\rangle}_{\mathsf{ter}} : \lceil \lfloor A \rfloor \rceil^{\langle\rangle}_{\mathsf{ty}}$$

The proof is complete, if we can ensure that $\lceil \lfloor s \rfloor \rceil^{\langle\rangle}_{\mathsf{ter}} = s$ and $\lceil \lfloor A \rfloor \rceil^{\langle\rangle}_{\mathsf{ty}} = A$. That is, we require a notion of cancellation for our translations. In total we will have to provide four such cancellation laws, the two we just mentioned for the round trip F–P–F, and another two for P–F–P.

Stating these cancellation laws requires a little care. Consider again the case of $\lceil \lfloor s \rfloor \rceil^{\langle\rangle}_{\mathsf{ter}} = s$. It will be used to establish a typing for s, so we must not already assume such a typing for s, for otherwise the law would be unusable. It is, however, perfectly reasonable to assume a typing for $\lfloor s \rfloor$, which holds in the situation where the law is going to be used.

We again have to generalise over arbitrary renamings, and also contexts, in order to prove these laws.

Let us first consider the P–F–P round trip, which, despite the partiality of the first translation step, is the easier of the two.

Lemma 24 (P–F–P Cancellation)

$$\lceil a[\xi] \rceil^{\Gamma}_{\mathsf{ty}} = A \quad \Longrightarrow \quad \lfloor A \rfloor = a[\xi]$$
$$\lceil a[\xi] \rceil^{\Gamma}_{\mathsf{ter}} = s \quad \Longrightarrow \quad \lfloor s \rfloor = a[\xi]$$

PROOF Both are by induction on a and mostly routine. Some minor groundwork is required to ensure that, e.g., $a[\xi_1] = a[\xi_2]$ whenever ξ_1 and ξ_2 agree on the free variables of a. The proof of the second implication requires the first. ∎

Next we consider the F–P–F round trip. The cancellation for the type translation does not cause any major issues once we assume that the result of the first translation step is a well-formed type in P under some Γ.

Lemma 25 (F–P–F Cancellation of Type Translation)

$$\Gamma \vdash^{\mathsf{ty}}_{\mathsf{P}} \lfloor A[\xi] \rfloor \quad \Longrightarrow \quad \lceil \lfloor A[\xi] \rfloor \rceil^{\Gamma}_{\mathsf{ty}} = A[\xi]$$

PROOF By induction on A. ∎

For the term translation, the situation is slightly more involved. We are again faced with the issue that a naive term translation step from F to P could potentially confuse free term and type variables. Since we are effectively proving the cancellation prior to typing, we cannot rely on typing to implicitly fix this issue (as we did for Lemma 17). In order to prevent this from happening here, we have to explicitly express that a pair or renamings sufficiently separates the free variables.

$$\begin{aligned}\Gamma \vDash_{\mathsf{P}} \xi \parallel \zeta \;:=\; &(\forall x a.\;\; \xi(x) : a \in \Gamma \implies a = *) \;\wedge \\ &(\forall x a.\;\; \zeta(x) : a \in \Gamma \implies a \neq *)\end{aligned}$$

Note in particular, that Γ is a PTS context, namely the one under which $\lfloor s[\xi,\zeta] \rfloor$ should be typable.

Lemma 26 (F–P–F Cancellation of Term Translation)

$$\Gamma \vdash^{\mathsf{ter}}_{\mathsf{P}} \lfloor s[\xi,\zeta] \rfloor : c \implies \Gamma \vDash_{\mathsf{P}} \xi \parallel \zeta \implies \lceil \lfloor s[\xi,\zeta] \rfloor \rceil^{\Gamma}_{\mathsf{ter}} = s[\xi,\zeta]$$

PROOF By induction on s and using Lemma 25 for the occurring instances of type translations. We also require two closure conditions for $\Gamma \vDash_{\mathsf{P}} \xi \parallel \zeta$ to handle the binder cases:

$$\frac{\Gamma \vDash_{\mathsf{P}} \xi \parallel \zeta}{\Gamma, * \vDash_{\mathsf{P}} \Uparrow \xi \parallel \zeta \circ +1} \qquad \frac{\Gamma \vDash_{\mathsf{P}} \xi \parallel \zeta \qquad \Gamma \vdash^{\mathsf{ty}}_{\mathsf{P}} c}{\Gamma, c \vDash_{\mathsf{P}} \xi \circ +1 \parallel \Uparrow \zeta}$$

Both are easy to verify. ∎

At this point we have all the ingredients we require to prove the inverse directions of the preservation results from Section 6. This in turn allows us to obtain the equivalences that state the full reduction of the typability problem from F to $\lambda 2$, and vice versa.

