Towards a computational interpretation of parametricity

Jean-Philippe Bernardy and Guilhem Moulin

Chalmers University of Technology and University of Gothenburg
{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 the Calculus of Constructions with a special parametricity rule (with computational content), and prove fundamental properties such as Church-Rosser's and strong normalization. The instances of the abstraction theorem can be both expressed and proved in the calculus itself.

1 Introduction

In his seminal paper, Reynolds [24] showed that types can be interpreted as predicates, in such a way that all inhabitants of a given type satisfy its interpretation as a predicate. A simple example is that if f has type $\forall a : \mathsf{type}.\, a \to a$ — the type of the polymorphic identity — then the following proposition holds

$$\forall a : \mathsf{type}.\, \forall \mathring{a} : a \to \mathsf{type}.\, \forall x : a.\, \mathring{a}\, x \to \mathring{a}\, (f\, a\, x)$$

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

The above result, *abstraction*, has proved useful for reasoning about functional programs [29, 11, 27][1]. It also has deep theoretical implications: for example, the correctness of the so-called Church encoding of data types [7, 30] depends on it.

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 [14] pioneered a different angle of study, of more syntactical nature: for each concrete term, a proof that it satisfies the relational interpretation of its type is constructed. That style has then been used by various authors, including Abadi et al. [1], Plotkin and Abadi [22] and Wadler [30]. Bernardy et al. [6] have also shown how terms, types, their relational interpretation as proofs and propositions can all be expressed (but *not* proved) in a single calculus. The calculi where this is possible must however be rich enough: in particular they must support dependent types. Many systems turn out to be suitable: the Calculus of Constructions [9] or Martin-Löf's Intuitionistic Type Theory [15] both satisfy the requirements.

[1] Even though the abstraction theorem as stated by Reynolds' is valid only for idealized languages (which for example are normalizing).

Still, even though we know that all terms 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 $\forall a : \mathsf{type}.\, a \to a$, namely

$$\forall f : (\forall a : \mathsf{type}.\, a \to a).\, \forall a : \mathsf{type}.\, \forall \mathring{a} : a \to \mathsf{type}.\, \forall x : a.\, \mathring{a}\, x \to \mathring{a}\, (f\, a\, x),$$

is not provable in type-theory.

In proof assistants based on type theory, one would like to rely on parametricity conditions to prove certain theorems. Indeed, the correctness of numerous functional programming techniques relies on parametricity. Examples include program transformation [11, 13], semantic program inversion [27] and generic programming [28]. 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 this need: 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 [23] describes the following representation for terms, using the Agda [19] proof assistant:

$$\begin{array}{l}\textbf{data}\ \mathsf{Term}\,(V : \mathsf{Set}) : \mathsf{Set}\ \textbf{where}\\ \quad \mathsf{var}\ : V \to \mathsf{Term}\, V\\ \quad \mathsf{app} : \mathsf{Term}\, V \to \mathsf{Term}\, V \to \mathsf{Term}\, V\\ \quad \mathsf{abs}\ : (\mathsf{Maybe}\, V \to \mathsf{Term}\, V) \to \mathsf{Term}\, V\end{array}$$

One can argue that terms defined as above must be well-formed 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. [6]. Only the parametricity axiom is missing to establish that all terms are well-formed. Chlipala [8] and Atkey et al. [3, p. 41] encounter the same type of situation.

In this paper, we aim to tackle the lack of support for parametricity in proof assistants. Technically, we propose to extend the Generalized Calculus of Constructions (CC_ω) to make all the parametricity 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 [26]. 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 CC_ω, we must provide a computation rule for parametricity.

Our technical contributions are as follows:

– We describe a dependently-typed λ-calculus with internalized parametricity. The calculus is summarized in definitions 1, 2, and 3; and motivated in Section 2. In particular, we show that the calculus internalizes parametricity.

– We prove that it satisfies the properties of a well-behaved λ-calculus with types. In particular, we prove the Church-Rosser property, subject reduction, and strong normalization.

2 Design, step by step

In this section we describe and motivate our design step by step, starting from the generalized calculus of constructions (CC_ω). The full and final definition of the system we will arrive at is described in definitions 1, 2, and 3.

2.1 CC_ω, and our notation

Our starting point is CC_ω [18], but familiarity with the calculus of constructions [9] is all we assume: we use CC_ω only because it is technically more convenient to have a type for each sort in the system. The presentation is in the style of pure type systems (PTS). Readers not familiar with PTSs are referred to Barendregt [4], but we give a brief reminder in the following paragraphs, as well as introduce our notation.

The system CC_ω features an infinite hierarchy of sorts, written here $\mathcal{S} = \{*, \Box_0, \Box_1, \dots\}$. The first sort, written $*$, is called the sort of propositions. Each sorts inhabits the its successor in the above list. Propositions are impredicative.

The various forms of quantifications, (and corresponding abstraction and application) are syntactically unified, and in general one needs to inspect sorts to identify which form is meant. In this paper, we sometimes need to have special treatment of quantifications over propositions (types of sort $*$). To support this, we tag the variables (resp. applications) with their sort (resp. the argument of the application). The concrete syntax is given in Definition 1. Sort annotation with $*$ are sometimes notationally inconvenient, in that case we use the following special syntax. Concretely, we have $A \overset{\bullet}{\to} B \triangleq \forall x^*\colon A.\, B$ when x does not appear in B, and $F \bullet A \triangleq F^* A$.

We also restrict our raw syntax to *well-sorted* terms: $(\lambda x^{\Box_0}.\, t)^* u$ and $x^{\Box_0}[x^* \mapsto u]$ are for instance ill-sorted, but $(\lambda x^{\Box_0}.\, t)^{\Box_0} u$ and $x^*[x^* \mapsto u]$ are not. We omit the annotations when they do not play any role, or can be inferred *e.g.*, for well typed terms.

The full syntax and typing rules is given in the following definitions. The remainder of the section explains our special construction for parametricity ($[\![\cdot]\!]$) and its typing rule, as well as the technical device of relational substitutions.

Definition 1 (Syntax).

Var	$\ni x, y, z, \mathring{x}, \mathring{y}, \mathring{z}$			
Sort	$\ni s$	$::= *$	$\mid \Box_0 \mid \Box_1 \mid \ldots$	
Term	$\ni a, b, f, t, u, A, B, C, \ldots$	$::=$	s	*sort*
		$\mid$	x^s	*variable*
		$\mid$	$[\![x^{\Box_0}]\!]$	*parametric witness*
		$\mid$	$A^s B$	*application*
		$\mid$	$\lambda x^s : A.\, B$	*abstraction*
		$\mid$	$\forall x^s : A.\, B$	*function space*
Context	$\ni \Gamma, \Delta$	$::=$	$-$	*empty context*
		$\mid$	$\Gamma, x^s : A$	*context extension*
relational substitution	$\ni \xi$	$::=$	$\varnothing$	*empty*
		$\mid$	$\xi, x^{\Box_0} \mapsto \mathring{x}$	*extension*

Definition 2 (Typing rules).

$$\frac{}{\vdash s_1 : s_2}\ (s_1, s_2) \in \mathcal{A} \qquad \textsc{Axiom}$$

$$\frac{\Gamma \vdash A : B \qquad \Gamma \vdash C : s}{\Gamma, x^s : C \vdash A : B} \qquad \textsc{Weakening}$$

$$\frac{\Gamma \vdash F : (\forall x^s : A.\, B) \qquad \Gamma \vdash a : A}{\Gamma \vdash F^s a : B[x^s \mapsto a]} \qquad \textsc{Application}$$

$$\frac{\Gamma, x^s : A \vdash b : B \qquad \Gamma \vdash (\forall x^s : A.\, B) : s'}{\Gamma \vdash (\lambda x^s : A.\, b) : (\forall x^s : A.\, B)} \qquad \textsc{Abstraction}$$

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

$$\frac{\Gamma \vdash A : B \qquad \Gamma \vdash B' : s \qquad B =_\beta B'}{\Gamma \vdash A : B'} \qquad \textsc{Conversion}$$

$$\frac{\Gamma \vdash A : s}{\Gamma, x^s : A \vdash x : A} \qquad \textsc{Start}$$

$$\frac{\Gamma \vdash A : \Box_0}{\Gamma, x^{\Box_0} : A \vdash [\![x]\!] : x \in [\![A]\!]_\varnothing} \qquad \textsc{Param}$$

with
$$\mathcal{A} = \{(*, \Box_0), (\Box_i, \Box_{1+i}) \mid i \in \mathbb{N}\}$$
$$\mathcal{R} = \{(*, *, *), (\Box_i, *, *), (*, \Box_i, \Box_i), (\Box_i, \Box_j, \Box_{i \sqcup j}) \mid i, j \in \mathbb{N}\}$$

2.2 Logical relations in CC_ω

In this section, we define a relational interpretation of terms and types of CC_ω. The material of this section is directly inspired from results of Bernardy et al. [6] and Bernardy and Lasson [5], which describe how to derive logical relations for any PTS [4]. In the following sections, we will show how it can be amended to suit our purposes.

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 [5] for details.

Likewise, extending the theory to support inductive types is straightforward and detailed by Bernardy et al. [6].

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

$$[\![-]\!] = -$$
$$[\![\Gamma, x : A]\!] = [\![\Gamma]\!], x : A, \mathring{x} : x \in [\![A]\!]$$

In fact, we will need to be explicit about the renaming relation existing between variables and their witness of parametricity, to ensure that the translations are well scoped[2]. (From now on we call such renamings *relational substitutions.*) Hence we subscript the translation with it:

$$[\![-]\!]_\xi = -$$
$$[\![\Gamma, x : A]\!]_{\xi, x \mapsto \mathring{x}} = [\![\Gamma]\!]_\xi, x : A, \mathring{x} : x \in [\![A]\!]_\xi$$

Until the next section, we make the assumption that a relational substitution contains information about all free variables in the term or context it is applied to.

The relational interpretation of types and its underlying intuition follow. Recall that, to interpret a type T, it suffices to give a proposition $C \in [\![T]\!]$ stating that C satisfies the interpretation of the type.

- 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.

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

- 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.

$$C \in [\![\forall x : A.\, B]\!]_\xi = \forall x : A.\, \forall \mathring{x} : x \in [\![A]\!]_\xi.\, (C\, x) \in [\![B]\!]_{\xi, x \mapsto \mathring{x}}$$

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

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

Finally we can give the translation from terms to proofs.

[2] If one were to always interpret x by $\mathring{x}$, the term $[\![[\![\lambda x.a]\!]]\!]$ would bind $\mathring{x}$ twice.

– The translation of a variable is done by looking up the corresponding parametric witness in the context.

$$[\![x]\!]_\xi = \xi(x)$$

– The case for abstraction adds a witness that the input satisfies the relational interpretation of its type and returning the relational interpretation of the body, mirroring the interpretation of product:

$$[\![\lambda x : A.\, B]\!]_\xi = \lambda x : A.\, \lambda \mathring{x} : x \in [\![A]\!]_\xi.\, [\![B]\!]_{\xi, x \mapsto \mathring{x}}$$

– The application follows the same pattern: the function is passed a witness that the argument satisfies the interpretation of its type.

$$[\![A\, B]\!]_\xi = [\![A]\!]_\xi\, B\, [\![B]\!]_\xi$$

– 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 ($z \in [\![T]\!]_\xi$).

$$[\![T]\!]_\xi = \lambda z : T.\, z \in [\![T]\!]_\xi$$

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 $[\![\cdot]\!]$.

