A computational interpretation of parametricity

Jean-Philippe Bernardy Guilhem Moulin
Chalmers University of Technology and University of Gothenburg
email: {bernardy,mouling}@chalmers.se

***Abstract*—Reynolds' abstraction theorem has recently been extended to lambda-calculi with dependent types. In this paper, we show how this theorem can be internalized. More precisely, we describe an extension of Pure Type Systems with a special parametricity rule (with computational content), and prove fundamental properties such as Church-Rosser's and strong normalization. All instances of the abstraction theorem can be both expressed and proved in the calculus itself. Moreover, one can apply parametricity to the parametricity rule: parametricity is itself parametric.**

I. Introduction

In his seminal paper, Reynolds [25] gave formal meaning to polymorphism, by interpreting types as predicates, in such a way that all inhabitants of a given type satisfy its interpretation as a predicate. Crucially, the type of types ($*$) is interpreted by the set of predicates. A simple example is that if a function f has type $\forall a : *. \, a \to a$ — the type of the polymorphic identity — then the following proposition holds[1]

$$\forall a : *. \forall \dot{a} : a \to *. \forall x : a. \, \dot{a}\, x \to \dot{a}\,(f\, a\, x)$$

(which implies that f must return exactly its argument).

The above result, *abstraction*, has proved useful for reasoning about functional programs [31] . It also has deep theoretical implications: for example, the induction principle over Church numerals can be deduced from it [32].

The study of parametricity is typically semantic, including the seminal work of Reynolds. There, the concern is to construct a model that captures the polymorphic character of a λ-calculus. Mairson [16] pioneered a different angle of study, of more syntactical nature: for each concrete term, a proof term which shows that it satisfies the relational interpretation of its type is constructed. That idea has then been used by various authors, including Abadi et al. [1], Plotkin and Abadi [23] and Wadler [32]. Bernardy et al. [8] have also shown how terms, types, their relational interpretation as proofs and propositions can all be expressed in a single calculus. The calculi where this is possible must however be rich enough: in particular they must support dependent types. Systems such as the Calculus of Constructions [10] or Martin-Löf's Intuitionistic Type Theory [17] both satisfy the requirements.

Still, even though we know that all terms of the above systems satisfy the parametricity condition, and each of these conditions can be expressed in the same system as the terms they concern, the very fact that any given term satisfies the relational interpretation of its type is *itself* not provable in the system. For example, the parametricity condition arising from the type $\forall a : *. \, a \to a$, namely

$$\forall f : (\forall a : *. \, a \to a). \quad \forall a : *. \forall \dot{a} : a \to *. \forall x : a. \, \dot{a}\, x \to \dot{a}\,(f\, a\, x),$$

is not provable in the Calculus of Constructions.

One would like to be able to rely on parametricity conditions within proof assistants themselves. Indeed, the correctness of numerous functional programming techniques relies on parametricity. Examples include program transformation [12, 14], testing [7], semantic program inversion [29] and generic programming [30]. Proof assistants based on type theory can already describe these techniques, and it would be useful to take advantage of parametricity to mechanize their proofs.

We are not the first to recognize the need for parametricity: it is for example a recurring topic in the field of mechanized metatheory, where precise encoding of variable bindings often makes use of polymorphism. For example, Pouillard [24] describes the following representation for terms, using the AGDA [20] proof assistant:

```
data Term (V : Set) : Set where
  var  : V → Term V
  app : Term V → Term V → Term V
  abs  : (Maybe V → Term V) → Term V
```

with the intent that closed terms should quantify universally over V. One can argue that terms defined as above must be well-scoped as follows: because V is an abstract type variable, the only way to obtain a type-correct argument to the var constructor is from the higher-order binding in the abs constructor. Pouillard formalizes the above argument within AGDA, using logical relations as defined by Bernardy et al. [8]. Only the parametricity axiom is missing to establish that all terms are well-scoped. Chlipala [9] and Atkey et al. [3] encounter the same type of situation.

Another application of parametricity is a new kind of *meta-programming*. Indeed, via the Curry-Howard isomorphism, proofs can be interpreted as programs, and in particular the proof that a term is parametric can be a useful program in its own right. Bernardy [5, p. 82] shows for example how to derive the vectors of length n from lists with a (modified) version of parametricity. Not only the type is derived, but operations on lists are transformed to their counterpart on vectors.

In this paper, we address the lack of support for parametricity in logical frameworks. Technically, we propose to extend the pure type systems (PTSs) to make all the parametricity

[1]We use the convention that the variable $\dot{a}$ is the predicate corresponding to the type variable a.

propositions provable internally. The aim is to pave the way for native support of parametricity in tools based on dependent types, such as AGDA or COQ [28]. The challenge is not to merely postulate the axiom and check its consistency with the rest of the system: because we want to retain the constructive character of these proof assistants, we must provide a computation rule for parametricity. This computational aspect is not only philosophically satisfying, it enables the usage of the parametricity as a meta-programming tool, as outlined above.

Furthermore, the new construction can itself be given a parametric interpretation, which preserves the abstraction theorem (in terms of itself). That is, in our calculus, *parametricity is parametric*. As we shall explain, to support this form of self-applicability, our design uses quantification over multi-dimensional objects. Beside parametricity, the use of multiple dimensions is a characteristic feature of our calculus.

Our technical contributions are as follows:

- We describe a dependently-typed λ-calculus with internalized parametricity. The calculus is summarized in definitions 5, 6, and 3; And motivated in Sec. II.
- We prove that it satisfies typical properties of a well-behaved λ-calculus with types. In particular, we prove the Church-Rosser property, subject reduction. We also prove strong normalization relatively to the corresponding calculus without parametricity, by translation into that calculus.
- An implementation of a type-checker for the calculus is made available for experimentation.

II. Design, step by step

In this section we describe and motivate our design step by step, starting from pure type systems.

A. Pure type systems, and our notation

We assume familiarity with pure type systems (PTSs), but we give a brief reminder in the following paragraphs, as well as an introduction to our notation. Readers are referred to Barendregt [4] for details.

PTSs are a family of λ-calculi, parameterized by a set of sorts $\mathcal{S}$, a set of axioms $\mathcal{A} \subseteq \mathcal{S} \times \mathcal{S}$ and set of rules $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S} \times \mathcal{S}$. The various syntactic forms of quantifications (and corresponding abstraction and application) are syntactically unified, and one needs to inspect sorts to identify which form is meant. The axioms $\mathcal{A}$ give the typing rules for sorts, and $\mathcal{R}$ determines which forms of quantification exist in the system. Many systems (*e.g.*, the Calculus of Constructions or System F) can be cast into the framework of PTSs. The syntax for PTS terms is the following:

$$\begin{array}{lccc}
\mathsf{Term} \ni A,\ldots,U & = & s & \text{sort} \\
 & | & x & \text{variable} \\
 & | & A\,B & \text{application} \\
 & | & \lambda x : A.\, B & \text{abstraction} \\
 & | & \forall x : A.\, B & \text{product}
\end{array}$$

The product $\forall x : A.\, B$ may be also written $A \to B$ when x does not occur free in B. In the rest of the paper we assume a given PTS specification $(\mathcal{S}, \mathcal{A}, \mathcal{R})$. We name the calculus arising from that specification $\mathcal{O}$.

The major part of the paper is devoted to the description and justification of another calculus, here called $\mathcal{P}$, parameterized over the same specification $(\mathcal{S}, \mathcal{A}, \mathcal{R})$, which extends $\mathcal{O}$ and contains our special construction for parametricity. The syntax and typing rules for $\mathcal{P}$ are given in definitions 5, 6, and 3.

B. Logical relations, from PTS to PTS

In this section we recall the relational interpretation of terms and types of the PTS $\mathcal{O}$ into another PTS, here called $[\![\mathcal{O}]\!]$. The material of this section is essentially the same as in Bernardy et al. [8] and Bernardy and Lasson [6].

In any PTS, types and terms can be interpreted as relations and proofs that the terms satisfy the relations. Each *type* can be interpreted as a predicate that its inhabitants satisfy; And each *term* can be turned into a proof that it satisfies the predicate of its type. Usual presentations of parametricity use binary relations, but for simplicity of notation we present here a unary version. The generalization to arbitrary arity is straightforward, and we refer the readers to [6] for details.

In the following we define what it means for a program C to satisfy the predicate generated by a type T (the proposition $C \in [\![T]\!]$); and the translation from a program C of type T to a proof $[\![C]\!]$ that C satisfies the predicate. A property of the translation is that whenever x is free in T, there is another free variable $\dot{x}$ in $[\![T]\!]$, which witnesses that x satisfies the parametricity condition of T ($\dot{x} : x \in [\![T]\!]$). This means that the translation extends contexts, as follows:

$$\begin{aligned}
[\![-]\!] &= - \\
[\![\Gamma, x : A]\!] &= [\![\Gamma]\!], x : A, \dot{x} : x \in [\![A]\!]
\end{aligned}$$

It is important to notice that this definition assumes a *global* renaming from each variable x to a fresh variable $\dot{x}$. (The renaming will be made local in further sections.)

The relational interpretation of a type T to a proposition $C \in [\![T]\!]$, defined as a syntactic translation from terms to terms by structural induction on T, as follows.

- Because types in a PTS are abstract (there is no pattern matching on types), any predicate over C can be used to witness that C satisfies the relational interpretation of a sort s.

$$C \in [\![s]\!] = C \to s$$

- If the type is a product ($\forall x : A.\, B$), then C is a function, and it satisfies the relational interpretation of its type iff it maps related inputs to related outputs.

$$C \in [\![\forall x : A.\, B]\!] = \forall x : A.\, \forall \dot{x} : x \in [\![A]\!].\, (C\,x) \in [\![B]\!]$$

- For any other syntactic form for a type T of sort s, (which, if it is well-typed, may be a variable or an application), T is interpreted as a predicate ($[\![T]\!] : T \to s$), and to check that C satisfies it, one can use application.

$$C \in [\![T]\!] = [\![T]\!]\, C$$

There remains to give the translation from a term C to a proof term $[\![C]\!]$, also defined by structural induction on the term C.

- The translation of a variable is done by looking up the corresponding parametric witness in the context.
$$[\![x]\!] = \dot{x}$$
- The case for abstraction adds a witness that the input satisfies the relational interpretation of its type and returns the relational interpretation of the body, mirroring the interpretation of product:
$$[\![\lambda x : A.\, B]\!] = \lambda x : A.\, \lambda \dot{x} : x \in [\![A]\!].\, [\![B]\!]$$
- The application follows the same pattern: the function is passed a witness that the argument satisfies the interpretation of its type.
$$[\![A\, B]\!] = [\![A]\!]\, B\, [\![B]\!]$$
- If the term has another syntactic form, then it is a type (T), thus we can use λ-abstraction to create a predicate and check that the abstracted variable z satisfies the relational interpretation of the type in the body ($z \in [\![T]\!]$).
$$[\![T]\!] = \lambda z : T.\, z \in [\![T]\!]$$
At this point the definition might appear circular; but in fact the form $\cdot \in [\![T]\!]$ invokes $[\![T]\!]$ *only* when T is an application, which is processed structurally by $[\![\cdot]\!]$.

Bernardy and Lasson [6] prove the following theorem:

Theorem 1 (Abstraction). *If* $\Gamma \vdash_{\mathcal{O}} A : B : s$, *then*
$$[\![\Gamma]\!] \vdash_{[\![\mathcal{O}]\!]} [\![A]\!] : (A \in [\![B]\!]) : s$$
where $[\![\mathcal{O}]\!]$ *is a PTS computed from* $\mathcal{O}$ *Furthermore, if* $\mathcal{O}$ *is consistent, then so is the* $[\![\mathcal{O}]\!]$.

Proof: By induction on the derivation. ■

A direct reading of the above result is as a typing judgement about translated terms: if A has type B, then $[\![A]\!]$ has type $A \in [\![B]\!]$. However, it can also be understood as an abstraction theorem for $\mathcal{O}$: if a program A has type B in Γ, then A satisfies the relational interpretation of its type ($A \in [\![B]\!]$). For arity 2, $[\![\Gamma]\!]$ contains two related environments (and witnesses that they are properly related), and $[\![A]\!]$ is a proof that the two possible interpretations of A (by picking variables out of each environment in $[\![\Gamma]\!]$) are related.

In general, the PTS $[\![\mathcal{O}]\!]$, where parametricity conditions are expressed, is more general than the source system $\mathcal{O}$. However, for rich enough systems, such as the calculus of constructions, they are equivalent (see [8, 6] for the conditions where this occurs).

C. Aim and example

Let us assume a PTS $\mathcal{Q}$, such that $\mathcal{Q}$ is equivalent to $[\![\mathcal{Q}]\!]$. Because both types and their parametricity conditions can be expressed in $\mathcal{Q}$, one can hope that for every term A of type B, the user of the system can get a witness $[\![A]\!]$ that it is parametric ($A \in [\![B]\!]$). Even though this hope is fulfilled for closed terms, we run out of luck for open terms, because the context where $[\![A]\!]$ is meaningful is "bigger" than that where A is: for each free variable $x : A$ in Γ, we need a variable $\dot{x} : x \in [\![A]\!]$ in $[\![\Gamma]\!]$. In other words, from $\Gamma \vdash_{\mathcal{Q}} A : B$ we have $[\![\Gamma]\!] \vdash_{\mathcal{Q}} [\![A]\!] : A \in [\![B]\!]$, but we really need $\Gamma \vdash_{\mathcal{Q}} [\![A]\!] : A \in [\![B]\!]$. Indeed, we do not want to add an explicit assumption that every variable is bound to a parametric value. The aim of this paper is to find a system $\mathcal{P}$ such that the following proposition is verified.

Proposition 1 (Internal parametricity).
$$\Gamma \vdash_{\mathcal{P}} A : B \Rightarrow \Gamma \vdash_{\mathcal{P}} [\![A]\!] : A \in [\![B]\!]$$

In that case, for any term A, users of $\mathcal{P}$ can invoke the fact that A is parametric, by writing $[\![A]\!]$.

Example 1. *For example, further assuming that* $\mathcal{P}$ *embeds the Calculus of Constructions, we can prove that any function of type* $\forall a : *.\, a \to a$ *is an identity, as we hinted at in the introduction. The formulation of the theorem within* $\mathcal{P}$ *and its proof term are as follows.*

$$\begin{aligned}&\mathit{identities} : \forall f : (\forall a : *.\, a \to a).\, \forall a : *.\, \forall x : a.\, \mathit{Eq}\, a\, (f\ a\ x)\, x\\ &\mathit{identities} = \lambda f.\lambda a.\lambda x.[\![f]\!]\, a\, (\mathit{Eq}\, a\, x)\, x\, (\mathit{refl}\, a\, x)\end{aligned}$$

where Eq *stands for Leibniz equality. (The type of* $[\![f]\!]$ *is* $\forall a : *.\, \forall \dot{a} : a \to *.\, \forall x : a.\, \dot{a}\, x \to \dot{a}\, (f\, a\, x)$.*)*

If $\mathit{identities}$ *is used on a concrete identity function, say* $i = \lambda a : *.\, \lambda x : a.\, x$, *then* $f\, a\, x$ *reduces to* x, *and the theorem specializes to reflexivity of equality:*

$$\mathit{identities}\, i\ :\ \forall a : *.\, \forall x : a.\, \mathit{Eq}\, a\, x\, x$$

after reduction the proof no longer mentions $[\![\cdot]\!]$*:*

$$\begin{aligned}&\mathit{identities}\, i\\ &\quad \to_\beta \lambda a.\lambda x.[\![f]\!][f \mapsto \lambda a : *.\, \lambda x : a.\, x]\, a\, (\mathit{Eq}\, a\, x)\, x\, (\mathit{refl}\, a\, x)\\ &\quad = \lambda a.\lambda x.[\![\lambda a : *.\, \lambda x : a.\, x]\!]\, a\, (\mathit{Eq}\, a\, x)\, x\, (\mathit{refl}\, a\, x)\\ &\quad = \lambda a.\lambda x.(\lambda a.\lambda \dot{a}.\lambda x.\lambda \dot{x}.\dot{x})\, a\, (\mathit{Eq}\, a\, x)\, x\, (\mathit{refl}\, a\, x)\\ &\quad \to_\beta \lambda a.\lambda x.\mathit{refl}\, a\, x\end{aligned}$$

D. Internalization

We have seen that the abstraction theorem (Th. 1) for PTSs gives us something very close to Prop. 1, except that for each free variable $x : A$ in Γ, we need an explicit witness that x is parametric ($\dot{x} : x \in [\![A]\!]$) in the environment.

However, we know that every closed term is parametric. Therefore, ultimately, we know that for each possible *concrete* term a that can be substituted for a free variable x, it is possible to construct a concrete term $[\![a]\!]$ to substitute for $\dot{x}$. This means that the witness of parametricity for x does not need to be given explicitly (if x is bound). Therefore we allow to access such a witness via the new syntactic form $[\![x]\!]$. This intuition justifies the addition of the substitution rule

$$[\![x]\!]\, [x \mapsto a] = [\![a]\!]$$

as well as the following typing rule, expressing that if x is found in the context, then it is valid to use $\lceil\!\lceil x \rceil\!\rceil$, which is the witness that x satisfies the parametricity condition of its type.

$$\frac{\Gamma \vdash A : s}{\Gamma, x : A \vdash \lceil\!\lceil x \rceil\!\rceil : x \in \llbracket A \rrbracket}$$

At the same time, we must amend the parametric interpretation to remember which variables are assigned an explicit witness, and which variables must wait for a concrete term. We write the list of variables given an explicit witness as an index to $\llbracket \cdot \rrbracket$. For example, abstraction is translated as follows:

$$\llbracket \lambda x : A.\, B \rrbracket_\xi = \lambda x : A.\, \lambda \dot{x} : x \in \llbracket A \rrbracket_\xi .\, \llbracket B \rrbracket_{\xi, x}$$

and other cases are modified accordingly. In particular, the interpretation of variables becomes[2]:

$$\begin{aligned} \llbracket x \rrbracket_\xi &= \dot{x} && \text{if } x \in \xi \\ \llbracket x \rrbracket_\xi &= \lceil\!\lceil x \rceil\!\rceil && \text{if } x \notin \xi \end{aligned}$$

(From here on, we may omit the index $\llbracket A \rrbracket$ to mean $\llbracket A \rrbracket_\varnothing$.)

The above construction solves the issue of context extension. That is, every term A of a PTS $\mathcal{Q}$ can be proven parametric by using $\llbracket A \rrbracket$ without extending the context where A is typeable. Another aspect of the result is that, assuming parametricity on variables, we have parametricity for all terms. This means that, in a practical language featuring parametricity, the parametric construction can be used on any term, but in normal forms $\lceil\!\lceil \cdot \rceil\!\rceil$ only appears on variables, maybe in a nested way.

Unfortunately, Prop. 1 does not quite hold at this stage, because there is an issue in applying abstraction to the new $\lceil\!\lceil \cdot \rceil\!\rceil$ construct, as we show on an example in the next section.

E. Parametricity of parametricity

The fact that all values are parametric is also captured by the following theorem (inside the calculus):

$$\begin{aligned} &\mathit{parametricity} : \forall A : *.\, \forall (a : A).\, a \in \llbracket A \rrbracket \\ &\mathit{parametricity} = \lambda A.\, \lambda a : A.\, \lceil\!\lceil a \rceil\!\rceil \end{aligned}$$

Since all terms are assumed parametric, in particular it should be possible to apply $\llbracket \cdot \rrbracket$ to the above term. For some closed type A, consider the term $\llbracket \mathit{parametricity}\, A \rrbracket$. It is convertible to $\lambda x : A.\, \lambda \dot{x} : x \in \llbracket A \rrbracket .\, \llbracket \lceil\!\lceil x \rceil\!\rceil \rrbracket_{\{x\}}$.

So far, we have not specified how to reduce the subterm $\llbracket \lceil\!\lceil x \rceil\!\rceil \rrbracket_{\{x\}}$ (where x is a free variable). Indeed, it is actually not possible to substitute x for a value in it, because x also appears as an index of $\llbracket \cdot \rrbracket$, and only variables can appear there (not arbitrary terms). This means that $\llbracket \lceil\!\lceil x \rceil\!\rceil \rrbracket_{\{x\}}$ is not an acceptable normal form: it must reduce to something else. A perhaps natural idea is to modify the reduction rules in such a way as to allow the following reduction, which exchanges the two occurrences of the parametric interpretation:

$$\llbracket \lceil\!\lceil x \rceil\!\rceil \rrbracket_{\{x\}} \longrightarrow \lceil\!\lceil \llbracket x \rrbracket_{\{x\}} \rceil\!\rceil .$$

[2]Careful readers might worry that we discard the index in the second case. An informal justification is that if x has no explicit witness, then the free variables of its type do not either. Therefore, types are preserved when doing the substitution.