Theorem 27 (Reduction of Typability from F to P)

$$\vdash^{\mathsf{ter}}_{\mathsf{F}} s : A \iff \vdash^{\mathsf{ter}}_{\mathsf{P}} \lfloor s \rfloor : \lfloor A \rfloor$$

PROOF Lemma 18 yields the forward implication. For the inverse direction we instantiate Lemma 23 to

$$\vdash^{\mathsf{ter}}_{\mathsf{P}} \lfloor s \rfloor : \lfloor A \rfloor \implies \vdash^{\mathsf{ter}}_{\mathsf{F}} \lceil \lfloor s \rfloor \rceil^{\langle\rangle}_{\mathsf{ter}} : \lceil \lfloor A \rfloor \rceil^{\langle\rangle}_{\mathsf{ty}}$$

The accumulated translations cancel according to Lemmas 25 and 26. Observe that $\langle\rangle \vDash_{\mathsf{P}} \mathsf{id} \parallel \mathsf{id}$ vacuously holds. ∎

Theorem 28 (Reduction of Typability from P to F)

$$\vdash^{\mathsf{ter}}_{\mathsf{P}} a : b \iff \vdash^{\mathsf{ter}}_{\mathsf{F}} \lceil a \rceil^{\langle\rangle}_{\mathsf{ter}} : \lceil b \rceil^{\langle\rangle}_{\mathsf{ty}}$$

PROOF Lemma 23 yields the forward implication. For the inverse direction we instantiate Lemma 18 to

$$\vdash^{\mathsf{ter}}_{\mathsf{F}} \ulcorner a \urcorner^{\langle\rangle}_{\mathsf{ter}} : \ulcorner b \urcorner^{\langle\rangle}_{\mathsf{ty}} \quad \Longrightarrow \quad \vdash^{\mathsf{ter}}_{\mathsf{P}} \left\lfloor \ulcorner a \urcorner^{\langle\rangle}_{\mathsf{ter}} \right\rfloor : \left\lfloor \ulcorner b \urcorner^{\langle\rangle}_{\mathsf{ty}} \right\rfloor$$

and use Lemma 24 to cancel the accumulated translations. ∎

Lemma 29 (Reduction of Typability from F to λ2)

$$\vdash^{\mathsf{ter}}_{\mathsf{F}} s : A \quad \Longleftrightarrow \quad \vdash_2 \lfloor s \rfloor : \lfloor A \rfloor \wedge \vdash_2 \lfloor A \rfloor : *$$

PROOF

(Thm. 27) $\vdash^{\mathsf{ter}}_{\mathsf{F}} s : A \iff \vdash^{\mathsf{ter}}_{\mathsf{P}} \lfloor s \rfloor : \lfloor A \rfloor$

(Thm. 13) $\iff \vdash_2 \lfloor s \rfloor : \lfloor A \rfloor \wedge \vdash_2 \lfloor A \rfloor : *$ ∎

Lemma 30 (Reduction of Typability from λ2 to F)

$$\vdash_2 a : b \wedge \vdash_2 b : * \quad \Longleftrightarrow \quad \vdash^{\mathsf{ter}}_{\mathsf{F}} \ulcorner a \urcorner^{\langle\rangle}_{\mathsf{ter}} : \ulcorner b \urcorner^{\langle\rangle}_{\mathsf{ty}}$$

PROOF

(Thm. 13) $\vdash_2 a : b \wedge \vdash_2 b : * \iff \vdash^{\mathsf{ter}}_{\mathsf{P}} a : b$

(Thm. 28) $\iff \vdash^{\mathsf{ter}}_{\mathsf{F}} \ulcorner a \urcorner^{\langle\rangle}_{\mathsf{ter}} : \ulcorner b \urcorner^{\langle\rangle}_{\mathsf{ty}}$ ∎

