Proof Reuse

Antoine Gaulin

June 27, 2022

1 Introduction

It is common in the literature to reuse proofs of previously established results in order to derive new theorems. A common pattern in papers is to start from a well understood language (often System F, LF, or the Calculus of Construction), add a new construct to it (e.g. subtyping, inductive types, etc.), and then show that the desirable properties of the original system are preserved. Most proofs (at least for the basic properties) are by induction on the structure of the hypothesized derivation. To conclude that the properties hold in the extension, it is then clearly sufficient to consider only the cases relevant to the new construct. However, in mechanization, one would need to work through all the previously established cases once more. This task is tedious and unnecessary.

We investigate a few ways to simplify the development of mechanized proofs, the key idea being to reuse proofs when possible. The aim is to start from the type system of Beluga [7, 8] and look at a few extensions that allow various forms of proof reuse. Through this process, we can also fix a major problem with how contexts are represented in Beluga, namely the inability to recover premisses needed for the formation of assumptions.

The first direction is to extend the data-level type theory (i.e. the logical framework LF [2]) with refinements, thus allowing a restricted form of subtyping to the language. This can then be lifted to context schemas, and then to the computation-level (i.e. the dependent contextual modal type theory [5]) in a mostly straightforward way. The main idea behind refinements is to "separate" a type into *sorts*. While types express syntactic properties of terms, sorts express semantic properties. They can therefore be used to enforce various properties on terms, while preserving type uniqueness. In this case, we obtain a notion of subsorting rather than subtyping. Ultimately, refinements allow a very limited form of proof reuse, and their usefulness is more in simplifying proofs.

The second direction is to add constructor subtyping [9, 1], which would be more accurately called *supertyping*. This idea is simple : if a type B has all the constuctors of another type A and possibly more, then B can be viewed as a supertype of A. Intuitively, this is because any object constructed with only the constructors defining A could also be constructed by the constructors defining B. In this setting, we get a notion of co-inheritance [9]. This can be seen as dual to the inheritance mechanisms of object oriented programming, in the sense that a co-inherited function is lifted from a subtype to its supertype, and then extended with the cases to cover the extra constructors (if needed).

The third direction is to add ornaments [4]. Here, we obtain systematic ways to enhance a type and/or its constructors with additional dependencies, as well as a lifting mechanism to lift proofs on a type to its ornamented type. Combining this with constructor subtyping, we should be able to present incremental development of languages and of their meta-theory, which would be closer to what is found in the literature.

Note. In what follows, the comments classified as **Remark** are clarifications or observations, and those classified as **Note** are either things that I didn't think of before I started typing this down, or places where I realized there is a mistake.

We start with refinements because it is the most invasive change to the language. This is due to the fact that we now want to assign both sorts and types to terms, which is done in a single judgment. This change is also present at the level of types, which are classified by both classes and kinds. Thus, almost every inference rule must be adapted, although they keep the same flavor. The core theory presented in this section is based on [7] and [8], and the addition of refinements closely follows what is shown in [3].

In the setting of refinements, every object should have a unique type (fully determined by its syntax), but possibly many different sorts. Each sort is restricted to a given type, and they express more specific properties that may of may not be satisfied by a given term of that type. In this sense, one may regard types as intrinsic properties, and sorts as extrinsic properties [6]. This allows us to specify properties without the need for additional types, which in turns simplifies the statement of theorems and their proofs. Additionally, there is a natural sub-sorting relation that is akin to logical implication, that is $S_1 \leq S_2 \sqsubset A$ if the property S_1 implies the property S_2 for any term of type A. In particular, if we have proven a result on terms of sort S_2, then we can reuse the proof on terms of sort S_1.

2 Motivating Examples

2.1 Values and Terms

In a framework without refinements, being a value is usually encoded as a property of terms, i.e. as a type of kind `tm -> type`. When refinements are added, it becomes simpler to have a sort of values refining the type of terms. So, an encoding of the λ-calculus with natural numbers could be the following :

```
LF term : type =
  | value : sort
  | zero : value
  | succ : (value -> value) ∧ (term -> term)
  | lam : (value -> term) -> value
  | app : term -> term -> term
;
```

Here, we make use of intersection sorts when specifying the `succ` constructors, which can be applied to arbitrary terms, but should only yield a value when it is applied to a value. Note also that the `lam` constructor is given sort `(value -> term) -> value`, which enforces a call-by-value semantic in the language. Now, let's define a big-step semantics for this small language :

```
LF big_step : term -> value -> type =
  | bs_zero : big_step zero zero
  | bs_succ : big_step M V -> big_step (succ M) (succ V)
  | bs_lam : big_step (lam M) (lam M)
  | bs_app : big_step F (lam M)
             -> big_step N V'
             -> big_step (M V') V
             -> big_step (app F N) V
```

Without refinements, our big-step semantics would instead have kind `term -> term -> type`, and the first thing that we would want is to prove that if `big_step M V`, then `V` is a value. Adding refinements allows us to specify this property directly and verify it automatically at type-checking, thus reducing the number of lemmas that one needs to prove about their language.

It should be noted that this improvement can be done without the use of refinements, by first defining a type of values and having a term constructor that lifts all values to terms, that is a constructor of type `value -> term`. However, this approach creates an unnecessary separation between values and terms, whereas refinements merely distinguish values as a special subset of terms. Due to this separation, if we want to prove a lemma that holds of arbitrary terms (possibly with some restriction), we often need to first prove a separate lemma just for values, so that we can use it to handle the case of values. When considering values as a sort refining terms, this should not be necessary since we

2.2 Bidirectional Typing

In bidirectional typing, we separate terms in two categories, normal and neutral, which correspond to introduction and elimination forms, respectively. The type of a neutral term is synthesized, while the type of a normal term is checked. So, a simply-typed λ-calculus with a base type and a constant would have the following syntax :

$$\begin{array}{rrl} \text{Types} & A, B ::= & \mathtt{b} \mid A \to B \\ \text{Normal terms} & M, N ::= & R \mid \mathtt{c} \mid \lambda x.M \\ \text{Neutral terms} & R ::= & x \mid R\ M \end{array}$$

To encode this without refinements, we would usually start with a type `tm : type` of terms, and define normal and neutral as predicates of kind `tm -> type`. Alternatively, we could encode normal and neutral terms directly, following the above syntax. However, this would make it more difficult to prove lemmas pertaining to all terms, as we would likely need to separate such a lemma in two parts, one for normal terms, and one for neutral terms. Using refinements, we can give the following encoding instead :

```
LF tp : type =
  | base : tp
  | arr : tp -> tp -> tp
;

LF tm : type =
  | normal : sort
  | neutral : sort
  | neutral ≤ normal
  | const : normal
  | lam : (term -> normal) -> normal
  | app : neutral -> normal -> neutral
;
```

In this definition, we use the subsorting mechanism to encode the fact that neutral terms can be considered as normal, rather than having a special constructor just for that. Now for the typing judgment, we can similarly use sorts to define a single type separated in two sorts :

```
LF has_type : tm -> tp -> type =
  | check : normal -> tp -> sort
  | synth : neutral -> tp -> sort
  | ht_switch : synth R A -> check R A
  | ht_const : check const base
  | ht_lam : ({x : neutral} synth x A -> check M B)
```

```
                 -> check (lam M) (arr A B)
      |ht_app : synth R (arr A B) -> check M A -> synth (app R M) B
    ;
```

So, the use of refinments allows us to give a much more compact encoding of the types.
Note. There should be at least one proof in the example section

3 Data-level

The data-level of Beluga includes the usual terms, types, and kinds, but also contexts and substitutions. We add to the type level a notion of sorts, and to the kind level a similar notion of classes. Contexts are classified using a notion of schema, which, in our extension, are built out of world declarations. A world is a record of assumptions satisfying certain properties. To ensure well-formedness of worlds, we use two syntactic categories : blocks and worlds. A block is a dependent record (i.e. Σ-type), and a world is a function space whose image is a block.

