Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 338 (2018) 79–95

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Machine-checked Proof of the Church-Rosser Theorem for the Lambda Calculus Using the Barendregt Variable Convention in Constructive Type Theory

Ernesto Copello[1,2] Nora Szasz[3] Álvaro Tasistro[3]

Universidad ORT Uruguay
Montevideo, Uruguay

Abstract

In this article we continue the work started in [3], deriving in Constructive Type Theory new induction principles for the λ-calculus, using (the historical) first order syntax with only one sort of names for both bound and free variables, and with α-conversion based upon name swapping. The principles provide a flexible framework for mimicking pen-and-paper proofs within the rigorous formal setting of a proof assistant. We here report on one successful application, namely a complete proof of the Church-Rosser Theorem. The whole development has been machine-checked using the system Agda [5].

Keywords: Lambda Calculus, Formal Metatheory, Type Theory

1 Introduction

Let us consider the following definition of the Lambda Calculus terms:

$$M, N ::= x \mid MN \mid \lambda x.M$$

Fig. 1. Lambda Calculus syntax

where x ranges over a denumerable set of variables V.

This syntax has been considered too concrete a level on which to formally develop the metatheory of the calculus. The reason is that there is no significant difference between terms that differ only in the choice of the names of the bound variables,

[1] Currently at The University of Iowa, Iowa City, USA, Email: ernesto-copello@uiowa.edu

[2] Partially supported by a scholarship granted by Agencia Nacional de Investigación e Innovación, Uruguay.

[3] Email: szasz,tasistro@ort.edu.uy

https://doi.org/10.1016/j.entcs.2018.10.006

i.e., that are α-equivalent; and therefore it becomes natural to identify such terms already at the syntactic level. In the classical setting this amounts to working on α-equivalence classes of terms. One way to do this is to reason generally on terms by choosing adequate representatives of the classes in question. A prominent illustration of this practice is given in Barendregt's book [1], and the criterion followed to choose the representatives has thenceforth received as standard name *the Barendregt Variable Convention* (BVC).

In previous work [3], we have justified a form of the BVC in Constructive Type Theory by the following method: We introduced a recursion principle allowing the definition of *strongly α-compatible* functions on concrete terms, i.e. functions on the syntax introduced above that equate α-equivalent terms. This principle allows one to specify the value of the function in the case of abstractions by considering only the case in which the bound variable does not belong to a given list of names. The principle is actually implemented by computing, for each given term, a canonical α-equivalent representative that satisfies such restriction. Then, the specified function actually operates on the computed representative. The α-conversion is implemented using the elementary operation of *name swapping* as in nominal techniques (see for instance [7] and [8]). Using this principle we have been able to define e.g. the substitution operation by considering, for abstractions, only the convenient case in which variable capture is avoided.

In addition, we also introduced a corresponding induction principle, which is presented in Figure 2. This principle requires the property being proved to be α-compatible, that is, preserved by α-conversion. In other words, it must be a property of the abstract terms arising from the identification of α-equivalent concrete ones. As before, the induction principle allows us to prove the case of abstraction considering only the case in which the bound name does not belong to a given list of names:

$$\frac{\begin{array}{l} P\ \alpha\text{-compatible} \\ (\forall x)\ P(x) \\ (\forall M, N)\ (P(M) \wedge P(N) \Rightarrow P(M\,N)) \\ (\exists xs, \forall M, \forall x \notin xs)(P(M) \Rightarrow P(\lambda x.M)) \end{array}}{(\forall M)\ P(M)}$$

Fig. 2. Alpha induction principle

In this formulation and hereafter we use the following convention for fixing the range of quantifiers: x, y, z, ... range over (names of) variables, M, N, ... range over λ-terms, and xs ranges over lists of variables.

The principles referred to above were actually derived from the ordinary structural induction on concrete terms. As first applications, we developed, using the Agda proof assistant [5], some metatheoretical results concerning substitution that will be mentioned in the next section.

In this work we present strengthened α-induction principles on λ-terms that can be used to satisfactorily deal with relations whose definitions involve name swapping, and that allow us to avoid finite lists of names in binding positions, not

just in abstractions but in any term. In this way we can scale the metatheory of the λ-calculus up to the Church-Rosser Theorem, formally reproducing a form of the BVC in the formalisation. The set of obtained principles provides a flexible framework quite able to pleasantly mimic pen-and-paper proofs within the rigorous formal setting of a proof assistant.

