Relating System F and $\lambda 2$: A Case Study in Coq, Abella and Beluga

Jonas Kaiser[1], Brigitte Pientka[2], and Gert Smolka[3]

1 **Saarland University, Saarbrücken, Germany**
`jkaiser@ps.uni-saarland.de`
2 **School of Computer Science, Montreal, Canada**
`bpientka@cs.mcgill.ca`
3 **Saarland University, Saarbrücken, Germany**
`smolka@ps.uni-saarland.de`

Abstract

We give three formalisations of a proof of the equivalence of the usual, two-sorted presentation of System F and its single-sorted pure type system (PTS) variant $\lambda 2$. This is established by reducing the typability problem of F to $\lambda 2$ and vice versa. A key challenge is the treatment of variable binding and contextual information. The formalisations all share the same high level proof structure using relations to connect the type systems. They do, however, differ significantly in their representation and manipulation of variables and contextual information. In Coq, we use pure de Bruijn indices and parallel substitutions. In Abella, we use higher-order abstract syntax (HOAS) and nominal constants of the ambient reasoning logic. In Beluga, we also use HOAS but within contextual modal type theory. Our contribution is twofold. First, we present and compare a collection of machine-checked solutions to a non-trivial theoretical result. Second, we propose our proof as a benchmark, complementing the POPLmark and ORBI challenges by testing how well a given proof assistant or framework handles complex contextual information involving multiple type systems.

1998 ACM Subject Classification F.4.1 Mathematical Logic – Lambda calculus and related systems

Keywords and phrases Pure Type Systems, System F, de Bruijn Syntax, Higher-Order Abstract Syntax, Contextual Reasoning

Digital Object Identifier 10.4230/LIPIcs.FSCD.2017.XY

1 Introduction

There are different presentations of "System F" in the literature and they are effectively considered equivalent, which is used to justify the transport of theoretical results between said presentations. The assumed notion of equivalence is primarily a reduction of the typability problem from one system to the other. While the existence of a suitable correspondence between the systems may appear likely or obvious, it turns out that actually proving it formally is surprisingly intricate. As long as the systems in question use the same expression syntax, the proofs are usually tedious but straightforward. If, on the other hand, not only the type systems, but also the syntactic languages differ, then establishing the correct correspondence becomes much more involved. The goal of this paper is to showcase various formalisation techniques to deal with the intricacies that arise in such an equivalence proof.

System F in its original form is due to Girard [13, 14], who introduced it in the context of proof theory. It was also independently discovered by Reynolds [25] as the polymorphic

$$\boxed{\mathsf{Ty_F}} \quad A, B ::= X \mid A \to B \mid \forall X.\, A \qquad \boxed{\mathsf{Tm_F}} \quad s, t ::= x \mid s\, t \mid \lambda x{:}A.\, s \mid s\, A \mid \Lambda X.\, s$$

$$\boxed{\mathsf{C}_\mathsf{F}^\mathsf{ty}} \quad \Delta ::= \emptyset \mid \Delta, X \qquad \boxed{\mathsf{C}_\mathsf{F}^\mathsf{tm}} \quad \Gamma ::= \bullet \mid \Gamma, x{:}A$$

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

$$\frac{\Delta; \Gamma \vdash_\mathsf{F}^\mathsf{tm} s : A \to B \qquad \Delta; \Gamma \vdash_\mathsf{F}^\mathsf{tm} t : A}{\Delta; \Gamma \vdash_\mathsf{F}^\mathsf{tm} s\, t : B} \qquad \frac{\Delta; \Gamma, x{:}A \vdash_\mathsf{F}^\mathsf{tm} s : B \qquad \Delta \vdash_\mathsf{F}^\mathsf{ty} A}{\Delta; \Gamma \vdash_\mathsf{F}^\mathsf{tm} \lambda x{:}A.\, s : A \to B}\ x \notin \operatorname{dom} \Gamma$$

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

Figure 1 Stratified System F: types, terms, contexts, type formation and typing.

$$\boxed{\mathsf{Tm}_\lambda} \quad a, b, c, d ::= * \mid \Box \mid x \mid a\, b \mid \lambda x{:}a.\, b \mid \Pi x{:}a.\, b \qquad \boxed{\mathsf{C}_\lambda} \quad \Psi ::= \bullet \mid \Psi, x{:}a$$

$$\frac{}{\Psi \vdash_\lambda * : \Box} \qquad \frac{x{:}a \in \Psi \qquad \Psi \vdash_\lambda a : u}{\Psi \vdash_\lambda x : a} \qquad \frac{\Psi \vdash_\lambda a : u \qquad \Psi, x{:}a \vdash_\lambda b : *}{\Psi \vdash_\lambda \Pi x{:}a.\, b : *}\ x \notin \operatorname{dom} \Psi$$

$$\frac{\Psi \vdash_\lambda a : \Pi x{:}c.\, d \qquad \Psi \vdash_\lambda b : c}{\Psi \vdash_\lambda a\, b : d[b/x]} \qquad \frac{\Psi \vdash_\lambda a : u \qquad \Psi, x{:}a \vdash_\lambda b : c \qquad \Psi, x{:}a \vdash_\lambda c : *}{\Psi \vdash_\lambda \lambda x{:}a.\, b : \Pi x{:}a.\, c}\ x \notin \operatorname{dom} \Psi$$

Figure 2 PTS: terms, contexts, and the type system $\lambda 2$; u ranges over the universes $*$ and $\Box$.

λ-calculus. For the purpose of this paper we consider two presentations that differ sufficiently to demonstrate the various complications. The first, called F and shown in Fig. 1, is the common two-sorted, or stratified, presentation, as for example given by Harper [16]. The second, given in Fig. 2, is the pure type system (PTS) $\lambda 2$, which appears as a corner in Barendregt's λ-cube [5].

In [12], Geuvers gives a proof sketch that valid typing judgements can be translated between these two presentations. In [18] we then gave the first proof, machine-checked in Coq, of the full reduction result. The proof presented here is a lot simpler for a number of reasons. We use relations on the two syntactic languages rather than translation functions, so we do not have to concern ourselves with cancellation laws. We can further establish the correspondence directly, which completely bypasses the need for an intermediate, stratified type system for the PTS syntax.

The relations that precisely establish the correspondence of our two systems are realised as inductive predicates (Fig. 3). The main advantage over the functional approach is that it allows us to focus on the meaningful, well-typed fragments of the two systems. Despite these simplifications we still have to face the metaphorical elephant in the room, namely variable binding and context manipulation. More precisely, we have to represent and handle contexts that track various kinds of information, like the set of defined variables, their associated types, their correspondence to other variables, and so on. To make the situation fully explicit: our contexts are dependent sequences of dependent records.