3.1 Syntax

$$
\begin{array}{rrl}
\text{Signatures} & \Sigma ::= & \cdot \mid \Sigma, D \\
\text{Declarations} & D ::= & \mathbf{s}{::}L \sqsubset \mathbf{a}{:}K \mid \mathbf{c}{::}S \sqsubset A \mid \mathbf{s}_1 \leq \mathbf{s}_2 \sqsubset \mathbf{a} \mid \mathbf{w}{:}W \mid \boldsymbol{\xi}{:}\Xi \\
\text{Meta-contexts} & \Delta ::= & \cdot \mid \Delta, u{::}S[\Psi] \sqsubset A[\Psi] \mid \Delta, p{::}S[\Psi] \sqsubset A[\Psi] \mid \Delta, s{:}\Psi_1[\Psi_2] \mid \Delta, \psi{:}\Xi \\
\\
\text{Kinds} & K ::= & \mathtt{Type} \mid \Pi x{:}A.K \\
\text{Classes} & L ::= & \mathtt{Sort} \mid \Pi x{::}S.L \mid \top \mid L_1 \wedge L_2 \\
\\
\text{Atomic type families} & P ::= & \mathbf{a} \mid P\ N \\
\text{Canonical type families} & A ::= & P \mid \Pi x{:}A_1.A_2 \\
\text{Atomic sort families} & Q ::= & \mathbf{s} \mid Q\ N \\
\text{Canonical sort families} & S ::= & Q \mid \Pi x{:}S_1.S_2 \mid \top \mid S_1 \wedge S_2 \\
\\
\text{Blocks} & B ::= & S \sqsubset A \mid \Sigma x{::}S \sqsubset A.B \\
\text{Worlds} & W ::= & B \mid \Pi x{::}S \sqsubset A.W \\
\text{Schema} & \Xi ::= & \varepsilon \mid \Xi + W \\
\\
\text{Heads} & H ::= & \mathbf{c} \mid x \mid \mathtt{proj}\ k\ x \mid \mathtt{Clo}(x, s[\sigma]) \mid \#p[\sigma] \mid \mathtt{proj}\ k\ \#p \\
\text{Spines} & \vec{M} ::= & \varepsilon \mid N; \vec{M} \\
\text{Normal terms} & N ::= & R \mid \lambda x.N \\
\text{Neutral terms} & R ::= & H\ \vec{M} \mid u[\sigma] \\
\\
\text{LF contexts} & \Psi ::= & \cdot \mid \Psi, x{::}S \sqsubset A \mid \Psi, x{:}(\mathbf{w}\vec{M}) \\
\text{Substitutions} & \sigma ::= & \cdot \mid \mathtt{wk}_\psi \mid s[\sigma] \mid \sigma; N
\end{array}
$$

Figure 1: Syntax of data-level

The updated syntax of the language is given in Figure 1. Most of the syntax that was already present in Beluga remains unchanged (kinds, types, terms, and substitution, to be precise). The main differences are in the contexts and signatures, where assumptions are endowed with a sort as well as a type. Most importantly, LF contexts have an additional construct to associate variables to a given world, instead of just a type. Finally, declarations are extended with subsorting and worlds.

In the syntax for sorts (and similarly for classes), $\top$ corresponds to all terms of the corresponding types, and $S_1 \wedge S_2$ is an intersection sort, so it classifies terms that can be classified by both S_1 and S_2.

In the syntax of worlds, records are denoted as $\langle \ell_i{::}S_i \sqsubset A_i \rangle_n$, where the subscript $n \geq 1$ indicates the number of fields. The Π's in front of records can either be a parameter referred to by some of the fields, or assumptions needed to ensure the well-formedness of a given world. Once we get to the sub-world judgment, we will see that using Π is perhaps a bit misleading since the rules do not obey the familiar contravariance of Π-types. So, we maybe we shouldn't think of them as functions, even though they seem like functions.

Concerning the labels of worlds, I decided to take them out of the world's syntax itself, and rather consider them as names in declarations. Ultimately, worlds should always be declared before they are used, so they would always be in the signature Σ. This is just to avoid redundancy.

3.2 Judgments

As previously mentionned, most of the judgments take a slightly different form in the presence of refinements. Let's first look at a quick summary of the judgments :

$\vdash \Sigma$ `sig`	Signature well-formedness
$\vdash_\Sigma \Delta$ `mctx`	Meta-context well-formedness
$\Delta \vdash_\Sigma \Psi$ `ctx`	LF context well-formedness
$\Delta; \Psi \vdash_\Sigma L \sqsubset K$	Class L refines kind K
$\Delta; \Psi \vdash_\Sigma Q \sqsubset P \Rightarrow L$	Atomic sort Q synthesizes atomic type P and class L
$\Delta; \Psi \vdash_\Sigma S \sqsubset A \Leftarrow \mathtt{Sort}$	Sort S refines type A
$\Delta; \Psi \vdash_\Sigma N \Leftarrow S \sqsubset A$	Normal term N checks against sort S refining type A
$\Delta; \Psi \vdash_\Sigma R \Rightarrow S \sqsubset A$	Neutral term R synthesizes sort S refining type A
$\Delta; \Psi \vdash_\Sigma \sigma \Leftarrow \Phi$	Substitution σ checks against LF context Φ
$\Delta; \Psi \vdash_\Sigma S_1 \leq S_2 \sqsubset A$	S_1 is a sub-sort of S_2 as refinements of A
$\Delta; \Psi \vdash_\Sigma W$ `world`	W is a well-formed world
$\Delta; \Psi \vdash_\Sigma \Xi$ `schema`	Ξ is a well-formed context schema
$\Delta; \Psi_1 \vdash_\Sigma \Psi_2 : \Xi$	LF context Ψ has schema Ξ
$\Delta; \Psi \vdash_\Sigma W_1 \leq W_2$	W_1 is a sub-world of W_2
$\Delta; \Psi \vdash_\Sigma \Xi_1 \leq \Xi_2$	Ξ_1 is a sub-schema of Ξ_2

Notes. (1) There might be too many contexts in some of these judgments. In particular, worlds and schemata should be closed, and the judgments for their well-formedness will only be used in signature formation, which requires all contexts to be empty.