The work that stands closest to the present one is [10], where Urban and Norrish show how to emulate the BVC when performing induction on relations over λ-terms. They illustrate the use of the induction principles by proving the Substitution Lemma for parallel reduction, and the Weakening Lemma of the typing relation. They present two induction principles on relations, one for the parallel reduction relation, and another one for the typing relation. When carrying out a proof by induction on these relations they are able to avoid a finite set of variable names as binders. To prove these strengthened induction principles they require that the relation definition rules satisfy the following preconditions: all functions and side conditions should be equivariant (i.e., preserved by name permutation), the side conditions must imply that all bound variables do not occur free in the conclusions, and all bound variables must be distinct. As a consequence, they have to modify the definition of the original relations to satisfy these preconditions in order to be able to prove the soundness of their induction principles.

All the definitions and proofs presented in the present article have been fully formalised in Constructive Type Theory [4] and machine-checked employing the system Agda [5]. The corresponding code is public, and it is available at:

https://github.com/ernius/formalmetatheory-nominal-Church-Rosser

In the subsequent text we give the proofs in English with a considerable level of detail so that they serve for making it clear how their formalisation was carried out.

The structure of the following sections is as follows: in Section 2 we recall the basic concepts of the λ-calculus, and some definitions and results from our previous work that are necessary for a better understanding of the material presented in this one. In Section 3 we introduce two new strengthened α-induction principles on λ-terms that will be useful to prove the main results of this work. Then, in Section 4 we present the notion of β-reduction and prove the Church-Rosser Theorem by using the standard method due to Tait and Martin-Löf which involves the formulation and study of the parallel β-reduction. The overall conclusions are exposed in Section 5.

2 Preliminaries

Variables belong to a denumerable set of names, and terms are inductively defined as in Figure 1.

The *freshness* relation states that a variable does not occur free in a term:

Definition 2.1 [Freshness]

$$\frac{x \neq y}{x\#y} \qquad \frac{x\#M \qquad x\#N}{x\#MN} \qquad \frac{x\#M}{x\#\lambda y.M} \qquad x\#\lambda x.M$$

$x \not\in_b M$ denotes that the variable x does not occur in a binding position in term M:

Definition 2.2 [$\not\in_b$]

$$x \not\in_b y \qquad \frac{x \not\in_b M \qquad x \not\in_b N}{x \not\in_b MN} \qquad \frac{x \neq y \qquad x \not\in_b M}{x \not\in_b \lambda y.M}$$

Next comes the operation of *swapping* of names. A finite sequence (composition) of name swaps constitutes a finite *name permutation*, which is the renaming mechanism to be used on terms. The action of swapping is first defined on names themselves:

Definition 2.3 [Swapping]

$$(x\ y)\ z\ = \begin{cases} y \text{ if } & z = x \\ x \text{ if } & z = y \\ z \text{ if } & x \neq z \wedge y \neq z, \end{cases}$$

and then it is directly extended to terms by swapping all names occurring in a term, including abstraction positions.

The permutation operation is just defined as the sequential application of a list of swaps. We usually use π to denote permutations, and the application of a permutation π to a term M is written $\pi\ M$. $(x\ y)\,\pi$ denotes the permutation consisting of the swap $(x\ y)$ followed by the permutation π.

In the next definition we give a syntax-directed definition of α-conversion ($\sim_\alpha$) based on the swapping operation.

Definition 2.4 [Alpha equivalence relation]

$$(\sim_\alpha \mathrm{v})\ \frac{}{x \sim_\alpha x} \qquad (\sim_\alpha \mathrm{a})\ \frac{M \sim_\alpha M' \qquad N \sim_\alpha N'}{MN \sim_\alpha M'N'}$$

$$(\sim_\alpha \lambda)\ \frac{(\forall z \not\in xs)\ \ (x\ z)M \sim_\alpha (y\ z)N}{\lambda x.M \sim_\alpha \lambda y.N}$$

In [3] we proved that this is an equivalence relation, preserved by the permutation operation (i.e. equivariant).