In the following we present three formalisations. They all follow the same basic proof structure, outlined in Sect. 2, but they each deal with the complexities of contextual information and variable binding in different ways. In Sects. 3, 4 and 5 we discuss in detail how this is managed in Coq [**?**], Abella [4] and respectively Beluga [24]. The Coq proof

$\boxed{\mathsf{C}^{\mathsf{ty}}_{\mathsf{R}}}$ $\quad \Theta ::= \bullet \mid \Theta, (X, y)$ $\qquad$ $\boxed{\mathsf{C}^{\mathsf{tm}}_{\mathsf{R}}}$ $\quad \Sigma ::= \bullet \mid \Sigma, (x, y)$

$$\frac{(X,y) \in \Theta}{\Theta \vdash X \sim y} \qquad \frac{\Theta \vdash A \sim a \qquad \Theta \vdash B \sim b}{\Theta \vdash A \to B \sim \Pi y{:}a.\, b}\ x \notin \Theta \qquad \frac{\Theta, (X,y) \vdash A \sim a}{\Theta \vdash \forall X.\, A \sim \Pi y{:}*.\, a}\ X, y \notin \Theta$$

$$\frac{(x,y) \in \Sigma}{\Theta; \Sigma \vdash x \approx y} \qquad \frac{\Theta; \Sigma \vdash s \approx a \qquad \Theta; \Sigma \vdash t \approx b}{\Theta; \Sigma \vdash s\, t \approx a\, b} \qquad \frac{\Theta; \Sigma \vdash s \approx a \qquad \Theta \vdash A \sim b}{\Theta; \Sigma \vdash s\, A \approx a\, b}$$

$$\frac{\Theta \vdash A \sim a \qquad \Theta; \Sigma, (x,y) \vdash s \approx b}{\Theta; \Sigma \vdash \lambda x{:}A.\, s \approx \lambda y{:}a.\, b}\ x, y \notin \Theta, \Sigma \qquad \frac{\Theta, (X,y); \Sigma \vdash s \approx a}{\Theta; \Sigma \vdash \Lambda X.\, s \approx \lambda y{:}*.\, a}\ X, y \notin \Theta, \Sigma$$

Figure 3 Inductive characterisation of $\sim$ and $\approx$; Θ and Σ track related type and terms variables.

employs pure de Bruijn syntax and parallel substitutions. Meanwhile, both Abella and Beluga allow us to work with higher-order abstract syntax (HOAS), albeit in two rather different background logics.

While the comparison of proofs from different systems in terms of code lines is only marginally meaningful, we were surprised to find that all three developments each take approximately 500 loc, with Beluga slightly on the shorter side and Coq somewhat on the longer. This, however, only covers establishing the correspondence itself. The systems require vastly different amounts of code to establish the separate meta theories for the two discussed type systems, due to different levels of background support.

Contributions of the paper:

1. We present and compare three different machine checked formalisations of the technically intricate reduction of typability from System F to the PTS $\lambda 2$ and vice versa.[1]
2. We propose that our equivalence proof serves as a benchmark for reasoning about and relating multiple type systems and languages involving variable binding. The key aspect of this benchmark is the representation and manipulation of contexts that track multiple kinds of information and exhibit complex dependency structures. As such it can be seen as a complement to the benchmarks proposed in [11, 10] and the POPLmark challenge [2].

2 Equivalence

The core challenge of the proof is the fact that F clearly distinguishes types and terms with separate syntactic sorts, Ty_{F} and Tm_{F} respectively, while $\lambda 2$ merges these into a single syntactic sort Tm_{λ}. The distinction still exists in $\lambda 2$ but it is semantically imposed through the type system, rather than at the level of syntax. Further consequences are the existence of two variable scopes in F, with separate abstraction and application mechanisms, while the same concepts are uniformly represented in $\lambda 2$ for a single variable scope. This extends to the formation of function spaces as well. For an in-depth discussion of the mismatches between the two systems see [18]. In essence the two systems differ in how explicit and readily available certain structural properties are. One half of the proof will thus have to reestablish implicit structures, which, as one would expect, is harder than removing it. This will lead to a certain asymmetry in proof effort for seemingly symmetrical lemma statements.

The basic idea of the proof presented here is to construct two relations $\sim$ and $\approx$ that put the types and respectively the terms of the two languages in correspondence (see Fig. 3). To

[1] The accompanying developments can be found at `https://www.ps.uni-saarland.de/extras/fscd17/`

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

$$\frac{x < N}{N \vdash^{\mathsf{ty}}_{\mathsf{F}} x_{\mathsf{ty}}} \qquad \frac{N \vdash^{\mathsf{ty}}_{\mathsf{F}} A \quad N \vdash^{\mathsf{ty}}_{\mathsf{F}} B}{N \vdash^{\mathsf{ty}}_{\mathsf{F}} A \to B} \qquad \frac{N+1 \vdash^{\mathsf{ty}}_{\mathsf{F}} A}{N \vdash^{\mathsf{ty}}_{\mathsf{F}} \forall. A} \qquad \frac{\Gamma_x = A \quad N \vdash^{\mathsf{ty}}_{\mathsf{F}} A}{N; \Gamma \vdash^{\mathsf{tm}}_{\mathsf{F}} x_{\mathsf{tm}} : A} \qquad \frac{N; \Gamma \vdash^{\mathsf{tm}}_{\mathsf{F}} s : \forall. A \quad N \vdash^{\mathsf{ty}}_{\mathsf{F}} B}{N; \Gamma \vdash^{\mathsf{tm}}_{\mathsf{F}} s\,B : A[B \cdot \mathsf{id}]}$$

$$\frac{N+1; \Gamma[+1] \vdash^{\mathsf{tm}}_{\mathsf{F}} s : A}{N; \Gamma \vdash^{\mathsf{tm}}_{\mathsf{F}} \Lambda.\, s : \forall. A} \qquad \frac{N; \Gamma, A \vdash^{\mathsf{tm}}_{\mathsf{F}} s : B \quad N \vdash^{\mathsf{ty}}_{\mathsf{F}} A}{N; \Gamma \vdash^{\mathsf{tm}}_{\mathsf{F}} \lambda A.\, s : A \to B} \qquad \frac{N; \Gamma \vdash^{\mathsf{tm}}_{\mathsf{F}} s : A \to B \quad N; \Gamma \vdash^{\mathsf{tm}}_{\mathsf{F}} t : A}{N; \Gamma \vdash^{\mathsf{tm}}_{\mathsf{F}} s\,t : B}$$

Figure 4 System F – de Bruijn encoding in Coq; term variable contexts Γ are lists of types.

obtain the desired equivalence results, we have to demonstrate that the relations exhibit the following properties:

1. $\sim$ is functional and injective.
2. $\sim$ is left-total and type-formation preserving on the well-formed types of F.
3. $\sim$ is right-total and type-formation preserving on the propositions of $\lambda 2$. A proposition of $\lambda 2$ is any term $a : \mathsf{Tm}_\lambda$ such that $\vdash_\lambda a : *$ holds.
4. $\approx$ is functional and injective.
5. $\approx$ is left-total and typing preserving on the well-typed terms of F.
6. $\approx$ is right-total and typing preserving on the proofs of $\lambda 2$. A proof of $\lambda 2$ is any term $b : \mathsf{Tm}_\lambda$ such that $\vdash_\lambda b : a$ holds for a a proposition of $\lambda 2$.

We can now formulate, and easily prove, the following equivalences:

▶ **Theorem 1** (Reductions from F to $\lambda 2$)**.**

$$\vdash^{\mathsf{ty}}_{\mathsf{F}} A \iff \exists!a.\ \vdash A \sim a \ \wedge\ \vdash_\lambda a : *$$
$$\vdash^{\mathsf{tm}}_{\mathsf{F}} s : A \iff \exists!ba.\ \vdash s \approx b \ \wedge\ \vdash A \sim a \ \wedge\ \vdash_\lambda b : a \ \wedge\ \vdash_\lambda a : *$$

Proof. The forward directions are simply the corresponding left-to-right preservation and left-totality results of $\approx$ and $\sim$. Uniqueness follows from functionality. For the inverse direction we use preservation (here from right to left) and uniqueness. ◀

▶ **Theorem 2** (Reductions from $\lambda 2$ to F)**.**

$$\vdash_\lambda a : * \iff \exists!A.\ \vdash A \sim a \ \wedge\ \vdash^{\mathsf{ty}}_{\mathsf{F}} A$$
$$\vdash_\lambda b : a \ \wedge\ \vdash_\lambda a : * \iff \exists!sA.\ \vdash s \approx b \ \wedge\ \vdash A \sim a \ \wedge\ \vdash^{\mathsf{tm}}_{\mathsf{F}} s : A$$

Proof. Dual to the previous result. ◀

For the remainder of the paper, we focus on how F, $\lambda 2$ and the two relations $\sim$ and $\approx$ are represented in our three proof systems, and how the various properties of the relations are obtained.