(2) It may be better to consider only LF contexts that have a schema rather than having a judgment for well-formed contexts and well-schemaed contexts. To achieve this, it would probably be necessary to enrich the notion of a context schema slightly. In particular, we would need a schema that specify a particular sort/type for the right-most element(s) of the context since that is frequently used in mechanization. The intuition for this comes from the fact that we usually don't consider terms that are not well-typed, or types that are not well-kinded, so it is odd to consider contexts that are not well-schemaed. On the other hand, since contexts can have multiple schemata, we may want to consider the classifier `ctx` as analogous to types and schemata as analogous to sorts. In this case, we could have a $\top$ schema to talk about arbitrary contexts, and merge the two judgments, just like we do for sorting/typing.

Before presenting the rules defining each of these judgments, let us go over some conventions that will simplify notation.

For all the judgments except signature validity, we omit the subscript Σ since the signature is fixed throughout any derivation. In all judgments except for meta-context validity, we assume that Δ is well-formed, and similarly we assume that Ψ is well-formed in all the remaining judgments except LF context validity. In practice, we would check that contexts are well-formed at the leaves of the proof trees.

For the synthesis judgments (those with $\Rightarrow$), the contexts, signatures and first object on the right of the turnstile are inputs, and the rest are outputs. For instance, in $\Delta;\Psi \vdash_\Sigma Q \sqsubset P \Rightarrow L$, both P and L are outputs. For the remaining jugments, everything is considered an input. In all judgments, we assume that every input is well-formed and in canonical form. To enforce this, we need to use hereditary substitutions.

Finally, we assume that all names of constants and variables are unique. Now, let's look at the inference rules.

$\boxed{\vdash \Sigma\ \texttt{sig}}$

$$\frac{}{\vdash \cdot\ \texttt{sig}} \qquad \frac{\vdash \Sigma\ \texttt{sig} \qquad \cdot;\cdot;\cdot \vdash_\Sigma L \sqsubset K}{\vdash \Sigma, \mathbf{s}{::}L \sqsubset \mathbf{a}{:}K\ \texttt{sig}}$$

$$\frac{\vdash \Sigma\ \texttt{sig} \qquad \cdot;\cdot;\cdot \vdash_\Sigma S \sqsubset A \Leftarrow \texttt{Sort}}{\vdash \Sigma, \mathbf{c}{::}S \sqsubset A\ \texttt{sig}} \qquad \frac{\vdash \Sigma\ \texttt{sig} \qquad \mathbf{s}_1 \sqsubset \mathbf{a}{::}L \sqsubset K \in \Sigma \qquad \mathbf{s}_2 \sqsubset \mathbf{a}{::}L \sqsubset K \in \Sigma}{\vdash \Sigma, \mathbf{s}_1 \leq \mathbf{s}_2 \sqsubset \mathbf{a}}$$

$$\frac{\vdash \Sigma\ \texttt{sig} \qquad \cdot;\cdot;\cdot \vdash_\Sigma W\ \texttt{world}}{\vdash \Sigma, \mathbf{w}{:}W} \qquad \frac{\vdash \Sigma\ \texttt{sig} \qquad \cdot;\cdot;\cdot \vdash_\Sigma \Xi\ \texttt{schema}}{\vdash \Sigma, \boldsymbol{\xi}{:}\Xi}$$

In the rules for signature formation, there is a notable change from what is shown in [3], namely that we don't have declarations of the form $\mathbf{a}{:}K$ or $\mathbf{c} : A$. I think those are unnecessary since we can just replace them with declarations of the form $\top \sqsubset \mathbf{a}{::}\top \sqsubset K$ and $\mathbf{c}{::}\top \sqsubset A$, respectively.

Note. After giving it some thought, this wouldn't work since all our declarations must introduce names. Nevertheless, we get a more uniform system by using $\top$ refinements in place of just kinds or types, so I would prefer to keep this approach and just add rules for these cases.

$\boxed{\vdash_\Sigma \Delta\ \texttt{mctx}}$

$$\frac{}{\vdash \cdot\ \texttt{mctx}} \qquad \frac{\vdash \Delta\ \texttt{mctx} \qquad \Delta;\Psi \vdash S \sqsubset A \Leftarrow \texttt{Sort}}{\vdash \Delta, u{::}S[\Psi] \sqsubset A[\Psi]}$$

$$\frac{\vdash \Delta\ \texttt{mctx} \qquad \Delta;\Psi \vdash S \sqsubset A \Leftarrow \texttt{Sort}}{\vdash \Delta, p{::}S[\Psi] \sqsubset A[\Psi]} \qquad \frac{\vdash \Delta\ \texttt{mctx} \qquad \Delta;\Psi_2 \vdash \Psi_1\ \texttt{ctx}}{\vdash \Delta, s{:}\Psi_1[\Psi_2]}$$

$$\frac{\vdash \Delta\ \texttt{sctx} \qquad \boldsymbol{\xi}{:}\Xi \in \Sigma}{\vdash \Delta, \psi{:}\Xi\ \texttt{sctx}}$$

Notes. (1) I'm still unsure about the distinction between meta-variables (u) and parameter variables (p). I think it's more about the way they are used? Specifically, p should only be substituted with an ordinary variable.

(2) In the rule for substitution variables, the premise $\Delta;\Psi_2 \vdash \Psi_1\ \texttt{ctx}$ does not match the usual LF context well-formedness judgment, which is of the form $\Delta \vdash \Psi\ \texttt{ctx}$ (i.e. without an LF context on the left

side of the turnstile). Maybe it should be a substitution judgment instead? Otherwise, the context validity judgment could be generalized in a straightforward way. The only paper that talks about substitution variables is [7], but the context formation rules are not given there.

$\boxed{\Delta \vdash_\Sigma \Psi \ \texttt{ctx}}$

$$\frac{}{\Delta \vdash \cdot \ \texttt{ctx}} \qquad \frac{\Delta \vdash \Psi \ \texttt{ctx} \qquad \Delta; \Psi \vdash S \sqsubset A \Leftarrow \texttt{Sort}}{\Delta \vdash \Psi, x{::}S \sqsubset A \ \texttt{ctx}}$$

$$\frac{\Delta \vdash \Psi \ \texttt{ctx} \qquad \mathbf{w}{:}\Pi(\overrightarrow{x{::}S \sqsubset A}).\langle \ell_i{::}S_i \sqsubset B_i \rangle_n \in \Sigma \qquad \Delta, \Psi \vdash \vec{M} \Leftarrow \overrightarrow{S \sqsubset A}}{\Delta \vdash \Psi, x{:}\mathbf{w}\ \vec{M}}$$

Remark. I've decided to use spines instead of substitutions for world parameters, mostly because users would want to refer to the terms during proofs, so that's what they would specify.