Theorem 1 (Abstraction). *If* $\Gamma \vdash_{\mathrm{CC}_\omega} A : B : s$*, then*

$$[\![\Gamma]\!]_\xi \vdash_{\mathrm{CC}_\omega} [\![A]\!]_\xi : (A \in [\![B]\!]_\xi) : s$$

where ξ *maps every variable in* Γ *to a globally fresh variable.*

Proof. The theorem is proved by induction on the derivation tree, similarly to [5]. Since we will revise the definition of $[\![\cdot]\!]$, we omit all details of the proof here.

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

2.3 Internalization

Both the source and target of $[\![\cdot]\!]$ are in the same system. Both the theorems and the proofs are expressible in that system. Therefore we can hope that for every term A of type B, the user of the system can get a witness $[\![A]\!]_\xi$ that it is parametric ($A \in [\![B]\!]_\xi$). Even though this hope is fulfilled for closed terms, we

run out of luck for open terms, because the context where $[\![A]\!]_\xi$ is meaningful is "bigger" than that where A is: for each free variable $x : A$ in Γ, we need a variable $\mathring{x} : x \in [\![A]\!]_\xi$ in $[\![\Gamma]\!]_\xi$.

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 $[\![a]\!]_\varnothing$ to substitute for $\mathring{x}$. 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 syntactic form $\lfloor\!\lfloor x \rfloor\!\rfloor$. This intuition justifies the addition the substitution rule

$$\lfloor\!\lfloor x \rfloor\!\rfloor [x \mapsto a] = [\![a]\!]_\varnothing$$

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

$$\frac{\Gamma \vdash A : s}{\Gamma, x : A \vdash \lfloor\!\lfloor x \rfloor\!\rfloor : x \in [\![A]\!]_\varnothing}$$

Using the above, we can now lift the restriction that relational substitutions must give a mapping for all free variables and define[3]:

$$\begin{aligned} [\![x]\!]_\xi &= \xi(x) && \text{if } x \in \xi \\ [\![x]\!]_\xi &= \lfloor\!\lfloor x \rfloor\!\rfloor && \text{if } x \notin \xi \end{aligned}$$

Conversely, contexts are not extended if the variable is not present in the mapping.

$$\begin{aligned} [\![-]\!]_\xi &= - \\ [\![\Gamma, x : A]\!]_{\xi, x \mapsto \mathring{x}} &= [\![\Gamma]\!]_\xi, x : A, \mathring{x} : x \in [\![A]\!]_\xi \\ [\![\Gamma, x : A]\!]_\xi &= [\![\Gamma]\!]_\xi, x : A && \text{if } x \notin \xi \end{aligned}$$

(From here on, we may omit the relational substitution argument to $[\![A]\!]$ to mean $[\![A]\!]_\varnothing$.)

The above construction is the only addition required to obtain full internalized parametricity. That is, assuming parametricity on variables, we have parametricity for all terms, as stated in the Theorem 2.

Theorem 2 (Parametricity).

$$\Gamma \vdash A : B \Rightarrow \Gamma \vdash [\![A]\!] : A \in [\![B]\!]$$

Proof. By induction on the derivation tree. The difference with the proof of the abstraction theorem is essentially that occurrences of Start for relevant variables are transformed to Param. (The definition of $[\![\cdot]\!]$ will be revised later, as well as this proof.)

[3] Careful readers might worry that we forget the substitution in the second case. An informal justification is that if x has no substitute in ξ, then the variables of its type do not either. Therefore, types are preserved when doing the substitution.

The fact that all values are parametric is also captured by the following, which is a theorem (an inhabited type inside the calculus):

$$\begin{aligned} &\mathit{parametricity} : \forall A : \mathsf{type}.\, \forall (a : A).\, a \in [\![A]\!] \\ &\mathit{parametricity} = \lambda A.\, \lambda a : A.\, [\![a]\!] \end{aligned}$$

Example 1. We can prove that any function of type $\forall a : \square_0.\, a \to a$ is an identity, as we hinted at in the introduction. The formulation of the theorem and its proof term are as follows.

$$\begin{aligned} &\mathit{identities} : \forall f : (\forall a : \square_0.\, a \to a).\, \forall a : \square_0.\, \forall x : a.\, \mathit{Eq}\, a\, (f\ a\ x)\, x \\ &\mathit{identities} = \lambda f.\lambda a.\lambda x.[\![f]\!]\, a\, (\mathit{Eq}\, a\, x)\, x\, (\mathit{refl}\, a\, x) \end{aligned}$$

When *identities* is used on a concrete identity function, say $i = \lambda a : \square_0.\, \lambda x : a.\, x$, the theorem specializes to reflexivity of equality:

$$\mathit{identities}\, i \ :\ \forall a : \square_0.\, \forall x : a.\, \mathit{Eq}\, a\, x\, x$$

and, using our extended definition of the substitution, after reduction the proof no longer mentions $[\![\cdot]\!]$:

$$\begin{aligned} \mathit{identities}\, i &\to_\beta \lambda a.\lambda x.[\![f]\!][f \mapsto \lambda a : \square_0.\, \lambda x : a.\, x]\, a\, (\mathit{Eq}\, a\, x)\, x\, (\mathit{refl}\, a\, x) \\ &= \lambda a.\lambda x.[\![\lambda a : \square_0.\, \lambda x : a.\, x]\!]\, a\, (\mathit{Eq}\, a\, x)\, x\, (\mathit{refl}\, a\, x) \\ &= \lambda a.\lambda x.(\lambda a.\lambda \mathring{a}.\lambda x.\lambda \mathring{x}.\mathring{x})\, a\, (\mathit{Eq}\, a\, x)\, x\, (\mathit{refl}\, a\, x) \\ &\to_\beta \lambda a.\lambda x.\mathit{refl}\, a\, x \end{aligned}$$

2.4 Nesting of $[\![\cdot]\!]$ and computational irrelevance

In this section we describe an issue with subject reduction in the system explained so far, and we tweak the system to fix the issue, by taking advantage of computational irrelevance in CC_ω.

Nesting $[\![\cdot]\!]$ and subject reduction For some closed type A, consider the term $[\![\mathit{parametricity}\, A]\!]$, which intuitively is a proof that parametricity *itself* satisfies the abstraction theorem. It is convertible to $\lambda a : A.\, \lambda \mathring{a} : a \in [\![A]\!].\, [\![[\![a]\!]]\!]_{\{x \mapsto \mathring{x}\}}$.

So far, we have not specified how to reduce the subterm $[\![[\![x]\!]]\!]_{\{x \mapsto \mathring{x}\}}$ (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 in a relational substitution, and only variables can appear there (not arbitrary terms). This means that $[\![[\![x]\!]]\!]_{\{x \mapsto \mathring{x}\}}$ is not an acceptable normal form: either it must reduce to something else, or it is should not be well-typed.

We first explore the possibility of reducing the term. A perhaps natural idea would be to modify the reduction rules in such a way to allow the following reduction, swapping the two occurrences of the parametric interpretation:

$$[\![[\![x]\!]]\!]_{\{x \mapsto \mathring{x}\}} \longrightarrow [\![[\![x]\!]_{\{x \mapsto \mathring{x}\}}]\!].$$

In that case the expression would further reduce to $[\![\mathring{x}]\!]$, which is a proper normal form. Unfortunately, allowing this reduction would *break subject reduction*. This can be checked by assuming $x : A$ (where A is any closed type, its actual form is irrelevant), and computing the types of the expression before and after reduction. By Theorem 2 we have $[\![x]\!] : [\![A]\!]\, x$. By a second application of Theorem 2, we get

$$\begin{aligned}[\![[\![x]\!]\!]_{\{x\mapsto\mathring{x}\}} &: [\![[\![A]\!]\, x]\!]_{\{x\mapsto\mathring{x}\}}\, [\![x]\!] \\ &: [\![[\![A]\!]]\!]_{\{x\mapsto\mathring{x}\}}\, x\, [\![x]\!]_{\{x\mapsto\mathring{x}\}}\, [\![x]\!] \\ &: [\![[\![A]\!]]\!]\, x\, [\![x]\!]_{\{x\mapsto\mathring{x}\}}\ [\![x]\!] \\ &: [\![[\![A]\!]]\!]\, x\, \mathring{x}\ [\![x]\!]\end{aligned}$$

On the other hand, $[\![x]\!]_{\{x\mapsto\mathring{x}\}} : [\![A]\!]\, x$, and by a another application of Theorem 2, we get

$$\begin{aligned}[\![[\![x]\!]_{\{x\mapsto\mathring{x}\}}]\!] &: [\![[\![A]\!]\, x]\!]\, [\![x]\!]_{\{x\mapsto\mathring{x}\}} \\ &: [\![[\![A]\!]]\!]\, x\, [\![x]\!]\, [\![x]\!]_{\{x\mapsto\mathring{x}\}} \\ &: [\![[\![A]\!]]\!]\, x\, [\![x]\!]\, \mathring{x}\end{aligned}$$

That is, in the above example, the reduction rule suggested above has the effect to swap the second and third arguments to $[\![[\![A]\!]]\!]$ in the type.

We are then left with the alternative to *prevent* the above situation from occurring, making the above term ill-typed. The solution we choose is to forbid all nested uses of $[\![\cdot]\!]$. Concretely, we make a twofold change to the system:

1. we make sure that all parametricity conditions inhabit the sort $*$, and
2. we prevent parametricity to be used at sort $*$.

One can easily check that the above is an invariant of our parametric interpretation (Definition 3). For instance, we ensure that explicit witnesses $\mathring{x}$ lie in sort $*$ for x of sort $\Box_0$.

A brief review of computational irrelevance Essentially, the change we propose in the above section takes advantage of a form of computational irrelevance. This notion is found in various flavors in the literature, but the version we use here is most similar to [20]. As Paulin-Mohring, we distinguish between informative (inhabiting $\Box_i$) and non-informative (inhabiting $*$) propositions: non-informative fragments cannot influence computation of informative ones, rather, they act as mere witnesses.

The above property of separation between $\Box_i$ and $*$ is already ensured by the structure of CC_ω: by invoking a proof $a : A : *$, it is impossible to gain any information about its structure, beyond its mere existence. If inductive constructions were to be added to the system, as in the calculus of inductive constructions [31], informative elimination from $*$ to $\Box_i$ would be forbidden, in order to avoid avoid logical inconsistency. (We discuss the issue further in Section 3.1.)

Amending the relational interpretation Even though the relational interpretation of terms as described in Section 2.2 works to represent terms of CC_ω inside CC_ω, it does not capture the irrelevance of $*$. To do so, we can proceed to change it as follows. (The end result is shown in Definition 3.) Because any function F of type $\forall x^*\colon A.\, B$ is guaranteed not to depend computationally on its argument, the results of the function are related regardless of whether the inputs are related. Therefore, the translation of $\forall x : A.\, B$ can be changed to reflect this observation: when the sort of A is $*$, we merely drop the parametricity witness $\mathring{x}$.

$$\begin{array}{ll} C \in [\![\forall x^{\Box_i} : A.\, B]\!]_\xi = \forall x^{\Box_i} : A.\, \forall \mathring{x}^*\colon x \in [\![A]\!]_\xi.\ C\, x \in [\![B]\!]_{\xi, x \mapsto \mathring{x}} \\ C \in [\![\forall x^*\colon A.\, B]\!]_\xi \quad = \forall x^*\colon A. & C\, x \in [\![B]\!]_\xi \end{array}$$

The translation of abstraction, application and context formation must then be adapted accordingly (equations (5), (6) and (14) in Definition 3), in each case by removing the witness $\mathring{x}$. (Dropping the witness from the translation roughly corresponds to having that all proofs are related: $C \in [\![A]\!]_\xi = \top$, where $\top$ represents truth.)