3 Coq

For the Coq proof we reuse the pure de Bruijn encoding of the two systems and the corresponding language local meta theory developed in [18], with major support from the Autosubst framework [27]. The language definitions are given in Figs. 4 and 5. The main feature of de Bruijn syntax is the absence of variable names. Variables are instead represented as numerical indices, where n references the nth enclosing binder of the correct scope. Dangling references represent free variables that instead reference positions in an enclosing context, indexed from right to left. Note that we keep the dot around as a notational

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

$$\frac{}{0 : a[+1] \in_\lambda \Psi, a} \qquad \frac{x : a \in_\lambda \Psi}{(x+1) : a[+1] \in_\lambda \Psi, b} \qquad \frac{}{\Psi \vdash_\lambda * : \square} \qquad \frac{x : a \in_\lambda \Psi \quad \Psi \vdash_\lambda a : u}{\Psi \vdash_\lambda x : a}$$

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

Figure 5 $\lambda 2$ – de Bruijn encoding in Coq; dependent contexts Ψ are lists of terms.

device to uniformly indicate the presence of a binding constructor, even if nothing remains to the left of it (e.g. $\Lambda.\, s$). It marks the precise spot where substitutions and indices have to be adjusted. The application of a parallel substitution σ to a term s is written $s[\sigma]$ where σ is a function from $\mathbb{N}$ that acts on all free variables of s at once. The Autosubst framework provides a normalisation procedure for such terms with applied substitutions. The existence of computable normal forms was demonstrated in [26]. Context morphism lemmas (CML) are a useful proof device to reason about judgements over pure de Bruijn syntax. In the following we will only focus on those aspects that have an immediate impact on our present proof. For an in-depth discussion of the interaction of pure de Bruijn syntax, parallel substitutions and CMLs we refer to [8, 26, 27, 1, 15, 18].

In the definition of F in Fig. 4, observe how the type variable context Δ degenerates to a plain natural number N. It is taken as an exclusive upper bound to the admissible type variable indices, hence $N = 0$ represents the empty context. The term variable context Γ is simply a list of types, since free variables are coded as context positions. The β-substitution $B \cdot \mathsf{id}$ used in the type specialisation rule maps the free index 0 to B and lowers all other indices by 1.

We further observe that for $\lambda 2$, defined in Fig. 5, context lookup is characterised inductively: $x : a \in_\lambda \Psi$. The need for this arises from the fact that the PTS contexts are dependent as well as the more general issue that in a de Bruijn setting, terms are not stable under context modifications. Hence, upon extraction of a term a from context Γ, all free variables of a have to be adjusted by an amount that depends on the position of a in Γ. The given inductive characterisation elegantly handles this complication.

Let us now consider the first equivalence proof. Similar to the way typing contexts are explicitly represented as lists of terms or types, we are going to track explicitly which variables are related in our definition of our relations $\sim^R$ and $\approx^R_S$, where the relational parameters R and S correspond to the contexts Θ and Σ from Fig. 3:

$$\frac{x\,R\,y}{x_{\mathsf{ty}} \sim^R y} \qquad \frac{A \sim^R a \quad B \sim^{R^{\Uparrow}} b}{A \to B \sim^R \Pi a.\, b} \qquad \frac{A \sim^{R^{\mathsf{ext}}} a}{\forall.\, A \sim^R \Pi *.\, a} \qquad \frac{x\,S\,y}{x_{\mathsf{tm}} \approx^R_S y}$$

$$\frac{s \approx^R_S a \quad t \approx^R_S b}{s\,t \approx^R_S a\,b} \qquad \frac{s \approx^R_S a \quad A \sim^R b}{s\,A \approx^R_S a\,b} \qquad \frac{A \sim^R a \quad s \approx^{R^{\Uparrow}}_{S^{\mathsf{ext}}} b}{\lambda A.\, s \approx^R_S \lambda a.\, b} \qquad \frac{s \approx^{R^{\mathsf{ext}}}_{S^{\Uparrow}} a}{\Lambda.\, s \approx^R_S \lambda *.\, a}$$

The parameters R and S track pairs of indices of type, and respectively, term variables. We technically represent them as lists of type $\mathsf{list}\,(\mathsf{var} \times \mathsf{var})$ and use $x\,R\,y$ to denote that the pair (x, y) is in R. The interesting part of this definition is how these auxiliary parameters have to be modified when binders are traversed, which we denoted above by $R^{\Uparrow}$ and R^{ext}. In order to precisely define these operations, let us recall the required action on a parallel

substitution σ that is pushed underneath a binder:

$$(\forall.\, A)[\sigma] \qquad \longmapsto \qquad \forall.\, A[0 \cdot \sigma \circ +1]$$

The $[0 \cdot \_]$ part ensures that any index referencing the presently traversed binder is preserved as such. Meanwhile the $[\_ \cdot \sigma \circ +1]$ part ensures that every index $n+1$ is mapped to $\sigma(n)[+1]$, where the $+1$ ensures that no free variables in the range of σ are accidentally captured by the traversed binder.

In our correspondence proof we traverse binders *almost* in lockstep. For the simple cases where we have a binder on both sides of the relation, and moreover, the bound variables actually correspond according to the information tracked in R, we define, analogously to the binder traversal for substitutions:

$$R^{\mathsf{ext}} := (0,0) :: \mathsf{bimap}\,(+1)\,(+1)\,R$$

where $\mathsf{bimap}\,f\,g\,R$ simply applies f to all left projections of R and g to all right projections.

The other possible scenario has a binder on the $\lambda 2$ side that has no counterpart in F with respect to the contextual information in R, like the *not-really dependent* PTS product that corresponds to an arrow type in F. As a consequence of this spurious binding, the $\lambda 2$ indices in R have to be shifted relative to their F counterparts. This one-sided index adjustment is obtained with

$$R^{\Uparrow} := \mathsf{bimap}\,\mathsf{id}\,(+1)\,R$$

▶ **Fact 3.** *Both R^{ext} and $R^{\Uparrow}$ preserve injectivity and functionality of R.* ◀

▶ **Lemma 4.** *The type relation $\sim^R$ is injective/functional, whenever R is injective/functional.*

Proof. Straightforward inductions using Fact 3. ◀

To obtain the same result for $\approx_S^R$ we additionally rely on R and S having disjoint ranges, that is, no PTS variable is considered related to both a type and a term variable. We denote this by $R \| S$.