In that case the expression further reduces to $\lceil\!\lceil \dot{x} \rceil\!\rceil$, which is a proper normal form. Unfortunately, this reduction *does not preserve types*. This can be checked by assuming $x : A$, and by computing the types of the expression before and after reduction. By Prop. 1 we have $\lceil\!\lceil x \rceil\!\rceil : \llbracket A \rrbracket\, x$. By Abstraction (giving an explicit parametric witness for x), we get

$$\begin{aligned} \llbracket \lceil\!\lceil x \rceil\!\rceil \rrbracket_{\{x\}} &: \llbracket \llbracket A \rrbracket\, x \rrbracket_{\{x\}}\, \lceil\!\lceil x \rceil\!\rceil \\ &: \llbracket \llbracket A \rrbracket \rrbracket_{\{x\}}\, x\, \llbracket x \rrbracket_{\{x\}}\, \lceil\!\lceil x \rceil\!\rceil \\ &: \llbracket \llbracket A \rrbracket \rrbracket\, x\, \llbracket x \rrbracket_{\{x\}}\, \lceil\!\lceil x \rceil\!\rceil \\ &: \llbracket \llbracket A \rrbracket \rrbracket\, x\, \dot{x}\, \lceil\!\lceil x \rceil\!\rceil \end{aligned}$$

On the other hand, by Abstraction we have $\llbracket x \rrbracket_{\{x\}} : \llbracket A \rrbracket\, x$, and by application of Prop. 1, we get

$$\begin{aligned} \lceil\!\lceil \llbracket x \rrbracket_{\{x\}} \rceil\!\rceil &: \llbracket \llbracket A \rrbracket\, x \rrbracket\, \llbracket x \rrbracket_{\{x\}} \\ &: \llbracket \llbracket A \rrbracket \rrbracket\, x\, \lceil\!\lceil x \rceil\!\rceil\, \llbracket x \rrbracket_{\{x\}} \\ &: \llbracket \llbracket A \rrbracket \rrbracket\, x\, \lceil\!\lceil x \rceil\!\rceil\, \dot{x} \end{aligned}$$

That is, in the above example, the reduction rule suggested above has the effect to swap the second and third arguments to $\llbracket \llbracket A \rrbracket \rrbracket$ in the type, which means that subject reduction would not hold if we were to have the above, naive rule.

However, one observes that, for a closed type A, the relation $\llbracket \llbracket A \rrbracket \rrbracket\, x$ is symmetric: a proof of $\llbracket \llbracket A \rrbracket \rrbracket\, x\, B\, C$ *is logically equivalent* to a proof of $\llbracket \llbracket A \rrbracket \rrbracket\, x\, C\, B$. Thus the swapping observed above is harmless, and it is suffices to deal with it in a technical fashion.

Example 2. *The relation* $\llbracket \llbracket (a : *) \to a \to a \rrbracket \rrbracket\, f$ *is symmetric for any* f.

In the light of this observation, we introduce a special-purpose operator (pronounced exchange) $\cdot \ddagger^\pi$, which applies a permutation to the arguments of relations, and which permutes their types in the same way.

$$\frac{\Gamma \vdash A : B}{\Gamma \vdash A \ddagger^\pi : B \ddagger^\pi}$$

Thanks to this operation we can now have a reduction relation that preserves types in the above situation:

$$\llbracket \lceil\!\lceil x \rceil\!\rceil \rrbracket_{\{x\}} \longrightarrow \lceil\!\lceil \llbracket x \rrbracket_{\{x\}} \rceil\!\rceil \ddagger^{(1,2)} .$$

Supporting this operation requires deep changes in the syntax, exposed in the next section.

F. A syntax for hypercubes

In order to support the swapping operation, we need to indicate the role of each of the arguments to the relations explicitly, in the syntax. That is, the type of $\llbracket \lceil\!\lceil x \rceil\!\rceil \rrbracket_{\{x\}}$ should be written

$$\llbracket \llbracket A \rrbracket \rrbracket \bullet \begin{pmatrix} x & \dot{x} \\ \lceil\!\lceil x \rceil\!\rceil & \cdot \end{pmatrix} .$$

Then, we can arrange to have the following:

$$\left(\llbracket \llbracket A \rrbracket \rrbracket \bullet \begin{pmatrix} x & \dot{x} \\ \lceil\!\lceil x \rceil\!\rceil & \cdot \end{pmatrix} \right) \ddagger^{(1,2)} =_\beta \llbracket \llbracket A \rrbracket \rrbracket \bullet \begin{pmatrix} x & \lceil\!\lceil x \rceil\!\rceil \\ \dot{x} & \cdot \end{pmatrix}$$

In general, we need to remember the grouping of arguments when applying the relational interpretation. Essentially, one iteration of the relational interpretation transforms an application of an argument into application of two arguments. After a second iteration, there will be four arguments, and 2^n after n iterations. We must change the syntax of the application to make these 2^n arguments appear grouped together. Abstraction and product follow the same pattern as application. Hence, we can arrange our bindings as oriented n-cubes in general. Using overbar to denote cube meta-variables, the syntax becomes the following:

$$\begin{array}{rcl} \mathsf{Term} & = & A\,\bar{B} \\ & | & \lambda\bar{x} : \bar{A}.\, B \\ & | & \forall\bar{x} : \bar{A}.\, B \\ & \ldots & \end{array}$$

A binding $\bar{x} : \bar{B}$ introduces 2^n variables x_i, where i is any bit-vector of size n, and n is the dimension of $\bar{B}$. Consider the binding $\bar{x} : \bar{B}$. If $\bar{B}$ has dimension zero, it stands for a single binding $x : B$. If it has dimension 1, it contains a type B_0, and a predicate B_1 over B_0. Abusing matrix notation, one could write

$$\bar{x} : \begin{pmatrix} B_0 \\ B_1 \end{pmatrix} \simeq \begin{pmatrix} x_0 : B_0 \\ x_1 : B_1\, x_0 \end{pmatrix}$$

At dimension two, the cube $\bar{B}$ contains a type B_{00}, two predicates B_{01} and B_{10} over B_{00}, and a relation B_{11}, between B_{00}, $B_{10}\, x_{00}$, and $B_{01}\, x_{00}$.

$$\bar{x} : \begin{pmatrix} B_{00} & B_{01} \\ B_{10} & B_{11} \end{pmatrix} \simeq \begin{pmatrix} x_{00} : B_{00} & x_{01} : B_{01}\, x_{00} \\ x_{10} : B_{10}\, x_{00} & x_{11} : B_{11}\, x_{00}\, x_{01}\, x_{10} \end{pmatrix}$$

We furthermore need a special syntax for the introduction, elimination and formation of relations, which correspond to application, abstraction and quantification over incomplete cubes (those that lack an element at index 1...1). Such a cube is found for example in the type of x_{11} above. Using a check to denote incomplete cube meta-variables:

$$\begin{array}{rcl} \mathsf{Term} & = & A \bullet \check{B} \\ & | & \lambda^{\bullet}\check{x} : \check{A}.\, B \\ & | & \check{A} \overset{\bullet}{\to} B \\ & \ldots & \end{array}$$

Using this syntax, we can finally write the type of x_{11} in the form we need: $B_{11}\bullet\begin{pmatrix} x_{00} & x_{01} \\ x_{10} & \cdot \end{pmatrix}$. The type of B_{11} is $\begin{pmatrix} B_{00} & B_{01} \\ B_{10} & \cdot \end{pmatrix} \overset{\bullet}{\to} s$. For a cube of arbitrary dimension, $x_{1...1} : B_{1...1}\bullet(\bar{x}/\!\!/1...1)$ and $B_i : (\bar{B}/\!\!/i) \overset{\bullet}{\to} s$, where $\bar{B}/\!\!/1...1$ denotes the cube $\bar{B}$ with the top vertex removed. Further generalizing, x_i is a witness that the sub-cube found by removing all the dimensions d such that $i_d = 0$ satisfies the relation B_i:

$$x_i : B_i\bullet(\bar{x}/\!\!/i)$$

where $\bar{B}/\!\!/i$ is the cube obtained by discarding the elements of the cube $\bar{B}$ for each dimension d where $i_d = 0$, and then removing the top vertex.

$$\bar{B}/\!\!/i = \left[\ j \mapsto B_{j\&i}\ \right]^{\|i\|}_{j\in\mathbf{2}^{\|i\|}-1}$$

where $\|i\| = \sum_d i_d$ and $\begin{array}{rl} j\&(0i) & = 0(j\&i) \\ (bj)\&(1i) & = b(j\&i) \end{array}$

B_i is then a relation over corresponding sub-cube of $\bar{B}$, which is written formally:

$$B_i : (\bar{B}/\!\!/i) \overset{\bullet}{\to} s$$

G. The interpretation of hypercubes

Having given the new syntax of terms, we can express the relational interpretation using this new syntax. The interpretation of a cube increases its dimension; to each element is associated its interpretation:

$$[\![\bar{A}]\!]_\xi = \left[\begin{array}{l} 0i \mapsto \{A_i\}_\xi \\ 1i \mapsto [\![A_i]\!]_\xi \end{array}\right]^{\dim \bar{A}+1}_{i\in\mathbf{2}^{\dim \bar{A}}}$$

where $\{A\}_\xi = A[x_i \mapsto x_{0i}, \forall x \in \xi]$ undoes the renaming from i to $0i$ performed above; However, we will freely omit this detail from here on, and ignore the leading 0's of our bit-vectors, identifying $0i$ with i.

If a binding $\bar{x}$ has been extended by the interpretation, a variable x_i is then interpreted as x_{1i}.

$$[\![x_i]\!]_{\xi,x} = x_{1i}$$

The interpretation of terms mentioning full cubes (of size 2^n for some n) is the following:

$$\begin{aligned} [\![A\,\bar{B}]\!]_\xi &= [\![A]\!]_\xi\, [\![\bar{B}]\!]_\xi \\ [\![\lambda\bar{x} : \bar{A}.\, B]\!]_\xi &= \lambda\bar{x} : [\![\bar{A}]\!]_\xi.\, [\![B]\!]_{\xi,x} \\ C \in [\![\forall\bar{x} : \bar{A}.\, B]\!]_\xi &= \forall\bar{x} : [\![\bar{A}]\!]_\xi.\, (C\ (\bar{x}/01...1)) \in [\![B]\!]_{\xi,x} \end{aligned}$$

The interpretation of the cubes of size $2^n - 1$ used for relations requires some care. Because the index 1...1 is missing in such a cube, applying the same method as for full cubes leaves two elements missing, at indices 1...1 and 01...1. The former is supposed to be missing (because the resulting cube is also incomplete), but the latter is dependent on the context. Hence we introduce the following notation for interpretation of incomplete cubes where the "missing element" is explicitly specified to be B:

$$([\![\check{A}]\!]_\xi \oplus B) = \left[\begin{array}{l} 0i \mapsto \{A_i\}_\xi \\ 1i \mapsto [\![A_i]\!]_\xi \\ 01...1 \mapsto B \end{array}\right]^{\dim \check{A}+1}_{i\in\mathbf{2}^{\dim \check{A}}-1}$$

The parametric interpretation of the special forms for relation formation, membership and product are as follows.

$$\begin{aligned} C \in [\![\check{A} \overset{\bullet}{\to} s]\!]_\xi &= ([\![\check{A}]\!]_\xi \oplus C) \overset{\bullet}{\to} s \\ C \in [\![A\bullet\check{B}]\!]_\xi &= [\![A]\!]_\xi\bullet([\![\check{B}]\!]_\xi \oplus C) \\ [\![\lambda^{\bullet}\check{x} : \check{A}.\, B]\!]_\xi &= \lambda^{\bullet}\check{x} : ([\![\check{A}]\!]_\xi \oplus (\lambda^{\bullet}\check{x} : \check{A}.\, B)).\, x_{01...1} \in [\![B]\!]_{\xi,x} \end{aligned}$$

They are a straightforward consequence of the usual parametric interpretation and our choice of grouping arguments in cubes. Readers familiar with realizability interpretations (in the style for example of [21]) will notice a similarity here: the interpretation of a function space adds a quantification; and the other forms behave accordingly. Note that the form $A\bullet\check{B}$ is always a type, and therefore we interpret it as such.

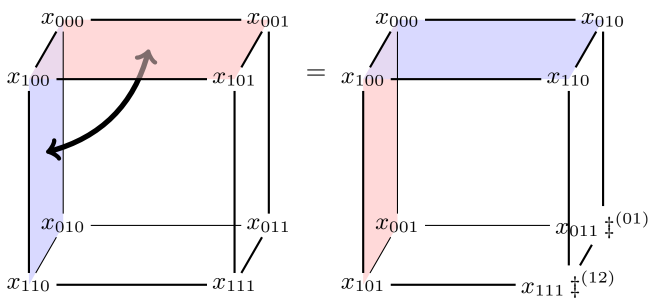
Example 3 (Repeated application of $[\![\cdot]\!]$)**.**

$$[\![\mathit{parametricity}\ A]\!] = \lambda \bar{a} : [\![(A)]\!].\ [\![a_1]\!] \ddagger^{(01)}$$

where $\bar{a} : [\![(A)]\!]$ *can be understood as* $\begin{pmatrix} a_0 \\ a_1 \end{pmatrix} : \begin{pmatrix} A \\ a_0 \in [\![A]\!] \end{pmatrix}$.

H. Exchanging dimensions

Given the above definition of cubes, it is straightforward to define an operation that applies an arbitrary permutation of its dimensions. For dimension $n = 0$ or $n = 1$, there is no non-trivial permutation. In the case of a square ($n = 2$), there is only one permutation, which is a simple swapping of the elements at indices 01 and 10. For higher dimensions ($n \geq 3$), the elements of the cube are multidimensional themselves (the dimension of an element at index i is $\|i\|$). Thus, one must take care to perform the exchange properly for each element. For instance, performing an exchange of dimensions 1 and 2 in a cube $\bar{x}$ for $n = 3$ involves exchanging dimensions 0 and 1 of the element x_{011}. Indeed, exchanging the dimensions 1 and 2 in the cube has the effect to exchange dimensions in the square occupied by x_{011}; so an exchange has to be performed on x_{011} to restore the cube structure. Geometrically, exchanging the dimensions as above corresponds to twisting the cube: two faces are swapped, and another is twisted. The situation is shown graphically in the following picture.



In general, applying a permutation π on the dimensions of a cube $\bar{C}$ is done as follows:

Definition 1 (Cube exchange)**.**

$$\bar{C} \ddagger^{\pi} = \left[\ i \mapsto C_{\pi(i)} \ddagger^{\pi/i}\ \right]_{i \in \mathbf{2}^{\dim \bar{C}}}^{\dim \bar{C}}$$

Where π/i stands for the permutation π restricted to the dimensions d where $i_d = 1$.
Incomplete cubes are permuted in the same way (simply omitting the top vertex).

Definition 2. *If π is a permutation $\{d \mapsto x_d\}$,*
$\pi/i = \mathrm{canon}\{d \mapsto x_d \mid i_d = 1\}$, *where* canon *maps the domain and co-domain of the function $\{d \mapsto x_d \mid i_d = 1\}$ to the set $\{0..\|i\| - 1\}$, preserving the order.*

Example 4. *If $\pi = \{0 \mapsto 0, 1 \mapsto 2, 2 \mapsto 1\}$ swaps dimensions 1 and 2, we have*

i	$\{d \mapsto \pi(d) \mid i_d = 1\}$	π/i
001	$\{2 \mapsto 1\}$	$\{0 \mapsto 0\}$
010	$\{1 \mapsto 2\}$	$\{0 \mapsto 0\}$
100	$\{0 \mapsto 0\}$	$\{0 \mapsto 0\}$
011	$\{1 \mapsto 2, 2 \mapsto 1\}$	$\{0 \mapsto 1, 1 \mapsto 0\}$
101	$\{0 \mapsto 0, 2 \mapsto 1\}$	$\{0 \mapsto 0, 1 \mapsto 1\}$
110	$\{0 \mapsto 0, 1 \mapsto 2\}$	$\{0 \mapsto 0, 1 \mapsto 1\}$

Applying a permutation to terms is then a matter of permuting all the cubes encountered:

$$\begin{aligned}
(A\,\bar{B}) \ddagger^{\pi}_{\xi} &= A \ddagger^{\pi}_{\xi}\ \bar{B} \ddagger^{\pi}_{\xi} \\
(\lambda \bar{x} : \bar{A}.\, B) \ddagger^{\pi}_{\xi} &= \lambda \bar{x} : \bar{A} \ddagger^{\pi}_{\xi}.\, B[\bar{x} \mapsto \bar{x} \ddagger^{\pi}] \ddagger^{\pi}_{\xi,x} \\
(\forall \bar{x} : \bar{A}.\, B) \ddagger^{\pi}_{\xi} &= \forall \bar{x} : \bar{A} \ddagger^{\pi}_{\xi}.\, B[\bar{x} \mapsto \bar{x} \ddagger^{\pi}] \ddagger^{\pi}_{\xi,x}
\end{aligned}$$

(and similarly for the incomplete cubes). There remains to explain the interaction with the special constructs, $[\![\cdot]\!]$ and $\cdot \ddagger^{\cdot}$ itself. We do so by listing four laws which hold in our calculus.

The first law is not surprising: the composition of exchanges is the exchange of the composition.

$$A \ddagger^{\rho} \ddagger^{\pi} =_{\beta} A \ddagger^{\rho \circ \pi} \tag{1}$$

Regarding the interactions between $[\![\cdot]\!]$ and $\cdot \ddagger^{\pi}$, recall first that the relational interpretation adds one dimension to cubes. By convention, the dimension added by $[\![\cdot]\!]$ is at index 0, and all other dimensions are shifted by one. Therefore, the relational interpretation of an exchange merely lifts the exchange out, and shifts indices by ones in its permutation, leaving dimension 0 intact.

$$[\![A \ddagger^{(x_1\ x_2 \cdots x_n)}]\!] =_{\beta} [\![A]\!] \ddagger^{(x_1+1\ x_2+1 \cdots x_n+1)} \tag{2}$$

The law that motivates the introduction of exchanges is the following:

$$[\![[\![A]\!]]\!]_{\xi} =_{\beta} [\![[\![A]\!]_{\xi}]\!] \ddagger^{(01)} \tag{3}$$

This law can also be explained by the convention that $[\![\cdot]\!]$ inserts always dimension 0. By commuting the uses of parametricity, dimensions are be swapped, and the exchange operator restores the order.

Lastly, one can also simplify exchanges in the presence of symmetric terms. We know that a term $[\![A]\!]^{n}$ is symmetric in its n first dimensions. Thus, applying a permutation that touches only dimensions $0..n-1$ to such a term has no effect. Formally, we have:

$$[\![A]\!]^{n} \ddagger^{(x_1\ x_2 \cdots x_m)} =_{\beta} [\![A]\!]^{n} \qquad \text{if } \forall i \in 1..m, x_i < n \tag{4}$$

We have seen before that it suffices to provide parametricity only for variables, and that the construct $[\![\cdot]\!]$ essentially acts as a “macro” on other constructs. The situation is not changed in the presence of dimension exchanges: (2) explains how to compute the parametricity witness of an exchange. For the $\cdot \ddagger^{\pi}$ construct, the situation is analogous: it suffices to provide the construct for variables, perhaps themselves enclosed by $[\![\cdot]\!]$. The reason is that the above laws give a way to compute the exchange for any term which is not a parametricity witness (The result is given in Def. 4). When we want to be explicit about exchange being the syntactic construct, we write simply

$\mathbf{x}\,\dagger^{\pi}$. The syntax fragment for parametricity and exchanges is as follows.

$$\begin{array}{llcll}
\mathsf{Var} & \ni x, y, z & & & \\
\mathsf{Param} & \ni \mathbf{x} & ::= & x & \text{variable} \\
 & & \mid & \llbracket \mathbf{x} \rrbracket & \text{parametric witness} \\
\mathsf{Term} & \ni a, \ldots, u & ::= & \mathbf{x}\,\dagger^{\pi} & \text{permutation of dimensions} \\
 & & \mid & \ldots &
\end{array}$$

I. Dimension checks

If a permutation acts on dimensions 0 to $n-1$, every cube it is applied to must exhibit at least n dimensions. So far we have not discussed this restriction, which is the final feature of the system to present. We choose to annotate sorts with the dimension of the type which inhabits it. The sort s at dimension n is written s^n. Hence, we can capture the restriction in the exchange rule.

$$\frac{\Gamma \vdash A : B \qquad \Gamma \vdash A : s^n}{\Gamma \vdash A\,\ddagger^{\pi} : B\,\ddagger^{\pi}}\ \dim(\pi) \leq n$$

If a type inhabits a sort of dimension n, all the quantifications found inside the type must at least be over cubes of dimension n. This is realized in the product rule as follows:

$$\frac{\Gamma \vdash \bar{A} : s_1^n \qquad \Gamma, \bar{x} : \bar{A} \vdash B : s_2^m}{\Gamma \vdash (\forall \bar{x} : \bar{A}.\, B) : s_3^{m \sqcap n}}$$
$$\textsc{Product}\ (s_1, s_2, s_3) \in \mathcal{R}$$

Similarly, relations found in the type must be over cubes of dimension n.