Theorem 31 (Equivalence of F and λ2) *The typability problem under the empty context can be reduced from* F *to* λ2*, and vice versa.*

$$\vdash^{\mathsf{ter}}_{\mathsf{F}} s : A \quad \Longleftrightarrow \quad \vdash_2 \lfloor s \rfloor : \lfloor A \rfloor \wedge \vdash_2 \lfloor A \rfloor : *$$
$$\vdash_2 a : b \wedge \vdash_2 b : * \quad \Longleftrightarrow \quad \vdash^{\mathsf{ter}}_{\mathsf{F}} \ulcorner a \urcorner^{\langle\rangle}_{\mathsf{ter}} : \ulcorner b \urcorner^{\langle\rangle}_{\mathsf{ty}}$$

PROOF Conjunction of Lemmas 29 and 30. ∎

8 Related Work

Context morphism lemmas appear in [14, 2, 18]. The basic idea appears to be due to McKinna: see footnote on p. 104 of [11]. Together with Goguen, he presents it in [14] as a proof device to obtain a generic substitutivity result for a PTS.

In [2] Adams adapts this approach to work with syntactic families, indexed with an upper bound on the set of free variables. In his work he also presents a number of high level design principles for the formal treatment of PTS theory that he collectively calls a "big-step" approach. His proofs are formalised in Coq.

Several formalised CMLs for a pure de Bruijn setting can also be found in the examples that accompany the Autosubst framework. The core principles are nicely laid out in [18].

An interesting remark can be found in [19] where certain aspects of the F*-language and its type system are discussed. F*'s support for inductions with custom termination arguments allows the authors to give a direct inductive prove of a CML, without establishing the renaming case first. We did not attempt to port this technique to our Coq proofs. The explicit formulation of the correct termination order is complicated, and additional overhead would be caused by simulating the required features of F* in Coq. Meanwhile, the gain would have been small, as most of our results only rely on renaming CMLs.

Several of the complications of our proofs stem from the various explicit representations of variable binding. Thus it is worth discussing a higher-order abstract syntax (HOAS) presentation as an alternative. Unfortunately the type theory of Coq is too strong to natively support HOAS-reasoning. Workarounds to this problem exist but neither Chlipalas parametric HOAS (PHOAS) [8] nor the various instances of the HYBRID framework [6, 7, 10] turned out to be viable options. It is not clear how PHOAS would handle the translation of syntax, and the most suitable variant of HYBRID [7] could not sensibly support the single-sorted PTS syntax. In addition both techniques introduce extra layers of abstraction that increase the cognitive gap between an informal paper presentation and a corresponding formalisation. It appears to us as if the gap is even bigger than the one for a pure de Bruijn presentation, though this is subjective and we may be biased.

9 Conclusion and Future Work

We have demonstrated that context morphism lemmas, which were originally designed as a proof device for a single type system, can be extended to a multi-system setting. Moreover, the technique is capable of handling the translation from one syntactic system to another.

We have used this technique to formalise the equivalence of two variants of System F, by showing that judgements are preserved under translations from one system to another and by establishing cancellation laws. That is, a judgement holds in one system if and only if its translated version holds in the other. The formalisation is about 1200 LOC, excluding the Autosubst framework.

Our result further demonstrates that the combination of pure de Bruijn syntax, parallel substitutions and context morphism lemmas leads to elegant proofs for non-trivial results.

There are a number of future research directions this work could take. An obvious next step is the extension of the equivalence result to further variants of System F. This would test the proposed techniques on new scenarios as well as strengthen the idea that all variants of F discussed in the literature are essentially the same.

Another open question is that of the faithfulness of the syntax translations with respect to β-reduction behaviour. It is unlikely that we defined an unfaithful translation that still satisfied the strong results we have shown in this work, but this is not a formal argument. Thus we have taken some preliminary steps to establish that (weak) β-reduction is preserved

under the syntactic translations. So far we have proven that $\lfloor \cdot \rfloor$ preserves weak β-reduction. The inverse direction will require additional work, since $\lambda 2$ has (ill-typed) reductions that F cannot mirror.

Finally it would be interesting to compare our proof to other formalisations using locally nameless syntax [3] or higher-order abstract syntax. With respect to the latter it might be interesting to first build a slim, elegant and flexible HOAS abstraction layer. A good starting point is the HYBRID variant of [6] which already uses de Bruijn syntax internally. It is, however, burdened by the somewhat cumbersome single-variable substitutions. It may be possible to improve upon this using the Autosubst library.