▶ **Fact 5.** *The property $R\|S$ is preserved under extending one relation and shifting the other, that is w.l.o.g.: $R\|S \implies R^{\Uparrow}\|S^{\mathsf{ext}}$.* ◀

▶ **Lemma 6.** *Disjointedness of ranges lifts from variable relations R and S to $\sim^R$ and $\approx_S^R$:*

$$R\|S \implies A \sim^R a \implies s \approx_S^R a \implies \bot$$

Proof. By induction on $A \sim^R a$ and discriminating on $s \approx_S^R a$, using Fact 5. ◀

▶ **Lemma 7.** *The term relation $\approx_S^R$ is functional, whenever R and S are functional. It is injective, whenever R and S are injective and $R\|S$ holds.*

Proof. Straightforward inductions. Injectivity relies on the premise $R\|S$ and Lemma 6 to discharge non-matching applications. Subderivations for $\sim^R$ are handled with Lemma 4. ◀

Proving the left and right totality and preservation results is slightly more interesting, as we have to generalise to open judgements and non-empty contexts. We achieve this with suitable proof invariants that are adapted from the notion of *generalised context morphisms* laid out in [18]. The key difference is that instead of a renaming ξ that maps from one context to another, we here consider a relation on variables that places two contexts in correspondence. All invariants are set up such that they vacuously hold when the initial context happens to be empty. We start with type formation and the direction from F to $\lambda 2$:

$$N \xrightarrow{R} \Psi := \forall x < N.\ \exists y.\ x\,R\,y \ \wedge\ y{:}* \in_\lambda \Psi$$

▶ **Fact 8.** *The invariant* $N \xrightarrow{R} \Psi$ *is preserved under corresponding extensions:*

$$N \xrightarrow{R} \Psi \implies N \xrightarrow{R^{\Uparrow}} \Psi, a \qquad N \xrightarrow{R} \Psi \implies N+1 \xrightarrow{R^{\mathsf{ext}}} \Psi, *$$ ◀

▶ **Lemma 9.** *The type-relation* $\sim^R$ *is left-total and preserves type formation:*

$$N \vdash^{\mathsf{ty}}_{\mathsf{F}} A \implies \forall R\, \Psi.\ N \xrightarrow{R} \Psi \implies \exists a.\ A \sim^R a \wedge \Psi \vdash_{\lambda} a : *$$

Proof. By induction on $N \vdash^{\mathsf{ty}}_{\mathsf{F}} A$. The two binder cases use Fact 8. ◀

For the inverse direction we establish preservation of type formation and right totality along the following invariant:

$$N \xleftarrow{R} \Psi := \forall y.\ y{:}* \in_{\lambda} \Psi \implies \exists x.\ x\, R\, y \wedge x < N$$

▶ **Fact 10.** *The invariant* $N \xleftarrow{R} \Psi$ *is preserved under corresponding extensions:*

$$N \xleftarrow{R} \Psi \implies \Psi \vdash_{\lambda} a : * \implies N \xleftarrow{R^{\Uparrow}} \Psi, a \qquad N \xleftarrow{R} \Psi \implies N+1 \xleftarrow{R^{\mathsf{ext}}} \Psi, *$$ ◀

▶ **Lemma 11.** *The type-relation* $\sim^R$ *is right-total and preserves type formation:*

$$\Gamma \vdash_{\lambda} a : * \implies \forall R\, N.\ N \xleftarrow{R} \Gamma \implies \exists A.\ A \sim^R a \wedge N \vdash^{\mathsf{ty}}_{\mathsf{F}} A$$

Proof. Induction on $\Gamma \vdash_{\lambda} a : *$, using Fact 10. One complication is the disambiguation of a given PTS-product $\Pi a.\, b$, where a is known to live in some universe u. Discriminating on u allows us to correctly choose either an arrow type $A \to B$, or a universal quantification $\forall.\, B$. Further requirements are the degeneracy of the universe $\Box$ ($*$ is the only inhabitant of $\Box$), as well as propagation and substitutivity for $\vdash_{\lambda}$. ◀

The preservation and totality results for $\approx^R_S$ make the overhead for explicitly tracking contextual information most apparent. Since some of the typing rules for applications ascribe types that are constructed from a non-trivial substitution operation, we require substitutivity results for the judgements under consideration; in particular, β-substitutivity for $\sim^R$:

▶ **Fact 12.** *The type relation* $\sim^R$ *is closed under* β*-substitutions:*

$$B \sim^R b \implies A \sim^{R^{\mathsf{ext}}} a \implies A[B \cdot \mathsf{id}] \sim^R a[b \cdot \mathsf{id}]$$ ◀

The proof of this fact is a lengthy but straightforward construction that first generalises the two concrete β-substitutions to arbitrary parallel substitutions σ and τ. The result is still not provable directly, as closure of $\sim^R$ under weakening is needed. This in turn is generalised to a statement for arbitrary renamings ξ and ζ in place of σ and τ. In essence we establish a CML for $\sim^R$.

We can now tackle the technically most intricate part of the proof. The invariant for preservation of typing from F to $\lambda 2$ is

$$\Gamma \xrightarrow[S]{R} \Psi := \forall x A.\ \Gamma_x = A \implies \exists y a.\ A \sim^R a \wedge x\, S\, y \wedge y{:}a \in_{\lambda} \Psi$$

▶ **Fact 13.** *The invariant* $\Gamma \xrightarrow[S]{R} \Psi$ *is preserved under corresponding extensions:*

$$\Gamma \xrightarrow[S]{R} \Psi \implies A \sim^R a \implies \Gamma, A \xrightarrow[S^{\mathsf{ext}}]{R^{\Uparrow}} \Psi, a \qquad \Gamma \xrightarrow[S]{R} \Psi \implies \Gamma[+1] \xrightarrow[S^{\Uparrow}]{R^{\mathsf{ext}}} \Psi, *$$ ◀

$$\frac{A\,\mathsf{ty} \qquad B\,\mathsf{ty}}{(A \to B)\,\mathsf{ty}} \qquad \frac{\mathbf{\Pi} x.\ x\,\mathsf{ty} \Rightarrow A\langle x\rangle\,\mathsf{ty}}{(\forall.\ A)\,\mathsf{ty}} \qquad \frac{s :_F \forall.\ B \qquad A\,\mathsf{ty}}{s\,A :_F B\langle A\rangle} \qquad \frac{s :_F A \to B \qquad t :_F A}{s\,t :_F B}$$

$$\frac{\mathbf{\Pi} x.\ x\,\mathsf{ty} \Rightarrow s\langle x\rangle :_F A\langle x\rangle}{\Lambda.\ s :_F \forall.\ A} \qquad \frac{A\,\mathsf{ty} \qquad \mathbf{\Pi} x.\ x :_F A \Rightarrow s\langle x\rangle :_F B}{\lambda A.\ s :_F A \to B}$$

Figure 6 HOAS specification of F in Abella.

▶ **Lemma 14.** *The term relation $\approx_S^R$ is left-total and preserves typing.*

$$N;\Gamma \vdash_{\mathsf{F}}^{\mathsf{tm}} s : A \implies \forall RS\Psi.\ R\ \mathsf{func} \implies N \xrightarrow{R} \Psi \implies \Gamma \xrightarrow[S]{R} \Psi \implies$$
$$\exists ba.\ A \sim^R a \wedge s \approx_S^R b \wedge \Psi \vdash_\lambda b : a \wedge \Psi \vdash_\lambda a : *$$

Proof. By induction on $N;\Gamma \vdash_{\mathsf{F}}^{\mathsf{tm}} s : A$. Both the invariant $\Gamma \xrightarrow[S]{R} \Psi$, as well as Lemma 9, are used to obtain related types in the variable case. Functionality of R allows us to equate these. ◀

The final part is the preservation of typing from $\lambda 2$ to F. Here we use:

$$\Gamma \xleftarrow[S]{R} \Psi := \forall ya.\ y{:}a \in_\lambda \Psi \implies \Psi \vdash_\lambda a : * \implies \exists xA.\ A \sim^R a \wedge x\,S\,y \wedge x{:}A \in \Gamma$$

▶ **Fact 15.** *The invariant $\Gamma \xleftarrow[S]{R} \Psi$ is preserved under corresponding extensions:*

$$\Gamma \xleftarrow[S]{R} \Psi \implies A \sim^R a \implies \Gamma, A \xleftarrow[S^{\mathsf{ext}}]{R^{\Uparrow}} \Psi, a \qquad \Gamma \xleftarrow[S]{R} \Psi \implies \Gamma[+1] \xleftarrow[S^{\Uparrow}]{R^{\mathsf{ext}}} \Psi, *$$ ◀

▶ **Lemma 16.** *The term relation $\approx_S^R$ is right-total and preserves typing:*

$$\Psi \vdash_\lambda a : * \implies \Psi \vdash_\lambda b : a \implies \forall RSN\Gamma.\ R\ \mathsf{inj} \implies N \xleftarrow{R} \Psi \implies \Gamma \xleftarrow[S]{R} \Psi \implies$$
$$\exists sA.\ A \sim^R a \wedge s \approx_S^R b \wedge N;\Gamma \vdash_{\mathsf{F}}^{\mathsf{tm}} s : A \wedge N \vdash_{\mathsf{F}}^{\mathsf{ty}} A$$

Proof. By induction on $\Psi \vdash_\lambda b : a$. The cases are mostly analogue to the previous result. Injectivity of R is required for the variable case. Note that we need to discriminate on the universes of product domains again (cf. Lemma 11), here to disambiguate the unified abstractions and applications correctly. ◀

At this point we make an interesting observation. The above proof demonstrates that the CML proof pattern not only generalises to a multi system setting [18] but also to relations in place of functional correspondences. This is what allowed us to quickly generate all the contextual invariants need for the various results.