J. Our calculus

The full definition of system $\mathcal{P}$, parameterized on a PTS specification $(\mathcal{S}, \mathcal{A}, \mathcal{R})$, is given in figures 1 and 2. For our system, the proof of parametricity (Prop. 1) goes through even for the case of the parametricity rule itself. The proof follows the structure of the abstraction theorem, with the difference that the START rule is translated to PARAM when an explicit witness is not available, and PARAM is translated to PARAM + EXCHANGE. (Full proofs are found in the appendix.) We have proved confluence, subject reduction, and strong normalization for our calculus. Since parametricity acts as a typing rule for $\llbracket\cdot\rrbracket$, subject reduction for our calculus stems directly from it. Strong normalization is proved by modeling the system in the PTS without parametricity. This model is done by introducing explicit witnesses of parametricity for all variables. Details of the proofs of these theorems are delayed until the appendix.

The syntactic changes made to the system require theorems to be adapted accordingly. In the case of Example 1, (proving that any function of type $\forall a : *.\, a \to a$ is an identity), the definition of Equality must be amended to make it inhabit $*^1$. This mostly involves augmenting the dimension of cubes by adding unit types as indices:

$$Eq : \forall a : *.\, a \to \begin{pmatrix} a \\ \cdot \end{pmatrix} \overset{\bullet}{\to} *^1$$
$$Eq = \lambda a : *.\, \lambda x : A.\, \lambda^{\bullet} \begin{pmatrix} y \\ \cdot \end{pmatrix} : \begin{pmatrix} a \\ \cdot \end{pmatrix} .\, \forall \begin{pmatrix} - \\ P \end{pmatrix} : \begin{pmatrix} \top \\ \begin{pmatrix} a \\ \cdot \end{pmatrix} \overset{\bullet}{\to} *^1 \end{pmatrix} .$$
$$\begin{pmatrix} \top \\ P \bullet \begin{pmatrix} x \\ \cdot \end{pmatrix} \end{pmatrix} \to P \bullet \begin{pmatrix} y \\ \cdot \end{pmatrix}$$

Definition 3 (Relational interpretation)**.**

$$\begin{aligned}
\llbracket \llbracket x \rrbracket^n \rrbracket_\xi &= \llbracket x \rrbracket^{n+1} && \textit{if } x \notin \xi \\
\llbracket \llbracket x_i \rrbracket^n \rrbracket_\xi &= \llbracket x_{1i} \rrbracket^n \dagger^{(0..n)} && \textit{if } x \in \xi \\
\llbracket \mathbf{x}\,\dagger^{\pi} \rrbracket_\xi &= \llbracket \mathbf{x} \rrbracket_\xi \dagger^{\pi+1} && \\
\llbracket \lambda \bar{x} : \bar{A}.\, B \rrbracket_\xi &= \lambda \bar{x} : \llbracket \bar{A} \rrbracket_\xi .\, \llbracket B \rrbracket_{\xi,x} && \\
\llbracket \lambda^{\bullet} \check{x} : \check{A}.\, B \rrbracket_\xi &= \lambda^{\bullet} \check{x} : (\llbracket \check{A} \rrbracket_\xi \oplus (\lambda^{\bullet} \check{x} : \check{A}.\, B)).\, x_{01\ldots1} \in \llbracket B \rrbracket_{\xi,x} && \\
\llbracket A\, \bar{B} \rrbracket_\xi &= \llbracket A \rrbracket_\xi \llbracket \bar{B} \rrbracket_\xi && \\
\llbracket T \rrbracket_\xi &= \lambda \check{z} : \begin{pmatrix} T \\ \cdot \end{pmatrix} .\, z_0 \in \llbracket T \rrbracket_\xi && \textit{if } T \textit{ is } \forall, \bullet \textit{ or } s^n
\end{aligned}$$

$$\begin{aligned}
C \in \llbracket s^n \rrbracket_\xi &= \begin{pmatrix} C \\ \cdot \end{pmatrix} \overset{\bullet}{\to} s^{n+1} && \\
C \in \llbracket \forall \bar{x} : \bar{A}.\, B \rrbracket_\xi &= \forall \bar{x} : \llbracket \bar{A} \rrbracket_\xi .\, (C\,(\bar{x}/01\ldots1)) \in \llbracket B \rrbracket_{\xi,x} && \\
C \in \llbracket \check{A} \overset{\bullet}{\to} s^n \rrbracket_\xi &= (\llbracket \check{A} \rrbracket_\xi \oplus C) \overset{\bullet}{\to} s^{n+1} && \\
C \in \llbracket A \bullet \check{B} \rrbracket_\xi &= \llbracket A \rrbracket_\xi \bullet (\llbracket \check{B} \rrbracket_\xi \oplus C) && \\
C \in \llbracket T \rrbracket_\xi &= \llbracket T \rrbracket_\xi \bullet \begin{pmatrix} C \\ \cdot \end{pmatrix} && \textit{if } T \textit{ is not } \forall, \bullet \textit{ nor } s^n
\end{aligned}$$

$$\begin{aligned}
\llbracket - \rrbracket_\xi &= - && \\
\llbracket \Gamma, x : A \rrbracket_{\xi,x} &= \llbracket \Gamma \rrbracket_\xi, x_0 : A, x_1 : x_0 \in \llbracket A \rrbracket_\xi && \textit{if } x \in \xi \\
\llbracket \Gamma, x : A \rrbracket_\xi &= \llbracket \Gamma \rrbracket_\xi, x : A && \textit{if } x \notin \xi
\end{aligned}$$

$$\llbracket \bar{A} \rrbracket_\xi = \left[\begin{array}{l} 0i \mapsto \{A_i\}_\xi \\ 1i \mapsto \llbracket A_i \rrbracket_\xi \end{array} \right]_{i \in \mathbf{2}^{\dim \bar{A}}}^{\dim \bar{A}+1}$$

$$(\llbracket \check{A} \rrbracket_\xi \oplus B) = \left[\begin{array}{l} 0i \mapsto \{A_i\}_\xi \\ 1i \mapsto \llbracket A_i \rrbracket_\xi \\ 01\ldots1 \mapsto B \end{array} \right]_{i \in \mathbf{2}^{\dim \check{A}} - 1}^{\dim \check{A}+1}$$

Definition 4 (Term exchange)**.**

$$\begin{aligned}
\llbracket x \rrbracket^n \dagger^{\rho} \ddagger^{\pi}_{\xi} &= \llbracket x \rrbracket^n \dagger^{\rho} && \textit{if } x \in \xi \\
\llbracket x \rrbracket^n \dagger^{\rho} \ddagger^{\pi}_{\xi} &= \llbracket x \rrbracket^n \dagger^{\mathrm{normal}_n(\pi \circ \rho)} && \textit{if } x \notin \xi \\
(A\, \bar{B}) \ddagger^{\pi}_{\xi} &= A \ddagger^{\pi}_{\xi}\ \bar{B} \ddagger^{\pi}_{\xi} && \\
(\lambda \bar{x} : \bar{A}.\, B) \ddagger^{\pi}_{\xi} &= \lambda \bar{x} : \bar{A} \ddagger^{\pi}_{\xi} .\, B[\bar{x} \mapsto \bar{x} \ddagger^{\pi}] \ddagger^{\pi}_{\xi,x} && \\
(\forall \bar{x} : \bar{A}.\, B) \ddagger^{\pi}_{\xi} &= \forall \bar{x} : \bar{A} \ddagger^{\pi}_{\xi} .\, B[\bar{x} \mapsto \bar{x} \ddagger^{\pi}] \ddagger^{\pi}_{\xi,x} && \\
(A \bullet \check{B}) \ddagger^{\pi}_{\xi} &= A \ddagger^{\pi}_{\xi} \bullet \bar{B} \ddagger^{\pi}_{\xi} && \\
(\lambda^{\bullet} \check{x} : \check{A}.\, B) \ddagger^{\pi}_{\xi} &= \lambda^{\bullet} \check{x} : \check{A} \ddagger^{\pi}_{\xi} .\, B[\bar{x} \mapsto \bar{x} \ddagger^{\pi}] \ddagger^{\pi}_{\xi,x} && \\
(\check{A} \overset{\bullet}{\to} s^n) \ddagger^{\pi}_{\xi} &= \check{A} \ddagger^{\pi}_{\xi} \overset{\bullet}{\to} s^n && \\
s^n \ddagger^{\pi}_{\xi} &= s^n &&
\end{aligned}$$

where $\mathrm{normal}_n(\pi)$ *removes all cycles of* π *entirely contained in* $0..n-1$.

Fig. 1. Relational interpretation of terms

Definition 5 (Syntax)**.**

Sort	$\ni s, s_1, s_2, s_3$	$::=$	$\mathcal{S}$	
Var	$\ni x, y, z$			
Param	$\ni \mathbf{x}$	$::=$	x	*variable*
		$\mid$	$[\![\mathbf{x}]\!]$	*parametric witness*
Term	$\ni a, \ldots, u$	$::=$	$\mathbf{x}\,\dagger^{\pi}$	*permutation of dimensions*
	$A, \ldots, U$	$\mid$	s^n	*sort at dimension n*
		$\mid$	$A\,\bar{B}$	*application (of hypercubes)*
		$\mid$	$\lambda\bar{x} : \bar{A}.\, B$	*abstraction (of hypercubes)*
		$\mid$	$\forall\bar{x} : \bar{A}.\, B$	*function space*
		$\mid$	$A\bullet\check{B}$	*relation membership*
		$\mid$	$\lambda^{\bullet}\check{x} : \check{A}.\, B$	*relation formation*
		$\mid$	$\check{A} \overset{\bullet}{\to} s^n$	*relation space*
Cube	$\ni \bar{A}$	$::=$	$\left[\ i \mapsto A_i\ \right]^{n}_{i \in \mathbf{2}^n}$	*cube of size 2^n*
Cube′	$\ni \check{A}$	$::=$	$\left[\ i \mapsto A_i\ \right]^{n}_{i \in \mathbf{2}^n - 1}$	*cube of size $2^n - 1$*
Context	$\ni \Gamma, \Delta$	$::=$	$-$	*empty context*
		$\mid$	$\Gamma, x : A$	*context extension*

Definition 6 (Typing rules)**.**

$$\frac{}{\vdash s_1^n : s_2^n}\ (s_1, s_2) \in \mathcal{A} \quad \textsc{Axiom} \qquad \frac{\Gamma \vdash A : B \qquad \Gamma \vdash C : s^n}{\Gamma, x : C \vdash A : B} \quad \textsc{Weakening} \qquad \frac{\Gamma \vdash F : (\check{A} \overset{\bullet}{\to} s^n) \qquad \Gamma \vdash \check{a} : \check{A}}{\Gamma \vdash F\bullet\check{a} : s^n} \quad \textsc{Rel-elim}$$

$$\frac{\Gamma, \check{x} : \check{A} \vdash B : s^n \qquad \Gamma \vdash \check{A} : s^n}{\Gamma \vdash (\lambda^{\bullet}\check{x} : \check{A}.\, B) : (\check{A} \overset{\bullet}{\to} s^n)} \quad \textsc{Rel-intro} \qquad \frac{\Gamma \vdash \check{A} : s_1^n}{\Gamma \vdash (\check{A} \overset{\bullet}{\to} s_1^n) : s_2^n} \quad \textsc{Rel-form}\ (s_1, s_2) \in \mathcal{A} \qquad \frac{\Gamma \vdash F : (\forall\bar{x} : \bar{A}.\, B) \qquad \Gamma \vdash \bar{a} : \bar{A}}{\Gamma \vdash F\,\bar{a} : B[\bar{x} \mapsto \bar{a}]} \quad \textsc{Application}$$

$$\frac{\Gamma, \bar{x} : \bar{A} \vdash b : B \qquad \Gamma \vdash (\forall\bar{x} : \bar{A}.\, B) : s^n}{\Gamma \vdash (\lambda\bar{x} : \bar{A}.\, b) : (\forall\bar{x} : \bar{A}.\, B)} \quad \textsc{Abstraction} \qquad \frac{\Gamma \vdash \bar{A} : s_1^n \qquad \Gamma, \bar{x} : \bar{A} \vdash B : s_2^m}{\Gamma \vdash (\forall\bar{x} : \bar{A}.\, B) : s_3^{m \sqcap n}} \quad \textsc{Product}\ (s_1, s_2, s_3) \in \mathcal{R} \qquad \frac{\Gamma \vdash A : B \qquad \Gamma \vdash B' : s^n \qquad B =_\beta B'}{\Gamma \vdash A : B'} \quad \textsc{Conversion}$$

$$\frac{\Gamma \vdash A : s^n}{\Gamma, x : A \vdash x : A} \quad \textsc{Start} \qquad \frac{\Gamma \vdash \mathbf{x} : A}{\Gamma \vdash [\![\mathbf{x}]\!] : \mathbf{x} \in [\![A]\!]_{\varnothing}} \quad \textsc{Param} \qquad \frac{\Gamma \vdash \mathbf{x} : A \qquad \Gamma \vdash A : s^n}{\Gamma \vdash \mathbf{x}\,\dagger^{\pi} : A\,\ddagger^{\pi}}\ \dim(\pi) \leq n \quad \textsc{Exchange}$$

- $\mathbf{2}^n$ *stands for all bit-vectors of size n; and* $\mathbf{2}^n - 1$ *stands for all bit-vectors of size n, except* 1...1.
- $\mathrm{ind}(\bar{A})$ *stands for* $\mathbf{2}^{\dim \bar{A}}$*; and* $\mathrm{ind}(\check{A})$ *stands for* $\mathbf{2}^{\dim \check{A}} - 1$.
- $\bar{x} : \bar{A}$ *stands for the bindings* $x_i : A_i\bullet(\bar{x}/\!\!/i)$ *where* $i \in \mathrm{ind}(\bar{A})$*; and* $\check{x} : \check{A}$ *stands for the bindings* $x_i : A_i\bullet(\check{x}/\!\!/i)$ *where* $i \in \mathrm{ind}(\check{A})$.
- *The typing judgement* $\Gamma \vdash \bar{a} : \bar{A}$ *stands for the conjunction of the judgements* $\Gamma \vdash a_i : A_i\bullet(\bar{a}/\!\!/i)$*, where* $i \in \mathrm{ind}(\bar{A})$*;*
  $\Gamma \vdash \check{a} : \check{A}$ *stands for the conjunction of the judgements* $\Gamma \vdash a_i : A_i\bullet(\check{a}/\!\!/i)$*, where* $i \in \mathrm{ind}(\check{A})$
- *Similarly,* $\bar{A} : s^n$ *stands for* $A_i : \bar{A}/\!\!/i \overset{\bullet}{\to} s^{\|i\|}$ *and* $\check{A} : s^n$ *stands for* $A_i : \check{A}/\!\!/i \overset{\bullet}{\to} s^{\|i\|}$.
- *The substitution* $\bar{x} \mapsto \bar{a}$ *stands for the parallel substitution* $x_i \mapsto a_i$ *for* $i \in \mathrm{ind}(\bar{a})$.

Fig. 2. Syntax and typing rules of $\mathcal{P}$, parameterized on a PTS specification $(\mathcal{S}, \mathcal{A}, \mathcal{R})$.

The proof term is almost itself needs fewer amendments:

$$\begin{aligned} \mathit{identities} &: \forall f : (\forall a : *.\, a \to a). \\ &\qquad \forall a : *.\, \forall x : a.\, Eq\, a\, (f\, a\, x)\bullet \begin{pmatrix} x \\ . \end{pmatrix} \\ \mathit{identities} &= \lambda f.\lambda a.\lambda x.[\![f]\!] \begin{pmatrix} a \\ Eq\, a\, x \end{pmatrix} \begin{pmatrix} x \\ \mathit{refl}\, a\, x \end{pmatrix} \end{aligned}$$

$$[\![f]\!] : \forall \begin{pmatrix} a_0 \\ a_1 \end{pmatrix} : \begin{pmatrix} * \\ a_0 \overset{\bullet}{\to} *^1 \end{pmatrix} . \forall \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} : \begin{pmatrix} a_0 \\ a_1 \end{pmatrix} . a_1\bullet(f\, a_0\, x_0)$$

III. Discussion

A. Alternative design: named dimensions

One may wonder if the introduction of the permutation operation on cubes is strictly necessary. In fact, it is a technical consequence of the convention that $[\![\cdot]\!]$ always adds dimension 0. Therefore, if we were to *name* dimensions, the need for permuting dimensions would disappear. However, we would instead have to deal with the renaming of dimensions, which is essentially as complex as permutations.

The situation is analogous to the issue of representation of variables in lambda-calculi. One can either use explicit names or De Brujin indices (and we have chosen indices here). Where abstraction introduces new variables, parametricity introduces new dimensions. and we have chosen indices here. A difference between variables and dimensions is that, whereas variable names are eliminated by application, case there is no obvious operation that eliminates dimensions.

B. *Extension: n-ary parametricity*

Here we have presented only unary parametricity, whereas most of the literature deals with binary parametricity. The generalization to arbitrary arity can be done in a straightforward way by mere syntactic duplication of relation indices, as detailed by Bernardy et al. [8] (Instead of x being interpreted as and index x_0 and a witness of parametricity x_1, we'd have many indices $x_0 ... x_{n-1}$ and a witness x_n that they are related by the parametric interpretation of the type.)

However, we have shown that by n iterations of unary parametricity, one gets n-ary parametricity between the faces of an n-cubes. If one were to *erase* the indices of these n faces (which are themselves cubes of dimension $n-1$), one would recover n-ary parametricity. This insight is developed by Bernardy and Lasson [6], with the difference that, in that work, sorts are used to determine which parts of terms must be erased. In future work we want to investigate the addition of an erasure operation based on the shape of cubes, which would make explicit the relation between parametricity at different arities within the calculus.

C. *Extension: inductive types*

Even though we considered only PTSs, it is straightforward to support the addition of inductive constructions. This can be done in the same way as Bernardy et al. [8]. Namely, the interpretation of an inductive definition is another inductive definition obtained by applying relational interpretation to every component of the original definition (type and constructors).

Example 5 (Inductive naturals and their interpretation)**.**

$$\begin{array}{l}\textbf{data}\ \mathsf{N} : \mathsf{Type}\ \textbf{where}\\ \quad \mathsf{zer} : \mathsf{N}\\ \quad \mathsf{suc} : \mathsf{N} \to \mathsf{N}\end{array}$$

$$\begin{array}{l}\textbf{data}\ [\![\mathsf{N}]\!] : \begin{pmatrix}\mathsf{N}\\ \cdot\end{pmatrix} \overset{\bullet}{\to} \mathsf{Type}\ \textbf{where}\\ \quad [\![\mathsf{zer}]\!] : [\![\mathsf{N}]\!] \bullet \begin{pmatrix}\mathsf{zer}\\ \cdot\end{pmatrix}\\ \quad [\![\mathsf{suc}]\!] : \begin{pmatrix}n\\ \dot{n}\end{pmatrix} : \begin{pmatrix}\mathsf{N}\\ [\![\mathsf{N}]\!]\end{pmatrix} \to [\![\mathsf{N}]\!] \bullet \begin{pmatrix}\mathsf{suc}\, n\\ \cdot\end{pmatrix}\end{array}$$

In our case we need also to verify that repetition of the interpretation yields symmetric relations, which is straightforward.

Usually, inductive definitions may inhabit any sort. In our case, if one wishes to introduce a definition at a dimension greater than zero, it must respect symmetry. That is, an inductive definition $I : s^n$ is legal only if, for any permutation π of the dimensions 0 to $n-1$, $I \ddagger^{\pi} = I$.

D. *Implementation*

An implementation of the calculus, realized in the Haskell programming language, is available for experimentation (http://hackage.haskell.org/package/uAgda).

E. *Related Work*

The relationship between logic and programming languages goes both ways. On the one hand, logical systems can be given meaning by assigning a computational interpretation to them. On the other hand, one would like to prove programs correct within formal logical systems. This means that one needs logical systems with enough power to support reasoning about programs. Seminal work featuring both sides of the connection includes [19] and [17].

The work described in this paper falls right into this tradition: we not only attempt to improve the support for reasoning about parametric reasoning in a logical framework, but also give parametricity a computational meaning. Furthermore, our parametricity construct is itself parametric. To our knowledge this has not been achieved before. Some logical frameworks with computational meaning have features which are related to parametricity, without subsuming it. Some logical systems have featured parametricity, but not in a computationally meaningful way.

a) Inductive families: In the first category, we must first mention inductive families, studied in different contexts by Pfenning and Paulin-Mohring [22], and Dybjer [11].

The computational realization of the induction principle over an inductive definition is a recursive function over the data, but which also carries information about the data being recursed over.