The substitution operation is defined using the α-compatible recursion principle mentioned in the previous section, and as a direct consequence of this, the following lemma is automatically derived:

Lemma 2.5 (α-compatibility of substitution)
$M \sim_\alpha M' \ \Rightarrow\ M[x{:=}N] = M'[x{:=}N]$.

The following results are successfully proved using the induction principles of [3].

Lemma 2.6 (Substitution preserves α-conversion)
$N \sim_\alpha N' \;\Rightarrow\; M[x{:=}N] \sim_\alpha M[x{:=}N']$.

Lemma 2.7 (Substitution under permutation)
$\pi\ (M[x{:=}N]) \sim_\alpha (\pi\ M)[(\pi\ x){:=}(\pi\ N)]$.

The next lemma shows that substitution commutes with abstraction up to α-conversion. This is so because the hypotheses ensure a fresh enough binder.

Lemma 2.8 (Substitution commutes with abstraction)
$x \neq y \wedge x\#N \;\Rightarrow\; (\lambda x.M)[y{:=}N] \sim_\alpha \lambda x.(M[y{:=}N])$.

As a consequence of the previous lemma, we obtain:

Lemma 2.9 (Substitution composition)
$x \neq y \wedge x\#P \;\Rightarrow\; M[x{:=}N][y{:=}P] \sim_\alpha M[y{:=}P][x{:=}N[y{:=}P]]$.

The following result was not proved in our previous work, so we exhibit it in detail.

Lemma 2.10 (Swapping substitution variable)
$x\#M \;\Rightarrow\; ((x\ y)M)[x{:=}N] \sim_\alpha M[y{:=}N]$.

Proof. We use our α-induction principle in Figure 2. First, for arbitrary names x, y and term N we consider the following predicate on terms:

$$\Pi(M) \equiv\ x\#M \;\Rightarrow\; ((x\ y)M)[x{:=}N] \sim_\alpha M[y{:=}N].$$

We have to prove that Π is α-compatible, that is, if $M{\sim_\alpha}P$ and $\Pi(M)$, then $\Pi(P)$. Assume $\Pi(M)$ and $x\#P$. Then, as freshness is preserved through $\sim_\alpha$, we have that $x\#M$. Then we proceed as follows:

$$\begin{array}{lcl} ((x\ y)P)[x{:=}N] & = & \{\sim_\alpha \text{ equivariance and Lemma 2.5}\} \\ ((x\ y)M)[x{:=}N] & \sim_\alpha & \{\Pi(M) \text{ and } x\#M\} \\ M[y{:=}N] & = & \{\text{Lemma 2.5}\} \\ P[y{:=}N]. & & \end{array}$$

Now we can proceed to the induction proper. We show the interesting case, namely the one of abstractions: We have $x\#\lambda z.M'$, where we choose $z \notin \{x, y\} \cup \mathrm{fv}(N)$. We need to prove $((x\ y)(\lambda z.M'))[x{:=}N] \sim_\alpha (\lambda z.M')[y{:=}N]$. As $x\#\lambda z.M'$ and $z \neq x$ we get $x\#M'$. Then we can reason as follows:

$$\begin{array}{rcl} ((x\ y)(\lambda z.M'))[x{:=}N] & = & \{\text{def. of swap}\} \\ (\lambda((x\ y)z).((x\ y)M'))[x{:=}N] & = & \{z \notin \{x, y\}\} \\ (\lambda z.((x\ y)M'))[x{:=}N] & \sim_\alpha & \{\text{Lemma 2.8}\} \\ \lambda z.(((x\ y)M')[x{:=}N]) & \sim_\alpha & \{\text{i.h.}\} \\ \lambda z.(M'[y{:=}N]) & \sim_\alpha & \{\text{Lemma 2.8}\} \\ (\lambda z.M')[y{:=}N] & & \end{array}$$

□

The preceding proof illustrates the usual pen-and-paper informal practice, which uses the BVC to assume binders fresh enough in some defined context, allowing us to apply substitution in a naive way without need of renaming.

The next result is a quite direct consequence of the previous lemma:

Lemma 2.11
$x\#\lambda y.M \;\Rightarrow\; ((x\ y)M)[x{:=}N] \sim_\alpha M[y{:=}N]$.