4 Abella

Abella supports the use of higher-order abstract syntax (HOAS) [20]. The main idea is to delegate variable binding at the object level to binding at the meta level. Take for example the term constructor $\mathsf{lam} : (\mathsf{tm} \to \mathsf{tm}) \to \mathsf{tm}$ that yields an abstraction of the untyped λ-calculus. The s in $\mathsf{lam}\,s$ is a function of the meta level. Substitution at the object level is implemented as application at the meta level, which we denote by $s\langle t\rangle$ to distinguish it from the various object level applications.

$$\frac{}{\mathcal{U}\,\Box} \qquad \frac{}{\mathcal{U}\,*} \qquad \frac{}{* :_\lambda \Box} \qquad \frac{a :_\lambda \Pi c.\, d \qquad b :_\lambda c}{a\,b :_\lambda d\langle b\rangle} \qquad \frac{a :_\lambda u \qquad \mathcal{U}\,u \qquad \mathbf{\Pi} x.\ x :_\lambda a \Rightarrow b\langle x\rangle :_\lambda *}{\Pi a.\, b :_\lambda *}$$

$$\frac{a :_\lambda u \qquad \mathcal{U}\,u \qquad \mathbf{\Pi} x.\ x :_\lambda a \Rightarrow c\langle x\rangle :_\lambda * \qquad \mathbf{\Pi} x.\ x :_\lambda a \Rightarrow b\langle x\rangle :_\lambda c\langle x\rangle}{\lambda a.\, b :_\lambda \Pi a.\, c}$$

Figure 7 HOAS specification of $\lambda 2$ in Abella.

Note that HOAS is not inductive, due to the negative occurrences of expression types. It also relies on intensional function spaces at the meta level in order to structurally analyse functions. Both aspects prevent Coq from natively supporting HOAS.

Abella is a system designed around a two-level logic approach. The lower *specification level*, essentially verbatim λProlog, is used to encode the object languages and their associated judgements. We then reason about these encoding at the *meta level*, using the logic $\mathcal{G}$, which is the intuitionistic predicative fragment of Church's simple type theory, extended with natural induction, (co)inductive predicates and nominal quantification ($\nabla x.\, s$). The axioms of $\mathcal{G}$ ensure that a ∇-quantified identifier is fresh for everything bound above it. We are going to use these freshness guarantees to faithfully represent object level variables.

The two levels are connected with a special inductive predicate that embeds the derivation of a λProlog judgement J from hypotheses $L = I_0, \ldots, I_n$ into $\mathcal{G}$, written $\{L \vdash J\}$, or simply $\{J\}$ in the absence of assumptions. Note that λProlog supports hypothetical ($J_1 \Rightarrow J_2$) and locally quantified ($\mathbf{\Pi} x.\ J[x]$) premises, which the embedding treats as follows:

$$\{L \vdash J_1 \Rightarrow J_2\} \quad \rightsquigarrow \quad \{L, J_1 \vdash J_2\} \qquad \{L \vdash \mathbf{\Pi} x.\ J[x]\} \quad \rightsquigarrow \quad \nabla x.\{L \vdash J[x]\}$$

For the quantification case, observe that the context L is usually bound at the outermost level, hence x is guaranteed to be fresh for L, satisfying the usual side condition for context extension rules. This process is often referred to as *mobility of binders*: object level binders are represented using λProlog quantification, which in turn is mapped to ∇-quantification in $\mathcal{G}$ and eventually opened with nominal constants n_i.

▶ **Fact 17.** *The embedding $\{L \vdash J\}$ satisfies cut and a nominal instantiation principle:*

$$\{L \vdash I\} \implies \{L, I \vdash J\} \implies \{L \vdash J\} \qquad \text{(cut)}$$

$$\forall s : \mathsf{typeof}\, n_i.\ \{L[n_i] \vdash J[n_i]\} \implies \{L[s] \vdash J[s]\} \qquad \text{(inst)}$$

Both are exposed as proof tactics to the user. ◀

Note that Fact 17 provides substitutivity for the various judgements of our object languages. It is also worth noting that only the judgements, but not the specification level types like Tm_λ and Ty_F, are embedded, so we cannot induct directly on our syntax definitions.

At this point it should be straightforward to understand the HOAS representation of our two systems given in Figs. 6 and 7. For F, the judgements $A\,\mathsf{ty}$ and $s :_F A$ encode type formation and respectively typing. For $\lambda 2$, the judgement $\mathcal{U}\,u$ is used to recognise universes, while $a :_\lambda b$ represents PTS typing.

As soon as we try to prove structural results about these definitions we are faced with a problem. Consider the following inversion principle for arrow type formation in F:

$$\{L \vdash A \to B\,\mathsf{ty}\} \implies \{L \vdash A\,\mathsf{ty}\} \wedge \{L \vdash B\,\mathsf{ty}\}$$

The premise may hold due to backchaining and $A \to B\,\mathsf{ty} \in L$. The problem is that L is too general and may contain arbitrary judgements, not even necessarily related to

type formation. Hence we need to somehow constrain L to only contain judgements of the form $n_i\,\mathsf{ty}$, where the n_i are nominals representing variables. Similarly we want to constrain typing contexts to only contain judgements of the form $n_i\,\mathsf{ty}$ or $n_i :_F A$. We specify this notion of well-formed contexts with auxiliary inductive $\mathcal{G}$-predicates. The well-formedness predicate for F typing contexts, written $\mathbb{C}_F^{\mathsf{tm}}(-)$, is defined as:

$$\frac{}{\mathbb{C}_F^{\mathsf{tm}}(\bullet)} \qquad \frac{\mathbb{C}_F^{\mathsf{tm}}(L)}{\mathbb{C}_F^{\mathsf{tm}}(L, x\,\mathsf{ty})}\; x \notin L \qquad \frac{\mathbb{C}_F^{\mathsf{tm}}(L) \qquad \{L \vdash A\,\mathsf{ty}\}}{\mathbb{C}_F^{\mathsf{tm}}(L, x :_F A)}\; x \notin L, A$$

The freshness conditions are implemented by locally ∇-quantifying the respective variable. Next we have to pair this definition with an inversion principle that reveals the structure and freshness properties of any $J \in L$ with $\mathbb{C}_F^{\mathsf{tm}}(L)$. At this point we are able to add $\mathbb{C}_F^{\mathsf{ty}}(L)$, defined analogously to $\mathbb{C}_F^{\mathsf{tm}}(L)$, as an extra premise to our inversion principle and then discard the spurious context extraction. This relies on the fact that $\nabla x.(A \to B) \neq x$ holds in $\mathcal{G}$. A key aspect of formalising our equivalence result in Abella is the correct choice of well-formedness predicates and associated inversion Lemmas.