One can draw a parallel with our interpretation of terms: computational constructs such as application and abstraction are interpreted as application and abstraction, but with additional information about the original term carried as an index. In fact, by taking advantage of extensionality, it is possible to see induction principles over data as a special case of parametricity, as shown for example by Wadler [32].

b) Extensionality: The ability to reason about functions can be supported not just by parametricity, but also by the principle of extensionality, which says that two functions are equal if they map equal inputs to equal outputs.

Altenkirch et al. [2] show how to integrate extensionality in a computationally meaningful way. Parallels can be drawn between the work of Altenkirch et al. and ours. Mainly, their equality relation depends on the structure of the types that it relates, in the same way as the interpretation of types we propose.

c) Higher-dimensional type-theories: Recent work on the interpretation of the equality-type in intensional type theory suggests that it should be modeled using higher-dimensional structures [15]. In this work we have found the need to introduce higher-dimensional objects as well, but to interpret parametricity instead of equality. Furthermore it has been known since [25] that for System F, the parametric interpretation of closed types coincides with equality. We hope that this work will help extending the connection to intensional type theory.

d) Meta-level reasoning: Miller and Tiu [18] propose a logical framework where one can reason precisely about terms and types of an object language. Their domain of application overlaps with ours. For example, one can prove in their framework that the object type $\forall a : \star.a$ is uninhabited. A characteristic of [18] is the total separation between object and host language. Here, we are able to unify both languages.

e) Parametricity: Plotkin and Abadi [23] have formulated a logic extended with axioms of parametricity. However, they are content with the consistency of parametricity, and do not give any computational interpretation for it.

As mentioned previously, it has been shown before *e.g.*, by Hasegawa [13], that parametricity is consistent with some models of System F. Consistency of parametricity with dependently-typed theories does not appear to have been proved previously.

In unpublished work, Takeuti [27] attempted to extend CC with parametricity, but Takeuti does not attempt to assign a computational content to his parametricity axioms, and only conjectures consistency of his system.

Bernardy et al. [8] have described logical relations from PTSs to PTSs, but in an *external* fashion: the abstraction theorem itself remains outside the system, as we have exposed in detail in Sec. II.

F. Future Work

A natural application of this work would be to integrate it in a real proof assistant (*e.g.*, COQ or AGDA). One could then experiment and analyze how much power parametricity gives on practical examples.

We also plan to analyze the relationship parametricity with other extensions of type-theory, such as for example (co)inductive constructions, extensionality, etc. Because parametricity is a powerful reasoning principle, it might be possible to understand other aspects of type-theory in terms of it. In particular, understanding data in terms of their Church-Encodings is a tempting application.

Acknowledgments We are grateful to Thierry Coquand, Peter Dybjer and Simon Huber for helpful comments and discussions.

REFERENCES

[1] M. Abadi, L. Cardelli, and P. Curien. Formal parametric polymorphism. In *Proc. of POPL'93*, pages 157–170. ACM, 1993.

[2] T. Altenkirch, C. McBride, and W. Swierstra. Observational equality, now! In *PLPV 2007*, pages 57–68. ACM, 2007.

[3] R. Atkey, S. Lindley, and J. Yallop. Unembedding domain-specific languages. In *Proc. of Haskell '09*, pages 37–48. ACM, 2009.

[4] H. P. Barendregt. Lambda calculi with types. *Handbook of logic in computer science*, 2:117–309, 1992.

[5] J.-P. Bernardy. *A theory of parametric polymorphism and an application*. Phd thesis, Chalmers Tekniska Högskola, 2011.

[6] J.-P. Bernardy and M. Lasson. Realizability and parametricity in pure type systems. In M. Hofmann, editor, *FoSSaCS*, volume 6604 of *LNCS*, pages 108–122. Springer, 2011.

[7] J.-P. Bernardy, P. Jansson, and K. Claessen. Testing polymorphic properties. In A. Gordon, editor, *European Symposium on Programming*, volume 6012 of *LNCS*, pages 125–144. Springer, 2010.

[8] J.-P. Bernardy, P. Jansson, and R. Paterson. Parametricity and dependent types. In *Proc. of ICFP 2010*, pages 345–356. ACM, 2010.

[9] A. Chlipala. Parametric higher-order abstract syntax for mechanized semantics. In *Proceeding of ICFP 2008*, pages 143–156. ACM, 2008.

[10] T. Coquand and G. Huet. The calculus of constructions. Technical report, INRIA, 1986.

[11] P. Dybjer. Inductive families. *Formal Aspects of Comp.*, 6(4): 440–465, 1994.

[12] A. Gill, J. Launchbury, and S. Peyton Jones. A short cut to deforestation. In *Proc. of FPCA*, pages 223–232. ACM, 1993.

[13] R. Hasegawa. Categorical data types in parametric polymorphism. *Mathematical Structures in Comp. Sci.*, 4(01):71–109, 1994.

[14] P. Johann. A generalization of short-cut fusion and its correctness proof. *Higher-Order and Symbol. Comp.*, 15(4):273–300, 2002.

[15] D. Licata and R. Harper. Canonicity for 2-dimensional type theory. In *Proc. of POPL 2012*. ACM, 2012.

[16] H. Mairson. Outline of a proof theory of parametricity. In *Proc. of FPCA 1991*, volume 523 of *LNCS*, pages 313–327. Springer-Verlag, 1991.

[17] P. Martin-Löf. *Intuitionistic type theory*. Bibliopolis, 1984.

[18] D. A. Miller and A. F. Tiu. A proof theory for generic judgments: An extended abstract. In *LICS 2003*, pages 118–127. IEEE, 2003.

[19] R. Milner. Logic for Computable Functions: description of a machine implementation. *Artificial Intelligence*, 1972.

[20] U. Norell. *Towards a practical programming language based on dependent type theory*. PhD thesis, Chalmers Tekniska Högskola, 2007.

[21] C. Paulin-Mohring. Extracting Fω's programs from proofs in the calculus of constructions. In *POPL'89*, pages 89–104. ACM, 1989.

[22] F. Pfenning and C. Paulin-Mohring. Inductively defined types in the calculus of constructions. In *MFPS*, volume 442 of *LNCS*, pages 209–228. Springer, 1990.

[23] G. Plotkin and M. Abadi. A logic for parametric polymorphism. In *Proc. of TLCA*, volume 664 of *LNCS*, page 361–375. Springer, 1993.

[24] N. Pouillard. Nameless, painless. In *Proc. of the 16th ACM SIGPLAN international conference on Funct. programming*, ICFP '11. ACM, 2011. to appear.

[25] J. C. Reynolds. Types, abstraction and parametric polymorphism. *Information processing*, 83(1):513–523, 1983.

[26] J. Svenningsson. *Scalable Program Analysis*. Phd thesis, Chalmers Tekniska Högskola, 2007.

[27] I. Takeuti. The theory of parametricity in lambda cube. Manuscript, 2004.

[28] The Coq development team. The Coq proof assistant, 2011.

[29] J. Voigtländer. Bidirectionalization for free! (Pearl). In *Proc. of POPL 2009*, pages 165–176. ACM, 2009.

[30] D. Vytiniotis and S. Weirich. Parametricity, type equality, and higher-order polymorphism. *J. Funct. Program.*, 20(02):175–210, 2010.

[31] P. Wadler. Theorems for free! In *Proc. of FPCA 1989*, pages 347–359. ACM, 1989.

[32] P. Wadler. The Girard–Reynolds isomorphism. *Theor. Comp. Sci.*, 375(1–3):201–226, 2007.

IV. Appendix

This section contains details of our proofs.

A. *Confluence*

To begin, we generalize some basic properties possessed by well-behaved λ-calculi. In particular, we prove the substitution lemma and the confluence property. We limit the detail of the proofs to the cases relevant to the specifics of our calculus.

Remark 1. *It follows immediately by induction on the structure of terms that substitution and reduction preserve sorts. This result will be silently used in the present section.*

Lemma 1. *If x does not occur free in A, then $\llbracket A\rrbracket_{\xi,x} = \llbracket A\rrbracket_{\xi}$ and $a \in \llbracket A\rrbracket_{\xi,x} = a \in \llbracket A\rrbracket_{\xi}$ for all a.*

Proof: Induction on the structure of A. ■

Lemma 2 ($\llbracket\cdot\rrbracket$ and substitution, part 1)**.** *If z is not in ξ, then*

$$\llbracket A[z_i \mapsto E]\rrbracket_{\xi} = \llbracket A\rrbracket_{\xi,z}[z_{0i} \mapsto E][z_{1i} \mapsto \llbracket E\rrbracket_{\xi}]$$
$$a[z_{0i} \mapsto E] \in \llbracket A[z_i \mapsto E]\rrbracket_{\xi} = (a \in \llbracket A\rrbracket_{\xi,z})[z_{0i} \mapsto E][z_{1i} \mapsto \llbracket E\rrbracket_{\xi}]$$

Proof: By induction on A.

Sort Trivial.

Abstraction (Product is similar)

$$\begin{aligned} &\llbracket(\lambda\bar{x} : \bar{A}.\, b)[z_i \mapsto E]\rrbracket_{\xi} \\ &= \llbracket\lambda\bar{x} : \bar{A}[z_i \mapsto E].\, b[z_i \mapsto E]\rrbracket_{\xi} \\ &= \lambda\bar{x} : \llbracket\bar{A}[z_i \mapsto E]\rrbracket_{\xi}.\, \llbracket b[z_i \mapsto E]\rrbracket_{\xi,x} \\ &= \lambda\bar{x} : \llbracket\bar{A}\rrbracket_{\xi,z}[z_{0i} \mapsto E][z_{1i} \mapsto \llbracket E\rrbracket_{\xi}]. \end{aligned}$$

$$\begin{aligned} &\llbracket b\rrbracket_{\xi,x,z}[z_{0i} \mapsto E][z_{1i} \mapsto \llbracket E\rrbracket_{\xi,x}] \\ &= (\lambda\bar{x} : \llbracket\bar{A}\rrbracket_{\xi,z}.\, \llbracket b\rrbracket_{\xi,x,z})[z_{0i} \mapsto E][z_{1i} \mapsto \llbracket E\rrbracket_{\xi,x}] \\ &= \llbracket\lambda\bar{x} : \bar{A}.\, b\rrbracket_{\xi,z}[z_{0i} \mapsto E][z_{1i} \mapsto \llbracket E\rrbracket_{\xi,x}] \end{aligned}$$

Rel-Intro (Rel-Form is similar)

$$\begin{aligned} &\llbracket(\lambda^{\bullet}\check{x} : \check{A}.\, B)[z_i \mapsto E]\rrbracket_{\xi} \\ &= \llbracket\lambda^{\bullet}\check{x} : \check{A}[z_i \mapsto E].\, B[z_i \mapsto E]\rrbracket_{\xi} \\ &= \lambda^{\bullet}\check{x} : (\llbracket\check{A}[z_i \mapsto E]\rrbracket_{\xi} \oplus \ldots).\, x_{01\ldots1} \in \llbracket B[z_i \mapsto E]\rrbracket_{\xi,x} \\ &= \lambda^{\bullet}\check{x} : (\llbracket\check{A}\rrbracket_{\xi} \oplus (\lambda^{\bullet}\check{x} : \check{A}.\, B))[z_{0i} \mapsto E][z_{1i} \mapsto \llbracket E\rrbracket_{\xi}]. \end{aligned}$$

$$\begin{aligned} &(x_{01\ldots1} \in \llbracket B\rrbracket_{\xi,x,z})[z_{0i} \mapsto E][z_{1i} \mapsto \llbracket E\rrbracket_{\xi}] \\ &= \llbracket\lambda^{\bullet}\check{x} : \check{A}.\, B\rrbracket_{\xi,z}[z_{0i} \mapsto E][z_{1i} \mapsto \llbracket E\rrbracket_{\xi}] \end{aligned}$$

Application

$$\begin{aligned} &\llbracket(F\,\bar{a})[z_i \mapsto E]\rrbracket_{\xi} \\ &= \llbracket F[z_i \mapsto E]\,\bar{a}[z_i \mapsto E]\rrbracket_{\xi} \\ &= \llbracket F[z_i \mapsto E]\rrbracket_{\xi}\, \llbracket\bar{a}[z_i \mapsto E]\rrbracket_{\xi} \\ &= F[z_i \mapsto E][z_{0i} \mapsto E][z_{1i} \mapsto \llbracket E\rrbracket_{\xi}] \end{aligned}$$

$$\begin{aligned} &\bar{a}[z_i \mapsto E][z_{0i} \mapsto E][z_{1i} \mapsto \llbracket E\rrbracket_{\xi}] \\ &= F\,\bar{a}[z_{0i} \mapsto E][z_{1i} \mapsto \llbracket E\rrbracket_{\xi}] \end{aligned}$$

Rel-Elim

$$\begin{aligned} &C[z_{0i} \mapsto E] \in \llbracket(A\bullet\check{B})[z_i \mapsto E]\rrbracket_{\xi} \\ &= \llbracket A[z_i \mapsto E]\rrbracket_{\xi}\bullet(\llbracket B[z_i \mapsto E]\rrbracket_{\xi} \oplus (C[z_{0i} \mapsto E])) \\ &= (\llbracket A\rrbracket_{\xi,z}\bullet(\llbracket\check{B}\rrbracket_{\xi,z} \oplus C))[z_{0i} \mapsto E][z_{1i} \mapsto \llbracket E\rrbracket_{\xi}] \\ &= (C \in \llbracket A\bullet\check{B}\rrbracket_{\xi,z})[z_{0i} \mapsto E][z_{1i} \mapsto \llbracket E\rrbracket_{\xi}] \end{aligned}$$

Variable

It is trivial if $x_i \neq z_i$. Now if $x_i = z_i$,

$$\begin{aligned} &\llbracket \llceil z_i\rrceil^{n}\, \dagger^{\pi}[z_i \mapsto E]\rrbracket_{\xi} \\ &= \llbracket\llbracket E\rrbracket^{n}\, \ddagger^{\pi}\rrbracket_{\xi} \\ &= \llbracket\llbracket E\rrbracket_{\xi}\rrbracket^{n}\, \ddagger^{(0\ldots n)}\, \ddagger^{\pi+1} \\ &= \llceil z_{1i}\rrceil^{n}\, \ddagger^{(0\ldots n)}\, \ddagger^{\pi+1}[z_{0i} \mapsto E][z_{1i} \mapsto \llbracket E\rrbracket_{\xi}] \\ &= \llbracket\llceil z_i\rrceil^{n}\rrbracket_{\xi,z}\, \ddagger^{\pi+1}[z_{0i} \mapsto E][z_{1i} \mapsto \llbracket E\rrbracket_{\xi}] \\ &= \llbracket\llceil z_i\rrceil^{n}\, \dagger^{\pi}\rrbracket_{\xi,z}[z_{0i} \mapsto E][z_{1i} \mapsto \llbracket E\rrbracket_{\xi}] \end{aligned}$$

■

Lemma 3 ($\llbracket\cdot\rrbracket$ and substitution, part 2)**.** *If ξ does not contain either z or any of the free variables of E, then*

$$\llbracket A[z_i \mapsto E]\rrbracket_{\xi} = \llbracket A\rrbracket_{\xi}[z_i \mapsto E],\ \textit{and}$$
$$a[z_i \mapsto E] \in \llbracket A[z_i \mapsto E]\rrbracket_{\xi} = (a \in \llbracket A\rrbracket_{\xi})[z_i \mapsto E]$$

Proof: By induction on A. The proof is similar to the one of Lem. 2, except for the variable case with $x_i = z_i$.

$$\begin{aligned} \llbracket\llceil z_i\rrceil^{n}\, \dagger^{\pi}[z_i \mapsto E]\rrbracket_{\xi} &= \llbracket\llbracket E\rrbracket^{n}\, \ddagger^{\pi}\rrbracket_{\xi} \\ &= \llbracket\llbracket E\rrbracket^{n}\, \ddagger^{\pi}\rrbracket_{\varnothing} \\ &= \llbracket E\rrbracket^{n+1}\, \ddagger^{\pi+1} \\ &= \llceil z_i\rrceil^{n+1}\, \dagger^{\pi+1}[z_i \mapsto E] \\ &= \llbracket\llceil z_i\rrceil^{n}\, \dagger^{\pi}\rrbracket_{\xi}[z_i \mapsto E] \end{aligned}$$

■

Lemma 4 (Symmetry)**.** *$\llbracket A\rrbracket^{n}$ is symmetric in its n first dimensions: $\llbracket A\rrbracket^{n}_{\xi}\, \ddagger^{\pi}_{\xi} = \llbracket A\rrbracket^{n}_{\xi}\, \ddagger^{\mathrm{normal}_n(\pi)}_{\xi}$*

Proof: By induction on the structure of A.

- If a is $\llceil x_i\rrceil^{m}\, \dagger^{\rho}$:
  - If $x \notin \xi$, $\llbracket A\rrbracket^{n}_{\xi}\, \ddagger^{\pi}_{\xi} = \llceil x_i\rrceil^{m+n}\, \dagger^{\rho+n}\, \ddagger^{\pi}_{\xi}$. Since $\rho + n$ and $(0\ldots n-1)$ are disjoint, we have $\mathrm{normal}_{m+n}((\rho+n)\circ\pi) = \mathrm{normal}_{m+n}((\rho+n)\circ \mathrm{normal}_n(\pi))$, and the result follows.
  - If $x \in \xi$, we have

$$\begin{aligned} &\llbracket A\rrbracket^{n}_{\xi}\, \ddagger^{\pi}_{\xi} \\ &= \llceil x_{1\ldots1i}\rrceil^{m}\, \ddagger^{(0\ldots m)+n}\, \ddagger^{\rho+n}\, \ddagger^{\pi}_{\xi} \\ &= \llceil x_{1\ldots1i}\rrceil^{m}\, \ddagger^{(0\ldots m)+n}\, \ddagger^{\rho+n}. \end{aligned}$$

- Otherwise: straightforward.

■

Lemma 5 ($\cdot\,\ddagger^{\cdot}$ and substitution)**.** *If ξ does not contain either z or any of the free variables of E, then*

$$A[z_i \mapsto E]\, \ddagger^{\pi}_{\xi} = A\, \ddagger^{\pi}_{\xi}[z_i \mapsto E]\ \textit{for all}\ \pi.$$

Proof: By induction on A. The only interested case is the one for variables, with $x_i = z_i$:

$$\begin{aligned}
\llbracket z_i \rrbracket^n \dagger^{\rho}[z_i \mapsto E] \ddagger^{\pi}_{\xi} &= \llbracket E \rrbracket^n \ddagger^{\rho} \ddagger^{\pi}_{\xi} \\
&= \llbracket E \rrbracket^n \ddagger^{\mathrm{normal}_n(\pi \circ \rho)} \qquad \text{by Lem. 4} \\
&= \llbracket z_i \rrbracket^n \ddagger^{\mathrm{normal}_n(\pi \circ \rho)}[z_i \mapsto E] \\
&= \llbracket z_i \rrbracket^n \dagger^{\rho} \ddagger^{\pi}_{\xi}[z_i \mapsto E]
\end{aligned}$$

■

Lemma 6 (Substitution)**.**

$$A[z \mapsto E][z' \mapsto E'] = A[z' \mapsto E'][z \mapsto E[z' \mapsto E']]$$

Proof: By induction on A; the only non-trivial case is for the parametric witnesses $\llbracket z \rrbracket^n$:

$$\begin{gathered}
\llbracket z \rrbracket^n \dagger^{\pi}[z \mapsto E][z' \mapsto E'] = \llbracket E \rrbracket^n_{\varnothing}[z' \mapsto E'] \ddagger^{\pi} \\
= \llbracket E[z' \mapsto E'] \rrbracket^n_{\varnothing} \ddagger^{\pi} = \llbracket z \rrbracket^n \dagger^{\pi}[z' \mapsto E'][z \mapsto E[z' \mapsto E']]
\end{gathered}$$

by lemmas 3 and 5. ■

We now check the Church-Rosser property holds, that is, we verify that the order in which the reductions are performed does not matter. To prove this property, we define a *parallel reduction* (Following the Tait/Martin-Löf technique), and show that the diamond property holds for this reduction.