$\boxed{\Delta \vdash_\Sigma \Psi_1 \leq \Psi_2}$

$$\frac{}{\Delta \vdash \cdot \leq \cdot} \qquad \frac{\Delta \vdash \Psi_1 \leq \Psi_2 \qquad \Delta; \Psi_2 \vdash S_1 \leq S_2 \sqsubset A}{\Delta \vdash (\Psi_1, x{::}S_1 \sqsubset A) \leq (\Psi_2, x{::}S_2)}$$

Note. I'm not sure if we should have $\Delta; \Psi_1 \vdash S_1 \leq S_2 \sqsubset A$ as the premise (instead of the judgment with Ψ_2). In a subtyping rule, we would use the most precise type in the context, but here it is the least precise context that is used.

$\boxed{\Delta; \Psi \vdash_\Sigma L \sqsubset K}$

$$\frac{}{\Delta; \Psi \vdash \texttt{Sort} \sqsubset \texttt{Type}} \qquad \frac{\Delta; \Psi \vdash S \sqsubset A \Leftarrow \texttt{Sort} \qquad \Delta; \Psi, x{:}S \sqsubset A \vdash L \sqsubset K}{\Delta; \Psi \vdash \Pi x{::}S.L \sqsubset \Pi x{:}A.K}$$

$$\frac{}{\Delta; \Psi \vdash \top \sqsubset K} \qquad \frac{\Delta; \Psi \vdash L_1 \sqsubset K \qquad \Delta; \Psi \vdash L_2 \sqsubset K}{\Delta; \Psi \vdash L_1 \wedge L_2 \sqsubset K}$$

Notes. (1) For the rule with $\top$, we probably need a premise stating that K is a well-formed kind, which implies that we need an extra judgment for kind validity (that would be exactly the same as what is already in Beluga).

(2) Rules for `Rec` kinds are missing.

$\boxed{\Delta; \Psi \vdash_\Sigma Q \sqsubset P \Rightarrow L}$

$$\frac{\mathbf{s}{::}L \sqsubset \mathbf{a}{:}K \in \Sigma}{\Delta; \Psi \vdash \mathbf{s} \sqsubset \mathbf{a} \Rightarrow L} \qquad \frac{\Delta; \Psi \vdash Q \sqsubset P \Rightarrow \Pi x{::}S.L \qquad \Delta; \Psi \vdash N \Leftarrow S \sqsubset A}{\Delta; \Psi \vdash Q\ N \sqsubset P\ N \Rightarrow [N/x]L}$$

$$\frac{\Delta; \Psi \vdash Q \sqsubset P \Rightarrow L_1 \wedge L_2}{\Delta; \Psi \vdash Q \sqsubset P \Rightarrow L_1} \qquad \frac{\Delta; \Psi \vdash Q \sqsubset P \Rightarrow L_1 \wedge L_2}{\Delta; \Psi \vdash Q \sqsubset P \Rightarrow L_2}$$

$\boxed{\Delta; \Psi \vdash_\Sigma S \sqsubset A \Leftarrow \texttt{Sort}}$

$$\frac{\Delta; \Psi \vdash Q \sqsubset P \Rightarrow \texttt{Sort}}{\Delta; \Psi \vdash Q \sqsubset P \Leftarrow \texttt{Sort}} \qquad \frac{\Delta; \Psi \vdash S \sqsubset A \Leftarrow \texttt{Sort} \qquad \Delta; \Psi, x{::}S \sqsubset A \vdash S' \sqsubset A'}{\Delta; \Psi \vdash \Pi x{::}S.S' \sqsubset \Pi x{:}A.A' \Leftarrow \texttt{Sort}}$$

$$\frac{}{\Delta; \Psi \vdash \top A \Leftarrow \texttt{Sort}} \qquad \frac{\Delta; \Psi \vdash S_1 \sqsubset A \Leftarrow \texttt{Sort} \qquad \Delta; \Psi \vdash S_2 \sqsubset A \Leftarrow \texttt{Sort}}{\Delta; \Psi \vdash S_1 \wedge S_2 \sqsubset A \Leftarrow \texttt{Sort}}$$

Note. Again, the rule for $\top$ should probably have a premise $\Delta; \Psi \vdash A \Leftarrow \texttt{Type}$, which requires adding a type well-formedness judgment.

$\boxed{\Delta; \Psi \vdash_\Sigma N \Leftarrow S \sqsubset A}$

$$\frac{\Delta; \Psi \vdash R \Rightarrow S \sqsubset A \qquad \Delta; \Psi \vdash S \leq S' \sqsubset A}{\Delta; \Psi \vdash R \Leftarrow S' \sqsubset A} \qquad \frac{\Delta; \Psi, x{::}S \sqsubset A \vdash N \Leftarrow S' \sqsubset A'}{\Delta; \Psi \vdash \lambda x.N \Leftarrow \Pi x{::}S.S' \sqsubset \Pi x{:}A.A'}$$

$$\frac{\Delta; \Psi \vdash N \Leftarrow S_1 \sqsubset A \qquad \Delta; \Psi \vdash N \Leftarrow S_2 \sqsubset A}{\Delta; \Psi \vdash N \Leftarrow S_1 \wedge S_2 \sqsubset A}$$

$\boxed{\Delta; \Psi \vdash_\Sigma R \Rightarrow S \sqsubset A}$

$$\frac{\mathbf{c}{::}S \sqsubset A \in \Sigma}{\Delta; \Psi \vdash \mathbf{c} \Rightarrow S \sqsubset A} \qquad \frac{x{::}S \sqsubset A \in \Psi}{\Delta; \Psi \vdash x \Rightarrow S \sqsubset A}$$

$$\frac{x{:}\mathbf{w}\ \vec{M} \in \Psi \qquad \mathbf{w}{:}\Pi(\overrightarrow{y{::}S \sqsubset A}).\langle \ell_i{::}S_i \sqsubset B_i \rangle_n \in \Sigma}{\Delta; \Psi \vdash \texttt{proj}\ k\ x \Rightarrow [\ell_{k-1}; ..., \ell_1; \overleftarrow{M}]S_k \sqsubset [\ell_{k-1}; ..., \ell_1; \overleftarrow{M}]B_k} \text{ (for } 1 \leq k \leq n)$$

$$\frac{\Delta; \Psi \vdash R \Rightarrow \Pi x{::}S_1 \sqsubset A_1.S_2 \sqsubset \Pi x{:}A_1.A_2 \qquad \Delta; \Psi \vdash N \Leftarrow S_1 \sqsubset A_1}{\Delta; \Psi \vdash R\ N \Rightarrow [N/x]S \sqsubset [N/x]A_2}$$

$$\frac{\Delta; \Psi \vdash R \Rightarrow S_1 \wedge S_2 \sqsubset A}{\Delta; \Psi \vdash R \Rightarrow S_1 \sqsubset A} \qquad \frac{\Delta; \Psi \vdash R \Rightarrow S_1 \wedge S_2 \sqsubset A}{\Delta; \Psi \vdash R \Rightarrow S_2 \sqsubset A}$$

Remarks. (1) In the rule for projections, $\overleftarrow{M}$ is just $\vec{M}$ backwards. This is necessary since we construct spines from right to left, and substitutions from left to right.
(2) Because of the last two rules regarding intersection sorts, the system is not deterministic.
Note. The rules for parameter variables, meta-variables, and closures are still missing.