Let us now consider the second equivalence proof. The relations $\sim$ and $\approx$ are defined as:

$$\frac{A \sim a \qquad \mathbf{\Pi} x.\; B \sim b\langle x\rangle}{A \to B \sim \Pi a.\, b} \qquad \frac{\mathbf{\Pi} xy.\; x \sim y \Rrightarrow A\langle x\rangle \sim a\langle y\rangle}{\forall.\, A \sim \Pi *.\, a} \qquad \frac{s \approx a \qquad t \approx b}{s\,t \approx a\,b}$$

$$\frac{A \sim a \qquad \mathbf{\Pi} xy.\; x \approx y \Rrightarrow s\langle x\rangle \approx b\langle y\rangle}{\lambda A.\, s \approx \lambda a.\, b} \qquad \frac{\mathbf{\Pi} xy.\; x \sim y \Rrightarrow s\langle x\rangle \approx a\langle y\rangle}{\Lambda.\, s \approx \lambda *.\, a} \qquad \frac{s \approx a \qquad A \sim b}{s\,A \approx a\,b}$$

We complement this definition with a well-formedness predicate $\mathbb{C}_{\approx}(L)$, which ascertains that L only contains judgements of the form $n_i \sim n_j$ or $n_i \approx n_j$, together with suitable lookup lemmas. For technical reasons we do not define $\mathbb{C}_{\sim}(-)$ and instead prove a strengthening lemma which holds due to the inferred subordination ordering of the two relations:

$$\mathbb{C}_{\approx}(L) \implies \{L, x \approx y \vdash A \sim a\} \implies \{L \vdash A \sim a\}$$

Before we go on, it is interesting to consider the information encapsulated in $\mathbb{C}_{\approx}(L)$. The obvious part is that L contains exactly the same information about corresponding variables that we had to track in Coq with the auxiliary parameters R and S. In addition, since L only contains pairings of fresh nominals, we immediately obtain that L is functional and injective. Lifting these properties to $\sim$ and then to $\approx$ are routine inductions.

When it comes to the totality and preservation statements, things become more interesting. The remaining four lemma statements are generalised over three different contexts belonging to the three involved judgements. Not only do we require these contexts to be well-formed, but we also have to connect them. We achieve this with a single ternary well-formedness predicate, $\mathbb{C}_R(- \mid - \mid -)$, defined as follows:

$$\frac{}{\mathbb{C}_R(\bullet \mid \bullet \mid \bullet)} \qquad \frac{\mathbb{C}_R(L_F \mid L_{\approx} \mid L_{\lambda}) \quad x, y \notin L_i}{\mathbb{C}_R(L_F, x\,\mathsf{ty} \mid L_{\approx}, x \sim y \mid L_{\lambda}, y :_{\lambda} *)} \qquad \frac{\mathbb{C}_R(L_F \mid L_{\approx} \mid L_{\lambda}) \quad x, y \notin L_i, A, a \quad \{L_F \vdash A\,\mathsf{ty}\} \quad \{L_{\approx} \vdash A \sim a\} \quad \{L_{\lambda} \vdash a :_{\lambda} *\}}{\mathbb{C}_R(L_F, x :_F A \mid L_{\approx}, x \approx y \mid L_{\lambda}, y :_{\lambda} a)}$$

When a binder is traversed, this ensures that all three contexts are extended with the same freshly chosen variables x and y. The definition is accompanied by three extraction lemmas, one for each of the three involved contexts, which provide the associated judgements from the two other contexts. Recall that in Coq we had four separate invariants for the four preservation and totality lemmas. Here, this information is uniformly encoded in $\mathbb{C}_R(L_F \mid L_{\approx} \mid L_{\lambda})$ and we use it for all four proofs.

▶ **Lemma 18.** *The type relation $\sim$ is total from F to $\lambda 2$ and preserves type formation.*

$$\{L_F \vdash A\,\mathsf{ty}\} \implies \forall L_{\approx} L_{\lambda}.\ \mathbb{C}_R(L_F \mid L_{\approx} \mid L_{\lambda}) \implies \exists a.\ \{L_{\approx} \vdash A \sim a\} \wedge \{L_{\lambda} \vdash a :_{\lambda} *\}$$

Proof. By induction on $\{L_F \vdash A\,\mathsf{ty}\}$. ◀

As a trivial corollary we obtain: $\{A\,\mathsf{ty}\} \implies \exists a.\ \{A \sim a\} \wedge \{a :_{\lambda} *\}$.

The proofs of the remaining three preservation results are surprisingly analogue, so we will not go into further detail. It is worth pointing out, though, that we again require the degeneracy of the PTS universe $\Box$, as well as propagation for both F and $\lambda 2$ in various places. The HOAS setup allows us to obtain these easily.

5 Beluga

The programming and proof environment Beluga [23, 24] is another system that supports HOAS. Object languages are encoded in the logical framework LF [17], while proofs about these are expressed as total programs in contextual modal type theory, Beluga's reasoning logic. The programs analyse LF derivation trees using pattern matching and higher-order unification. In Beluga, it is necessary to give programs (i.e. proof terms) explicitly, which are proof checked as part of type checking. This stands in contrast to Coq and Abella, both of which are interactive systems with a tactic language for proof term construction.

The biggest difference, however, lies with Beluga's treatment of object level variables. In particular, at the outermost level, there is no such thing as a free object variable. This is in stark contrast to Coq's dangling de Bruijn indices and Abella's global nominal constants. In Beluga we instead deal with *contextual objects*, written $[\,\Gamma \vdash K\,]$, that is objects K (like types, terms, typing derivations) paired with contexts Γ in which they are meaningful [19, 22].

As an example, consider the function type $X \to X$ where X is a free type variable. This function type is ill-formed under the empty type variable context $\Delta = \emptyset$. In Coq and Abella we can express this type, as $0_{\mathsf{ty}} \to 0_{\mathsf{ty}}$ and respectively $n_0 \to n_0$. We can then show that assuming well-formedness under the empty context entails absurdity, $0 \overset{\mathsf{ty}}{\mid_{\mathsf{F}}} 0_{\mathsf{ty}} \to 0_{\mathsf{ty}} \implies \bot$ and $\{\bullet \vdash n_0 \to n_0\,\mathsf{ty}\} \implies \bot$. Meanwhile in Beluga, we observe that the contextual object $[\,\bullet \vdash x \to x\,]$ is not even syntactically well-formed, since $x \notin \bullet$. An immediate consequence of this is that the type formation judgement for F, which ensures that the context is covering all free type variables, becomes redundant. Hence Beluga's definition of F is obtained from Fig. 6 by removing all references to type formation. The definition of $\lambda 2$ is identical to the one for Abella (Fig.7).

Recall that context management was completely manual in Coq. Each judgement required well-chosen generalisations and custom invariants to accurately track contextual information. In Abella the situation was noticeably better, as contexts at the object level were kept implicit and handled by the system. At the reasoning level they did, however, surface as explicit, unstructured sequences of judgements. The desired contextual structure then had to be imposed with auxiliary predicates, together with copious amounts of inversion lemmas.

Beluga contexts, on the other hand, are sequence of not necessarily homogeneous declarations. Each declaration can depend on prior declarations and encapsulate multiple pieces of related information using a dependent record. Contexts are first class citizens and *context schema* ascription, $\Gamma : S$, is used to ensure that a given Γ satisfies certain structural constraints.