After this change, the relational interpretation never "reaches" a type of sort $*$ (nor an inhabitant of such a type), given that one starts with a type of sort $\Box_i$ (or an inhabitant of such a type). Therefore, we have now more freedom in the choice of the relational interpretation of $\Box_i$. As hinted in the previous section, we choose types of sort $*$:

$$C \in [\![\Box_i]\!]_\xi = C \to *$$

However, having the single sort $*$ for the relational interpretation of the whole hierarchy $\Box_i$ leads to a technical issue (we discuss it in Section 3.2), hence we choose to only allow the relational interpretation for types of sort $\Box_0$ (or inhabitants of such types).

The abstraction theorem remains valid with the above modifications, with no essential change to the proof (see Section 4.2 for details).

Even though we forbid direct nesting of $[\![\cdot]\!]$, indirect nesting is still possible. For instance, parametricity can be used to prove a proposition $a \in [\![A]\!]$ that a program F relies on. That is, if $F : (a \in [\![A]\!]) \to B$ with $a : A : \Box_0$, then $[\![F\, [\![a]\!]]\!]$ is acceptable. Indeed, because the relational interpretation simply ignores proof arguments (see Definition 3), the above expression reduces to $[\![F]\!]\, [\![a]\!]$, and the nesting disappears.

Definition 3 (Relational interpretation).

$$[\![x]\!]_\xi = \mathring{x} \qquad \textit{if } x \in \xi \qquad (1)$$

$$[\![x]\!]_\xi = [\![x]\!] \qquad \textit{otherwise} \qquad (2)$$

$$[\![\lambda x^{\Box_0}\colon A.\, B]\!]_\xi = \lambda x^{\Box_0}\colon A.\, \lambda \mathring{x}^*\colon x \in [\![A]\!]_\xi.\, [\![B]\!]_{\xi, x \mapsto \mathring{x}} \qquad (x \notin \xi) \qquad (3)$$

$$[\![A\, B]\!]_\xi = [\![A]\!]_\xi\, B \bullet [\![B]\!]_\xi \qquad (4)$$

$$[\![\lambda x^*\colon A.\, B]\!]_\xi = \lambda x^*\colon A.\, [\![B]\!]_\xi \qquad (x \notin \xi) \qquad (5)$$

$$[\![A \bullet B]\!]_\xi = [\![A]\!]_\xi \bullet B \qquad (6)$$

$$[\![T]\!]_\xi = \lambda z^{\Box_0}\colon T.\, z \in [\![T]\!]_\xi \qquad \textit{if } T \textit{ is } \forall \textit{ or } \Box_0 \qquad (7)$$

$$C \in [\![\Box_0]\!]_\xi = C \to *$$ (8)

$$C \in [\![\forall x^{\Box_0}\colon A.\, B]\!]_\xi = \forall x^{\Box_0}\colon A.\, \forall \mathring{x}^*\colon x \in [\![A]\!]_\xi.\, (C\,x) \in [\![B]\!]_{\xi, x \mapsto \mathring{x}} \quad (x \notin \xi)$$ (9)

$$C \in [\![\forall x^*\colon A.\, B]\!]_\xi = \forall x^*\colon A.\, (C \bullet x) \in [\![B]\!]_\xi \quad (x \notin \xi)$$ (10)

$$C \in [\![T]\!]_\xi = [\![T]\!]_\xi\, C \quad \textit{if } T \textit{ is not } \forall \textit{ nor a sort}$$ (11)

$$[\![-]\!]_\xi = -$$ (12)

$$[\![\Gamma, x : A]\!]_{\xi, x \mapsto \mathring{x}} = [\![\Gamma]\!]_\xi, x^{\Box_0}\colon A, \mathring{x}^*\colon x \in [\![A]\!]_\xi \quad \textit{if } A : \Box_0$$ (13)

$$[\![\Gamma, x : A]\!]_{\xi, x \mapsto \mathring{x}} = [\![\Gamma]\!]_\xi, x^*\colon A \quad \textit{if } A : *$$ (14)

$$[\![\Gamma, x^s : A]\!]_\xi = [\![\Gamma]\!]_\xi, x^s : A \quad \textit{if } x \notin \xi$$ (15)

2.5 Summary

The changes made to relational interpretation do not compromise the abstraction theorem, nor parametricity. The proofs need only to be amended to ignore irrelevant variables. (Full proofs are found in the appendix.) In particular, Example 1 remains valid.

This concludes the description of the design of our calculus, and its motivation. In summary, we have added a parametricity rule, whose computational content is given by the systematic construction of parametricity witnesses from terms. Proofs that are irrelevant (in the sense of [20]) are always known to satisfy the relational interpretation of their type: this is built into the definition of the relational interpretation.

We have proved confluence, subject reduction, and strong normalization for our calculus. Since parametricity acts as a typing rule for $[\![\cdot]\!]$, and subject reduction for our calculus stems directly from it. Strong normalization is proved by modelling the system in CC_ω. This model is done by introducing explicit witnesses of parametricity for all relevant variables. Details of the proofs of these theorems and parametricity are delayed until the appendix.

3 Discussion

3.1 Extension: inductive types

Even though we considered only PTSs (without inductive constructions), it is straightforward to support them, provided usual precautions regarding computational relevance are respected. That is, if inductive constructions were to be added to the system, informative elimination from $*$ to $\Box_0$ would be forbidden. This is consistent with what happens in the Calculus of Inductive Constructions [31], and its implementation in the Coq system.

Forbidding this kind of elimination is necessary for our abstraction theorem to hold. Indeed, if elimination were permitted from $*$ to $\Box_0$, one could in particular lift proofs to programs. Showing that such a lifting satisfies the parametricity condition arising from its type would require a witness that the proof itself satisfies a similar condition, but no witness of this fact is available, given our interpretation of proofs.

However, systems with computational irrelevance are often extended with the possibility to transform an (irrelevant) proof of falsity ($\bot : *$) to any type $\alpha : \Box_0$.

$$\mathsf{botElim} : \bot \overset{\bullet}{\to} \forall \alpha : \Box_0 . \, \alpha$$

Such an elimination *is* compatible with our theory of parametricity. Indeed, we can give a relational interpretation of $\mathsf{botElim}$ as follows:

$$\begin{aligned} &[\![\mathsf{botElim}]\!] : \forall b : \bot . \forall \alpha : \Box_0 . \forall \mathring{\alpha} : \alpha \to * . \, \mathring{\alpha} \, (\mathsf{botElim}\bullet b \, \alpha) \\ &[\![\mathsf{botElim}]\!] = \lambda b . \mathsf{botElim}\bullet b \, (\forall \alpha : \Box_0 . \forall \mathring{\alpha} : \alpha \to * . \, \mathring{\alpha} \, (\mathsf{botElim}\bullet b \, \alpha)) \end{aligned}$$

The trick is that, even though there is no witness that b satisfies any relational interpretation, it is itself a proof of falsity, and therefore no further condition is needed to prove its parametricity (or indeed anything else).

3.2 Sort hierarchies

We have allowed parametricity to be only used at values of types inhabiting $\Box_0$. We already justified in detail why parametricity at sort $*$ is disallowed, but what about $\Box_1$, $\Box_2$, and so on? For the axiom $\vdash \Box_i : \Box_{i+1}$, the abstraction theorem says: $(\lambda z : \Box_i . z \to *) : \Box_i \to *$, which is only possible if $z : \Box_i \vdash z \to * : *$, which is wrong.

One way to allow parametricity at higher sorts would be to extend the division between relevant and irrelevant sorts. That is, have a parallel hierarchy of sorts for propositions $*_i : *_{i+1}$ with impredicative rules $(s, *_i, *_i)$. The parametricity conditions for types of $\Box_i$ would live in $*_i$. In that case, the abstraction theorem would yield $z : \Box_i \vdash z \to *_i : *_{i+1}$, which would hold by assumption.

3.3 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 [17] and [15].

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. To our knowledge this has not been attempted 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.

Inductive families In the first category, we must first mention inductive families, studied in different contexts by Pfenning and Paulin-Mohring [21], and Dybjer [10].

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, it is possible to see induction principles over data as a special case of parametricity, as shown for example by Wadler [30].

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. Many 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. Additionally, Altenkirch et al. also rely on a separation between proofs and programs, in a similar fashion to what we present here.

Meta-level reasoning Miller and Tiu [16] propose a logical framework where one can reason precisely about terms and types of an object language. Their domain of application clearly overlaps with ours. For example, one can prove in their framework that the object type $\forall a : \star.a$ is uninhabited. A characteristic of [16] is the total separation between object and host language. Here, we are able to unify both languages, even though we must keep some amount of semantic separation between proofs and programs: programs cannot depend on the structure of proofs.

Parametricity Plotkin and Abadi [22] 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 [12], 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 [25] attempted to extend CC with parametricity. Takeuti asserted parametricity at all types, with a similar definition of the parametric interpretation as ours. His system also seems to feature a notion of irrelevance: two types in $\Box_0$ appear to always be related. Takeuti does not attempt to assign a computational content to his parametricity axioms, and only conjectures consistency of his system.

3.4 Future Work

A natural extension of this work would be to integrate it in the Coq system. One could experiment and measure how much power does parametricity give on practical examples. The presentation we give here fits naturally with the existing Coq implementation, except for that fact that we allow parametricity only on the first level of the hierarchy of sorts. Lifting the limitation could be done as outlined in Section 3.2, but then it would not blend well with the current Coq type system.

Another limitation of this work is the impossibility to apply parametricity to proofs. That is, programs can depend on parametricity proofs, and these proofs have a computational meaning, but programs cannot depend on the structure of proofs (only their existence). The limitation does not appear to prevent use of parametricity to prove correctness of functional programs, but it prevents deeper use of parametricity, for example justifying impredicative Church-encodings of data (which necessarily live in the $*$ sort). Therefore, we think that lifting the restriction can bring fundamental benefits. For example, by nesting parametricity, one can generate parametricity of arbitrary arity [5]. Because implementing unary parametricity would then be enough, nesting has the possibility to simplify the system. In the process, we hope to clarify design decisions and reveal more fundamental properties of the relational interpretation.

Bibliography

[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 *Proc. of the 2007 workshop on Programming languages meets program verification*, pages 57–68. ACM, 2007.

[3] R. Atkey, S. Lindley, and J. Yallop. Unembedding domain-specific languages. In *Proc. of the 2nd ACM SIGPLAN symposium on Haskell*, 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 and M. Lasson. Realizability and parametricity in pure type systems. In M. Hofmann, editor, *Foundations Of Software Sci. And Computational Structures*, volume 6604 of *LNCS*, pages 108–122. Springer, 2011.

[6] J.-P. Bernardy, P. Jansson, and R. Paterson. Parametricity and dependent types. In *Proc. of the 15th ACM SIGPLAN international conference on Funct. programming*, pages 345–356. ACM, 2010.

[7] C. Böhm and A. Berarducci. Automatic synthesis of typed lambda-programs on term algebras. *Theor. Comp. Sci.*, 39(2-3):135–154, 1985.

[8] A. Chlipala. Parametric higher-order abstract syntax for mechanized semantics. In *Proceeding of the 13th ACM SIGPLAN international conference on Funct. programming*, ICFP '08, pages 143–156. ACM, 2008.