$\boxed{\Delta; \Psi_1 \vdash_\Sigma \sigma \Leftarrow \Psi_2}$

$$\frac{}{\Delta; \Psi_1 \vdash \cdot \Leftarrow \cdot} \qquad \frac{\Delta; \Psi_1 \vdash \sigma \Leftarrow \Psi_2' \qquad s{::}\Psi_2[\Psi_2'] \in \Delta}{\Delta; \Psi_1 \vdash s[\sigma] \Leftarrow \Psi_2}$$

$$\frac{}{\Delta; \psi, \Psi \vdash \texttt{wk}_\psi \Leftarrow \psi} \qquad \frac{\Delta; \Psi_1 \vdash \sigma \Leftarrow \Psi_2 \qquad \Delta; \Psi_1 \vdash N \Leftarrow S \sqsubset A}{\Delta; \Psi_1 \vdash (\sigma; N) \Leftarrow (\Psi_2, x{::}S \sqsubset A)}$$

$$\frac{\Delta; \Psi \vdash \vec{M} \Leftarrow \overrightarrow{S \sqsubset A} \qquad \Delta; \Psi, (\overrightarrow{x{::}S \sqsubset A}) \vdash B \ \texttt{block}}{\Delta; \Psi_1 \vdash (\sigma; B) \Leftarrow (\Psi_2, x{:}\mathbf{w}\ \vec{M})}$$

Notes. (1) The rule for substitution of $x : \mathbf{w}\ \vec{M}$ could be better. In particular, the world's signature should appear in the premises to ensure well-formedness.

(2) The rule for substitution variables could be enhanced with a context relation. (Later, I'll ignore substitution variables for now)

Remark. The premise $\Delta; \Psi \vdash \overrightarrow{S \sqsubset A} \Leftarrow \texttt{Sort}$ checks that all the sorts are valid, given that the previous ones are. Formally, this auxilliary judgment is given by the following two rules :

$$\frac{\Delta; \Psi \vdash S \sqsubset A \Leftarrow \texttt{Sort}}{\Delta; \Psi \vdash S \sqsubset A; \varepsilon \Leftarrow \texttt{Sort}} \qquad \frac{\Delta; \Psi \vdash \overrightarrow{S \sqsubset A} \Leftarrow \texttt{Sort} \qquad \Delta; \Psi, (\overrightarrow{x{::}S \sqsubset A}) \vdash S' \sqsubset A' \Leftarrow \texttt{Sort}}{\Delta; \Psi \vdash S' \sqsubset A'; \overrightarrow{S \sqsubset A} \Leftarrow \texttt{Sort}}$$

$\boxed{\Delta; \Psi \vdash_\Sigma S_1 \leq S_2 \sqsubset A}$

$$\frac{\Delta; \Psi \vdash S \sqsubset A}{\Delta; \Psi \vdash S \leq S \sqsubset A} \qquad \frac{\Delta; \Psi \vdash S_1 \leq S_2 \sqsubset A \qquad \Delta; \Psi \vdash S_2 \leq S_3 \sqsubset A}{\Delta; \Psi \vdash S_1 \leq S_3 \sqsubset A}$$

$$\frac{\Delta; \Psi \vdash S \sqsubset A}{\Delta; \Psi \vdash S \leq \top \sqsubset A} \qquad \frac{\Delta; \Psi \vdash S_2 \leq S_1 \sqsubset A \qquad \Delta; \Psi \vdash S_1' \leq S_2' \sqsubset A'}{\Delta; \Psi \vdash \Pi x{::}S_1.S_1' \leq \Pi x{::}S_2.S_2' \sqsubset \Pi x{:}A.A'}$$

$$\frac{\Delta; \Psi \vdash S \leq S_1 \sqsubset A \qquad \Delta; \Psi \vdash S \leq S_2 \sqsubset A}{\Delta; \Psi \vdash S \leq S_1 \wedge S_2 \sqsubset A}$$

$$\frac{\Delta; \Psi \vdash S_1 \leq S \sqsubset A \qquad \Delta; \Psi \vdash S_2 \sqsubset A \Leftarrow \texttt{Sort}}{\Delta; \Psi \vdash S_1 \wedge S_2 \sqsubset A} \qquad \frac{\Delta; \Psi \vdash S_2 \leq S \sqsubset A \qquad \Delta; \Psi \vdash S_1 \sqsubset A \Leftarrow \texttt{Sort}}{\Delta; \Psi \vdash S_1 \wedge S_2 \sqsubset A}$$

$\boxed{\Delta; \Psi \vdash_\Sigma B\ \texttt{block}}$

$$\frac{\Delta; \Psi \vdash S \sqsubset A \Leftarrow \texttt{Sort}}{\Delta; \Psi \vdash (S \sqsubset A)\ \texttt{block}} \qquad \frac{\Delta; \Psi \vdash S \sqsubset A \Leftarrow \texttt{Sort} \qquad \Delta; \Psi, x{::}S \sqsubset A \vdash B\ \texttt{block}}{\Delta; \Psi \vdash (\Sigma x{::}S \sqsubset A.B)\ \texttt{block}}$$

$\boxed{\Delta; \Psi \vdash_\Sigma W\ \texttt{world}}$

$$\frac{\Delta; \Psi \vdash B\ \texttt{block}}{\Delta; \Psi \vdash B\ \texttt{world}} \qquad \frac{\Delta; \Psi \vdash S \sqsubset A \Leftarrow \texttt{Sort} \qquad \Delta; \Psi, x{::}S \sqsubset A \vdash W\ \texttt{world}}{\Delta; \Psi \vdash (\Pi x{::}S \sqsubset A.W)\ \texttt{world}}$$

$\boxed{\Delta; \Psi \vdash_\Sigma \Xi\ \texttt{schema}}$

$$\frac{}{\Delta; \Psi \vdash \varepsilon\ \texttt{schema}} \qquad \frac{\Delta; \Psi \vdash \Xi\ \texttt{schema} \qquad \mathbf{w}{:}W \in \Sigma \qquad \mathbf{w} \notin \Xi}{\Delta; \Psi \vdash \Xi + \mathbf{w}\ \texttt{schema}}$$

Note. The judgments for world and schema validity and only used in signature formations, which requires the contexts to be empty. So, they may not be needed at all in this case.