Schemas are Beluga's main device to enforce invariants on contextual information. We use propagation for $\lambda 2$ as an example. The requisite schema is:

$$S_{\lambda W} := [x{:}\ \mathsf{Tm}_{\lambda},\ x :_{\lambda} *] + [x{:}\ \mathsf{Tm}_{\lambda},\ x :_{\lambda} a,\ a :_{\lambda} *]$$

Note how it separates PTS variables into type and term variables via the associated typing information, which already imposes the necessary semantic contextual information. The proof of propagation is straightforward. We implement a total recursive function k satisfying:

$$k \,:\, \forall \Gamma{:}S_{\lambda W}.\, [\,\Gamma \vdash a :_\lambda b\,] \implies [\,\Gamma \vdash \mathsf{type\_correct}\, b\,]$$

The contextual predicate $[\,\Gamma \vdash \mathsf{type\_correct}\, b\,]$ encodes that b is either $\Box$ or it can be typed with some universe u. We pattern match $\mathcal{D} : [\,\Gamma \vdash a :_\lambda b\,]$ and obtain seven cases. The first four are structural, recursively descending into sub-derivations. The only part that is non-obvious is the traversal of binders, where various pieces of information are added to the context. These have to be packaged into declaration blocks in order to satisfy $S_{\lambda W}$ for the recursive call. More interesting though are the three base cases. When we compare the matched type against $S_{\lambda W}$, we observe three ways in which $\mathcal{D}$ could have been obtained from the context. The first and third are trivial as they unify b with $*$ which can be typed with the universe $\Box$. For the remaining case we know that b can be typed with the universe $*$, contextual information that was packaged together with the matched judgement. Note that throughout the construction we exploit that Beluga natively supports substitution into parametric sub-derivations.

We now start with our third equivalence proof. The definitions of $\sim$ and $\approx$ exactly coincide with Abella. We are going to primarily concern ourselves with the schemas, which best illustrate Beluga's tracking of contextual information.

We begin with the functionality and injectivity properties of $\sim$ and $\approx$. Since equality is not native in Beluga we have to define equality predicates for each syntactic sort to express our statements. The tightest invariants that hold for the rules defining $\sim$ and $\approx$ can be expressed with the following context schemas:

$$S_\sim := [x{:}\ \mathsf{Ty_F},\, y{:}\ \mathsf{Tm}_\lambda,\, x \sim y] + [y{:}\ \mathsf{Tm}_\lambda]$$
$$S_\approx := [x{:}\ \mathsf{Ty_F},\, y{:}\ \mathsf{Tm}_\lambda,\, x \sim y] + [x{:}\ \mathsf{Tm_F},\, y{:}\ \mathsf{Tm}_\lambda,\, x \approx y]$$

Due to the subordination ordering of $\sim$ and $\approx$, Beluga is capable of automatically strengthening from $S_\approx$ to $S_\sim$, and weaken vice versa (see also [28]).

▶ **Lemma 19.** *There exist total recursive functions $f_{\mathsf{ty}}, f_{\mathsf{tm}}, i_{\mathsf{ty}}$ and i_{tm}, satisfying*

$$f_{\mathsf{ty}} \,:\, \forall \Gamma{:}S_\sim.\, [\,\Gamma \vdash A \sim a\,] \implies [\,\Gamma \vdash A \sim a'\,] \implies [\,\Gamma \vdash a =_\lambda a'\,]$$
$$i_{\mathsf{ty}} \,:\, \forall \Gamma{:}S_\sim.\, [\,\Gamma \vdash A \sim a\,] \implies [\,\Gamma \vdash A' \sim a\,] \implies [\,\Gamma \vdash A =^{\mathsf{ty}}_{\mathsf{F}} A'\,]$$
$$f_{\mathsf{tm}} \,:\, \forall \Gamma{:}S_\approx.\, [\,\Gamma \vdash s \approx a\,] \implies [\,\Gamma \vdash s \approx a'\,] \implies [\,\Gamma \vdash a =_\lambda a'\,]$$
$$i_{\mathsf{tm}} \,:\, \forall \Gamma{:}S_\approx.\, [\,\Gamma \vdash s \approx a\,] \implies [\,\Gamma \vdash s' \approx a\,] \implies [\,\Gamma \vdash s =^{\mathsf{tm}}_{\mathsf{F}} s'\,]$$

Proof. Each by induction on the first premise and pattern matching on the second. In the variable cases we have matched against two context records r and r'. Since x and y are local to r and one of them is shared between the two matched records, unification infers $r = r'$, closing the case. Note that f_{tm} and i_{tm} contain calls to f_{ty} and respectively i_{ty}, which relies on context strenghtening. For i_{tm} we also again require disjointedness of the two relations, which is easily obtainable under contexts satisfying $S_\approx$. ◀

For the remaining four totality and preservation proofs we have to deal with two complications. First, we have to remove assumptions and conclusions referring to F type formation.

Second, we have to define predicates that capture the existential nature of the four conclusions, including relevant typing information, with custom predicates. We have, for example,

$$\mathsf{exists\_rel\_proof}\ s\ A \iff \exists ba.\ s \approx b \wedge A \sim a \wedge b :_\lambda a \wedge a :_\lambda *$$

The required context schemas are quite involved:

$$\begin{aligned}
S^{\rightarrow}_{\sim W} &:= [x{:}\ \mathsf{Ty_F},\ y{:}\ \mathsf{Tm}_\lambda,\ x \sim y,\ y :_\lambda *] + [y{:}\ \mathsf{Tm}_\lambda,\ y :_\lambda a] \\
S^{\leftarrow}_{\sim W} &:= [x{:}\ \mathsf{Ty_F},\ y{:}\ \mathsf{Tm}_\lambda,\ x \sim y,\ y :_\lambda *] + [y{:}\ \mathsf{Tm}_\lambda,\ y :_\lambda a,\ A \sim a,\ a :_\lambda *] \\
S_{\approx W} &:= [x{:}\ \mathsf{Ty_F},\ y{:}\ \mathsf{Tm}_\lambda,\ x \sim y,\ y :_\lambda *] + [x{:}\ \mathsf{Tm_F},\ y{:}\ \mathsf{Tm}_\lambda,\ x \approx y,\ x :_F A,\ y :_\lambda a,\ A \sim a]
\end{aligned}$$

▶ **Lemma 20.** *There exist total recursive functions* $p^{\rightarrow}_{\sim}, p^{\leftarrow}_{\sim}, p^{\rightarrow}_{\approx}$ *and* $p^{\leftarrow}_{\approx}$*, satisfying*

$$\begin{aligned}
p^{\rightarrow}_{\sim} &: \forall \Gamma{:}S^{\rightarrow}_{\sim}.\ \forall A{:}[\Gamma \vdash \mathsf{Ty_F}].\ [\Gamma \vdash \mathsf{exists_rel_prop}\ A] \\
p^{\leftarrow}_{\sim} &: \forall \Gamma{:}S^{\leftarrow}_{\sim}.\ [\Gamma \vdash a :_\lambda *] \implies [\Gamma \vdash \mathsf{exists_rel_type}\ a] \\
p^{\rightarrow}_{\approx} &: \forall \Gamma{:}S_{\approx}.\ [\Gamma \vdash s :_F A] \implies [\Gamma \vdash \mathsf{exists_rel_proof}\ s\ A] \\
p^{\leftarrow}_{\approx} &: \forall \Gamma{:}S_{\approx}.\ [\Gamma \vdash b :_\lambda a] \implies [\Gamma \vdash a :_\lambda *] \implies [\Gamma \vdash \mathsf{exists_rel_term}\ b\ a]
\end{aligned}$$

Proof. The first is by induction on $A : [\Gamma \vdash \mathsf{Ty_F}]$ (recall that this was not possible in Abella), the others are by induction on the first premise. The proofs are quite technical but mostly straightforward. The construction of $p^{\leftarrow}_{\approx}$, needs $\lambda 2$ propagation. Interestingly, neither propagation in F nor the degeneracy of $\Box$ are needed, as unification automatically handles the respective occurrences. ◀

The most interesting part of the Beluga development appears to be the particularly rich structure and interdependencies of the various schemas. We would like to point out in particular, that while the schemas $S_\sim$ and $S_\approx$ could likely be inferred automatically by inspecting the involved type families, this does not appear to work for those schemas with auxiliary well-typedness assumptions (subscript W). This contradicts the common believe that schema inference should in principle always be possible.