Definition 7 (Parallel nested reduction)**.**

$$\textsc{Refl}\ \frac{}{A \triangleright A} \qquad \beta\ \frac{b \triangleright b' \qquad \bar{a} \triangleright \bar{a}'}{(\lambda \bar{x} : \bar{A}.\, b)\, \bar{a} \triangleright b'[\bar{x} \mapsto \bar{a}']}$$

$$\beta^{\bullet}\ \frac{b \triangleright b' \qquad \breve{a} \triangleright \breve{a}'}{(\lambda^{\bullet} \breve{x} : \breve{A}.\, b) {\bullet} \breve{a} \triangleright b'[\breve{x} \mapsto \breve{a}']}$$

$$\textsc{App-Cong}\ \frac{F \triangleright F' \qquad \bar{a} \triangleright \bar{a}'}{F\, \bar{a} \triangleright F'\, \bar{a}'}$$

$$\textsc{App}^{\bullet}\textsc{-Cong}\ \frac{F \triangleright F' \qquad \breve{a} \triangleright \breve{a}'}{F {\bullet} \breve{a} \triangleright F' {\bullet} \breve{a}'}$$

$$\textsc{Abs-Cong}\ \frac{\bar{A} \triangleright \bar{A}' \qquad b \triangleright b'}{\lambda \bar{x} : \bar{A}.\, b \triangleright \lambda \bar{x} : \bar{A}'.\, b'}$$

$$\textsc{Abs}^{\bullet}\textsc{-Cong}\ \frac{\breve{A} \triangleright \breve{A}' \qquad b \triangleright b'}{\lambda^{\bullet} \breve{x} : \breve{A}.\, b \triangleright \lambda^{\bullet} \breve{x} : \breve{A}'.\, b'}$$

$$\textsc{All-Cong}\ \frac{\bar{A} \triangleright \bar{A}' \qquad B \triangleright B'}{\forall \bar{x} : \bar{A}.\, B \triangleright \forall \bar{x} : \bar{A}'.\, B'}$$

$$\textsc{All}^{\bullet}\textsc{-Cong}\ \frac{\breve{A} \triangleright \breve{A}'}{\breve{A} \overset{\bullet}{\to} s^n \triangleright \breve{A}' \overset{\bullet}{\to} s^n}$$

with $\bar{A} \triangleright \bar{A}'$ iff. for all i, $A_i \triangleright A'_i$ (and similarly for $\breve{A} \triangleright \breve{A}'$).

Lemma 7 (Congruence of $\llbracket \cdot \rrbracket$)**.** *If $A \triangleright A'$, then for all ξ,*

- $\llbracket A \rrbracket_{\xi} \triangleright \llbracket A' \rrbracket_{\xi}$*, and*
- $a \in \llbracket A \rrbracket_{\xi} \triangleright a' \in \llbracket A' \rrbracket_{\xi}$ *for all $a \triangleright a'$*

Proof: By induction on $A \triangleright A'$:

- The case of Refl is trivial.
- For β, one expects

$$\llbracket (\lambda \bar{x} : \bar{A}.\, b)\, \bar{a} \rrbracket_{\xi} \triangleright \llbracket b'[\bar{x} \mapsto \bar{a}'] \rrbracket_{\xi},$$

knowing $b \triangleright b'$ and $\bar{a} \triangleright \bar{a}'$.

$$\begin{array}{cl}
 & \llbracket (\lambda \bar{x} : \bar{A}.\, b)\, \bar{a} \rrbracket_{\xi} \\
= & \{\text{by def. of } \llbracket \cdot \rrbracket_{\xi}\} \\
 & (\lambda \bar{x} : \llbracket \bar{A} \rrbracket_{\xi}.\, \llbracket b \rrbracket_{\xi,x})\, \llbracket \bar{a} \rrbracket_{\xi} \\
\triangleright & \{\text{by } \beta,\ \textsc{Refl} \text{ and IH}\} \\
 & \llbracket b' \rrbracket_{\xi,x}[\bar{x} \mapsto \llbracket \bar{a}' \rrbracket_{\xi}] \\
= & \{\text{by Lem. 2}\} \\
 & \llbracket b'[\bar{x} \mapsto \bar{a}'] \rrbracket_{\xi}
\end{array}$$

- The cases of $\beta^{\bullet}$ is similar.
- The cases of $\star$-Cong are straightforward using the definition of $\llbracket \cdot \rrbracket$.

■

Lemma 8 (Congruence of $\cdot \ddagger^{\cdot}_{\cdot}$)**.** *If $A \triangleright A'$, then for all ξ and π, one has $A \ddagger^{\pi}_{\xi} \triangleright A' \ddagger^{\pi}_{\xi}$*

Proof: By induction on $A \triangleright A'$. The only interesting cases are for the β and $\beta^{\bullet}$-reductions. For β ($\beta^{\bullet}$ is similar), we have

$$\begin{array}{cl}
 & ((\lambda \bar{x} : \bar{A}.\, b)\, \bar{a}) \ddagger^{\pi}_{\xi} \\
= & \{\text{by def. of } \cdot \ddagger^{\pi}_{\xi}\} \\
 & (\lambda \bar{x} : \bar{A} \ddagger^{\pi}_{\xi}.\, b[\bar{x} \mapsto \bar{x} \ddagger^{\pi}_{\xi}] \ddagger^{\pi}_{\xi,x})\, \bar{a} \ddagger^{\pi}_{\xi} \\
\triangleright & \{\text{by } \beta,\ \textsc{Refl} \text{ and IH}\} \\
 & b'[\bar{x} \mapsto \bar{x} \ddagger^{\pi}_{\xi}] \ddagger^{\pi}_{\xi,x}[\bar{x} \mapsto \bar{a}' \ddagger^{\pi}_{\xi}] \\
= & b' \ddagger^{\pi}_{\xi,x}[\bar{x} \ddagger^{\pi}_{\xi} \mapsto \bar{a}' \ddagger^{\pi}_{\xi}] \\
= & b'[\bar{x} \mapsto \bar{a}'] \ddagger^{\pi}_{\xi}
\end{array}$$

■

Lemma 9 (Congruence of substitution)**.** *If $A \triangleright A'$ and $E \triangleright E'$, then $A[z \mapsto E] \triangleright A'[z \mapsto E']$.*

Proof: By induction on $A \triangleright A'$:

- For Refl, the expected result follows from an induction on A (using n times Lem. 7 and Lem. 8 for the case $\llbracket z \rrbracket^n \ddagger^{\pi}$).
- For β, one expects

$$((\lambda \bar{x} : \bar{A}.\, b)\, \bar{a})[z \mapsto E] \triangleright b'[\bar{x} \mapsto \bar{a}'][z \mapsto E],$$

knowing $b \triangleright b'$ and $\bar{a} \triangleright \bar{a}'$. We have

$$\begin{array}{cl}
 & ((\lambda \bar{x} : \bar{A}.\, b)\, \bar{a})[z \mapsto E] \\
= & \{\text{by def. of the substitution}\} \\
 & (\lambda \bar{x} : A[z \mapsto E].\, b[z \mapsto E])\, \bar{a}[z \mapsto E] \\
\triangleright & \{\text{by } \beta \text{ and IH}\} \\
 & b'[z \mapsto E'][\bar{x} \mapsto \bar{a}'[z \mapsto E']] \\
= & \{\text{by Lem. 6}\} \\
 & b'[\bar{x} \mapsto \bar{a}'][z \mapsto E']
\end{array}$$

- The cases of $\beta^{\bullet}$ is similar.
- The cases of $\star$-Cong stem from straightforward uses of induction hypotheses.

■

Theorem 2 (Diamond property). *The rewriting system ($\rhd$) has the diamond property. That is, for each A, B, B' such that $B \lhd A \rhd B'$, there exists C such that $B \rhd C \lhd B'$*

Proof: By inductions on the derivations:

- If one of the derivations ends with Refl, one has either $A = B$, or $A = B'$. We pick $C = B'$ in the former case and $C = B$ in the latter.
- If one of the derivations ends with Abs-Cong, All-Cong, Abs$^\bullet$-Cong or All$^\bullet$-Cong, the other one has to end with the same rule, and the result is a straightforward use of the induction hypothesis.
- If one of the derivations ends with App-Cong, the other one has to end with App-Cong, or with β. The first case is straightforward; in the second one, one has
$$(\lambda\bar{x} : \bar{A}'.\, b')\,\bar{a}' \;\lhd\; (\lambda\bar{x} : \bar{A}.\, b)\,\bar{a} \;\rhd\; b''[\bar{x} \mapsto \bar{a}'']$$
with $\lambda\bar{x} : \bar{A}'.\, b' \lhd \lambda\bar{x} : \bar{A}.\, b, \quad b \rhd b'' \quad$ and $\quad \bar{a}' \lhd \bar{a} \rhd \bar{a}''$
The situation is summarized in the diagram below. In more details, the end of the derivation of $\lambda\bar{x} : \bar{A}'.\, b' \lhd \lambda\bar{x} : \bar{A}.\, b$ has to be either Abs-Cong, or Refl. In the first case (the last one is similar), one has $\bar{A}' \lhd \bar{A}$ and $b' \lhd b$.
By induction hypothesis there exist b''', $\bar{a}'''$ such that $b' \rhd b''' \lhd b''$ and $\bar{a}' \rhd \bar{a}''' \lhd \bar{a}''$.
The result follows by β and Lem. 9:
$$(\lambda\bar{x} : \bar{A}'.\, b')\,\bar{a}' \;\rhd\; b'''[\bar{x} \mapsto \bar{a}'''] \;\lhd\; b''[\bar{x} \mapsto \bar{a}'']$$
$$\begin{tikzcd}
 & (\lambda\bar{x} : \bar{A}.\, b)\,\bar{a} \arrow[dl, "\textsc{Abs-Cong}", "\bar{a}' \lhd \bar{a},\, b' \lhd b"'] \arrow[dr, "\beta", "b \rhd b'',\, \bar{a} \rhd \bar{a}''"'] & \\
(\lambda\bar{x} : \bar{A}'.\, b')\,\bar{a}' \arrow[dr, dashed, "b' \rhd b''',\, \bar{a}' \rhd \bar{a}'''", "\beta"'] & & b''[\bar{x} \mapsto \bar{a}''] \arrow[dl, dashed, "b''' \lhd b'',\, \bar{a}''' \lhd \bar{a}''"', "\text{Lem. 9}"] \\
 & b'''[\bar{x} \mapsto \bar{a}'''] &
\end{tikzcd}$$
- The case for App$^\bullet$-Cong is similar.
- If both derivations end with the same β or $\beta^\bullet$ rule, the result is a straightforward use of the induction hypothesis and Lem. 9.

■

Corollary 1 (Church-Rosser property). *Our calculus system has the confluence (Church-Rosser) property that is, for each A, B, B' such that $B \longleftarrow^\star A \longrightarrow^\star B'$, there exists C such that $B \longrightarrow^\star C \longleftarrow^\star B'$*

Proof: Direct consequence of Th. 2, noticing $\rhd^\star = \longrightarrow^\star$. ■

B. Abstraction

In this section we check that our main goal, the integration of parametricity (See Prop. 1), is achieved by the design that we propose. At the same time, we check that the abstraction theorem also holds for our calculus. We do so by proving Lem. 10, which subsumes both theorems.

Theorem 3 (Abstraction).

1) $\Gamma \vdash A : B \quad \Rightarrow \quad [\![\Gamma]\!]_\xi \vdash [\![A]\!]_\xi : (A \in [\![B]\!]_\xi)$*, where ξ contains all the variables in Γ.*
2) Furthermore, if the original judgement makes no use of Param*, the resulting judgement does not either.*

Proof:

1) Direct consequence of Lem. 10.1.
2) In the proof of Lem. 10.1, if ξ is full, then the target derivation trees contains Param iff Param occurs in the derivation tree for $\Gamma \vdash A : B$.

■

Theorem 4 (Parametricity).
$$\Gamma \vdash A : B \;\Rightarrow\; \Gamma \vdash [\![A]\!] : (A \in [\![B]\!])$$

This theorem means that every term satisfies the parametricity condition of its type, even if it contains free variables.

Proof: Take ξ empty in Lem. 10.1. ■

Definition 8. *ξ conforms to Γ when ξ contains exactly a suffix of Γ.*

Remark 2. *If $[\![\cdot]\!]$ preserves conforming indices: if ξ conforms to Γ and A is well-typed in Γ, the definition of $[\![A]\!]_\xi$ makes only recursive calls with conforming substitutions.*

Sketch: By induction on the typing derivation. In the definition of $[\![\cdot]\!]$, of every bound variable in a term is put into the index in recursive calls. ■

Lemma 10 (Generalized abstraction). *Assuming that ξ conforms to Γ,*
1) $\Gamma \vdash A : B \;\Rightarrow\; [\![\Gamma]\!]_\xi \vdash [\![A]\!]_\xi : A \in [\![B]\!]_\xi$
2) $\Gamma \vdash A : B \;\Rightarrow\; [\![\Gamma]\!]_\xi \vdash A : B$
3) $\Gamma \vdash B : s^n \;\Rightarrow\; [\![\Gamma]\!]_\xi, x : B \vdash x \in [\![B]\!]_\xi : s^{n+1}$

Proof: The proof is done by simultaneous induction on the derivation tree, and is similar to the proofs of the abstraction theorem done by Bernardy and Lasson [6]. The new parts occur in the special handling of the Start and Param rules. The proof of each sub-lemma can be sketched as follows:

1) The cases of abstraction and application stem from the fact that their respective relational interpretation follows the same pattern as the relational interpretation of the product. The case of a variable x (Start) is more tricky: if $x \in \xi$, then the context contains an explicit witness of parametricity for x. This witness is used to justify the translated judgement. If $x \notin \xi$, then we can use the parametricity rule on x to translate the typing judgement. The Param rule is handled similarly, with the additional complexity that an exchange of dimensions must be added when $x \notin \xi$.
2) This sub-lemma is used to justify weakening of contexts in the other sub-lemmas. It is a consequence of the thinning

lemma and the fact that the interpretation of types in always well-typed (see the third item below).

3) This sub-lemma expresses that if T is a well-sorted type, then $x \in \llbracket T \rrbracket$ is also well-sorted. It is easy to convince oneself of that result by checking that the translation of a type always yields a relation, and that the translation of a relation is itself a relation.

Since this result is the angular stone of our development, we give yet more detail (construction of the target derivation tree) in the second appendix. ■

Remark 3. *In summary, and roughly speaking,* Lem. *10 replaces the occurrences of* START *(resp.* PARAM*) for variables not in ξ by* PARAM *(resp. nested* PARAM + EXCHANGE *). Occurrences on* START *(resp.* PARAM*) for variables in ξ are preserved.*

Lemma 11. *Let $\pi = (01)$ and $\omega = \xi \cap \zeta$; then*

$$\llbracket \llbracket A \rrbracket_\xi \rrbracket_\zeta [\bar{x} \mapsto \bar{x} \ddagger^\pi \mid x \in \omega] \ddagger^\pi_\omega =_\beta \llbracket \llbracket A \rrbracket_\zeta \rrbracket_\xi$$

Proof: By induction on the structure of A. We limit details to a few interesting cases.

- Variable, Exchange and Param cases are treated all at one by proving the lemma for $A = \lceil\!\lceil x \rceil\!\rceil^n \ddagger^\rho$, for any n and ρ. We calculate the left and right-hand-side in each of the following cases.
  - $x \in \xi$, $x \in \zeta$. (then $x \in \omega$)

$$\begin{aligned}
&\llbracket \llbracket \lceil\!\lceil x \rceil\!\rceil^n \ddagger^\rho \rrbracket_\xi \rrbracket_\zeta [\bar{x} \mapsto \bar{x} \ddagger^\pi \mid x \in \omega] \ddagger^\pi_\omega \\
&= \llbracket \llbracket \lceil\!\lceil x \rceil\!\rceil^n \ddagger^\rho \rrbracket_x \rrbracket_x [\bar{x} \mapsto \bar{x} \ddagger^\pi] \\
&= \llbracket \llbracket \lceil\!\lceil x \rceil\!\rceil^n \rrbracket_x \rrbracket_x \ddagger^{\rho+2} [\bar{x} \mapsto \bar{x} \ddagger^\pi] \\
&= \lceil\!\lceil x_{11} \rceil\!\rceil^n \ddagger^{(\ldots)} \ddagger^{\rho+2} [\bar{x} \mapsto \bar{x} \ddagger^\pi] \\
&= \lceil\!\lceil x_{11} \rceil\!\rceil^n \ddagger^{(\ldots)} \ddagger^{\rho+2} \\
&= \llbracket \llbracket \lceil\!\lceil x \rceil\!\rceil^n \rrbracket_x \rrbracket_x \ddagger^{\rho+2} \\
&= \llbracket \llbracket \lceil\!\lceil x \rceil\!\rceil^n \ddagger^\rho \rrbracket_x \rrbracket_x \\
&= \llbracket \llbracket \lceil\!\lceil x \rceil\!\rceil^n \ddagger^\rho \rrbracket_\zeta \rrbracket_\xi
\end{aligned}$$

  - $x \notin \xi$, $x \notin \zeta$. (then $x \notin \omega$)

$$\begin{aligned}
&\llbracket \llbracket \lceil\!\lceil x \rceil\!\rceil^n \ddagger^\rho \rrbracket_\xi \rrbracket_\zeta [\bar{x} \mapsto \bar{x} \ddagger^\pi \mid x \in \omega] \ddagger^\pi_\omega \\
&= \lceil\!\lceil \lceil\!\lceil \lceil\!\lceil x \rceil\!\rceil^n \ddagger^\rho \rceil\!\rceil \rceil\!\rceil \ddagger^\pi \\
&= \lceil\!\lceil x \rceil\!\rceil^{n+2} \ddagger^{\rho+2} \ddagger^\pi \\
&\quad \{\text{by } \pi \text{ disjoint from } \rho+2\} \\
&= \lceil\!\lceil x \rceil\!\rceil^{n+2} \ddagger^\pi \ddagger^{\rho+2} \\
&\quad \{\text{by Eq. (4)}\} \\
&= \lceil\!\lceil x \rceil\!\rceil^{n+2} \ddagger^{\rho+2} \\
&= \lceil\!\lceil \lceil\!\lceil \lceil\!\lceil x \rceil\!\rceil^n \ddagger^\rho \rceil\!\rceil \rceil\!\rceil \\
&= \llbracket \llbracket \lceil\!\lceil x \rceil\!\rceil^n \ddagger^\rho \rrbracket_\zeta \rrbracket_\xi
\end{aligned}$$

  - $x \in \xi$, $x \notin \zeta$. (then $x \notin \omega$)

$$\begin{aligned}
&\llbracket \llbracket \lceil\!\lceil x \rceil\!\rceil^n \ddagger^\rho \rrbracket_\xi \rrbracket_\zeta [\bar{x} \mapsto \bar{x} \ddagger^\pi \mid x \in \omega] \ddagger^\pi_\omega \\
&= \lceil\!\lceil \llbracket \lceil\!\lceil x \rceil\!\rceil^n \ddagger^\rho \rrbracket_x \rceil\!\rceil \ddagger^\pi \\
&= \lceil\!\lceil \llbracket \lceil\!\lceil x \rceil\!\rceil^n \rrbracket_x \rceil\!\rceil \ddagger^{\rho+2} \ddagger^\pi \\
&= \lceil\!\lceil \lceil\!\lceil x_1 \rceil\!\rceil^n \ddagger^{(0..n)} \rceil\!\rceil \ddagger^{\rho+2} \ddagger^\pi \\
&= \lceil\!\lceil x_1 \rceil\!\rceil^{n+1} \ddagger^{(1..n+1)} \ddagger^{\rho+2} \ddagger^\pi \\
&= \{\text{by } \pi \text{ disjoint from } \rho+2\} \\
&\quad \lceil\!\lceil x_1 \rceil\!\rceil^{n+1} \ddagger^{(1..n+1)\circ(01)} \ddagger^{\rho+2} \\
&= \lceil\!\lceil x_1 \rceil\!\rceil^{n+1} \ddagger^{(0..n+1)} \ddagger^{\rho+2} \\
&= \llbracket \lceil\!\lceil x \rceil\!\rceil^{n+1} \rrbracket_x \ddagger^{\rho+2} \\
&= \llbracket \lceil\!\lceil \lceil\!\lceil x \rceil\!\rceil^n \rceil\!\rceil \rrbracket_x \ddagger^{\rho+2} \\
&= \llbracket \lceil\!\lceil \lceil\!\lceil x \rceil\!\rceil^n \ddagger^\rho \rceil\!\rceil \rrbracket_x \\
&= \llbracket \llbracket \lceil\!\lceil x \rceil\!\rceil^n \ddagger^\rho \rrbracket_\zeta \rrbracket_\xi
\end{aligned}$$

  - $x \notin \xi$, $x \in \zeta$ (then $x \notin \omega$). Symmetric to the above case.
- Abstraction.

$$\begin{aligned}
&\llbracket \llbracket \lambda \bar{x} : A.\, b \rrbracket_\xi \rrbracket_\zeta [\bar{y} \mapsto \bar{y} \ddagger^\pi \mid y \in \omega] \ddagger^\pi_\omega \\
&= \lambda \bar{x} : \llbracket \llbracket A \rrbracket_\zeta \rrbracket_\xi \ddagger^\pi_\omega [\bar{y} \mapsto \bar{y} \ddagger^\pi \mid y \in \omega]. \\
&\quad \llbracket \llbracket b \rrbracket_{\zeta,x} \rrbracket_{\xi,x} [\bar{y} \mapsto \bar{y} \ddagger^\pi \mid y \in \omega][\bar{x} \mapsto \bar{x} \ddagger^\pi] \ddagger^\pi_{\omega,x} \\
&= \lambda \bar{x} : \llbracket \llbracket A \rrbracket_\zeta \rrbracket_\xi \ddagger^\pi_\omega [\bar{y} \mapsto \bar{y} \ddagger^\pi \mid y \in \omega]. \\
&\quad \llbracket \llbracket b \rrbracket_{\zeta,x} \rrbracket_{\xi,x} [\bar{y} \mapsto \bar{y} \ddagger^\pi \mid y \in \omega, x] \ddagger^\pi_{\omega,x} \\
&= \lambda \bar{x} : \llbracket \llbracket A \rrbracket_\xi \rrbracket_\zeta .\, \llbracket \llbracket b \rrbracket_{\xi,x} \rrbracket_{\zeta,x} \\
&= \llbracket \llbracket \lambda \bar{x} : A.\, b \rrbracket_\zeta \rrbracket_\xi
\end{aligned}$$

■

Note that (3) is a corollary of the above lemma.