3 Alpha Induction Principles

In this section we introduce two new α-induction principles. The first one is presented in Figure 3.

$$\frac{\begin{array}{l} P\ \alpha\text{-compatible} \\ (\forall x)\ P(x) \\ (\forall M, N)\ (P(M) \wedge P(N) \Rightarrow P(M\,N)) \\ (\exists xs, \forall M, \forall x{\notin}xs)\ ((\forall \pi)\ P(\pi\ M) \Rightarrow P(\lambda x.M)) \end{array}}{(\forall M)\ P(M)}$$

Fig. 3. Alpha induction principle with permutations

It is a strengthened version of the one shown in Figure 2, where the induction hypothesis of the abstraction case allows us to assume the property for all permutations of names in the body. This principle is useful to deal with relations which make use of the permutation operation in their definitions. We will show an example of this situation in the proof of Lemma 4.7 in the next section.

The α-induction principle with permutations shown in Figure 3 is proved using the one in Figure 4, which was derived in [3] from simple structural induction on λ-terms, in very much the same way as complete induction on natural numbers is derived from ordinary mathematical induction.

$$\frac{\begin{array}{l} (\forall x)\ P(x) \\ (\forall M, N)\ (P(M) \wedge P(N) \Rightarrow P(M\,N)) \\ (\forall M, x)\ ((\forall \pi)\ P(\pi\ M)\ \Rightarrow P(\lambda x.M)) \end{array}}{(\forall M)\ P(M)}$$

Fig. 4. Strong permutation induction principle

Proof. (Alpha induction principle with permutations).
The variable and application cases are direct. For the abstraction case, given any term M and variable x, we must prove $P(\lambda x.M)$ knowing:

$$(\forall \pi)\ P(\pi\ M) \tag{1a}$$
$$(\exists xs, \forall M', \forall y{\notin}xs)\ ((\forall \pi')\ P(\pi'\ M') \Rightarrow P(\lambda y.M')) \tag{1b}$$

Let xs be a list of names as in (1b). Let us further pick y not in xs and also fresh in $\lambda x.M$. Then for all M', π', $P(\pi'\ M') \Rightarrow P(\lambda y.M')$ holds. So taking $M' =$

$(x\ y)M$ we have that $(\forall \pi')P(\pi'\ ((x\ y)M)) \Rightarrow P(\lambda y.(x\ y)M)$. Now, as $\pi'((x\ y)M) = ((x\ y)\pi')M$, we can use (1a) to get $P(\lambda y.(x\ y)M)$ from (1b), and finally $P(\lambda x.M)$ because P is α-compatible and $\lambda x.M \sim_\alpha \lambda y.(x\ y)M$. This last α-equivalence holds because we have chosen y fresh in $\lambda x.M$. □

The next induction principle (Figure 5) enables us to assume bound variables not in a given finite list of names xs through the entire induction, and not only for the abstraction case.

$$\frac{\begin{array}{l} P\ \alpha\text{-compatible} \\ (\forall x)\ P(x) \\ (\forall M, N)\ ((\forall y \in xs, y \notin_b MN) \wedge P(M) \wedge P(N) \Rightarrow P(M\,N)) \\ (\forall M, x)\ ((\forall y \in xs, y \notin_b \lambda x.M) \wedge P(M) \Rightarrow P(\lambda x.M)) \end{array}}{(\forall M)\ P(M)}$$

Fig. 5. Strengthened α-induction principle

Proof. (Strengthened α-induction principle).
To derive this principle we introduce a *rewrite* function such that, given a list of names xs and a term M, $rewrite\,(xs, M)$ returns a term α-convertible with M that does not contain any element of xs as binder.
To prove $P(M)$ for any term M, we proceed as follows. Given a list of names xs, an α-compatible predicate P, and the following hypotheses:

$$\begin{array}{l} (\forall x)\ P(x) \\ (\forall M, N)\ ((\forall y \in xs, y \notin_b MN) \wedge P(M) \wedge P(N) \Rightarrow P(M\,N)) \\ (\forall M, x)\ ((\forall y \in xs, y \notin_b \lambda x.M) \wedge P(M) \Rightarrow P(\lambda x.M)) \end{array} \tag{2}$$

we prove the following predicate Π by structural induction on terms:

$$\Pi(M) \equiv ((\forall x \in xs)\ \Rightarrow x \notin_b M) \Rightarrow P(M) \tag{3}$$

Then, we use this predicate Π on the term $rewrite\,(xs, M)$, which has no bound variables in xs to obtain $P(rewrite\,(xs, M))$. Finally, as P is α-compatible and $rewrite\,(xs, M) \sim_\alpha M$ we get that $P(M)$ holds for any M.