The schemas can be further arranged in a hierarchy (Fig. 8). A context satisfying $S_\sim$ can always be weakened to one sitting lower in the hierarchy. The hierarchy also induces a strengthening relationship, going upwards, as long as the subordination order judgements under said contexts is respected.

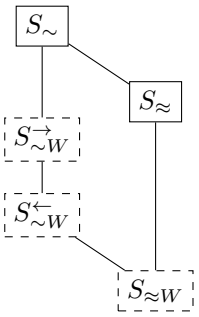

Figure 8 Hierarchy of Context Schemas

6 Conclusion

We have considered a technically interesting proof and demonstrated how various formalisation techniques deal with the arising intricacies. The development of three different formalisations allowed us to gain deep insights into the inherent complexities of the proof. In particular we were able to separate these from technical artefacts due to the chosen formalisation technique. Two examples of inherent complications are the not quite perfectly aligned binding structures and the missing typing information required to disambiguate the uniform PTS applications.

Our set of developments demonstrates, that the various formalisation techniques can be arranged in a hierarchy of abstraction layers. At the lowest level we have pure de Bruijn with a lot of representation freedom, which, however, has to be manged manually. Higher up in the hierarchy sit the HOAS techniques, which hide a lot of the technicalities and provide

a more meaningful abstraction. In comparison with Abella, Beluga appears to deliver the theoretically nicer interface, with the added features of contextual reasoning and the ability to perform inductions directly over the HOAS syntax. Practically though, both systems are relatively young with certain usability issues. Among the two, Abella's tactic language certainly gives it a head start.

We observe that our proof contains a number of challenges that, taken as a whole, constitute a nice benchmark for systems designed to reason about type systems. It tests in particular, how well multiple type systems with binding constructs can be brought into correspondence. We hence propose it as a complement to the POPLmark challenge [2] on the one hand, which solely tests how well a system can reason about a single type system, and the ORBI benchmarks [11, 10] which cover small-scale contextual reasoning. It is our believe that such a benchmark could be very useful to those who seek to develop or improve frameworks for reasoning about type systems and similar syntactic systems with variable binding and complex contextual information.

6.1 Future Work

We would like to continue this line of work into three largely orthogonal directions.

First, we would like to widen the scope of the benchmark itself and include a correspondence result for the computational behaviour of the two systems. Equi-reducebility is at least as interesting a problem as equi-typability, and likely to pose its own set of challenges. One of these is the fact that, prior to typing, $\lambda 2$ has a lot more β-redices than its stratified counterpart, hence some form of typing information will have to be tracked along with the relational assumptions. This will lead to new forms of contexts that are likely comparable in complexity to those discussed here.

Second, we would like to test further frameworks against our benchmark. One candidate is the locally nameless approach [3]. It provides an abstraction layer that sits somewhere between pure de Bruijn and HOAS. Our benchmark could test how stable this layer really is. Another is the HYBRID framework [9] that aims to bring HOAS to Coq. At present however, it is neither equipped with Abella's ∇ nor Beluga's contextual types, so it remains to be seen if it can handle our challenge. It would also be interesting to see how a system like Twelf [21] would handle the rich context structures.

Finally, we consider scaling the challenge both down to the simply typed λ-calculus, as well as up to F_ω, to obtain a better understanding of where exactly certain complications originate from.

References

1 Martín Abadi, Luca Cardelli, Pierre-Louis Curien, and Jean-Jacques Lévy. Explicit substitutions. *JFP*, 1(4):375–416, 1991.

2 Brian E. Aydemir, Aaron Bohannon, Matthew Fairbairn, J. Nathan Foster, Benjamin C. Pierce, Peter Sewell, Dimitrios Vytiniotis, Geoffrey Washburn, Stephanie Weirich, and Steve Zdancewic. Mechanized metatheory for the masses: The POPLmark challenge. In *TPHOL*, LNCS 3603, pages 50–65. Springer, 2005.

3 Brian E. Aydemir, Arthur Charguéraud, Benjamin C. Pierce, Randy Pollack, and Stephanie Weirich. Engineering formal metatheory. In *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. *JFR*, 7(2):1–89, 2014.

5 Henk Barendregt. Introduction to generalized type systems. *JFP*, 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 2006*, pages 63–77. Springer, 2007.

7 Venanzio Capretta and Amy P. Felty. Higher-order abstract syntax in type theory. In *Logic Colloquium 2006*, pages 65–90. Cambridge Univ. Press, 2009.

8 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.

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

10 Amy P. Felty, Alberto Momigliano, and Brigitte Pientka. The next 700 challenge problems for reasoning with higher-order abstract syntax representations: Part 2 - a survey. *Journal of Automated Reasoning*, 55(4):307–372, 2015.

11 Amy P. Felty and Brigitte Pientka. Reasoning with higher-order abstract syntax and contexts: A comparison. In *ITP 2010*, LNCS 6172, pages 227–242. Springer, 2010.

12 Jan Herman Geuvers. Logics and type systems. Proefschrift, Katholieke Univ. Nijmegen, 1993.

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

14 Jean-Yves Girard, Paul Taylor, and Yves Lafont. *Proofs and types*. Cambridge Univ. Press, 1989.

15 Healfdene Goguen and James McKinna. Candidates for substitution. Technical Report ECS-LFCS-97-358, Univ. of Edinburgh, 1997.

16 Robert Harper. *Practical foundations for programming languages*. Cambridge Univ. Press, 2013.

17 Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *Journal of the ACM*, 40(1):143–184, January 1993.

18 Jonas Kaiser, Tobias Tebbi, and Gert Smolka. Equivalence of System F and $\lambda 2$ in Coq based on context morphism lemmas. In *CPP 2017*. ACM, 2017.

19 Aleksandar Nanevski, Frank Pfenning, and Brigitte Pientka. Contextual modal type theory. *ACM Transactions on Computational Logic*, 9(3):1–49, 2008.

20 Frank Pfenning and Conal Elliott. Higher-order abstract syntax. In *PLDI 1988*, pages 199–208. ACM, 1988.

21 Frank Pfenning and Carsten Schürmann. System description: Twelf - A meta-logical framework for deductive systems. In *CADE 1999*, pages 202–206. Springer, 1999.

22 Brigitte Pientka. A type-theoretic foundation for programming with higher-order abstract syntax and first-class substitutions. In *POPL 2008*, pages 371–382. ACM Press, 2008.

23 Brigitte Pientka. Beluga: programming with dependent types, contextual data, and contexts. In *FLOPS 2010*, LNCS 6009, pages 1–12. Springer, 2010.

24 Brigitte Pientka and Andrew Cave. Inductive Beluga:Programming Proofs (System Description). In *CADE 2015*, LNCS 9195, pages 272–281. Springer, 2015.

25 John Charles Reynolds. Towards a theory of type structure. In *Paris Colloque sur la Programmation*, LNCS 19, pages 408–423. Springer, 1974.

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

27 Steven Schäfer, Tobias Tebbi, and Gert Smolka. Autosubst: Reasoning with de Bruijn terms and parallel substitutions. In *ITP 2015*, LNCS 9236, pages 359–374. Springer, 2015.

28 Roberto Virga. *Higher-Order Rewriting with Dependent Types*. PhD thesis, Dep. of Math. Sciences, Carnegie Mellon Univ., 1999. CMU-CS-99-167.