$\boxed{\Delta \vdash_\Sigma \Psi : \Xi}$

$$\frac{}{\Delta \vdash \cdot{:}\varepsilon} \qquad \frac{\psi{:}\Xi \in \Delta}{\Delta \vdash \psi{:}\Xi} \qquad \frac{\Delta \vdash \Psi{:}\Xi_1 \qquad \Delta \vdash \Xi_1 \leq \Xi_2}{\Delta \vdash \Psi{:}\Xi_2}$$

$$\frac{\Delta \vdash \Psi : \Xi \qquad \mathbf{w}{:}\Pi\overrightarrow{x{::}S \sqsubset A}.\langle \ell_i{::}S_i \sqsubset B_i \rangle_n \in \Xi \qquad \Delta; \Psi \vdash \vec{M} \Leftarrow \overrightarrow{S \sqsubset A}}{\Delta \vdash (\Psi, x{:}\mathbf{w}\ \vec{M}) : \Xi}$$

Note. The judgment $\Delta; \Psi \vdash \vec{M} \Leftarrow \overrightarrow{S \sqsubset A}$ is just checking each of the terms in the spine $\vec{M}$ against the corresponding type (which may depend on the previous terms). Rules will be added soon.

$\boxed{\Delta; \Psi \vdash_\Sigma B_1 \leq B_2}$

$$\frac{\Delta; \Psi \vdash S_1 \leq S_2 \sqsubset A}{\Delta; \Psi \vdash (S_1 \sqsubset A) \leq (S_2 \sqsubset A)} \qquad \frac{\Delta; \Psi \vdash B_1 \leq B_2 \qquad \Delta; \Psi \vdash S \sqsubset A \Leftarrow \texttt{Sort}}{\Delta; \Psi \vdash \Sigma x{::}S \sqsubset A.B_1 \leq B_2}$$

$$\frac{\Delta; \Psi \vdash S_1 \leq S_2 \sqsubset A \qquad \Delta; \Psi, x{::}S_1 \sqsubset A \vdash B_1 \leq B_2}{\Delta; \Psi \vdash (\Sigma x{::}S_1 \sqsubset A).B_1 \leq (\Sigma x{::}S_2 \sqsubset A).B_2}$$

$\boxed{\Delta; \Psi \vdash_\Sigma W_1 \leq W_2}$

$$\frac{\Delta; \Psi \vdash B_1 \leq B_2 \quad \text{(as blocks)}}{\Delta; \Psi \vdash B_1 \leq B_2 \quad \text{(as worlds)}} \qquad \frac{\Delta; \Psi \vdash W_1 \leq W_2 \qquad \Delta; \Psi \vdash S \sqsubset A \Leftarrow \texttt{Sort}}{\Delta; \Psi \vdash \Pi x{::}S \sqsubset A.W_1 \leq W_2}$$

$$\frac{\Delta; \Psi \vdash S_1 \leq S_2 \sqsubset A \qquad \Delta; \Psi, x{::}S_1 \sqsubset A \vdash W_1 \leq W_2}{\Delta; \Psi \vdash (\Pi x{::}S_1 \sqsubset A).W_1 \leq (\Pi x{::}S_2 \sqsubset A).W_2}$$

Remark. We stress that the last rule for sub-worlds does not satisfy the usual contravariance associated to subtyping of Π-types, which indicates that we may not want to consider Π-worlds as functions. In addition, we also have this rule stating that having extra parameters yields sub-worlds, which is also not the case when subtyping function spaces.

$\boxed{\Delta; \Psi \vdash_\Sigma \Xi_1 \leq \Xi_2}$

$$\frac{\Delta; \Psi \vdash \Xi\ \texttt{schema}}{\Delta; \Psi \vdash \varepsilon \leq \Xi} \qquad \frac{\Delta; \Psi \vdash \Xi_1 \leq \Xi_2 \qquad \Delta; \Psi \vdash W_1 \leq W_2}{\Delta; \Psi \vdash \Xi_1 + W_1 \leq \Xi_2 + W_2} \qquad \frac{\Delta; \Psi \vdash \Xi_1 + \Xi_2\ \texttt{schema}}{\Delta; \Psi \vdash \Xi_1 + \Xi_2 \leq \Xi_2 + \Xi_1}$$

4 Computation-level

We can lift the data-level refinements to the computation-level to obtain a restricted notion of refinements where the user does not directly specify any sorts. It could be interesting to have a full blown refinement type system, and it should not complicate matters too much.

Contexts	$\Gamma ::=$	$\cdot \mid \Gamma, y{::}\mu \sqsubset \kappa$
Kinds	$\kappa ::=$	$\mathtt{ctype} \mid \Pi x{:}A[\Psi].\kappa$
Classes	$\zeta ::=$	$\mathtt{csort} \mid \Pi x{::}S[\Psi].\zeta \mid \top \mid \zeta_1 \wedge \zeta_2$
Types	$\tau ::=$	$A[\Psi] \mid \tau_1 \to \tau_2 \mid \Pi\psi{:}\Xi.\tau \mid \Pi^{\Box}u{:}A[\Psi].\tau$
Sorts	$\mu ::=$	$S[\Psi] \mid \mu_1 \to \mu_2 \mid \Pi\psi{:}\Xi.\mu \mid \Pi^{\Box}u{::}S[\Psi].\mu \mid \top \mid \mu_1 \wedge \mu_2$
Checked expressions	$e ::=$	$i \mid \mathtt{rec}\ f.e \mid \mathtt{fn}\ y.e \mid \Lambda\psi.e \mid \lambda^{\Box}u.e \mid \mathtt{box}(\Psi.M) \mid \mathtt{case}\ i\ \mathtt{of}\ bs$
Synthesized expressions	$i ::=$	$y \mid i\ e \mid i\ \lceil\Psi\rceil \mid i\ \lceil\Psi.N\rceil \mid (e{::}\mu \sqsubset \tau)$
Branch	$b ::=$	$\Pi\Delta.\mathtt{box}(\Psi.M){::}S[\Psi] \sqsubset A[\Psi] \mapsto e$
Branches	$bs ::=$	$\cdot \mid (b \mid bs)$

Figure 2: Syntax of computation level

4.1 Syntax

The syntax for the computation level is given in Figure 2. Again, it is essentially the same as what is already in Beluga, except that we have sorts and classes. However, in this case, we consider a restricted version that is fully induced by the refinements (and schemata) of the data level.

Notes. (1) In practice, we allow user-defined computation-level type families, but they are not present in the current formulation. To add them, we would need to extend the syntax for types and have addition declarations. In this case, it wouldn't be much more work to add user-defined sorts.

(2) For sorts, we may want to consider $\Pi^{\Box}u{::}S[\Psi_1] \sqsubset A[\Psi_2].\mu$ instead of having both LF contexts be identical, probably with the condition that Ψ_1 is contained in Ψ_2 (up to renaming). In particular, we could have $\Psi_1{:}\Xi_1$ and $\Psi_2{:}\Xi_2$, where $\Xi_1 \leq \Xi_2$.

(3) It may be better to have $\Pi x{:}A[\Psi].\kappa$ kinds, and similarly for classes.