In turn, the proof of $\Pi(M)$ by structural induction on M is straightforward because of the syntax directed definition of $\notin_b$:

- Variable case: Direct.
- Application case: We need to prove $\Pi(MN)$ for any M, N, such that $\Pi(M)$ and $\Pi(N)$ hold. That is, we have to prove $P(MN)$, given that any variable x in xs satisfies that $x \notin_b (MN)$. Then, by the syntax directed definition of $\notin_b$, we directly have that $x \notin_b M$ and $x \notin_b N$, and so we are able to use the induction hypothesis on M and N to get $P(M)$ and $P(N)$. So, we have that all the premises in the second assertion in (2) hold, and hence its conclusion $P(MN)$.
- Abstraction case: We must prove $\Pi(\lambda y.M)$, that is, we need to prove $P(\lambda y.M)$ knowing that every variable x in xs satisfies that $x \notin_b \lambda y.M$. By the definition of

$\notin_b$, we have that $x \neq y$ and $x \notin_b M$. We can apply the last result to the induction hypothesis $\Pi(M)$ to get $P(M)$. Finally, we get the desired result using the third assertion in (2).

□

4 Parallel Beta Reduction

The β-reduction relation ($\rightarrow_\beta$) is defined as the compatible (with the syntactic constructors) closure of the β-contraction $(\lambda x.M)N \rhd_\beta M[x:=N]$. The classical proof of confluence of β-reduction by Tait and Martin-Löf rests upon the property of confluence of the so-called parallel reduction, which can apply several β-contractions "in parallel" in one single step. We present our definition in Figure 6.

$$(\rightrightarrows \mathrm{v})\ \frac{}{x \rightrightarrows x} \qquad (\rightrightarrows \mathrm{a})\ \frac{M \rightrightarrows M' \qquad N \rightrightarrows N'}{MN \rightrightarrows M'N'}$$

$$(\rightrightarrows \lambda)\ \frac{(\exists xs, \forall z \notin xs)\ (x\ z)M \rightrightarrows (y\ z)N}{\lambda x.M \rightrightarrows \lambda y.N}$$

$$(\rightrightarrows \beta)\ \frac{\lambda x.M \rightrightarrows \lambda y.P' \qquad N \rightrightarrows P'' \qquad P'[y:=P''] \sim_\alpha P}{(\lambda x.M)N \rightrightarrows P}$$

Fig. 6. Parallel reduction relation

The first three rules have the same form as the ones defining the α-conversion relation presented in Definition 2.4, which provides evidence that we want this parallel reduction to be compatible with α-conversion, that is, if $M \rightrightarrows N$, $M \sim_\alpha M'$ and $N \sim_\alpha N'$ then $M' \rightrightarrows N'$. We will prove this property in Lemmas 4.5 and 4.6. Finally, note that the $(\rightrightarrows\beta)$ rule has an extra premise involving α-conversion. The reason for this is that our substitution operation modifies the bound names in terms as a consequence of being defined with our α-recursion principle. Without that additional premise we would not be able to prove that $\rightrightarrows$ is α-compatible on its right hand side.

We start by proving some basic properties:

Lemma 4.1 (Reflexivity of $\rightrightarrows$)
$M \rightrightarrows M$.

Proof. Direct application of the permutation induction principle in Figure 4. □

Lemma 4.2 (Equivariance of $\rightrightarrows$)
$M \rightrightarrows N \ \Rightarrow\ \pi M \rightrightarrows \pi N$.