C. Subject reduction

In this section we prove subject reduction (preservation of types). We start by discussing basic properties generally attributed to PTSs.

The weakening of contexts behaves in our calculus exactly in the same way as in all PTSs. Indeed, the usual thinning lemma holds.

Lemma 12 (Thinning). *Let Γ and Δ be legal contexts such that $\Gamma \subseteq \Delta$. Then $\Gamma \vdash A : B \Longrightarrow \Delta \vdash A : B$.*

Proof: As in [4, lem. 5.2.12]. ■

The generation lemma for our calculus must account for the new parametricity construct.

Lemma 13 (Generation). *The statement of the lemma is the same as that of the generation lemma for PTS [4, lem. 5.2.13], but with the additional case for the* PARAM *rule:*

- *If $\Gamma \vdash \lceil\!\lceil x \rceil\!\rceil : C$ then there exists B such that $\Gamma \vdash B : s^n$, $(x : B) \in \Gamma$, and $C =_\beta x \in \llbracket B \rrbracket$.*

Proof: As in [4]:

- We follow the derivation $\Gamma \vdash \lceil\!\lceil x \rceil\!\rceil : C$ until $\lceil\!\lceil x \rceil\!\rceil$ is introduced. It can only be done by the following rule

$$\frac{\Delta \vdash B : s_n}{\Delta, x : B \vdash \lceil\!\lceil x \rceil\!\rceil : x \in \llbracket B \rrbracket} \text{ PARAM}$$

with $C =_\beta x \in \llbracket B \rrbracket$, and $(\Delta, \bar{x} : B) \subseteq \Gamma$. The conclusion stems from Lem. 12.

■

Theorem 5 (Subject reduction)**.** *If $A \longrightarrow A'$ and $\Gamma \vdash A : T$, then $\Gamma \vdash A' : T$.*

Proof: Most of the technicalities of the proof by Barendregt [4], concern β-reduction, and are not changed by our addition of parametricity.

Hence we discuss here only the handling of the parametricity construct: our task is to check that substitution a concrete term a for x in $\llbracket x \rrbracket$ preserves the type of the expression.

Facing a term such as $\llbracket x \rrbracket$ in context Γ, we know by generation that it must have type $x \in \llbracket B \rrbracket$ (for some type B valid in Γ, and $x : B$). We can then prove that substituting a term a of type B' (where B' is convertible to B) for x preserves the type of the expression. Indeed, the expression then reduces to $\llbracket a \rrbracket$, which has type $a \in \llbracket B' \rrbracket$ by Th. 4. In turn, $a \in \llbracket B' \rrbracket$ is convertible to $x \in \llbracket B \rrbracket$ by Lem. 7. ■

D. Strong normalization

In this section we present a formalization of the intuitive model presented in Sec. II-D. We do this via a transformation, from our system $\mathcal{P}$ to the naked PTS $\mathcal{O}$.

Each term is mapped to a term where parametricity witnesses are passed explicitly. Simultaneously, contexts are extended with explicit witnesses: in a first approximation, each binding $x : A$ is replaced by a multiple binding $x : A, \breve{x} : x \in \llbracket A \rrbracket$. This means that $\llbracket x \rrbracket$ can be interpreted by the corresponding variable $\breve{x}$ in the context. The following table shows how some example terms[3] can be interpreted (for the sake of readability we omit type annotations in the abstractions, since they play no role in these examples):

original term A	its interpretation $\llparenthesis A \rrparenthesis$
$\lambda x.\, \llbracket x \rrbracket$	$\lambda x.\, \lambda \breve{x}.\, \breve{x}$
$(\lambda x.\, \llbracket x \rrbracket)\, (y\, z)$	$(\lambda x.\, \lambda \breve{x}.\, \breve{x})\, (y\, z)\, (\breve{y}\, z\, \breve{z})$
$(\lambda x.\, \llbracket x \rrbracket)(\lambda y.\, \llbracket y \rrbracket)$	$(\lambda x.\, \lambda \breve{x}.\, \breve{x}) \quad (\lambda y.\, \lambda \breve{y}.\, \breve{y})$
	$(\lambda y.\, \lambda \breve{y}.\, \lambda \breve{y}'.\, \lambda \breve{y}'_2.\, \breve{y}'_2)$

Given that the interpretation (written $\llparenthesis \cdot \rrparenthesis$) is sound with respect to $\mathcal{O}$ (Lem. 16) and that it preserves reductions (Lem. 15), we obtain strong normalization (Th. 6). The rest of the section is devoted to defining the model formally, and arguing for its soundness.

In general, the transformation is not trivial, because of the interaction between functions and their arguments, occurring in the APP rule. If a function uses parametricity on one of its argument, calls to the function must also compute explicit parametricity witnesses. (This may in turn trigger the need for more explicit witnesses at the call site). Further, if the function is passed to another function, this will create further needs for explicit witnesses.

As we have seen above, each binding $x : A$ should be replaced be $x : A, \ldots, \breve{x}^n : x \in \llbracket A \rrbracket^n$ for some n. Our main task is to compute a n that would be big enough to make all the parametricity witnesses $\llbracket x \rrbracket^k$ explicit. To do so, we use an intermediate representation of the typing derivation, containing some *constraints* on the n's, by annotation of the derivation tree, as in Fig. 3. We assume without loss of generality that variable names are distinct, so the n's are given by a (partial) valuation $\epsilon : \llbracket \mathsf{Var} \rrbracket \to \mathbb{N}$. This annotation of the derivation with constraints is an instance of a technique known as type-based analysis [26].

$$\frac{\mathfrak{t} :: \Gamma \vdash \bar{a} : \bar{A} \qquad \begin{array}{c}\Gamma \vdash F : (\forall \bar{x} : \bar{A}.\, B) \\ \{e + \epsilon(x) \leq \epsilon(y) \mid e \leq \epsilon(y) \in \mathfrak{t}\}\end{array}}{\Gamma \vdash F\, \bar{a} : B[\bar{x} \mapsto \bar{a}]}\ \textsc{Application}$$

$$\frac{\mathfrak{t} :: \Gamma \vdash \breve{a} : \breve{A} \qquad \begin{array}{c}\Gamma \vdash F : (\forall^\bullet \breve{x} : \breve{A}.\, s^n) \\ \{e + \epsilon(x) \leq \epsilon(y) \mid e \leq \epsilon(y) \in \mathfrak{t}\}\end{array}}{\Gamma \vdash F \bullet \breve{a} : s^n}\ \textsc{Rel-Elim}$$

$$\frac{\Gamma \vdash A : s^m \qquad n \leq \epsilon(x)}{\Gamma, x_i : A \vdash \llbracket x_i \rrbracket^n \ddagger^\pi : (x_i \in \llbracket A \rrbracket^n) \ddagger^\pi}\ \dim \pi \leq m + n \qquad \textsc{Param}/n$$

Fig. 3. Typing rules extended with constraints on the valuation ϵ. Rules omitted here (see Def. 6) remain unchanged. The notation $e \leq \epsilon(y) \in \mathfrak{t}$ expresses that the constraint appears in the tree $\mathfrak{t}$. (For the sake of conciseness, we merged the rules START, PARAM and EXCHANGE into PARAM/n here.)

The APPLICATION and REL-ELIM rules require special care: Indeed, we need to "lift" the inequalities of the right sub-tree $\mathfrak{t}$, since if F has to be extended to a term of type $\forall \llbracket x : A \rrbracket^n.\, B$, then it has to be fed with n extra parametricity witnesses $\llbracket a \rrbracket \cdots \llbracket a \rrbracket^n$, hence the context has to be extended enough to contain $\breve{y}^n$, for each y free in a. Note that the constraints $e + \epsilon(x) \leq \epsilon(y)$ that we add in the APPLICATION and REL-ELIM rule are more restrictive than the corresponding $e \leq \epsilon(y)$ that are in $\mathfrak{t}$, so one can simply ignore the latter.

We need to check that the system of constraints has a solution. In fact, the simplex it defines is unbound: indeed, the only place where a variable appears at the left-hand side of a constraint is in APPLICATION and REL-ELIM when we "lift by x" the constraints in the sub-tree $\mathfrak{t}$; It cannot create any cycle, since x does not appear in $\mathfrak{t}$.

With our notion of cubes instead of usual bindings, extending the context with an explicit witness corresponds to adding one dimension to the cube. However, we *a priory* only need to access one of the new vertices, the one that has the new dimension set to one. Hence in general, each of the $2^{\dim \bar{A}}$ vertices x_i of a binding $\bar{x} : \bar{A}$ will be extended with $x_{ji} : x_i \in \llbracket A_i \bullet (\bar{x} /\!\!/ i) \rrbracket^k$ for $0 \leq k \leq \epsilon(x)$ and $j = 0^{\epsilon(x) - k} 1^k$.

Permutations on variables yield yet another difficulty, as one can see in the example $\lambda x : A.\, \lambda y_1 : x \in \llbracket A \rrbracket.\, \llbracket y_1 \rrbracket \ddagger^{(12)}$. (The cubes have been flattened for the sake of readability.) Here, $\llbracket y_1 \rrbracket \ddagger^{(12)} : \llbracket A \rrbracket^2\, x\, y_1\, \llbracket x \rrbracket$ while $\llbracket y_1 \rrbracket : \llbracket A \rrbracket^2\, x\, \llbracket x \rrbracket\, y_1$.

[3]The third of them demonstrates nested parametricity.

Our solution is to not only extend the context with explicit parametricity witnesses, but also with explicit *permuted* parametricity witnesses. Hence a possible interpretation of the previous term in the naked system $\mathcal{O}$ is the following:

$$\begin{aligned}&\lambda x_0 : A.\, \lambda x_1 : (\llbracket A \rrbracket\, x_0).\\ &\qquad \lambda y_{01} : (\llbracket A \rrbracket\, x_0).\, \lambda y_{11} : (\llbracket A \rrbracket^2\, x_0\, x_1\, y_{01}).\\ &\qquad\qquad \lambda y_{11}^{(12)} : (\llbracket A \rrbracket^2\, x_0\, y_{01}\, x_1).\quad y_{11}^{(12)}\end{aligned}$$

We are not focusing on the minimal extension here, and we add witnesses for each possible permutation. It is however possible to refine this extension, since for instance the relations are symmetric in the new dimensions, hence we can ignore permutation cycles that are entirely contained into these new dimensions.

Definition 9 (Explicit binding of witnesses)**.** *Writing $\mathfrak{S}_n$ to be the group of permutations on $\{0, \dots, n-1\}$,*

$$\begin{aligned}\llparenthesis s^n \rrparenthesis &= s\\ \llparenthesis \llbracket x_i \rrbracket^n \dagger^\pi \rrparenthesis &= x_{ji}^\pi \quad \textit{where } j = \overbrace{0\dots0\underbrace{1\dots1}_{n}}^{\epsilon(x)}\\ \llparenthesis \lambda \bar{x} : \bar{A}.\, B \rrparenthesis &= \lambda \llparenthesis \bar{x} : \bar{A} \rrparenthesis .\, \llparenthesis B \rrparenthesis\\ \llparenthesis \forall \bar{x} : \bar{A}.\, B \rrparenthesis &= \forall \llparenthesis \bar{x} : \bar{A} \rrparenthesis .\, \llparenthesis B \rrparenthesis\\ \llparenthesis F\,{}^x\bar{a} \rrparenthesis &= \llparenthesis F \rrparenthesis\, \{ \llparenthesis \llbracket a_i \rrbracket^k \ddagger^\pi \rrparenthesis \mid i \in \mathrm{ind}(\bar{a}), k \leq \epsilon(x),\\ &\qquad \pi \in \mathfrak{S}_{k+\dim \bar{a}} \quad \}\\ \llparenthesis \lambda^\bullet \check{x} : \check{A}.\, B \rrparenthesis &= \lambda \llparenthesis \check{x} : \check{A} \rrparenthesis .\, \llparenthesis B \rrparenthesis\\ \llparenthesis \forall^\bullet \check{x} : \check{A}.\, s^n \rrparenthesis &= \forall \llparenthesis \check{x} : \check{A} \rrparenthesis .\, s\\ \llparenthesis F \bullet^x \check{A} \rrparenthesis &= \llparenthesis F \rrparenthesis\, \{ \llparenthesis \llbracket a_i \rrbracket^k \ddagger^\pi \rrparenthesis \mid i \in \mathrm{ind}(\check{a}), k \leq \epsilon(x),\\ &\qquad \pi \in \mathfrak{S}_{k+\dim \check{a}} \quad \}\end{aligned}$$

$$\begin{aligned}\llparenthesis - \rrparenthesis &= -\\ \llparenthesis \Gamma, x_i : A \rrparenthesis &= \llparenthesis \Gamma \rrparenthesis, \llparenthesis x_i : A \rrparenthesis\end{aligned}$$

We introduce a new macro $\llparenthesis x_i : A \rrparenthesis$, which expands to the following multiple bindings:

$$\begin{aligned}\llparenthesis x_i : A : s^n \rrparenthesis = \{ x_{ji}^\pi : \llparenthesis (x_i \in \llbracket A \rrbracket^k) \ddagger^\pi \rrparenthesis \mid\ & k \leq \epsilon(x),\\ & j = 0^{\epsilon(x)-k} 1^k,\\ & \pi \in \mathfrak{S}_{k+n} \quad \}\end{aligned}$$

Bindings of cube variables are merely "flattened", using our previously defined macro:

$$\begin{aligned}\llparenthesis \bar{x} : \bar{A} \rrparenthesis &= \{ \llparenthesis x_i : A_i \bullet (\bar{x} /\!\!/ i) \rrparenthesis \mid i \in \mathrm{ind}(\bar{A}) \}\\ \llparenthesis \check{x} : \check{A} \rrparenthesis &= \{ \llparenthesis x_i : A_i \bullet (\check{x} /\!\!/ i) \rrparenthesis \mid i \in \mathrm{ind}(\check{A}) \}\end{aligned}$$

The essence of the model defined by $\llparenthesis \cdot \rrparenthesis$ is that a parametricity witness $\llbracket x_i \rrbracket^n \dagger^\pi$ is adequately modeled by the variable $x^\pi_{0\dots01\dots1i}$, that is, if x has type A, then $x_1 : x \in \llbracket A \rrbracket$, etc.

Lemma 14 ($\llparenthesis \cdot \rrparenthesis$ and substitution)**.**

$$\begin{aligned}\llparenthesis A[x_i \mapsto a_i] \rrparenthesis = \llparenthesis A \rrparenthesis [\, & x_{ji}^\pi \mapsto \llparenthesis \llbracket a_i \rrbracket^k \ddagger^\pi \rrparenthesis,\\ & k \leq \epsilon(x), j = 0^{\epsilon(x)-k} 1^k, \pi \in \dots]\end{aligned}$$

Proof: By induction on A; We illustrate how the proof proceeds by showing (only) the case for variables, since other cases stem from straightforward uses of the induction hypotheses.

$$\begin{aligned}\llparenthesis \llbracket x_i \rrbracket^n \dagger^\pi [x_i \mapsto a_i] \rrparenthesis &= \llparenthesis \llbracket a_i \rrbracket^n \ddagger^\pi \rrparenthesis\\ &= x_{ji}^\pi [x_{ji}^\pi \mapsto \llparenthesis \llbracket a_i \rrbracket^n \ddagger^\pi \rrparenthesis]\\ &= \llparenthesis \llbracket x_i \rrbracket^n \dagger^\pi \rrparenthesis [x_{ji}^\pi \mapsto \llparenthesis \llbracket a_i \rrbracket^k \ddagger^\pi \rrparenthesis, \dots]\end{aligned}$$

■

Lemma 15 (Congruence of $\llparenthesis \cdot \rrparenthesis$)**.** *If $\Gamma \vdash A : B$ with $A \longrightarrow A'$, then*

$$\llparenthesis A \rrparenthesis \longrightarrow^+ \llparenthesis A' \rrparenthesis$$

.

Proof: By induction on $A \longrightarrow A'$. ■

Finally we can show the soundness of our model, by proving that the transformation yields well-typed terms in $\mathcal{O}$.

Lemma 16. *Let $\Gamma \vdash A : B : s^n$, k such that $k \leq \epsilon(x)$ for each free variable x, and $\pi \in \mathfrak{S}_{n+k}$ Then*

$$\llparenthesis \Gamma \rrparenthesis \vdash_{\mathcal{O}} \llparenthesis \llbracket A \rrbracket^k \ddagger^\pi \rrparenthesis : \llparenthesis (A \in \llbracket B \rrbracket^k) \ddagger^\pi \rrparenthesis .$$

In particular, if $\Gamma \vdash A : B$ then $\llparenthesis \Gamma \rrparenthesis \vdash_{\mathcal{O}} \llparenthesis A \rrparenthesis : \llparenthesis B \rrparenthesis$.

Proof: Even though we need the stronger induction hypothesis, we only prove the latter application here: $\Gamma \vdash A : B \Longrightarrow \llparenthesis \Gamma \rrparenthesis \vdash_{\mathcal{O}} \llparenthesis A \rrparenthesis : \llparenthesis B \rrparenthesis$. By induction on the derivation:

AXIOM

Trivial.