[9] T. Coquand and G. Huet. The calculus of constructions. Technical report, INRIA, 1986.
[10] P. Dybjer. Inductive families. *Formal Aspects of Computing*, 6(4):440–465, 1994.
[11] A. Gill, J. Launchbury, and S. Peyton Jones. A short cut to deforestation. In *Proc. of FPCA*, pages 223–232. ACM, 1993.
[12] R. Hasegawa. Categorical data types in parametric polymorphism. *Mathematical Structures in Comp. Sci.*, 4(01):71–109, 1994.
[13] P. Johann. A generalization of short-cut fusion and its correctness proof. *Higher-Order and Symbol. Comput.*, 15(4):273–300, 2002.
[14] H. Mairson. Outline of a proof theory of parametricity. In *Proc. of FPCA 1991*, volume 523 of *LNCS*, pages 313–327. Springer-Verlag, 1991.
[15] P. Martin-Löf. *Intuitionistic type theory.* Bibliopolis, 1984.
[16] D. A. Miller and A. F. Tiu. A proof theory for generic judgments: An extended abstract, 2003.
[17] R. Milner. Logic for Computable Functions: description of a machine implementation. *Artificial Intelligence*, 1972.
[18] A. Miquel. *Le Calcul des Constructions implicite: syntaxe et sémantique.* Thèse de doctorat, Université Paris 7, 2001.
[19] U. Norell. *Towards a practical programming language based on dependent type theory.* PhD thesis, Chalmers Tekniska Högskola, 2007.
[20] C. Paulin-Mohring. Extracting Fω's programs from proofs in the calculus of constructions. In *POPL'89*, pages 89–104. ACM, 1989.
[21] F. Pfenning and C. Paulin-Mohring. Inductively defined types in the calculus of constructions. In *Mathematical Foundations of Programming Semantics*, pages 209–228. 1990.
[22] G. Plotkin and M. Abadi. A logic for parametric polymorphism. In *Proc. of the International Conference on Typed Lambda Calculi and Applications*, volume 664 of *LNCS*, page 361–375. Springer, 1993.
[23] N. Pouillard. Nameless, painless. In *Proc. of the 16th ACM SIGPLAN international conference on Funct. programming*, ICFP '11. ACM, 2011. to appear.
[24] J. C. Reynolds. Types, abstraction and parametric polymorphism. *Information processing*, 83(1):513–523, 1983.
[25] I. Takeuti. The theory of parametricity in lambda cube. Manuscript, 2004.
[26] The Coq development team. The Coq proof assistant, 2011.
[27] J. Voigtländer. Bidirectionalization for free! (Pearl). In *Proc. of POPL 2009*, pages 165–176. ACM, 2009.
[28] D. Vytiniotis and S. Weirich. Parametricity, type equality, and higher-order polymorphism. *J. Funct. Program.*, 20(02):175–210, 2010.
[29] P. Wadler. Theorems for free! In *Proc. of FPCA 1989*, pages 347–359. ACM, 1989.
[30] P. Wadler. The Girard–Reynolds isomorphism. *Theor. Comp. Sci.*, 375(1–3): 201–226, 2007.
[31] B. Werner. *Une théorie des constructions inductives.* PhD thesis, Université de Paris 7, 1994.

4 Appendix

This section contains details of our proofs.

4.1 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 t, then $[\![t]\!]_{\xi,x\mapsto\mathring{x}} = [\![t]\!]_\xi$.*

Proof. Induction on the structure of t.

Lemma 2 ($[\![\cdot]\!]$ and substitution, part 1). *if $z^{\Box_0}$ is not in ξ, then*

$$[\![t[z^{\Box_0} \mapsto u]]\!]_\xi = [\![t]\!]_{\xi,z\mapsto\mathring{z}}[z^{\Box_0} \mapsto u][\mathring{z}^* \mapsto [\![u]\!]_\xi]$$

Proof. By induction on t. Only the variable case ($t = x$) will be explained here, as the other ones boil down to easy equational reasoning.

Sort trivial.
Abstraction (Product is similar)

$$\begin{aligned}
&[\![(\lambda x^{\Box_0}\colon A.\, b)[z^{\Box_0} \mapsto u]]\!]_\xi \\
&\quad = [\![\lambda x^{\Box_0}\colon A[z^{\Box_0} \mapsto u].\, b[z^{\Box_0} \mapsto u]]\!]_\xi \\
&\quad = \lambda x^{\Box_0}\colon A[z^{\Box_0} \mapsto u].\, \lambda\mathring{x}^*\colon x \in [\![A[z^{\Box_0} \mapsto u]]\!]_\xi.\, [\![b[z^{\Box_0} \mapsto u]]\!]_{\xi,x\mapsto\mathring{x}} \\
&\quad = \lambda x^{\Box_0}\colon A[z^{\Box_0} \mapsto u].\, \lambda\mathring{x}^*\colon x \in [\![A]\!]_{\xi,z\mapsto\mathring{z}}[z^{\Box_0} \mapsto u][\mathring{z}^* \mapsto [\![u]\!]_\xi]. \\
&\qquad\qquad [\![b]\!]_{\xi,z\mapsto\mathring{z},x\mapsto\mathring{x}}[z^{\Box_0} \mapsto u][\mathring{z}^* \mapsto [\![u]\!]_{\xi,x\mapsto\mathring{x}}] \\
&\quad = \lambda x^{\Box_0}\colon A[z^{\Box_0} \mapsto u].\, \lambda\mathring{x}^*\colon x \in [\![A]\!]_{\xi,z\mapsto\mathring{z}}[z^{\Box_0} \mapsto u][\mathring{z}^* \mapsto [\![u]\!]_\xi]. \\
&\qquad\qquad [\![b]\!]_{\xi,z\mapsto\mathring{z},x\mapsto\mathring{x}}[z^{\Box_0} \mapsto u][\mathring{z}^* \mapsto [\![u]\!]_\xi] \\
&\quad = (\lambda x^{\Box_0}\colon A.\, \lambda\mathring{x}^*\colon x \in [\![A]\!]_{\xi,z\mapsto\mathring{z}}.\, [\![b]\!]_{\xi,z\mapsto\mathring{z},x\mapsto\mathring{x}}) \\
&\qquad\qquad [u^{\Box_0} \mapsto z][\mathring{z}^* \mapsto [\![u]\!]_\xi] \\
&\quad = [\![\lambda x^{\Box_0}\colon A.\, b]\!]_{\xi,z\mapsto\mathring{z}}[z^{\Box_0} \mapsto u][\mathring{z}^* \mapsto [\![u]\!]_\xi]
\end{aligned}$$

Application

$$\begin{aligned}
&[\![(F\,a)[z^{\square_0} \mapsto u]]\!]_\xi \\
&\qquad = [\![F[z^{\square_0} \mapsto u]\,a[z^{\square_0} \mapsto u]]\!]_\xi \\
&\qquad = [\![F[z^{\square_0} \mapsto u]]\!]_\xi\, a[z^{\square_0} \mapsto u]\bullet[\![a[z^{\square_0} \mapsto u]]\!]_\xi \\
&\qquad = [\![F]\!]_{\xi,z\mapsto\mathring{z}}[z^{\square_0} \mapsto u][\mathring{z}^* \mapsto [\![u]\!]_\xi]\, a[z \mapsto u] \\
&\qquad\qquad \bullet[\![a]\!]_{\xi,z\mapsto\mathring{z}}[z^{\square_0} \mapsto u][\mathring{z}^* \mapsto [\![u]\!]_\xi] \\
&\qquad = [\![F]\!]_{\xi,z\mapsto\mathring{z}}[z^{\square_0} \mapsto u][\mathring{z}^* \mapsto [\![u]\!]_\xi]\, a[z^{\square_0} \mapsto u][\mathring{z}^* \mapsto [\![u]\!]_\xi] \\
&\qquad\qquad \bullet[\![a]\!]_{\xi,z\mapsto\mathring{z}}[z^{\square_0} \mapsto u][\mathring{z}^* \mapsto [\![u]\!]_\xi] \\
&\qquad = ([\![F]\!]_{\xi,z\mapsto\mathring{z}}\, a\bullet[\![a]\!]_{\xi,z\mapsto\mathring{z}})[z^{\square_0} \mapsto u][\mathring{z}^* \mapsto [\![u]\!]_\xi] \\
&\qquad = [\![F\,a]\!]_{\xi,z\mapsto\mathring{z}}[z^{\square_0} \mapsto u][\mathring{z}^* \mapsto [\![u]\!]_\xi]
\end{aligned}$$

Abstraction, Product and Application are straightforward for $(*, \square_0, \square_0)$

Variable

- If $x = z$:
  $[\![z[z^{\square_0} \mapsto u]]\!]_\xi = [\![u]\!]_\xi = \mathring{z}[z^{\square_0} \mapsto u][\mathring{z}^* \mapsto [\![u]\!]_\xi]$
  $= [\![z]\!]_{\xi,z\mapsto\mathring{z}}[z^{\square_0} \mapsto u][\mathring{z}^* \mapsto [\![u]\!]_\xi]$
- If $x \neq z$ and $x \in \xi$:
  $[\![x[z^{\square_0} \mapsto u]]\!]_\xi = \mathring{x} = \mathring{x}[z^{\square_0} \mapsto u][\mathring{z}^* \mapsto [\![u]\!]_\xi]$
  $= [\![x]\!]_{\xi,z\mapsto\mathring{z}}[z^{\square_0} \mapsto u][\mathring{z}^* \mapsto [\![u]\!]_\xi]$
- If $x \neq z$ and $x \notin \xi$:
  $[\![x[z^{\square_0} \mapsto u]]\!]_\xi = \llbracket x \rrbracket = \llbracket x \rrbracket[z^{\square_0} \mapsto u][\mathring{z}^* \mapsto [\![u]\!]_\xi]$ □

Lemma 3 ($[\![\cdot]\!]$ and substitution, part 2). *if ξ does not contain either z or any of the free variables of u, then*

$$[\![t[z \mapsto u]]\!]_\xi = [\![t]\!]_\xi[z \mapsto u]$$

Proof. By induction on t. Only the variable case ($t = x$) will be proved here.

Abstraction, Product, Application and Sort same as above.

Variable

- If $x = z$:
  $[\![z[z^{\square_0} \mapsto u]]\!]_\xi = [\![u]\!]_\xi = [\![u]\!]_\varnothing = \llbracket z \rrbracket[z^{\square_0} \mapsto u]$
  since $z \notin \xi$
- If $x \neq z$ and $x \in \xi$:
  $[\![x[z^* \mapsto u]]\!]_\xi = \mathring{x} = [\![x]\!]_\xi[z^* \mapsto u]$
- If $x \neq z$ and $x \notin \xi$:
  $[\![x[z^* \mapsto u]]\!]_\xi = \llbracket x \rrbracket = \llbracket x \rrbracket[z^* \mapsto u]$ □

If z is a proof variable, the previous lemma can be slightly generalized:

Lemma 4 ($[\![\cdot]\!]$ and substitution, part 3).

$$[\![t[z^* \mapsto u]]\!]_\xi = [\![t]\!]_\xi[z^* \mapsto u]$$

Proof. Same as above (by induction on t), except for the variable case, where t cannot be the *proof* variable z.

Lemma 5 (Substitution).

$$t[z \mapsto u][z' \mapsto u'] = t[z' \mapsto u'][z \mapsto u[z' \mapsto u']]$$

Proof. By induction on t; the only non-trivial case is for the parametric witnesses $[\![z]\!]$:

$$\begin{aligned}[\![z]\!][z^{\square_0} \mapsto u][z' \mapsto u'] &= [\![u]\!]_\varnothing[z' \mapsto u'] \\ &= [\![u[z' \mapsto u']]\!]_\varnothing = [\![z]\!][z' \mapsto u'][z^{\square_0} \mapsto u[z' \mapsto u']]\end{aligned}$$

by Lemma 3.