Proof. By induction on the definition of $\rightrightarrows$.
The variable and application cases are direct. In the abstraction case, we have to prove $\lambda(\pi\ x).(\pi\ M) \rightrightarrows \lambda(\pi\ y).(\pi\ N)$ from the premise of the rule $(\rightrightarrows\lambda)$ and the corresponding induction hypothesis. We can in addition exclude the variable

z mentioned in the premise from the domain of the permutation π and reason as follows:

$$\begin{array}{ll}
(x\ z)M \rightrightarrows (y\ z)N & \Rightarrow \{\text{i.h.}\} \\
\pi((x\ z)M) \rightrightarrows \pi((y\ z)N) & \Rightarrow \{\text{def. of perm.}\} \\
((\pi\, x)\ (\pi\, z))(\pi\, M) \rightrightarrows ((\pi\, y)\ (\pi\, z))(\pi\, N) & \Rightarrow \{\text{as } z \notin \mathrm{dom}(\pi) \text{ then } (\pi\, z) = z\} \\
((\pi\ x)\ z)(\pi\ M) \rightrightarrows ((\pi\ y)\ z)(\pi\ N) & \Rightarrow \{(\rightrightarrows \lambda)\ \text{rule}\} \\
\lambda(\pi\ x).(\pi\ M) \rightrightarrows \lambda(\pi\ y).(\pi\ N). &
\end{array}$$

In the $(\rightrightarrows \beta)$ case we must prove $(\lambda(\pi\ x).(\pi\ M))(\pi\ N) \rightrightarrows \pi\ P$ from premises $\lambda x.M \rightrightarrows \lambda y.P'$, $N \rightrightarrows P''$ and $P \sim_\alpha P'[y{:=}P'']$. By direct application of the induction hypotheses corresponding to the first two premises we get:

$$\begin{array}{c}
\lambda(\pi\ x).(\pi\ M) \rightrightarrows \lambda(\pi\ y).(\pi\ P') \\
\text{and } \pi\ N \rightrightarrows \pi\ P''
\end{array} \tag{4}$$

Then, using the third premise we can reason as follows:

$$\begin{array}{ll}
P \sim_\alpha P'[y{:=}P''] & \Rightarrow \{\sim_\alpha \text{ equivariance}\} \\
\pi\ P \sim_\alpha \pi\ (P'[y{:=}P'']) & \Rightarrow \{\text{Lemma 2.7}\} \\
\pi\ P \sim_\alpha (\pi\ P')[(\pi\ y){:=}(\pi\ P'')] & \Rightarrow \{\sim_\alpha \text{ symmetry}\} \\
(\pi\ P')[(\pi\ y){:=}(\pi\ P'')] \sim_\alpha \pi\ P &
\end{array}$$

We obtain the desired result using the $(\rightrightarrows\beta)$ rule with (4) and this last result as premises. □

As a direct consequence of the previous lemma we derive the following result:

Corollary 4.3 (Preservation of $\rightrightarrows$ under abstraction)
$M \rightrightarrows N \ \Rightarrow\ \lambda x.M \rightrightarrows \lambda x.N.$

The following lemmas state that our parallel reduction relation is preserved by α-equivalence. Both results are proved by easy inductions on the parallel reduction relation.

Lemma 4.4 (Right α-compatibility of $\rightrightarrows$)
$M \rightrightarrows N \wedge N \sim_\alpha P \ \Rightarrow\ M \rightrightarrows P.$

Lemma 4.5 (Left α-compatibility of $\rightrightarrows$)
$M \sim_\alpha N \wedge N \rightrightarrows P \ \Rightarrow\ M \rightrightarrows P.$

We can now prove in a direct manner that α-conversion is included in the parallel reduction.

Lemma 4.6
$\sim_\alpha \subseteq \rightrightarrows$.

Proof. Given $M \sim_\alpha N$, as $\rightrightarrows$ is reflexive by Lemma 4.1, we also know $M \rightrightarrows M$. Then using Lemma 4.4 we obtain the desired result.
□

As $\rightrightarrows$ basically applies β-contractions, no free variable should be introduced at any step, therefore freshness is preserved.