WEAKENING

$$\dfrac{\dfrac{\dfrac{\Gamma\vdash A:B}{\text{induction}}}{\llparenthesis\Gamma\rrparenthesis \vdash \llparenthesis A\rrparenthesis : \llparenthesis B\rrparenthesis} \qquad \dfrac{\dfrac{\dfrac{\Gamma\vdash C:s^n}{\text{induction}}}{\llparenthesis\Gamma\rrparenthesis\vdash\llparenthesis C\rrparenthesis : s}}{\llparenthesis\Gamma\rrparenthesis,\llparenthesis x_i:C\rrparenthesis\ \text{legal}}\ \text{Lem. 10.3}}{\llparenthesis\Gamma\rrparenthesis,\llparenthesis x_i:C\rrparenthesis\vdash\llparenthesis A\rrparenthesis:\llparenthesis B\rrparenthesis}\ \text{Thinning}$$

APPLICATION (REL-ELIM is similar)

$$\text{by def.}\ \dfrac{\text{by def.}\ \dfrac{\dfrac{\dfrac{\Gamma\vdash F:(\forall\bar{x}:\bar{A}.\,B)}{\text{induction}}}{\llparenthesis\Gamma\rrparenthesis\vdash\llparenthesis F\rrparenthesis:(\forall\llparenthesis\bar{x}:\bar{A}\rrparenthesis.\,\llparenthesis B\rrparenthesis)}}{\llparenthesis\Gamma\rrparenthesis\vdash\llparenthesis F\rrparenthesis:(\forall\{x^{\pi}_{ji}:\ldots\}.\,\llparenthesis B\rrparenthesis)} \qquad \dfrac{\dfrac{\dfrac{\Gamma\vdash\bar{a}:\bar{A}}{\Gamma\vdash a_i:A_i\bullet(\bar{x}/\!\!/i)}\ \text{by def.}}{\text{induction}}}{\llparenthesis\Gamma\rrparenthesis\vdash\llparenthesis\llbracket a_i\rrbracket^{\|j\|}\ddagger^{\pi}\rrparenthesis:\llparenthesis(a_i\in\llbracket A_i\bullet(\bar{x}/\!\!/i)\rrbracket^{\|j\|})\ddagger^{\pi}\rrparenthesis}}{\dfrac{\llparenthesis\Gamma\rrparenthesis\vdash\llparenthesis F\rrparenthesis\,\{\llparenthesis\llbracket a_i\rrbracket^{\|j\|}\ddagger^{\pi}\rrparenthesis\mid\ldots\}:\llparenthesis B\rrparenthesis[x^{\pi}_{ji}\mapsto\llparenthesis\llbracket a_i\rrbracket^{\|j\|}\ddagger^{\pi}\rrparenthesis,\ldots]}{\llparenthesis\Gamma\rrparenthesis\vdash\llparenthesis F\,\bar{a}\rrparenthesis:\llparenthesis B[\bar{x}\mapsto\bar{a}]\rrparenthesis}\ \text{by def., Lem. 14}}\ \text{(many-)APP.}$$

ABSTRACTION (REL-INTRO is similar)

$$\text{by def.}\ \dfrac{\dfrac{\dfrac{\dfrac{\Gamma,\bar{x}:\bar{A}\vdash b:B}{\text{induction}}}{\llparenthesis\Gamma\rrparenthesis,\llparenthesis\bar{x}:\bar{A}\rrparenthesis\vdash\llparenthesis b\rrparenthesis:\llparenthesis B\rrparenthesis}\qquad\vdots}{\llparenthesis\Gamma\rrparenthesis\vdash(\lambda\llparenthesis\bar{x}:\bar{A}\rrparenthesis.\,\llparenthesis b\rrparenthesis:(\forall\llparenthesis\bar{x}:\bar{A}\rrparenthesis.\,\llparenthesis B\rrparenthesis}\ \text{(many-)ABS.}}{\llparenthesis\Gamma\rrparenthesis\vdash\llparenthesis\lambda\bar{x}:\bar{A}.\,b\rrparenthesis:\llparenthesis\forall\bar{x}:\bar{A}.\,B\rrparenthesis}$$

PRODUCT (REL-FORM is similar)

$$\text{by def.}\ \dfrac{\dfrac{\dfrac{\text{by def.}\ \dfrac{\Gamma\vdash\bar{A}:s_1^m}{\Gamma,\ldots\vdash A_i\bullet(\bar{x}/\!\!/i):s_1^m}}{\text{induction}}}{\llparenthesis\Gamma\rrparenthesis,\ldots\vdash\llparenthesis(x_i\in\llbracket A_i\bullet(\bar{x}/\!\!/i)\rrbracket^{\|j\|})\ddagger^{\pi}\rrparenthesis:s_1}\qquad\dfrac{\dfrac{\Gamma,\bar{x}:\bar{A}\vdash B:s_2^n}{\text{induction}}}{\llparenthesis\Gamma\rrparenthesis,\llparenthesis\bar{x}:\bar{A}\rrparenthesis\vdash\llparenthesis B\rrparenthesis:s_2}}{\dfrac{\llparenthesis\Gamma\rrparenthesis\vdash(\forall\{x^{\pi}_{ji}:\llparenthesis(x_i\in\llbracket A_i\bullet(\bar{x}/\!\!/i)\rrbracket^{\|j\|})\ddagger^{\pi}\rrparenthesis\mid\ldots\}.\,\llparenthesis B\rrparenthesis):s_3}{\llparenthesis\Gamma\rrparenthesis\vdash\llparenthesis\forall\bar{x}:\bar{A}.\,B\rrparenthesis:s_3}}\ \text{(many-)PROD.}$$

CONVERSION

$$\dfrac{\dfrac{\dfrac{\Gamma\vdash A:B}{\text{induction}}}{\llparenthesis\Gamma\rrparenthesis\vdash\llparenthesis A\rrparenthesis:\llparenthesis B\rrparenthesis}\qquad\dfrac{\dfrac{\Gamma\vdash B':s^n}{\text{induction}}}{\llparenthesis\Gamma\rrparenthesis\vdash\llparenthesis B'\rrparenthesis:s}\qquad\dfrac{\dfrac{B=_\beta B'}{\text{Corollary 1, Lem. 15}}}{\llparenthesis B\rrparenthesis=_\beta\llparenthesis B'\rrparenthesis}}{\llparenthesis\Gamma\rrparenthesis\vdash\llparenthesis A\rrparenthesis:\llparenthesis B'\rrparenthesis}\ \text{CONV.}$$

START, PARAM, EXCHANGE

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma\vdash A:s^m}{\text{induction}}}{\llparenthesis\Gamma\rrparenthesis,\llparenthesis x_i:A\rrparenthesis\ \text{legal}}}{\llparenthesis\Gamma\rrparenthesis,\llparenthesis x_i:A\rrparenthesis\vdash x^{\pi}_{ji}:\llparenthesis(x_i\in\llbracket A\rrbracket^n)\ddagger^{\pi}\rrparenthesis}\ \text{Thinning, START}}{\llparenthesis\Gamma\rrparenthesis,\llparenthesis x_i:A\rrparenthesis\vdash\llparenthesis\llbracket x_i\rrbracket^n\dagger^{\pi}\rrparenthesis:\llparenthesis(x_i\in\llbracket A\rrbracket^n)\ddagger^{\pi}\rrparenthesis}\ \text{by def.}$$

■

Theorem 6 (Strong normalization)**.** *If $\mathcal{O}$ is strongly normalizing, then so is $\mathcal{P}$, our system extended with* PARAM.

Proof: Assume $\Gamma\vdash A:B$ and consider a chain of reductions $A\longrightarrow^n A'$. We have $\llparenthesis A\rrparenthesis\longrightarrow^m\llparenthesis A'\rrparenthesis$, and $m\geq n$ by Lem. 15. We also have that $\llparenthesis A\rrparenthesis$ is typeable in $\mathcal{O}$, by Lem. 16. Therefore, only finite chains of reductions are possible. ■

V. GENERALIZED ABSTRACTION: PROOF DETAILS

The proof depends on the following lemmas:

1) $\Gamma\vdash A:B\Rightarrow\llbracket\Gamma\rrbracket_\xi\vdash\llbracket A\rrbracket_\xi:A\in\llbracket B\rrbracket_\xi$
2) $\Gamma\vdash B:s^n\Rightarrow\llbracket\Gamma\rrbracket_\xi,x:B\vdash x\in\llbracket B\rrbracket_\xi:s^{n+1}$
3) $\Gamma\vdash A:B\Rightarrow\llbracket\Gamma\rrbracket_\xi\vdash A:B$

The lemmas are proved by transforming derivation trees. They mutually depend on each other, (but only for structurally smaller statements, hence the recursion is sound). For each lemma, each rule is treated. The rule being handled is written before the corresponding part of the resulting derivation.

In the proofs, the application of each sub-lemma to an arbitrary derivation $\Gamma \vdash A : B$ are written as follows:

1) $\llbracket \Gamma \vdash A : B \rrbracket_\xi$
2) $\{\Gamma \vdash A : B\}_\xi$
3) $|\Gamma \vdash A : B|$

We only give further details for items 1 and 2 in the following.

A. $\Gamma \vdash A : B \Rightarrow \llbracket\Gamma\rrbracket_\xi \vdash \llbracket A\rrbracket_\xi : A \in \llbracket B\rrbracket_\xi$

•AXIOM; REL-ELIM; REL-FORM; PRODUCT. In this case, the definition of $\llbracket A\rrbracket_\xi$ falls through: a new relation is introduced. The proof relies on the next sub-lemma.

$$\Gamma \vdash A : s^n$$

$$\dfrac{\dfrac{\begin{array}{c}\vdots\,\{\Gamma \vdash A : s^n\}_\xi \\ \llbracket\Gamma\rrbracket_\xi, z_0 : A \vdash z_0 \in \llbracket A\rrbracket_\xi : s^{n+1}\end{array} \quad \begin{array}{c}\vdots\,|\Gamma \vdash A : s^n| \\ \llbracket\Gamma\rrbracket_\xi \vdash A : s^n\end{array}}{\llbracket\Gamma\rrbracket_\xi \vdash \lambda^{\bullet}\check{z} : \begin{pmatrix}A\\ \cdot\end{pmatrix}. z_0 \in \llbracket A\rrbracket_\xi : \begin{pmatrix}A\\ \cdot\end{pmatrix} \overset{\bullet}{\to} s^{n+1}}\text{Rel-I}}{\llbracket\Gamma\rrbracket_\xi \vdash \llbracket A\rrbracket_\xi : A \in \llbracket s^n\rrbracket_\xi}\textsc{def}$$

•WEAKENING

$$\dfrac{\Gamma \vdash A : B \quad \Gamma \vdash C : s^n}{\Gamma, x : C \vdash A : B}\textsc{wk}$$

-$x \notin \xi$

$$\dfrac{\dfrac{\begin{array}{c}\vdots\,\llbracket\Gamma \vdash A : B\rrbracket_\xi \\ \llbracket\Gamma\rrbracket_\xi \vdash \llbracket A\rrbracket_\xi : A \in \llbracket B\rrbracket_\xi\end{array} \quad \begin{array}{c}\vdots\,|\Gamma \vdash C : s^n| \\ \llbracket\Gamma\rrbracket_\xi \vdash C : s^n\end{array}}{\llbracket\Gamma\rrbracket_\xi, x : C \vdash \llbracket A\rrbracket_\xi : A \in \llbracket B\rrbracket_\xi}\textsc{wk}}{\llbracket\Gamma, x : C\rrbracket_\xi \vdash \llbracket A\rrbracket_\xi : A \in \llbracket B\rrbracket_\xi}\textsc{def}$$

-$x \in \xi$

$$\dfrac{\dfrac{\begin{array}{c}\vdots\,\llbracket\Gamma \vdash C : s^n\rrbracket_\xi \\ \llbracket\Gamma\rrbracket_\xi \vdash \llbracket C\rrbracket_\xi : C \in \llbracket s^n\rrbracket_\xi\end{array} \quad \begin{array}{c}\vdots\,|\Gamma \vdash C : s^n| \\ \llbracket\Gamma\rrbracket_\xi \vdash C : s^n\end{array}}{\dfrac{\llbracket\Gamma\rrbracket_\xi, x_0 : C \vdash \llbracket C\rrbracket_\xi : C \in \llbracket s^n\rrbracket_\xi}{\llbracket\Gamma\rrbracket_\xi, x_0 : C \vdash \llbracket C\rrbracket_\xi : \begin{pmatrix}C\\ \cdot\end{pmatrix} \overset{\bullet}{\to} s^{n+1}}\textsc{def}}\textsc{wk} \quad \dfrac{\begin{array}{c}\vdots\,|\Gamma \vdash C : s^n| \\ \llbracket\Gamma\rrbracket_\xi \vdash C : s^n\end{array}}{\llbracket\Gamma\rrbracket_\xi, x_0 : C \vdash \begin{pmatrix}x_0\\ \cdot\end{pmatrix} : \begin{pmatrix}C\\ \cdot\end{pmatrix}}\textsc{st}}{\llbracket\Gamma\rrbracket_\xi, x_0 : C \vdash \llbracket C\rrbracket_\xi {}^{\bullet}\begin{pmatrix}x_0\\ \cdot\end{pmatrix} : s^{n+1}}\textsc{app}$$

$$\dfrac{\dfrac{\dfrac{\begin{array}{c}\vdots\,\llbracket\Gamma \vdash A : B\rrbracket_\xi \\ \llbracket\Gamma\rrbracket_\xi \vdash \llbracket A\rrbracket_\xi : A \in \llbracket B\rrbracket_\xi\end{array} \quad \begin{array}{c}\vdots\,|\Gamma \vdash C : s^n| \\ \llbracket\Gamma\rrbracket_\xi \vdash C : s^n\end{array}}{\llbracket\Gamma\rrbracket_\xi, x_0 : C \vdash \llbracket A\rrbracket_\xi : A \in \llbracket B\rrbracket_\xi}\textsc{wk} \quad \vdots}{\llbracket\Gamma\rrbracket_\xi, \bar{x} : \begin{pmatrix}C\\ \llbracket C\rrbracket_\xi\end{pmatrix} \vdash \llbracket A\rrbracket_\xi : A \in \llbracket B\rrbracket_\xi}\textsc{wk}}{\llbracket\Gamma, x : C\rrbracket_\xi \vdash \llbracket A\rrbracket_\xi : A \in \llbracket B\rrbracket_\xi}\textsc{def}$$

•REL-INTRO

$$\dfrac{\Gamma, \check{z} : \check{A} \vdash B : s^n \quad \Gamma \vdash \check{A} : s^n}{\Gamma \vdash (\lambda^{\bullet}\check{z} : \check{A}. B) : \check{A} \overset{\bullet}{\to} s^n}\text{Rel-I}$$

$$\dfrac{\dfrac{\begin{array}{c}\vdots\{\Gamma,\check z:\check A\vdash B:s^n\}_\xi \qquad \wr\\ \llbracket\Gamma\rrbracket_\xi,\check z:\check A, z_{01\ldots1}:B\vdash z_{01\ldots1}\in\llbracket B\rrbracket_\xi:s^{n+1}\end{array}}{\llbracket\Gamma\rrbracket_\xi,\check z:(\llbracket\check A\rrbracket_\xi\oplus(\lambda^{\bullet}\check z:\check A.\,B))\vdash z_{01\ldots1}\in\llbracket B\rrbracket_\xi:s^{n+1}}\,{\scriptstyle\mathrm{WK}}\quad \begin{array}{c}\wr\\ \llbracket\Gamma\rrbracket_\xi\vdash(\llbracket\check A\rrbracket_\xi\oplus(\lambda^{\bullet}\check z:\check A.\,B)):s^{n+1}\end{array}}{\dfrac{\llbracket\Gamma\rrbracket_\xi\vdash(\lambda^{\bullet}\check z:(\llbracket\check A\rrbracket_\xi\oplus(\lambda^{\bullet}\check z:\check A.\,B)).\,z_{01\ldots1}\in\llbracket B\rrbracket_\xi):((\llbracket\check A\rrbracket_\xi\oplus(\lambda^{\bullet}\check z:\check A.\,B))\overset{\bullet}{\to}s^{n+1})}{\llbracket\Gamma\rrbracket_\xi\vdash\llbracket\lambda^{\bullet}\check z:\check A.\,B\rrbracket_\xi:(\lambda^{\bullet}\check z:\check A.\,B\in\llbracket\check A\overset{\bullet}{\to}s^n\rrbracket_\xi)}\,{\scriptstyle\mathrm{DEF}}}\,\text{Rel-I}$$

●APPLICATION

$$\dfrac{\Gamma\vdash F:(\forall\bar x:\bar A.\,B)\quad\Gamma\vdash\bar a:\bar A}{\Gamma\vdash F\,\bar a:B[x\mapsto\bar a]}\,{\scriptstyle\mathrm{APP}}$$

$$\dfrac{\dfrac{\dfrac{\begin{array}{c}\vdots\llbracket\Gamma\vdash F:(\forall\bar x:\bar A.\,B)\rrbracket_\xi\\ \llbracket\Gamma\rrbracket_\xi\vdash\llbracket F\rrbracket_\xi:F\in\llbracket\forall\bar x:\bar A.\,B\rrbracket_\xi\end{array}}{\llbracket\Gamma\rrbracket_\xi\vdash\llbracket F\rrbracket_\xi:(\forall\bar x:\llbracket\bar A\rrbracket_\xi.\,(F\,\bar x)\in\llbracket B\rrbracket_{\xi,x})}\,{\scriptstyle\mathrm{DEF}}\quad\begin{array}{c}\vdots\llbracket\Gamma\vdash\bar a:\bar A\rrbracket_\xi\\ \llbracket\Gamma\rrbracket_\xi\vdash\llbracket\bar a\rrbracket_\xi:\llbracket\bar A\rrbracket_\xi\end{array}}{\dfrac{\llbracket\Gamma\rrbracket_\xi\vdash\llbracket F\rrbracket_\xi\,\llbracket\bar a\rrbracket_\xi:((F\,\bar x)\in\llbracket B\rrbracket_{\xi,x})[\bar x\mapsto\bar a]}{\llbracket\Gamma\rrbracket_\xi\vdash\llbracket F\rrbracket_\xi\,\llbracket\bar a\rrbracket_\xi:(F\,\bar a)\in\llbracket B[\bar x\mapsto\bar a]\rrbracket_\xi}\,\text{Lemma 2}}\,{\scriptstyle\mathrm{APP}}}{\llbracket\Gamma\rrbracket_\xi\vdash\llbracket F\,\bar a\rrbracket_\xi:(F\,\bar a)\in\llbracket B[\bar x\mapsto\bar a]\rrbracket_\xi}\,{\scriptstyle\mathrm{DEF}}$$

●ABSTRACTION

$$\dfrac{\Gamma,z:\bar A\vdash b:B\quad\Gamma\vdash(\forall\bar x:\bar A.\,B):s^n}{\Gamma\vdash(\lambda\bar z:\bar A.\,b):(\forall\bar x:\bar A.\,B)}\,{\scriptstyle\mathrm{ABS}}$$

$$\dfrac{\dfrac{\dfrac{\begin{array}{c}\vdots\llbracket\Gamma,z:\bar A\vdash b:B\rrbracket_{\xi,z}\\ \llbracket\Gamma,z:\bar A\rrbracket_{\xi,z}\vdash\llbracket b\rrbracket_{\xi,z}:b\in\llbracket B\rrbracket_{\xi,z}\end{array}}{\llbracket\Gamma\rrbracket_\xi,z:\llbracket\bar A\rrbracket_\xi\vdash\llbracket b\rrbracket_{\xi,z}:b\in\llbracket B\rrbracket_{\xi,z}}\,{\scriptstyle\mathrm{DEF}}\quad\begin{array}{c}\wr\\ \llbracket\Gamma\rrbracket_\xi\vdash(\forall\bar z:\llbracket\bar A\rrbracket_\xi.\,b\in\llbracket B\rrbracket_{\xi,z}):s^{n+1}\end{array}}{\llbracket\Gamma\rrbracket_\xi\vdash(\lambda\bar z:\llbracket\bar A\rrbracket_\xi.\,\llbracket b\rrbracket_{\xi,z}):(\forall\bar z:\llbracket\bar A\rrbracket_\xi.\,b\in\llbracket B\rrbracket_{\xi,z})}\,{\scriptstyle\mathrm{ABS}}}{\llbracket\Gamma\rrbracket_\xi\vdash\llbracket\lambda\bar z:\bar A.\,b\rrbracket_\xi:(\lambda\bar z:\bar A.\,b)\in\llbracket\forall\bar z:\bar A.\,B\rrbracket_\xi}\,{\scriptstyle\mathrm{DEF}}$$

●CONVERSION

$$\dfrac{\Gamma\vdash A:B'\quad\Gamma\vdash B:s^n\quad B'=_\beta B}{\Gamma\vdash A:B}\,{\scriptstyle\mathrm{CONV}}$$

$$\dfrac{\begin{array}{c}\vdots\{\Gamma\vdash B:s^n\}_\xi\\ \llbracket\Gamma\rrbracket_\xi,x:B\vdash x\in\llbracket B\rrbracket_\xi:s^{n+1}\end{array}\quad\dfrac{\begin{array}{c}\vdots|\Gamma\vdash A:B'|\\ \llbracket\Gamma\rrbracket_\xi\vdash A:B'\end{array}\quad\begin{array}{c}\vdots|\Gamma\vdash B:s^n|\\ \llbracket\Gamma\rrbracket_\xi\vdash B:s^n\end{array}\quad B'=_\beta B}{\llbracket\Gamma\rrbracket_\xi\vdash A:B}\,{\scriptstyle\mathrm{CONV}}}{\llbracket\Gamma\rrbracket_\xi\vdash A\in\llbracket B\rrbracket_\xi:s^{n+1}}\,\text{subst}$$

$$\dfrac{\begin{array}{c}\vdots\llbracket\Gamma\vdash A:B'\rrbracket_\xi\\ \llbracket\Gamma\rrbracket_\xi\vdash\llbracket A\rrbracket_\xi:A\in\llbracket B'\rrbracket_\xi\end{array}\quad\dfrac{B'=_\beta B}{A\in\llbracket B'\rrbracket_\xi=_\beta A\in\llbracket B\rrbracket_\xi}\,\text{Lemma 7}}{\llbracket\Gamma\rrbracket_\xi\vdash\llbracket A\rrbracket_\xi:A\in\llbracket B\rrbracket_\xi}\,{\scriptstyle\mathrm{CONV}}$$

●START

$$\dfrac{\Gamma\vdash A:s^n}{\Gamma,x:A\vdash x:A}\,{\scriptstyle\mathrm{ST}}$$

-$x\notin\xi$

Since ξ conforms to Γ, no variable of ξ is in Γ.