We now check that the modifications made to CC_ω do not break the Church-Rosser property, 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. Note that the case for redex-elimination differs slightly from the usual one, in that it may perform multiple reductions in a row, if they occur on the head of the redex. We had to extend the formulation of the β rule since the parametric interpretation does not preserve the number of redexes: relevant redexes result in two redexes (see equations (3) and (4) in Definition 3). This multiplication of redexes does not compromise the soundness of the technique, because it occurs only at the leaves of the relation. (If we had Σ-types in the syntax, we could uncurrify the term first, to keep only one redex).

Definition 4 (Parallel nested reduction).

$$\textsc{Refl}\ \frac{}{A \triangleright A}$$

$$\beta\ \frac{t \triangleright t' \qquad u_i \triangleright u_i'}{(\lambda {x_1}^{s_1} : A_1 \ldots \lambda {x_n}^{s_n} : A_n.\, t)^{s_1} u_1 \cdots^{s_n} u_n \triangleright t'[{x_1}^{s_1} \mapsto u_1'] \cdots [{x_n}^{s_n} \mapsto u_n']}$$

$$\textsc{App-Cong}\ \frac{t \triangleright t' \qquad u \triangleright u'}{t^s u \triangleright {t'}^s u'} \qquad \textsc{Abs-Cong}\ \frac{A \triangleright A' \qquad t \triangleright t'}{\lambda x^s : A.\, t \triangleright \lambda x^s : A'.\, t'}$$

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

Lemma 6 (Congruence of $[\![\cdot]\!]$). *If $A \triangleright A'$, then $[\![A]\!]_\xi \triangleright [\![A']\!]_\xi$ for all ξ.*

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

– The case of Refl is trivial.

– For β, we proceed by induction on n (the number of β-reductions); for $n = 1$, one expects

$$[\![(\lambda x^s : A.\, t)^s u]\!]_\xi \triangleright [\![t'[x^s \mapsto u']]\!]_\xi,$$

knowing that $t \triangleright t'$ and $u \triangleright u'$. The sort s can either be $*$ or $\Box_0$, as in the other cases our term would be ill-sorted.

- For $*$,

$$\begin{array}{ll} & [\![(\lambda x^*\colon A.\, t)\bullet u]\!]_\xi \\ = & \quad \{\text{by def. of } [\![\cdot]\!]_\xi\} \\ & (\lambda x^*\colon A.\, [\![t]\!]_\xi)\bullet u \\ \triangleright & \quad \{\text{by } \beta, \textsc{Refl} \text{ and IH}\} \\ & [\![t']\!]_\xi[x^* \mapsto u'] \\ = & \quad \{\text{by Lemma 4}\} \\ & [\![t'[x^* \mapsto u']]\!]_\xi \end{array}$$

- For $\Box_0$,

$$\begin{array}{ll} & [\![(\lambda x^{\Box_0}\colon A.\, t)\, u]\!]_\xi \\ = & \quad \{\text{by def. of } [\![\cdot]\!]_\xi\} \\ & (\lambda x^{\Box_0}\colon A.\, \lambda \mathring{x}^*\colon x \in [\![A]\!]_\xi.\, [\![t]\!]_{\xi, x \mapsto \mathring{x}})\, u\bullet [\![u]\!]_\xi \\ \triangleright & \quad \{\text{by } \beta, \textsc{Refl} \text{ and IH}\} \\ & [\![t']\!]_{\xi, x \mapsto \mathring{x}}[x^{\Box_0} \mapsto u'][\mathring{x}^* \mapsto [\![u']\!]_\xi] \\ = & \quad \{\text{by Lemma 2}\} \\ & [\![t'[x^{\Box_0} \mapsto u']]\!]_\xi \end{array}$$

If $n = 0$ (no β reduction), the conclusion stems from Refl. The inductive case is similar to the one for $n = 1$.

– The cases of $\star$-Cong are straightforward using the definition of $[\![\cdot]\!]$. □

Lemma 7 (Congruence of substitution). *If $t \triangleright t'$ and $u \triangleright u'$, then $t[z \mapsto u] \triangleright t'[z \mapsto u']$.*

Proof. By induction on $t \triangleright t'$:

– For Refl, the expected result follows from an induction on t (using Lemma 6 for the case $[\![z]\!]$).

– For β, one expects

$$((\lambda x : A.\, t)\, e)[z \mapsto u] \triangleright t'[x \mapsto e'][z \mapsto u'],$$

knowing that $t \triangleright t'$ and $e \triangleright e'$. We have

$$\begin{array}{ll} & ((\lambda x : A.\, t)\, e)[z \mapsto u] \\ = & \quad \{\text{by def. of the substitution}\} \\ & (\lambda x : A[z \mapsto u].\, t[z \mapsto u])\, e[z \mapsto u] \\ \triangleright & \quad \{\text{by } \beta \text{ and IH}\} \\ & t'[z \mapsto u'][x \mapsto e'[z \mapsto u']] \\ = & \quad \{\text{by Lemma 5}\} \\ & t'[x \mapsto e'][z \mapsto u'] \end{array}$$

– The cases of $\star$-Cong are straightforward. □

Theorem 3 (Diamond property). *The rewriting system ($\triangleright$) has the diamond property. That is, for each x, y, y' such that $y \triangleleft x \triangleright y'$, there exists z such that $y \triangleright z \triangleleft y'$*

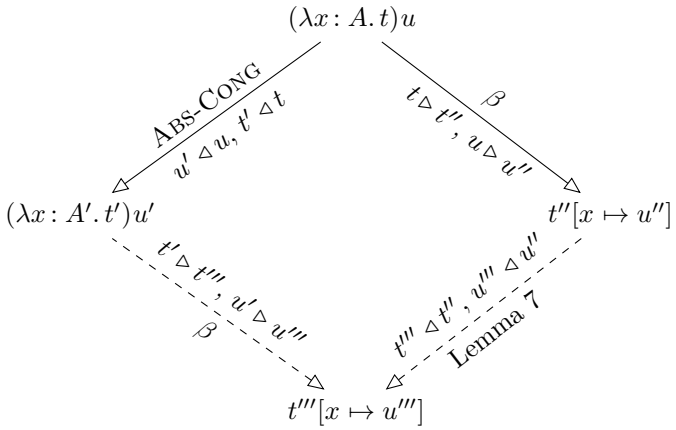
Proof. By inductions on the derivations:

- If one of the derivations ends with Refl, one has either $x = y$, or $x = y'$. We pick $z = y'$ in the former case and $z = y$ in the latter.
- If one of the derivations ends with Abs-Cong or All-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

$$\begin{gathered}(\lambda x^s : A'.\, t')^s u' \;\triangleleft\; (\lambda x^s : A.\, t)^s u \;\triangleright\; t''[x^s \mapsto u''] \\ \text{with } \lambda x^s : A'.\, t' \;\triangleleft\; \lambda x^s : A.\, t, \quad t \triangleright t'' \quad \text{and} \quad u' \triangleleft u \triangleright u''\end{gathered}$$

  The situation is summarized in the diagram below. In more details, the end of the derivation of $\lambda x^s : A'.\, t' \triangleleft \lambda x^s : A.\, t$ can be either Abs-Cong, or Refl in the first case (the last one is similar), one has $A' \triangleleft A$ and $t' \triangleleft t$.
  By induction hypothesis there exist t''', u''' such that $t' \triangleright t''' \triangleleft t''$ and $u' \triangleright u''' \triangleleft u''$.
  The result follows by β and Lemma 7:

$$(\lambda x^s : A'.\, t')^s u' \;\triangleright\; t'''[x^s \mapsto u'''] \;\triangleleft\; t''[x^s \mapsto u'']$$



- If both derivations end with the same β rule, the result is a straightforward use of the induction hypothesis and Lemma 7. □

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

Proof. Direct consequence of Theorem 3, noticing $\triangleright^\star = \longrightarrow^\star$.

4.2 Abstraction

In this section we check that our main goal, the integration of parametricity, is achieved by the design that we propose. At the same time, we check that the abstraction theorem also holds. We do so by proving Lemma 8, which subsumes both theorems.

Theorem 4 (Abstraction).

1. *$\Gamma \vdash A : B : \square_0 \;\Rightarrow\; [\![\Gamma]\!]_\xi \vdash [\![A]\!]_\xi : (A \in [\![B]\!]_\xi) : *$, where ξ maps all the variables in Γ to fresh ones.*
2. *Furthermore, if the original judgement makes no use of* Param*, nor does the resulting judgement.*

Proof.

1. Direct consequence of Lemma 8.1.
2. In the proof of Lemma 8.1, if ξ is full, then the target derivation trees contains Param iff Param occurs in the derivation tree for $\Gamma \vdash A : B$. □

Theorem 5 (Parametricity).

$$\Gamma \vdash A : B : \square_0 \;\Rightarrow\; \Gamma \vdash [\![A]\!] : (A \in [\![B]\!]) : *$$

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

Proof. Take ξ empty in Lemma 8.1.

Definition 5. *ξ conforms to Γ when ξ maps exactly a suffix of Γ (restricted to $\square_0$-variables) to relational variables.*

Remark 2. $[\![\cdot]\!]$ preserves conforming substitutions.

Proof (sketch). By induction on the typing derivation. In the definition of $[\![\cdot]\!]$, of every bound variable in a term is assigned an explicit relational interpretation.

Lemma 8 (Generalized abstraction). *Assuming that ξ conforms to Γ,*

1. *$\Gamma \vdash A : B : \square_0$ and $B \neq *\;\Rightarrow\; [\![\Gamma]\!]_\xi \vdash [\![A]\!]_\xi : A \in [\![B]\!]_\xi : *$*
2. *$\Gamma \vdash A : B : s \;\Rightarrow\; [\![\Gamma]\!]_\xi \vdash A : B : s$*
3. *$\Gamma \vdash B : \square_0$ and $B \neq *\;\Rightarrow\; [\![\Gamma]\!]_\xi, x : B \vdash x \in [\![B]\!]_\xi : *$*

(where ξ contains all the variables of sort $\square_0$ bound in Γ).

Proof. The proof is done by simultaneous induction on the derivation tree, and is very similar to the proofs done by Bernardy et al. [6] and Bernardy and Lasson [5]. The new parts occur in the special handling of $*$ and of the Start rule. 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 $\mathring{x}$, and looking up this second variable in the context justifies the translated judgement. If $x \notin \xi$, then we can use the parametricity rule on x to translate the typing judgement. Note that using parametricity is possible because, by assumption, x cannot be a proof. PARAM can be ignored, because direct nesting of parametricity is forbidden.
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 $x \in [\![T]\!]$ is also well-sorted. It is easy to convince oneself of that result by checking that the translation of a type always yields a predicate. □

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

Remark 3. In summary, and roughly speaking, Lemma 8 replaces the occurrences of START for variables not in ξ by PARAM. Occurrences on START for variables in ξ are preserved.

4.3 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 9 (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 10 (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 $\Gamma \vdash [\![x]\!] : C$ then there exists B such that $\Gamma \vdash B : \Box_0$, $(x : B) \in \Gamma$, and $C =_\beta x \in [\![B]\!]$.*

Proof. As in [4]:

– we follow the derivation $\Gamma \vdash \llparenthesis x \rrparenthesis : C$ until $\llparenthesis x \rrparenthesis$ is introduced. It can only be done by the following rule

$$\frac{\Delta \vdash B : \Box_0}{\Delta, x : B \vdash \llparenthesis x \rrparenthesis : x \in [\![B]\!]} \textsc{Param}$$

with $C =_\beta x \in [\![B]\!]$, and $(\Delta, x^{\Box_0} : B) \subseteq \Gamma$. The conclusion stems from Lemma 9. □

Theorem 6 (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 $\llparenthesis x \rrparenthesis$ preserves the type of the expression.