4.2 Judgments

We have the follow computation level judgments :

$\Delta \vdash_\Sigma \Gamma\ \mathtt{cctx}$	Γ is a well-formed context
$\Delta; \Gamma \vdash_\Sigma \zeta \sqsubset \kappa$	Class ζ refines kind κ
$\Delta; \Gamma \vdash_\Sigma \mu \sqsubset \tau \Leftarrow \mathtt{csort}$	Sort μ refines type τ
$\Delta; \Gamma \vdash_\Sigma e \Leftarrow \mu \sqsubset \tau$	Expression e checks agains sort μ refining type τ
$\Delta; \Gamma \vdash_\Sigma i \Rightarrow \mu \sqsubset \tau$	Expression i synthesizes saor μ refining type τ
$\Delta; \Gamma \vdash_\Sigma b \Leftarrow_{\mu' \sqsubset \tau'} \mu \sqsubset \tau$	Branch b checks against μ refining τ when analyzing a μ' refining τ'
$\Delta; \Gamma \vdash_\Sigma \mu_1 \leq \mu_2 \sqsubset \tau$	μ_1 is a subsort of μ_2

Again, we omit the subscript Σ since it is fixed throughout any derivation, and we assume that all inputs are well-formed. The system should be decidable if we follow the same input/output convention as in the data level (i.e. the synthesized sorts and types are outputs, and everything else is an input). The judgments are defined via the following rules :

Notes. (1) Just as in the data-level, it may be necessary to have judgments for kind and type well-formedness.

(2) Since we are just lifting everything to the computation level, it may be redundant to have $\top$ and intersection computation-level sorts (and classes) since they could probably always be inferred from $\top$ and intersection data-level sorts (and classes).

(3) The rules for refinement should be enhanced with (LF) context relations

$\boxed{\Delta \vdash_\Sigma \Gamma\ \texttt{cctx}}$

$$\frac{}{\Delta \vdash \cdot\ \texttt{cctx}} \qquad \frac{\Delta \vdash \Gamma\ \texttt{cctx} \qquad \Delta; \Gamma \vdash \mu \sqsubset \tau \Leftarrow \tau}{\Delta \vdash \Gamma, y{::}\mu \sqsubset \tau\ \texttt{cctx}}$$

$\boxed{\Delta; \Gamma \vdash_\Sigma \zeta \sqsubset \kappa}$

$$\frac{}{\Delta; \Gamma \vdash \texttt{csort} \sqsubset \texttt{ctype}} \qquad \frac{}{\Delta; \Gamma \vdash \top \sqsubset \kappa}$$

$$\frac{\Delta; \Psi \vdash S \sqsubset A \qquad \Delta; \Gamma, y{::}S[\Psi] \sqsubset A[\Psi] \vdash \zeta \sqsubset \kappa}{\Delta; \Gamma \vdash \Pi y{::}S[\Psi].\zeta \sqsubset \Pi y{:}A[\Psi].\kappa}$$

$$\frac{\Delta; \Gamma \vdash \zeta_1 \sqsubset \kappa \qquad \Delta; \Gamma \vdash \zeta_2 \sqsubset \kappa}{\Delta; \Gamma \vdash \zeta_1 \wedge \zeta_2 \sqsubset \kappa}$$

Note. For the Π rule, it indeed seems better to have the boxed LF contexts since we establish $S \sqsubset A$ at the data-level.

$\boxed{\Delta; \Gamma \vdash_\Sigma \mu \sqsubset \tau \Leftarrow \texttt{csort}}$

$$\frac{\Delta; \Psi \vdash S \sqsubset A \Leftarrow \texttt{Sort}}{\Delta; \Gamma \vdash S[\Psi] \sqsubset A[\Psi] \Leftarrow \texttt{csort}}$$

$$\frac{\Delta; \Gamma \vdash \mu_1 \sqsubset \tau_1 \Leftarrow \texttt{csort} \qquad \Delta; \Gamma, y{::}\mu_1 \sqsubset \tau_1 \vdash \mu_2 \sqsubset \tau_2 \Leftarrow \texttt{csort}}{\Delta; \Gamma \vdash \mu_1 \rightarrow \mu_2 \sqsubset \tau_1 \rightarrow \tau_2 \Leftarrow \texttt{csort}}$$

$$\frac{\Delta; \Psi \vdash S \sqsubset A \Leftarrow \texttt{Sort} \qquad \Delta, u{::}S[\Psi] \sqsubset A[\Psi]; \Gamma \vdash \mu \sqsubset \tau \Leftarrow \texttt{csort}}{\Delta; \Gamma \vdash \Pi^\Box u{::}S[\Psi].\mu \sqsubset \Pi^\Box u{:}A[\Psi].\tau \Leftarrow \texttt{csort}}$$

$$\frac{\Delta \vdash \Xi_1 \leq \Xi_2 \qquad \Delta, \psi{:}\Xi_1; \Gamma \vdash \mu \sqsubset \tau}{\Delta; \Gamma \vdash \Pi\psi{:}\Xi_1.\mu \sqsubset \Pi\psi{:}\Xi_2.\tau \Leftarrow \texttt{csort}}$$

Notes. (1) Missing rules for $\top$ and intersection sorts.

(2) It is not really necessary to mention `csort` everywhere in this judgment if we never check against other classes (although it's more likely that the other sorts are missing).

$\boxed{\Delta; \Gamma \vdash_\Sigma e \Leftarrow \mu \sqsubset \tau}$

$$\frac{\Delta; \Gamma \vdash i \Rightarrow \mu \sqsubset \tau \qquad \Delta; \Gamma \vdash \mu \leq \mu' \sqsubset \tau}{\Delta; \Gamma \vdash i \Leftarrow \mu' \sqsubset \tau} \qquad \frac{\Delta; \Psi \vdash M \Leftarrow S \sqsubset A}{\Delta; \Gamma \vdash \texttt{box}(\Psi.M) \Leftarrow S[\Psi] \sqsubset A[\Psi]}$$

$$\frac{\Delta; \Gamma, f{::}\mu \sqsubset \tau \vdash e \Leftarrow \mu \sqsubset \tau}{\Delta; \Gamma \vdash \texttt{rec}\ f.e \Leftarrow \mu \sqsubset \tau} \qquad \frac{\Delta; \Gamma, y{::}\mu_1 \sqsubset \tau_1 \vdash e \Leftarrow \mu_2 \sqsubset \tau_2}{\Delta; \Gamma \vdash \texttt{fn}\ y.e \Leftarrow \mu_1 \rightarrow \mu_2 \sqsubset \tau_1 \rightarrow \tau_2}$$

$$\frac{\Delta, u{::}S[\Psi] \sqsubset A[\Psi]; \Gamma \vdash e \Leftarrow \mu \sqsubset \tau}{\Delta; \Gamma \vdash \lambda^{\Box} u.e \Leftarrow \Pi^{\Box} u{::}S[\Psi].\mu \sqsubset \Pi^{\Box} u{:}A[\Psi].\tau} \qquad \frac{\Delta, \psi{:}\Xi; \Gamma \vdash e \Leftarrow \mu \sqsubset \tau}{\Delta; \Gamma \vdash \Lambda\psi.e \Leftarrow \Pi\psi{:}\Xi.\mu \sqsubset \Pi\psi{:}\Xi.\tau}$$

$$\frac{\Delta; \Gamma \vdash i \Rightarrow A[\Psi] \qquad \text{for all } k \;\; \Delta; \Gamma \vdash b_k \Leftarrow_{S[\Psi] \sqsubset A[\Psi]} \mu \sqsubset \tau}{\Delta; \Gamma \vdash \mathtt{case}\ i\ \mathtt{of}\ b_1 \mid ... \mid b_n \Leftarrow \mu \sqsubset \tau}$$