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma\vdash A:s^n}{\Gamma,x:A\vdash x:A}\,{\scriptstyle\mathrm{ST}}}{\Gamma,x:A\vdash\llbracket x\rrbracket:x\in\llbracket A\rrbracket}\,{\scriptstyle\mathrm{PARAM}}}{\llbracket\Gamma\rrbracket_\xi,x:A\vdash\llbracket x\rrbracket:x\in\llbracket A\rrbracket_\xi}\,\text{(conforms)}}{\llbracket\Gamma,x:A\rrbracket_\xi\vdash\llbracket x\rrbracket_\xi:x\in\llbracket A\rrbracket_\xi}\,{\scriptstyle\mathrm{DEF}}$$

-$x\in\xi$

$$\dfrac{\dfrac{\begin{array}{c}\vdots\{\Gamma\vdash A:s^n\}_\xi\\ \llbracket\Gamma\rrbracket_\xi,x_0:A\vdash x_0\in\llbracket A\rrbracket_\xi:s^{n+1}\end{array}}{\llbracket\Gamma\rrbracket_\xi,x_0:A,x_1:x_0\in\llbracket A\rrbracket_\xi\vdash x_1:x_0\in\llbracket A\rrbracket_\xi}\,{\scriptstyle\mathrm{ST}}}{\llbracket\Gamma,x:A\rrbracket_\xi\vdash\llbracket x\rrbracket_\xi:x\in\llbracket A\rrbracket_\xi}\,{\scriptstyle\mathrm{DEF}}$$

•PARAM

$$\dfrac{\Gamma \vdash x : A}{\Gamma \vdash \llbracket x \rrbracket : x \in \llbracket A \rrbracket}\,\textsc{param}$$

Let π be (01).

$$\llbracket \Gamma \rrbracket_\xi \vdash \llbracket x \rrbracket \in \llbracket x \in \llbracket A \rrbracket \rrbracket_\xi : s^{n+1}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots\,\llbracket \Gamma \vdash x : A \rrbracket_\xi}{\llbracket \Gamma \rrbracket_\xi \vdash \llbracket x \rrbracket_\xi : x \in \llbracket A \rrbracket_\xi}}{\llbracket \Gamma \rrbracket_\xi \vdash \llbracket \llbracket x \rrbracket_\xi \rrbracket : \llbracket x \rrbracket_\xi \in \llbracket x \in \llbracket A \rrbracket_\xi \rrbracket}\,\textsc{param}}{\llbracket \Gamma \rrbracket_\xi \vdash \llbracket x \rrbracket_\xi \ddagger^\pi : (\llbracket x \rrbracket_\xi \in \llbracket x \in \llbracket A \rrbracket_\xi \rrbracket) \ddagger^\pi}\,\textsc{exchange}}{\llbracket \Gamma \rrbracket_\xi \vdash \llbracket \llbracket x \rrbracket \rrbracket_\xi : (\llbracket x \rrbracket_\xi \in \llbracket x \in \llbracket A \rrbracket_\xi \rrbracket) \ddagger^\pi}\,\textsc{def}
\qquad
\dfrac{\dfrac{\dfrac{\dfrac{\text{Eq. (3)}}{\llbracket \llbracket A \rrbracket \rrbracket_\xi =_\beta \llbracket \llbracket A \rrbracket_\xi \rrbracket \ddagger^\pi}}{\llbracket \llbracket A \rrbracket \rrbracket_\xi \bullet \begin{pmatrix} x & \llbracket x \rrbracket \\ \llbracket x \rrbracket_\xi & . \end{pmatrix} =_\beta \llbracket \llbracket A \rrbracket_\xi \rrbracket \ddagger^\pi \bullet \begin{pmatrix} x & \llbracket x \rrbracket \\ \llbracket x \rrbracket_\xi & . \end{pmatrix}}\,\beta\text{-}\textsc{Rel-Elim}}{\llbracket \llbracket A \rrbracket \rrbracket_\xi \bullet \begin{pmatrix} x & \llbracket x \rrbracket \\ \llbracket x \rrbracket_\xi & . \end{pmatrix} =_\beta (\llbracket \llbracket A \rrbracket_\xi \rrbracket \bullet \begin{pmatrix} x & \llbracket x \rrbracket_\xi \\ \llbracket x \rrbracket & . \end{pmatrix}) \ddagger^\pi}\,\textsc{def}}{\llbracket x \rrbracket \in \llbracket x \in \llbracket A \rrbracket \rrbracket_\xi =_\beta (\llbracket x \rrbracket_\xi \in \llbracket x \in \llbracket A \rrbracket_\xi \rrbracket) \ddagger^\pi}\,\textsc{def}$$

$$\dfrac{\cdots}{\llbracket \Gamma \rrbracket_\xi \vdash \llbracket \llbracket x \rrbracket \rrbracket_\xi : \llbracket x \rrbracket \in \llbracket x \in \llbracket A \rrbracket \rrbracket_\xi}\,\textsc{conv}$$

•EXCHANGE Trivial.

B. $\Gamma \vdash B : s^n \Rightarrow \llbracket \Gamma \rrbracket_\xi, x : B \vdash x \in \llbracket B \rrbracket_\xi : s^{n+1}$

•AXIOM

$$\dfrac{}{\vdash s_1^n : s_2^n}\,\textsc{ax}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash s_1^n : s_2^n}\,\textsc{ax}}{x : s_1^n \vdash x : s_1^n}\,\textsc{st}}{x : s_1^n \vdash \begin{pmatrix} x \\ . \end{pmatrix} : s_1^{n+1}}\,\textsc{def}}{x : s_1^n \vdash \begin{pmatrix} x \\ . \end{pmatrix} \overset{\bullet}{\to} s_1^{n+1} : s_2^{n+1}}\,\text{Rel-F}}{x : s_1^n \vdash x \in \llbracket s_1^n \rrbracket_\xi : s_2^{n+1}}\,\textsc{def}$$

•START

$$\dfrac{\Gamma \vdash s^n : s_2^n}{\Gamma, y : s^n \vdash y : s^n}\,\textsc{st}$$

-$y \notin \xi$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma \vdash s^n : s_2^n}{\Gamma, y : s^n \vdash y : s^n}{\scriptstyle\text{ST}}}{\Gamma, y : s^n \vdash \llbracket y \rrbracket : y \in \llbracket s^n \rrbracket}{\scriptstyle\text{PARAM}} \quad \dfrac{\Gamma \vdash s^n : s_2^n}{\Gamma, y : s^n \vdash y : s^n}{\scriptstyle\text{ST}}}{\Gamma, y : s^n, x : y \vdash \llbracket y \rrbracket : y \in \llbracket s^n \rrbracket}{\scriptstyle\text{WK}}}{\Gamma, y : s^n, x : y \vdash \llbracket y \rrbracket : y \overset{\bullet}{\to} s^{n+1}}{\scriptstyle\text{DEF}} \quad \dfrac{\dfrac{\Gamma \vdash s^n : s_2^n}{\Gamma, y : s^n \vdash y : s^n}{\scriptstyle\text{ST}}}{\Gamma, y : s^n, x : y \vdash x : y}{\scriptstyle\text{ST}}}{\Gamma, y : s^n, x : y \vdash \llbracket y \rrbracket \bullet \begin{pmatrix} x \\ . \end{pmatrix} : s^{n+1}}\text{Rel-E}}{\llbracket \Gamma \rrbracket_\xi, y : s^n, x : y \vdash \llbracket y \rrbracket \bullet \begin{pmatrix} x \\ . \end{pmatrix} : s^{n+1}}\text{(conforms)}}{\llbracket \Gamma, y : s^n \rrbracket_\xi, x : y \vdash x \in \llbracket y \rrbracket_\xi : s^{n+1}}{\scriptstyle\text{DEF}}$$

-$y \in \xi$

$$\dfrac{\dfrac{\begin{matrix}\wr \\ \wr \\ \Gamma, y : s^n, \dot{y} : y \overset{\bullet}{\to} s^{n+1}, x : y \vdash \dot{y} : y \overset{\bullet}{\to} s^{n+1}\end{matrix} \quad \dfrac{\begin{matrix}\wr \\ \wr \\ \Gamma, y : s^n, \dot{y} : y \overset{\bullet}{\to} s^{n+1} \vdash y : s^n\end{matrix}}{\Gamma, y : s^n, \dot{y} : y \overset{\bullet}{\to} s^{n+1}, x : y \vdash x : y}{\scriptstyle\text{ST}}}{\llbracket \Gamma \rrbracket_\xi, y : s^n, \dot{y} : y \overset{\bullet}{\to} s^{n+1}, x : y \vdash \dot{y} \bullet \begin{pmatrix} x \\ . \end{pmatrix} : s^{n+1}}\text{Rel-E}}{\llbracket \Gamma, y : s^n \rrbracket_\xi, x : y \vdash x \in \llbracket y \rrbracket_\xi : s^{n+1}}{\scriptstyle\text{DEF}}$$

●WEAKENING

$$\dfrac{\Gamma \vdash B : s^n \quad \Gamma \vdash C : s^m}{\Gamma, y : C \vdash B : s^n}{\scriptstyle\text{WK}}$$

-$y \notin \xi$

$$\dfrac{\dfrac{\begin{matrix}\vdots \{\Gamma \vdash B : s^n\}_\xi \\ \llbracket \Gamma \rrbracket_\xi, x : B \vdash x \in \llbracket B \rrbracket_\xi : s^{n+1}\end{matrix} \quad \begin{matrix}\vdots |\Gamma \vdash C : s^m| \\ \llbracket \Gamma \rrbracket_\xi \vdash C : s^m\end{matrix}}{\llbracket \Gamma \rrbracket_\xi, y : C, x : B \vdash x \in \llbracket B \rrbracket_\xi : s^{n+1}}{\scriptstyle\text{THINNING}}}{\llbracket \Gamma, y : C \rrbracket_\xi, x : B \vdash x \in \llbracket B \rrbracket_\xi : s^{n+1}}{\scriptstyle\text{DEF}}$$

-$y \in \xi$

$$\dfrac{\dfrac{\begin{matrix}\vdots \{\Gamma \vdash B : s^n\}_\xi \\ \llbracket \Gamma \rrbracket_\xi, x : B \vdash \llbracket A \rrbracket_\xi : A \in \llbracket B \rrbracket_\xi\end{matrix} \quad \begin{matrix}\vdots |\Gamma \vdash C : s^m| \\ \llbracket \Gamma \rrbracket_\xi \vdash C : s^m\end{matrix} \quad \begin{matrix}\vdots \{\Gamma \vdash C : s^m\}_\xi \\ \llbracket \Gamma \rrbracket_\xi, y : C \vdash y \in \llbracket C \rrbracket_\xi : s^{m+1}\end{matrix}}{\llbracket \Gamma \rrbracket_\xi, y : C, \dot{y} : y \in \llbracket C \rrbracket_\xi, x : B \vdash x \in \llbracket B \rrbracket_\xi : s^{n+1}}{\scriptstyle\text{THINNING}}}{\llbracket \Gamma, y : C \rrbracket_\xi, x : B \vdash x \in \llbracket B \rrbracket_\xi : s^{n+1}}{\scriptstyle\text{DEF}}$$

●REL-ELIM

$$\dfrac{\Gamma \vdash F : \check{A} \overset{\bullet}{\to} s^n \quad \Gamma \vdash \check{a} : \check{A}}{\Gamma \vdash F \bullet \check{a} : s^n}\text{Rel-E}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\begin{matrix}\vdots \llbracket \Gamma \vdash F : \check{A} \overset{\bullet}{\to} s^n \rrbracket_\xi \\ \llbracket \Gamma \rrbracket_\xi \vdash \llbracket F \rrbracket_\xi : F \in \llbracket \check{A} \overset{\bullet}{\to} s^n \rrbracket_\xi\end{matrix} \quad \begin{matrix}\vdots |\Gamma \vdash F \bullet \check{a} : s^n| \\ \llbracket \Gamma \rrbracket_\xi \vdash F \bullet \check{a} : s^n\end{matrix}}{\llbracket \Gamma \rrbracket_\xi, z_0 : F \bullet \check{a} \vdash \llbracket F \rrbracket_\xi : F \in \llbracket \check{A} \overset{\bullet}{\to} s^n \rrbracket_\xi}{\scriptstyle\text{WK}}}{\llbracket \Gamma \rrbracket_\xi, z_0 : F \bullet \check{a} \vdash \llbracket F \rrbracket_\xi : (\llbracket \check{A} \rrbracket_\xi \oplus F) \overset{\bullet}{\to} s^{n+1}}{\scriptstyle\text{DEF}} \quad \dfrac{\begin{matrix}\wr \\ \wr \\ \llbracket \Gamma \rrbracket_\xi, z_0 : F \bullet \check{a} \vdash \llbracket \check{a} : \check{A} \rrbracket_\xi\end{matrix} \quad \dfrac{\dfrac{\begin{matrix}\vdots |\Gamma \vdash F \bullet \check{a} : s^n| \\ \llbracket \Gamma \rrbracket_\xi, z_0 \vdash F \bullet \check{a} : s^n\end{matrix}}{\llbracket \Gamma \rrbracket_\xi, z_0 : F \bullet \check{a} \vdash z_0 : F \bullet \check{a}}{\scriptstyle\text{ST}}}{\llbracket \Gamma \rrbracket_\xi, z_0 : F \bullet \check{a} \vdash z_0 : F \bullet (\llbracket \check{a} \rrbracket_\xi /\!\!/ 01 ... 1)}{\scriptstyle\text{DEF}}}{\llbracket \Gamma \rrbracket_\xi, z_0 : F \bullet \check{a} \vdash (\llbracket \check{a} \rrbracket_\xi \oplus z_0) : (\llbracket \check{A} \rrbracket_\xi \oplus F)}{\scriptstyle\text{DEF}}}{\llbracket \Gamma \rrbracket_\xi, z_0 : F \bullet \check{a} \vdash \llbracket F \rrbracket_\xi \bullet (\llbracket \check{a} \rrbracket_\xi \oplus z_0) : s^{n+1}}\text{Rel-E}}{\llbracket \Gamma \rrbracket_\xi, z_0 : F \bullet \check{a} \vdash z_0 \in \llbracket F \bullet \check{a} \rrbracket_\xi : s^{n+1}}{\scriptstyle\text{DEF}}$$

●REL-INTRO Absurd: the type is a relation ($\check{A} \overset{\bullet}{\to} s^n$), which cannot be a sort.

●Rel-Form

$$\dfrac{\Gamma \vdash \check{A} : s_1^n}{\Gamma \vdash (\check{A} \overset{\bullet}{\to} s_1^n) : s_2^n}\,\text{Rel-F}$$

$$\dfrac{\dfrac{\dfrac{\begin{array}{c}\wr\\ \llbracket\Gamma\rrbracket_\xi, z_0 : (\check{A} \overset{\bullet}{\to} s_1^n) \vdash \llbracket \check{A} : s_1^n\rrbracket_\xi\end{array} \quad \begin{array}{c}\wr\\ \llbracket\Gamma\rrbracket_\xi, z_0 : (\check{A} \overset{\bullet}{\to} s_1^n) \vdash z_0 : \check{A} \overset{\bullet}{\to} s_1^n\end{array}}{\llbracket\Gamma\rrbracket_\xi, z_0 : (\check{A} \overset{\bullet}{\to} s_1^n) \vdash (\llbracket\check{A}\rrbracket_\xi \oplus z_0) : s_1^{n+1}}\,\text{DEF}}{\llbracket\Gamma\rrbracket_\xi, z_0 : (\check{A} \overset{\bullet}{\to} s_1^n) \vdash (\llbracket\check{A}\rrbracket_\xi \oplus z_0) \overset{\bullet}{\to} s_1^{n+1} : s_2^{n+1}}\,\text{Rel-F}}{\llbracket\Gamma\rrbracket_\xi, z_0 : (\check{A} \overset{\bullet}{\to} s_1^n) \vdash z_0 \in \llbracket\check{A} \overset{\bullet}{\to} s_1^n\rrbracket_\xi : s_2^{n+1}}\,\text{DEF}$$

●Application

$$\dfrac{\dfrac{\Gamma \vdash \bar{A} : s_1^m}{\Gamma \vdash F : \bar{A} \to s^n}\,\text{gen} \quad \Gamma \vdash \bar{a} : \bar{A}}{\Gamma \vdash F\,\bar{a} : s^n}\,\text{APP}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\begin{array}{c}\vdots\;\llbracket\Gamma \vdash F : \bar{A} \to s^n : s^{n+1}\rrbracket_\xi\\ \llbracket\Gamma\rrbracket_\xi \vdash \llbracket F\rrbracket_\xi : F \in \llbracket\bar{A} \to s^n\rrbracket_\xi\end{array}}{\llbracket\Gamma\rrbracket_\xi \vdash \llbracket F\rrbracket_\xi : \bar{a} \in \llbracket\bar{A}\rrbracket_\xi \to F\,\bar{a} \in \llbracket s^n\rrbracket_\xi}\,\text{DEF} \quad \begin{array}{c}\vdots\;\llbracket\Gamma \vdash \bar{a} : \bar{A} : s^n\rrbracket_\xi\\ \llbracket\Gamma\rrbracket_\xi \vdash \llbracket\bar{a}\rrbracket_\xi : \bar{a} \in \llbracket\bar{A}\rrbracket_\xi\end{array}}{\dfrac{\llbracket\Gamma\rrbracket_\xi \vdash \llbracket F\rrbracket_\xi\,\llbracket\bar{a}\rrbracket_\xi : F\,\bar{a} \in \llbracket s^n\rrbracket_\xi}{\llbracket\Gamma\rrbracket_\xi \vdash \llbracket F\rrbracket_\xi\,\llbracket\bar{a}\rrbracket_\xi : \begin{pmatrix}F\,\bar{a}\\ .\end{pmatrix} \overset{\bullet}{\to} s^{n+1}}\,\text{DEF}}\,\text{APP} \quad \dfrac{\begin{array}{c}\vdots\;|\Gamma \vdash F\,\bar{a} : s^n|\\ \llbracket\Gamma\rrbracket_\xi \vdash F\,\bar{a} : s^n\end{array}}{\llbracket\Gamma\rrbracket_\xi, x : F\,\bar{a} \vdash x : F\,\bar{a}}\,\text{ST}}{\llbracket\Gamma\rrbracket_\xi, x : F\,\bar{a} \vdash \llbracket F\rrbracket_\xi\,\llbracket\bar{a}\rrbracket_\xi \begin{pmatrix}x\\ .\end{pmatrix} : s^{n+1}}\,\text{Rel-E}}{\llbracket\Gamma\rrbracket_\xi, x : F\,\bar{a} \vdash x \in \llbracket F\,\bar{a}\rrbracket_\xi : s^{n+1}}\,\text{DEF}$$

●Abstraction (Absurd.)
●Product

$$\dfrac{\Gamma \vdash \bar{A} : s_1^m \quad \Gamma, x : \bar{A} \vdash B : s_2^n}{\Gamma \vdash (\forall \bar{x} : \bar{A}.\, B) : s_3^{m \sqcap n}}\,(s_1, s_2, s_3)$$

$$\dfrac{\dfrac{\begin{array}{c}\wr\\ \llbracket\Gamma\rrbracket_\xi, f : (\forall \bar{x} : \bar{A}.\, B) \vdash \llbracket\bar{A}\rrbracket_\xi : s_1^{1+m}\end{array} \quad \dfrac{\begin{array}{c}\wr\\ \llbracket\Gamma\rrbracket_\xi, f : (\forall \bar{x} : \bar{A}.\, B), x : \llbracket\bar{A}\rrbracket_\xi \vdash f\,x_0 : B\end{array} \quad \dfrac{\begin{array}{c}\vdots\;\{\Gamma, x : \bar{A} \vdash B : s_2^n\}_{\xi,x}\\ \llbracket\Gamma\rrbracket_\xi, x : \llbracket\bar{A}\rrbracket_\xi, z : B \vdash z \in \llbracket B\rrbracket_{\xi,x} : s_2^{1+n}\end{array}}{\llbracket\Gamma\rrbracket_\xi, f : (\forall \bar{x} : \bar{A}.\, B), x : \llbracket\bar{A}\rrbracket_\xi, z : B \vdash z \in \llbracket B\rrbracket_{\xi,x} : s_2^{1+n}}\,\text{WK}}{\llbracket\Gamma\rrbracket_\xi, f : (\forall \bar{x} : \bar{A}.\, B), x : \llbracket\bar{A}\rrbracket_\xi \vdash (f\,x_0) \in \llbracket B\rrbracket_{\xi,x} : s_2^{1+n}}\,\text{subst}}{\llbracket\Gamma\rrbracket_\xi, f : (\forall \bar{x} : \bar{A}.\, B) \vdash (\forall \bar{x} : \llbracket\bar{A}\rrbracket_\xi.\, (f\,x_0) \in \llbracket B\rrbracket_{\xi,x}) : s_3^{1+m \sqcap n}}\,(s_1, s_2, s_3)}{\llbracket\Gamma\rrbracket_\xi, f : (\forall \bar{x} : \bar{A}.\, B) \vdash f \in \llbracket\forall \bar{x} : \bar{A}.\, B\rrbracket_\xi : s_3^{1+m \sqcap n}}\,\text{DEF}$$

●Conversion

$$\dfrac{\Gamma \vdash B : s^n \quad \Gamma \vdash s^n : s^{n+1} \quad s^n =_\beta s^n}{\Gamma \vdash B : s^n}\,\text{CONV}$$

Trivial.
●Param Absurd: the type of the parametricity witness is $z \in \llbracket B\rrbracket_\xi$, which cannot be a sort s^n.
●Exchange Trivial.
●Param Impossible.