Facing a term such as $\llparenthesis x \rrparenthesis$ in context Γ, we know by generation that it must have type $x \in [\![B]\!]$ (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 $[\![a]\!]$, which has type $a \in [\![B']\!]$ by Theorem 5. In turn, $a \in [\![B']\!]$ is convertible to $x \in [\![B]\!]$ by Lemma 6.

4.4 Strong normalization

In this section we present a formalization of the intuitive model presented in Section 2.3. We do this via a transformation, from the system extended with parametricity to the naked CC_ω system.

Each term is mapped to a term where parametricity witnesses are passed explicitly. Simultaneously, contexts are extended with explicit witnesses: each binding $x : A : \Box_0$ is replaced by a multiple binding $x : A, \breve{x} : x \in [\![A]\!]$. This means that $\llparenthesis x \rrparenthesis$ can be interpreted by the corresponding variable $\breve{x}$ in the context. The following table shows how some example terms 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 $\langle\!\langle A \rangle\!\rangle$
$\lambda x.\, \llparenthesis x \rrparenthesis$	$\lambda x.\, \lambda \breve{x}.\, \breve{x}$
$(\lambda x.\, \llparenthesis x \rrparenthesis)\,(y\ z)$	$(\lambda x.\, \lambda \breve{x}.\, \breve{x})\,(y\ z)\,(\breve{y}\ z\ \breve{z})$

Given that the interpretation $(\langle\!\langle \cdot \rangle\!\rangle)$ is sound with respect to CC_ω (Lemma 14) and that it preserves reductions (Lemma 15), we obtain strong normalization (Theorem 7). The rest of the section is devoted to defining the model formally, and arguing for its soundness.

The situation is that we have two transformations ($[\![\cdot]\!]$ and $\langle\!\langle \cdot \rangle\!\rangle_\Gamma$) which can both introduce parametricity witnesses. Because $[\![\cdot]\!]$ will be applied to the result of $\langle\!\langle \cdot \rangle\!\rangle_\Gamma$, there is a danger that a single variable may end up being given two

parametricity witnesses. We want to avoid such an occurrence, otherwise we would have to prove these two witnesses equal, which is technically cumbersome. Therefore, to define our interpretation, we need to identify the bindings $x : a : \square_0$ for which an explicit witness $\mathring{x} : x \in [\![A]\!]$ has already been introduced. We do so via a special syntactic marking on the bindings of variables which are given an explicit witness of parametricity by either transformation. This marking is syntactically realized by underlining binders of abstractions and products, as well as the argument of a corresponding application.

Concretely, we slightly modify $[\![\cdot]\!]$ by marking the duplicated binding (cases for abstraction and product); correspondingly the first argument is marked as well, in the case of the application. Similarly, we mark the new binding in the case for $[\![T]\!]_\xi$, and the case for $C \in [\![T]\!]_\xi$ has to be amended accordingly.

$$\begin{aligned}
[\![\lambda x^{\square_0}\colon A.\, B]\!]_\xi &= \lambda \underline{x : A}.\, \lambda \mathring{x}^*\colon x \in [\![A]\!]_\xi.\, [\![B]\!]_{\xi, x \mapsto \mathring{x}} \\
[\![A\, B]\!]_\xi &= [\![A]\!]_\xi\, \underline{B}\bullet [\![B]\!]_\xi \\
[\![T]\!]_\xi &= \lambda \underline{z : T}.\, z \in [\![T]\!]_\xi \\
\hline
C \in [\![\square_0]\!]_\xi &= \underline{C} \to * \\
C \in [\![\forall x^{\square_0}\colon A.\, B]\!]_\xi &= \forall \underline{x : A}.\, \forall \mathring{x}^*\colon x \in [\![A]\!]_\xi.\, (C\, x) \in [\![B]\!]_{\xi, x \mapsto \mathring{x}} \\
C \in [\![T]\!]_\xi &= [\![T]\!]_\xi\, \underline{C} \\
\hline
[\![\Gamma, x^{\square_0}\colon A]\!]_{\xi, x \mapsto \mathring{x}} &= [\![\Gamma]\!]_\xi, \underline{x : A}, \mathring{x}^*\colon x \in [\![A]\!]_\xi
\end{aligned}$$

Additionally, since nesting $[\![\cdot]\!]$ is forbidden (it yields ill-typed terms), we are free to choose the behavior of $[\![\cdot]\!]$ on the new marked bindings; avoiding the duplication of parametricity witnesses. The result is as follows:

$$\begin{aligned}
[\![\lambda \underline{x : A}.\, \lambda \mathring{x}^*\colon A'.\, B]\!]_\xi &= \lambda \underline{x : A}.\, \lambda \mathring{x}^*\colon A'.\, [\![B]\!]_{\xi, x \mapsto \mathring{x}} \\
[\![A\, \underline{B}\bullet B']\!]_\xi &= [\![A]\!]_\xi\, \underline{B}\bullet B' \\
\hline
C \in [\![\forall \underline{x : A}.\, \forall \mathring{x}^*\colon A'.\, B]\!]_\xi &= \forall \underline{x : A}.\, \forall \mathring{x}^*\colon A'.\, (C\, x) \in [\![B]\!]_{\xi, x \mapsto \mathring{x}} \\
\hline
[\![\Gamma, \underline{x : A}, \mathring{x}^*\colon A']\!]_\xi &= [\![\Gamma]\!]_\xi, \underline{x : A}, \mathring{x}^*\colon A'
\end{aligned}$$

It is useful to stress that we only change the behavior of $[\![\cdot]\!]$ on cases that were forbidden in our extended system. This means that the newly introduced binding, as well as the changes, are only relevant in this proof of normalization. This means that all the previous results remain valid. In fact, the users of the system are free to remain entirely oblivious to the special marking, and the presentation of the interpretation of the system done in the previous sections need not being amended at all. On the other hand, one may wonder why we change the behavior of $[\![\cdot]\!]$ only in cases that are forbidden. The answer is that we are going to apply it to terms in CC_ω, which do not contain any occurrence of $[\![\cdot]\!]$, and therefore nested uses of $[\![\cdot]\!]$ can safely occur.

We can define our interpretation function as follows, adding explicit witnesses only if necessary that is, only for *non-marked* bindings (resp. arguments of applications) of sort $\Box_0$.

Definition 6 (Explicit binding of witnesses).

$$
\begin{aligned}
\langle\!|s|\!\rangle_\Gamma &= s \\
\langle\!|x|\!\rangle_\Gamma &= x \\
\langle\!|\llbracket x \rrbracket|\!\rangle_\Gamma &= \breve{x} \\
\\
\langle\!|\lambda x^{\Box_0}\colon A.\, B|\!\rangle_\Gamma &= \lambda \underline{x : \langle\!|A|\!\rangle_\Gamma}.\, \lambda \breve{x}^*\colon x \in [\![\langle\!|A|\!\rangle_\Gamma]\!]_{\chi(\Gamma)}.\, \langle\!|B|\!\rangle_{\Gamma,x} \\
\langle\!|\lambda x^*\colon A.\, B|\!\rangle_\Gamma &= \lambda x^*\colon \langle\!|A|\!\rangle_\Gamma.\, \langle\!|B|\!\rangle_\Gamma \\
\langle\!|\lambda \underline{x : A}.\, \lambda \mathring{x}^*\colon A'.\, B|\!\rangle_\Gamma &= \lambda \underline{x : \langle\!|A|\!\rangle_\Gamma}.\, \lambda \breve{x}^*\colon \langle\!|A'|\!\rangle_\Gamma.\, \langle\!|B|\!\rangle_{\Gamma,x} \\
\\
\langle\!|A\, B|\!\rangle_\Gamma &= \langle\!|A|\!\rangle_\Gamma\, \underline{\langle\!|B|\!\rangle_\Gamma} \bullet [\![\langle\!|B|\!\rangle_\Gamma]\!]_{\chi(\Gamma)} \\
\langle\!|A \bullet B|\!\rangle_\Gamma &= \langle\!|A|\!\rangle_\Gamma \bullet \langle\!|B|\!\rangle_\Gamma \\
\langle\!|A\, \underline{B} \bullet B'|\!\rangle_\Gamma &= \langle\!|A|\!\rangle_\Gamma\, \underline{\langle\!|B|\!\rangle_\Gamma} \bullet \langle\!|B'|\!\rangle_\Gamma \\
\\
\langle\!|\forall x^{\Box_0}\colon A.\, B|\!\rangle_\Gamma &= \forall \underline{x : \langle\!|A|\!\rangle_\Gamma}.\, \forall \breve{x}^*\colon x \in [\![\langle\!|A|\!\rangle_\Gamma]\!]_{\chi(\Gamma)}.\, \langle\!|B|\!\rangle_{\Gamma,x} \\
\langle\!|\forall x^*\colon A.\, B|\!\rangle_\Gamma &= \forall x^*\colon \langle\!|A|\!\rangle_\Gamma.\, \langle\!|B|\!\rangle_\Gamma \\
\langle\!|\forall \underline{x : A}.\, \forall \mathring{x}^*\colon A'.\, B|\!\rangle_\Gamma &= \forall \underline{x : \langle\!|A|\!\rangle_\Gamma}.\, \forall \breve{x}^*\colon \langle\!|A'|\!\rangle_\Gamma.\, \langle\!|B|\!\rangle_{\Gamma,x} \\
\hline
\langle\!|-|\!\rangle &= - \\
\langle\!|\Gamma, x^{\Box_0}\colon A|\!\rangle &= \langle\!|\Gamma|\!\rangle, \underline{x : A}, \mathring{x}^*\colon x \in [\![\langle\!|A|\!\rangle_\Gamma]\!]_{\chi(\Gamma)} \\
\langle\!|\Gamma, x^*\colon A|\!\rangle &= \langle\!|\Gamma|\!\rangle, x^*\colon \langle\!|A|\!\rangle_\Gamma \\
\langle\!|\Gamma, \underline{x : A}, \mathring{x}^*\colon A'|\!\rangle &= \langle\!|\Gamma|\!\rangle, \underline{x : \langle\!|A|\!\rangle_\Gamma}, \breve{x}^*\colon \langle\!|A'|\!\rangle_\Gamma
\end{aligned}
$$

The above assumes that for each variable x, there is a globally fresh variable $\breve{x}$. The relational substitution mapping each x to $\breve{x}$ is written χ.

Definition 7. $\chi(\Gamma) = \{z \mapsto \breve{z} | z \in \Gamma\}$

The essence of the model defined by $\langle\!|\cdot|\!\rangle_\Gamma$ is that a parametricity witness $\llbracket x \rrbracket$ is adequately modeled by the variable $\breve{x}$, that is, if x has type A, then $\breve{x} : x \in [\![A]\!]$. The situation is somewhat more complex though, since the type of x will itself undergo translation by $\langle\!|\cdot|\!\rangle_\Gamma$, as well as the type of the witness. Hence, we must ultimately prove that $x \in [\![\langle\!|A|\!\rangle_\Gamma]\!]_{\chi(\Gamma)} = \langle\!|x \in [\![A]\!]|\!\rangle_\Gamma$. In fact, the lemma should be further generalized before it can be proven:

Lemma 11 ($\langle\!|\cdot|\!\rangle_\Gamma$ and $[\![\cdot]\!]$ commute). *If*

- $\Gamma \vdash A : B$,
- $\Delta \subseteq \Gamma$

then $[\![\langle\!|A|\!\rangle_\Gamma]\!]_{\chi(\Gamma)} = \langle\!|[\![A]\!]_{\chi(\Delta)}|\!\rangle_\Gamma$

Proof. By induction on A.