Alternatively, we could imagine replaying this proof in some form in another proof system, say one that natively supports a HOAS approach. This would allow us to analyse which parts of the proof are inherently complicated and which issues are artefacts of the de Bruijn setup. The Abella proof system [4] could be an interesting candidate for such a comparison.

References

[1] Martín Abadi, Luca Cardelli, Pierre-Louis Curien, and Jean-Jacques Lévy. Explicit substitutions. *Journal of Functional Programming*, 1(4):375–416, 1991.

[2] Robin Adams. Formalized metatheory with terms represented by an indexed family of types. In *Types for Proofs and Programs*, volume 3839 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2006.

[3] Brian E. Aydemir, Arthur Charguéraud, Benjamin C. Pierce, Randy Pollack, and Stephanie Weirich. Engineering formal metatheory. In *Proceedings of POPL 2008*, pages 3–15. ACM, 2008.

[4] David Baelde, Kaustuv Chaudhuri, Andrew Gacek, Dale Miller, Gopalan Nadathur, Alwen Tiu, and Yuting Wang. Abella: A system for reasoning about relational specifications. *Journal of Formalized Reasoning*, 7(2):1–89, 2014.

[5] Henk Barendregt. Introduction to generalized type systems. *Journal of Functional Programming*, 1(2):125–154, 1991.

[6] Venanzio Capretta and Amy P. Felty. Combining de Bruijn indices and higher-order abstract syntax in coq. In *Types for Proofs and Programs: International Workshop, TYPES 2006, Nottingham, UK, April 18-21, 2006, Revised Selected Papers*, pages 63–77. Springer, 2007.

[7] Venanzio Capretta and Amy P. Felty. Higher-order abstract syntax in type theory. In S. Barry Cooper, Herman Geuvers, Anand Pillay, and Jouko Väänänen, editors, *Logic Colloquium 2006:*, pages 65–90. Cambridge University Press, 2009.

[8] Adam Chlipala. Parametric higher-order abstract syntax for mechanized semantics. In *ACM Sigplan Notices*, volume 43, pages 143–156. ACM, 2008.

[9] Nicolaas Govert de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. *Indagationes Mathematicae (Proceedings)*, 75(5):381 – 392, 1972.

[10] Amy P. Felty and Alberto Momigliano. Hybrid - A definitional two-level approach to reasoning with higher-order abstract syntax. *J. Autom. Reasoning*, 48(1):43–105, 2012.

[11] Jan Herman Geuvers. Logics and type systems. Proefschrift, Katholieke Universiteit Nijmegen, 1993.

[12] Jean-Yves Girard. Interprétation fonctionelle et élimination des coupures de l'arithmétique d'ordre supérieur. Thèse de doctorat d'état, Université Paris VII, 1972.

[13] Jean-Yves Girard, Paul Taylor, and Yves Lafont. *Proofs and types*. Cambridge University Press, 1989.

[14] Healfdene Goguen and James McKinna. Candidates for substitution. Technical Report ECS-LFCS-97-358, University of Edinburgh, 1997.

[15] Robert Harper. *Practical foundations for programming languages*. Cambridge University Press, 2013.

[16] John Charles Reynolds. Towards a theory of type structure. In *Paris Colloque sur la Programmation*, volume 19 of *Lecture Notes in Computer Science*, pages 408–423. Springer, 1974.

[17] Steven Schäfer, Gert Smolka, and Tobias Tebbi. Completeness and decidability of de Bruijn substitution algebra in Coq. In *Proceedings of CPP 2015*, pages 67–73. ACM, 2015.

[18] Steven Schäfer, Tobias Tebbi, and Gert Smolka. Autosubst: Reasoning with de Bruijn terms and parallel substitutions. In *Interactive Theorem Proving, Proceedings*, volume 9236 of *Lecture Notes in Computer Science*, pages 359–374. Springer, 2015.

[19] Nikhil Swamy, Cătălin Hriţcu, Chantal Keller, Aseem Rastogi, Antoine Delignat-Lavaud, Simon Forest, Karthikeyan Bhargavan, Cédric Fournet, Pierre-Yves Strub, Markulf Kohlweiss, Jean-Karim Zinzindohoue, and Santiago Zanella-Béguelin. Dependent types and multi-monadic effects in F*. In *Proceedings of POPL 2016*, pages 256–270. ACM, 2016.