$\boxed{\Delta; \Gamma \vdash_{\Sigma} i \Rightarrow \mu \sqsubset \tau}$

$$\frac{\Delta; \Gamma \vdash e \Leftarrow \mu \sqsubset \tau}{\Delta; \Gamma \vdash (e{::}\mu \sqsubset \tau) \Rightarrow \mu \sqsubset \tau} \qquad \frac{y{::}\mu \sqsubset \tau \in \Gamma}{\Delta; \Gamma \vdash y \Rightarrow \mu \sqsubset \tau}$$

$$\frac{\Delta; \Gamma \vdash i \Rightarrow \mu_1 \to \mu_2 \sqsubset \tau_1 \to \tau_2 \qquad \Delta; \Gamma \vdash e \Leftarrow \mu_1 \sqsubset \tau_1}{\Delta; \Gamma \vdash i\ e \Rightarrow \mu_2 \sqsubset \tau_2}$$

$$\frac{\Delta; \Gamma \vdash i \Rightarrow \Pi\psi{:}\Xi_1.\mu \sqsubset \Pi\psi{:}\Xi_2.\tau \qquad \Delta; \Gamma \vdash \Psi : \Xi_1}{\Delta; \Gamma \vdash i\ \lceil \Psi \rceil \Rightarrow [\![\Psi/\psi]\!](\mu \sqsubset \tau)}$$

$$\frac{\Delta; \Gamma \vdash i \Rightarrow \Pi^{\Box} u{::}S[\Psi].\mu \sqsubset \Pi^{\Box} u{:}A[\Psi].\tau \qquad \Delta; \Psi \vdash M \Leftarrow S \sqsubset A}{\Delta; \Gamma \vdash i\ \lceil \Psi.M \rceil \Rightarrow [\![\Psi.M/u]\!](\mu \sqsubset \tau)}$$

$\boxed{\Delta; \Gamma \vdash_{\Sigma} b \Leftarrow_{\mu' \sqsubset \tau'} \mu \sqsubset \tau}$

$$\frac{\Delta; \Gamma \vdash M_k \Leftarrow S_k \sqsubset A_k \qquad \begin{array}{ll} \Delta, \Delta_k & \vdash \Psi \doteq \Psi_k/(\theta_1, \Delta') \\ \Delta' & \vdash [\![\theta_1]\!](S \sqsubset A) \doteq [\![\theta_1]\!](S_k \sqsubset A_k)/(\theta_2, \Delta'') \\ \Delta''; [\![\theta_2]\!][\![\theta_1]\!]\Gamma & \vdash [\![\theta_2]\!][\![\theta_1]\!]e_k \Leftarrow [\![\theta_2]\!][\![\theta_1]\!](\mu \sqsubset \tau) \end{array}}{\Delta; \Gamma \vdash \Pi\Delta_k.\mathtt{box}(\Psi.M_k) : S_k[\Psi_k] \sqsubset A_k[\Psi_k] \mapsto e_k \Leftarrow_{S[\Psi] \sqsubset A[\Psi]} \mu \sqsubset \tau}$$

$\boxed{\Delta; \Gamma \vdash_{\Sigma} \mu_1 \leq \mu_2 \sqsubset \tau}$

$$\frac{\Delta; \Psi \vdash S \sqsubset A}{\Delta; \Gamma \vdash S[\Psi] \sqsubset A[\Psi]}$$

$$\frac{\Delta; \Gamma \vdash \mu_2 \leq \mu_1 \sqsubset \tau \qquad \Delta; \Gamma, y{::}\mu_2 \sqsubset \tau \vdash \mu_1' \leq \mu_2' \sqsubset \tau'}{\Delta; \Gamma \vdash \mu_1 \to \mu_1' \leq \mu_2 \to \mu_2' \sqsubset \tau \to \tau'}$$

$$\frac{\Delta; \Psi \vdash S_2 \leq S_1 \sqsubset A \qquad \Delta, u{::}S_2[\Psi] \sqsubset A[\Psi]; \Gamma \vdash \mu_1 \leq \mu_2 \sqsubset \tau}{\Delta; \Gamma \vdash \Pi^{\Box} u{::}S_1[\Psi].\mu_1 \leq \Pi^{\Box} u{::}S_2[\Psi].\mu_2 \sqsubset \Pi^{\Box} u{:}A[\Psi].\tau}$$

$$\frac{\Delta \vdash \Xi_1 \leq \Xi_2 \qquad \Delta \vdash \Xi_2 \leq \Xi \qquad \Delta, \psi{:}\Xi_1; \Gamma \vdash \mu_1 \leq \mu_2 \sqsubset \tau}{\Delta; \Gamma \vdash \Pi\psi{:}\Xi_1.\mu_1 \leq \psi{:}\Xi_2.\mu_2 \sqsubset \Pi\psi{:}\Xi.\tau}$$

5 Operational Semantics

References

[1] Barthe, G., and Frade, M. J. Constructor subtyping. In *Programming Languages and Systems* (Berlin, Heidelberg, 1999), S. D. Swierstra, Ed., Springer Berlin Heidelberg, pp. 109–127.

[2] Harper, R., Honsell, F., and Plotkin, G. D. A framework for defining logics. In *Proceedings of the Symposium on Logic in Computer Science (LICS'87), Ithaca, New York, USA, June 22-25, 1987* (1987), IEEE Computer Society, pp. 194–204.

[3] Lovas, W., and Pfenning, F. Refinement types for logical frameworks and their interpretation as proof irrelevance. *Logical Methods in Computer Science 6*, 4 (dec 2010).

[4] McBride, C. Ornamental algebras, algebraic ornaments, 2011.

[5] Nanevski, A., Pfenning, F., and Pientka, B. Contextual modal type theory. *ACM Trans. Comput. Log. 9*, 3 (2008), 23:1–23:49.

[6] Pfenning, F. Church and Curry: Combining Intrinsic and Extrinsic Typing, 6 2000.

[7] Pientka, B. A type-theoretic foundation for programming with higher-order abstract syntax and first-class substitutions. In *Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2008, San Francisco, California, USA, January 7-12, 2008* (2008), G. C. Necula and P. Wadler, Eds., ACM, pp. 371–382.

[8] Pientka, B., and Dunfield, J. Programming with proofs and explicit contexts. In *Proceedings of the 10th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, July 15-17, 2008, Valencia, Spain* (2008), S. Antoy and E. Albert, Eds., ACM, pp. 163–173.

[9] Poll, E. Subtyping and inheritance for inductive types. In *Proceedings of TYPES'97 Workshop on Subtyping, inheritance and modular development of proofs, Durham, UK* (September 1997).