- The base case on variables shows how we rely on consistently having $\breve{x}$ as a witness for x.
  $x \in \Delta$
  $$\langle\!|\llbracket x \rrbracket_{\chi(\Delta)}|\!\rangle_\Gamma = \langle\!|\breve{x}|\!\rangle_\Gamma = \breve{x} = \llbracket x \rrbracket_{\chi(\Gamma)} = \llbracket \langle\!| x |\!\rangle_\Gamma \rrbracket_{\chi(\Gamma)}$$
  $x \notin \Delta$
  $$\langle\!|\llbracket x \rrbracket_{\chi(\Delta)}|\!\rangle_\Gamma = \langle\!|\llbracket\!| x |\!\rrbracket|\!\rangle_\Gamma = \breve{x} = \llbracket x \rrbracket_{\chi(\Gamma)} = \llbracket \langle\!| x |\!\rangle_\Gamma \rrbracket_{\chi(\Gamma)}$$

Application Case Fa, the other ones being trivial

$$\begin{array}{ll} & \llbracket \langle\!| F\,a |\!\rangle_\Gamma \rrbracket_{\chi(\Gamma)} \\ = & \quad \{\text{by def. of } \langle\!|\cdot|\!\rangle_\Gamma\} \\ & \llbracket \langle\!| F |\!\rangle_\Gamma\ \underline{\langle\!| a |\!\rangle_\Gamma}\bullet\llbracket \langle\!| a |\!\rangle_\Gamma \rrbracket_{\chi(\Gamma)} \rrbracket_{\chi(\Gamma)} \\ = & \quad \{\text{by def. of } \llbracket\cdot\rrbracket\} \\ & \llbracket \langle\!| F |\!\rangle_\Gamma \rrbracket_{\chi(\Gamma)}\ \underline{\langle\!| a |\!\rangle_\Gamma}\bullet\llbracket \langle\!| a |\!\rangle_\Gamma \rrbracket_{\chi(\Gamma)} \\ = & \quad \{\text{by IH}\} \\ & \langle\!|\llbracket F \rrbracket_{\chi(\Delta)}|\!\rangle_\Gamma\ \underline{\langle\!| a |\!\rangle_\Gamma}\bullet\langle\!|\llbracket a \rrbracket_{\chi(\Delta)}|\!\rangle_\Gamma \\ = & \quad \{\text{by def. of } \langle\!|\cdot|\!\rangle_\Gamma\} \\ & \langle\!|\llbracket F \rrbracket_{\chi(\Delta)}\ \underline{a}\bullet\llbracket a \rrbracket_{\chi(\Delta)}|\!\rangle_\Gamma \\ = & \quad \{\text{by def. of } \llbracket\cdot\rrbracket\} \\ & \langle\!|\llbracket F\,a \rrbracket_{\chi(\Delta)}|\!\rangle_\Gamma \end{array}$$

- The case of the abstraction $\lambda x^{\square_0}\colon A.\, b$ shows how bindings are handled, and illustrates why the non-duplication of parametricity witnesses is important.

$$\begin{array}{ll} & \langle\!|\llbracket \lambda x^{\square_0}\colon A.\, b \rrbracket_{\chi(\Delta)}|\!\rangle_\Gamma \\ = & \quad \{\text{by def. of } \llbracket\cdot\rrbracket\} \\ & \langle\!|\lambda \underline{x : A}.\, \lambda \mathring{x}^*\colon x \in \llbracket A \rrbracket_{\chi(\Delta)}.\, \llbracket b \rrbracket_{\chi(\Delta), x \mapsto \mathring{x}}|\!\rangle_\Gamma \\ = & \quad \{\text{by } \alpha\text{-renaming, and def. of } \langle\!|\cdot|\!\rangle_\Gamma\} \\ & \lambda \underline{x : \langle\!| A |\!\rangle_\Gamma}.\, \lambda \breve{x}^*\colon \langle\!| x \in \llbracket A \rrbracket_{\chi(\Delta)} |\!\rangle_\Gamma.\, \langle\!|\llbracket b \rrbracket_{\chi(\Delta, x)}|\!\rangle_{\Gamma, x} \\ = & \quad \overline{\{\text{by IH}\}} \\ & \lambda \underline{x : \langle\!| A |\!\rangle_\Gamma}.\, \lambda \breve{x}^*\colon x \in \llbracket \langle\!| A |\!\rangle_\Gamma \rrbracket_{\chi(\Gamma)}.\, \llbracket \langle\!| b |\!\rangle_{\Gamma, x} \rrbracket_{\chi(\Gamma, x)} \\ = & \quad \overline{\{\text{by def. of } \llbracket\cdot\rrbracket\}} \\ & \llbracket \lambda \underline{x : \langle\!| A |\!\rangle_\Gamma}.\, \lambda \breve{x}^*\colon x \in \llbracket \langle\!| A |\!\rangle_\Gamma \rrbracket_{\chi(\Gamma)}.\, \langle\!| b |\!\rangle_{\Gamma, x} \rrbracket_{\chi(\Gamma)} \\ = & \quad \overline{\{\text{by def. of } \langle\!|\cdot|\!\rangle_\Gamma\}} \\ & \llbracket \langle\!| \lambda x^{\square_0}\colon A.\, b |\!\rangle_\Gamma \rrbracket_{\chi(\Gamma)} \end{array}$$

Product Same as above.
Otherwise straightforward. □

Another important lemma is that $\langle\!|\cdot|\!\rangle$ applied to a context adds all necessary witnesses. Formally:

Lemma 12. $\llbracket \langle\!| \Gamma |\!\rangle \rrbracket_\xi = \langle\!| \Gamma |\!\rangle$

Proof. By induction on Γ. An easy consequence of the fact that we leave unchanged the bindings already equipped with an explicit parametricity witness.

Lemma 13 ($\langle\!| \cdot |\!\rangle_\Gamma$ and substitution).

- $\langle\!| A[z^* \mapsto a] |\!\rangle_\Gamma = \langle\!| A |\!\rangle_\Gamma[z \mapsto \langle\!| a |\!\rangle_\Gamma]$
- $\langle\!| A[z^{\Box_0} \mapsto a] |\!\rangle_\Gamma = \langle\!| A |\!\rangle_{\Gamma,z}[z \mapsto \langle\!| a |\!\rangle_\Gamma][\breve{z} \mapsto [\![\langle\!| a |\!\rangle_\Gamma]\!]_{\chi(\Gamma)}]$

Proof. By (simultaneous) induction on A; We illustrate how the proof proceeds by showing (only) the case for $\lambda x^{\Box_0}\colon A.\, b$, with z of sort $\Box_0$ (The second item of the lemma).

Abstraction Case $\lambda x^{\Box_0}\colon A.\, b$, the other ones being trivial

$$
\begin{aligned}
& \langle\!| (\lambda x^{\Box_0}\colon A.\, b)[z^{\Box_0} \mapsto a] |\!\rangle_\Gamma \\
= \quad & \{\text{by def. of the substitution}\} \\
& \langle\!| \lambda x^{\Box_0}\colon A[z \mapsto a].\, b[z \mapsto a] |\!\rangle_\Gamma \\
= \quad & \{\text{by def. of } \langle\!| \cdot |\!\rangle_\Gamma\} \\
& \lambda \underline{x : \langle\!| A[z \mapsto a] |\!\rangle_\Gamma}.\, \lambda \breve{x}^*\colon x \in [\![\langle\!| A[z \mapsto a] |\!\rangle_\Gamma]\!]_{\chi(\Gamma)}. \\
& \qquad\qquad\qquad\qquad\qquad\qquad \langle\!| b[z \mapsto a] |\!\rangle_{\Gamma,x} \\
= \quad & \{\text{by IH and Lemmas 2 and 4}\} \\
& (\lambda \underline{x : \langle\!| A |\!\rangle_{\Gamma,z}}.\, \lambda \breve{x}^*\colon x \in [\![\langle\!| A |\!\rangle_{\Gamma,z}]\!]_{\chi(\Gamma,z)}.\, \langle\!| b |\!\rangle_{\Gamma,z,x}) \\
& \qquad\qquad\qquad\qquad [z \mapsto \langle\!| a |\!\rangle_\Gamma][\breve{z} \mapsto [\![\langle\!| a |\!\rangle_\Gamma]\!]_{\chi(\Gamma)}] \\
= \quad & \{\text{by def. of } \langle\!| \cdot |\!\rangle_\Gamma\} \\
& \langle\!| \lambda x^{\Box_0}\colon A.\, b |\!\rangle_{\Gamma,z}[z \mapsto \langle\!| a |\!\rangle_\Gamma][\breve{z} \mapsto [\![\langle\!| a |\!\rangle_\Gamma]\!]_{\chi(\Gamma)}]
\end{aligned}
$$

Product Same as above.
Otherwise straightforward. □

Finally we can show the soundness of our model, by proving that the transformation yields well-typed terms in CC_ω.

Lemma 14. $\Gamma \vdash A : B \Longrightarrow \langle\!| \Gamma |\!\rangle \vdash_{\mathrm{CC}_\omega} \langle\!| A |\!\rangle_\Gamma : \langle\!| B |\!\rangle_\Gamma$

Proof. By induction on the derivation.

Most cases are straightforward given the above definition of $\langle\!| \cdot |\!\rangle_\Gamma$, and use the abstraction theorem, wherever $\langle\!| \cdot |\!\rangle_\Gamma$ uses $[\![\cdot]\!]$.

Three cases merit further mention:

- The case of application essentially relies on the fact that if $\langle\!| a |\!\rangle_\Gamma$ is typeable in $\langle\!| \Gamma |\!\rangle$, then the explicit witness of parametricity, $[\![\langle\!| a |\!\rangle_\Gamma]\!]$, is typeable in the same context. This is a consequence of Lemma 12.
- The conversion case relies on Lemma 13.
- The base case of parametricity relies on Lemma 11, and is informally explained above.

Details of the application and parametricity cases are given in Figure 1. □

Axiom The axiom case is trivial: the tree is untouched. $\dfrac{}{\vdash s_1 : s_2}$ Axiom

Weakening Case $\Gamma, x^{\square_0}\!: C \vdash A : B$

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash A : B}{\text{induction}}}{\langle\!|\Gamma|\!\rangle \vdash \langle\!|A|\!\rangle_\Gamma : \langle\!|B|\!\rangle_\Gamma} \quad \dfrac{\dfrac{\Gamma \vdash C : \square_0}{\text{induction}}}{\langle\!|\Gamma|\!\rangle \vdash \langle\!|C|\!\rangle_\Gamma : \square_0} \quad \dfrac{\dfrac{\Gamma \vdash C : \square_0}{\text{induction}}}{\langle\!|\Gamma|\!\rangle_\Gamma \vdash \langle\!|C|\!\rangle_\Gamma : \square_0}\ \text{Lemma 8.3}}{\langle\!|\Gamma|\!\rangle_\Gamma, x : \langle\!|C|\!\rangle_\Gamma \vdash x \in [\![\langle\!|C|\!\rangle_\Gamma]\!]_{\chi(\Gamma)} : *}\ 2\times\text{Weak.}$$
$$\dfrac{\langle\!|\Gamma|\!\rangle, \underline{x : \langle\!|C|\!\rangle_\Gamma}, \breve{x}^*\!: x \in [\![\langle\!|C|\!\rangle_\Gamma]\!]_{\chi(\Gamma)} \vdash \langle\!|A|\!\rangle_\Gamma : \langle\!|B|\!\rangle_\Gamma}{\langle\!|\Gamma|\!\rangle, \underline{x : \langle\!|C|\!\rangle_\Gamma}, \breve{x}^*\!: x \in [\![\langle\!|C|\!\rangle_\Gamma]\!]_{\chi(\Gamma)} \vdash \langle\!|A|\!\rangle_{\Gamma,x} : \langle\!|B|\!\rangle_{\Gamma,x}}\ x \text{ is free in neither } A \text{ nor } B$$

Application Case $F^{\square_0} a$

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash F : (\forall x^{\square_0}\!: A.\, B)}{\text{induction}}}{\langle\!|\Gamma|\!\rangle \vdash \langle\!|F|\!\rangle_\Gamma : (\forall \underline{x : \langle\!|A|\!\rangle_\Gamma}.\, \forall \breve{x}^*\!: x \in [\![\langle\!|A|\!\rangle_\Gamma]\!]_{\chi(\Gamma)}.\, \langle\!|B|\!\rangle_{\Gamma,x})} \quad \dfrac{\dfrac{\Gamma \vdash a : A}{\text{induction}}}{\langle\!|\Gamma|\!\rangle \vdash \langle\!|a|\!\rangle_\Gamma : \langle\!|A|\!\rangle_\Gamma} \quad \dfrac{\dfrac{\dfrac{\Gamma \vdash a : A}{\text{induction}}}{\langle\!|\Gamma|\!\rangle \vdash \langle\!|a|\!\rangle_\Gamma : \langle\!|A|\!\rangle_\Gamma}\ \text{Lem. 8.1}}{[\![\langle\!|\Gamma|\!\rangle]\!]_{\chi(\Gamma)} \vdash [\![\langle\!|a|\!\rangle_\Gamma]\!]_{\chi(\Gamma)} : \langle\!|a|\!\rangle_\Gamma \in [\![\langle\!|A|\!\rangle_\Gamma]\!]_{\chi(\Gamma)}}\ \text{Lem. 12}}{\langle\!|\Gamma|\!\rangle \vdash [\![\langle\!|a|\!\rangle_\Gamma]\!]_{\chi(\Gamma)} : \langle\!|a|\!\rangle_\Gamma \in [\![\langle\!|A|\!\rangle_\Gamma]\!]_{\chi(\Gamma)}}\ 2\times\text{App.}$$
$$\dfrac{\langle\!|\Gamma|\!\rangle \vdash \langle\!|F|\!\rangle_\Gamma\, \underline{\langle\!|a|\!\rangle_\Gamma}\bullet[\![\langle\!|a|\!\rangle_\Gamma]\!]_{\chi(\Gamma)} x : \langle\!|B|\!\rangle_\Gamma[x \mapsto \langle\!|a|\!\rangle_\Gamma][\breve{x} \mapsto [\![\langle\!|a|\!\rangle_\Gamma]\!]_{\chi(\Gamma)}]}{\langle\!|\Gamma|\!\rangle \vdash \langle\!|F\, a|\!\rangle_\Gamma : \langle\!|B[x \mapsto a]|\!\rangle_\Gamma}\ \text{Lemma 13}$$

Abstraction Case $\lambda x^{\square_0}\!: A.\, b$

$$\dfrac{\dfrac{\dfrac{\Gamma, x : A \vdash b : B}{\text{induction}}}{\langle\!|\Gamma|\!\rangle, \underline{x : A}, \breve{x}^*\!: x \in [\![\langle\!|A|\!\rangle_\Gamma]\!]_{\chi(\Gamma)} \vdash \langle\!|b|\!\rangle_{\Gamma,x} : \langle\!|B|\!\rangle_{\Gamma,x}} \quad \dfrac{\dfrac{\Gamma \vdash (\forall x^{\square_0}\!: A.\, B) : s}{\text{induction}}}{\langle\!|\Gamma|\!\rangle \vdash (\forall \underline{x : \langle\!|A|\!\rangle_\Gamma}.\, \forall \breve{x}^*\!: x \in [\![\langle\!|A|\!\rangle_\Gamma]\!]_{\chi(\Gamma)}.\, \langle\!|B|\!\rangle_{\Gamma,x}) : s}}{\text{by def.}\ \dfrac{\langle\!|\Gamma|\!\rangle \vdash (\lambda \underline{x : \langle\!|A|\!\rangle_\Gamma}.\, \lambda \breve{x}^*\!: x \in [\![\langle\!|A|\!\rangle_\Gamma]\!]_{\chi(\Gamma)}.\, \langle\!|b|\!\rangle_{\Gamma,x}) : (\forall \underline{x : \langle\!|A|\!\rangle_\Gamma}.\, \forall \breve{x}^*\!: x \in [\![\langle\!|A|\!\rangle_\Gamma]\!]_{\chi(\Gamma)}.\, \langle\!|B|\!\rangle_{\Gamma,x})}{\langle\!|\Gamma|\!\rangle \vdash \langle\!|\lambda x^{\square_0}\!: A.\, b|\!\rangle_\Gamma : \langle\!|\forall x^{\square_0}\!: A.\, B|\!\rangle_\Gamma}}\ 2\times\text{Abs.}$$

Product Case $\forall x^{\square_0}\!: A.\, B$

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash A : \square_0}{\text{induction}}}{\langle\!|\Gamma|\!\rangle \vdash \langle\!|A|\!\rangle_\Gamma : \square_0} \quad \text{Lemma 8.3}\ \dfrac{\dfrac{\dfrac{\Gamma \vdash A : \square_0}{\text{induction}}}{\langle\!|\Gamma|\!\rangle \vdash \langle\!|A|\!\rangle_\Gamma : \square_0}}{\langle\!|\Gamma|\!\rangle, x : \langle\!|A|\!\rangle_\Gamma \vdash x \in [\![\langle\!|A|\!\rangle_\Gamma]\!]_{\chi(\Gamma)} : *} \quad \dfrac{\dfrac{\Gamma, x : A \vdash B : s_2}{\text{induction}}}{\langle\!|\Gamma|\!\rangle, \underline{x : A}, \breve{x}^*\!: x \in [\![\langle\!|A|\!\rangle_\Gamma]\!]_{\chi(\Gamma)} \vdash \langle\!|B|\!\rangle_{\Gamma,x} : s_2}}{\text{by def.}\ \dfrac{\langle\!|\Gamma|\!\rangle \vdash (\forall \underline{x : \langle\!|A|\!\rangle_\Gamma}.\, \forall \breve{x}^*\!: x \in [\![\langle\!|A|\!\rangle_\Gamma]\!]_{\chi(\Gamma)}.\, \langle\!|B|\!\rangle_{\Gamma,x}) : s_3}{\langle\!|\Gamma|\!\rangle \vdash \langle\!|\forall x^{\square_0}\!: A.\, B|\!\rangle_\Gamma : s_3}}\ 2\times\text{Prod.}$$

Conversion

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash A : B}{\text{induction}}}{\langle\!|\Gamma|\!\rangle \vdash \langle\!|A|\!\rangle_\Gamma : \langle\!|B|\!\rangle_\Gamma} \quad \dfrac{\dfrac{\Gamma \vdash B' : s}{\text{induction}}}{\langle\!|\Gamma|\!\rangle \vdash \langle\!|B'|\!\rangle_\Gamma : s} \quad \dfrac{B =_\beta B'}{\text{Corollary 1, Lemma 15}}}{\langle\!|\Gamma|\!\rangle \vdash \langle\!|A|\!\rangle_\Gamma : \langle\!|B'|\!\rangle_\Gamma}\ \text{Conv.}$$

Start

Easy. (Add one weakening for $\Gamma \vdash x : A : \square_0$).

Param

$$\text{Start}\ \dfrac{\text{Lemma 8.3}\ \dfrac{\dfrac{\dfrac{\Gamma \vdash A : \square_0}{\text{induction}}}{\langle\!|\Gamma|\!\rangle \vdash \langle\!|A|\!\rangle_\Gamma : \square_0}}{\langle\!|\Gamma|\!\rangle, \underline{x : \langle\!|A|\!\rangle_\Gamma} \vdash x \in [\![\langle\!|A|\!\rangle_\Gamma]\!]_{\chi(\Gamma)} : *}}{\langle\!|\Gamma|\!\rangle, \underline{x : \langle\!|A|\!\rangle_\Gamma}, \breve{x}^*\!: x \in [\![\langle\!|A|\!\rangle_\Gamma]\!]_{\chi(\Gamma)} \vdash \breve{x} : x \in [\![\langle\!|A|\!\rangle_\Gamma]\!]_{\chi(\Gamma)}}$$
$$\text{Lemma 11}\ \dfrac{\langle\!|\Gamma|\!\rangle, \underline{x : \langle\!|A|\!\rangle_\Gamma}, \breve{x}^*\!: x \in [\![\langle\!|A|\!\rangle_\Gamma]\!]_{\chi(\Gamma)} \vdash \breve{x} : \langle\!|x \in [\![A]\!]|\!\rangle_\Gamma}{\text{by def.}\ \dfrac{}{\langle\!|\Gamma, x^{\square_0}\!: A|\!\rangle \vdash \langle\!|\llbracket x \rrbracket|\!\rangle_{\Gamma,x} : \langle\!|x \in [\![A]\!]|\!\rangle_{\Gamma,x}}}$$

Fig. 1. Soundness of model: proof details.

Lemma 15 (Congruence of $\langle\!| \cdot |\!\rangle_\Gamma$). *If $\Gamma \vdash A : B$ with $A \longrightarrow A'$, then*

$$\langle\!| A |\!\rangle_\Gamma \longrightarrow^+ \langle\!| A' |\!\rangle_\Gamma.$$

Proof. Each occurrence of the App rule is translated to at least one occurrence of App in the typing derivation.

Theorem 7 (Strong normalization). *If CC_ω is strongly normalizing, then so is the system extended with* Param.

Proof. Assume $\Gamma \vdash A : B$ and consider a chain of reductions $A \longrightarrow^n A'$. We have $\langle\!| A |\!\rangle_\Gamma \longrightarrow^m \langle\!| A' |\!\rangle_\Gamma$, and $m \geq n$ by Lemma 15. We also have that $\langle\!| A |\!\rangle_\Gamma$ is typeable in CC_ω, by Lemma 14. Therefore, only finite chains of reductions are possible.

5 Generalized abstraction: Proof details

The proof of Lemma 8 depends on the following lemmas:

1. $\Gamma \vdash A : B : \Box_0$ and $B \neq *$ $\Rightarrow$ $[\![\Gamma]\!]_\xi \vdash [\![A]\!]_\xi : A \in [\![B]\!]_\xi : *$
2. $\Gamma \vdash A : B : s$ $\Rightarrow$ $[\![\Gamma]\!]_\xi \vdash A : B : s$
3. $\Gamma \vdash B : \Box_0$ and $B \neq *$ $\Rightarrow$ $[\![\Gamma]\!]_\xi, x : B \vdash x \in [\![B]\!]_\xi : *$

The lemmas are proved by transforming derivation trees. They mutually depend on each other, (but only for structurally smaller statement: the recursion is sound). In the proofs, the trees generated by each sub-lemma are written as follows:

- Lemma 1: $[\![\Gamma \vdash A : B]\!]_\xi$,
- Lemma 2: $|\Gamma \vdash A : B|$,
- Lemma 3: $\{\Gamma \vdash B : \Box_0\}_\xi$.

We only give further detail for 1. and 3.

Note that proofs never need to handle the Param case, because nesting of parametricity is forbidden. (The premise of the theorems is always invalidated.)