Lemma 4.7 ($\rightrightarrows$ preserves freshness)
$x\#M \wedge M \rightrightarrows N \;\Rightarrow\; x\#N.$

Proof. We use the α-induction principle with permutations (Figure 3) on the term M. In order to apply this principle we must prove, for any variable x, that the predicate

$$\Pi(M) \equiv \; (\forall N)\; (x\#M \wedge M \rightrightarrows N \;\Rightarrow\; x\#N).$$

is α-compatible, which follows from the α-compatibility of both freshness and parallel reduction. Now, for the main result, we only show the interesting abstraction case of the induction (i.e. for a term $\lambda y.M'$). We therefore have that $x\#\lambda y.M'$ and $\lambda y.M' \rightrightarrows \lambda z.N'$, and we must prove $x\#\lambda z.N'$. Now, $\lambda y.M' \rightrightarrows \lambda z.N'$ must be the result of an application of the $(\rightrightarrows\lambda)$ rule, so we get its premise $(\forall w \notin xs)\; (y\; w)M' \rightrightarrows (z\; w)N'$. The α-induction principle allows us to exclude some variables for the abstraction case, so we can also assume $y \neq x$. Using this inequality and the hypothesis $x\#\lambda y.M'$ we get by definition that $x\#M'$. Now let u be a variable such that $u\#N', u \notin xs$ and $u \neq x$; then $x\#(y\; u)M'$ because $x \neq y$, $x \neq u$ and $x\#M'$. We can apply the premise of the $(\rightrightarrows\lambda)$ rule with u, as $u \notin xs$, and we get $(y\; u)M' \rightrightarrows (z\; u)N'$. We use the induction hypothesis on M' and permutation $(y\; u)$ with the previous two results to get $x\; \#(z\; u)N'$. We also have that $\lambda u.(z\; u)N' \sim_\alpha \lambda z.N'$ because $u\; \#N'$. Then, as $\sim_\alpha$ preserves freshness, we get the desired result. □

We can now prove the following inversion lemmas, which state that the original definition of parallel reduction by Takahashi [9] (which we note $\rightrightarrows_T$ in the next definition) can be derived from ours. These lemmas will be useful in the proof of the diamond property of our relation $\rightrightarrows$.

$$\frac{}{x \rightrightarrows_T x} \qquad \frac{M \rightrightarrows_T M' \qquad N \rightrightarrows_T N'}{MN \rightrightarrows_T M'N'} \qquad \frac{M \rightrightarrows_T M'}{\lambda x.M \rightrightarrows_T \lambda x.M'}$$

$$\frac{M \rightrightarrows_T M' \qquad N \rightrightarrows_T N'}{(\lambda x.M)N \rightrightarrows_T M'[x{:=}N']}$$

Fig. 7. Takahashi's parallel reduction relation.

Lemma 4.8 ($\rightrightarrows$ λ-inversion)
$\lambda x.M \rightrightarrows M' \;\Rightarrow\; (\exists M'')\; (M \rightrightarrows M'' \;\wedge \lambda x.M \rightrightarrows \lambda x.M'' \wedge\; M' \sim_\alpha \lambda x.M'').$

Proof. By definition of $\rightrightarrows$ it must be the case that $\lambda x.M \rightrightarrows M'$ is a result of an application of $(\rightrightarrows \lambda)$ rule; then we have that M' is in an abstraction $\lambda y.N$, and that there exists a list of variables xs such that $(\forall z \notin xs)\; (x\; z)M \rightrightarrows (y\; z)N$. We take $M'' = (x\; y)N$, and prove that M'' satisfies the three properties of the thesis.

- Let z be a variable such that $z \notin xs$ and $z\#\lambda y.M'$. By definition of $\#$, $x\#\lambda x.M$, and then, as parallel reduction preserves freshness, $x\#\lambda y.N$ also holds. So:

$$\begin{array}{ll} (x\ z)M \rightrightarrows (y\ z)N & \Rightarrow \{\rightrightarrows \text{ equivariance}\} \\ (x\ z)(x\ z)M \rightrightarrows (x\ z)(y\ z)N & \Rightarrow \{\text{swap self inverse}\} \\ M \rightrightarrows (x\ z)(y\ z)N & \Rightarrow \{(*)\} \\ M \rightrightarrows (x\ y)N & \end{array}$$

(*) Here we apply Lemma 4.4 with the premise $(x\ z)(y\ z)N \sim_\alpha (x\ y)N$. This swapping cancellation property requires z and x to be fresh enough, as it is the case.

- We apply Lemma 4.4 with $\lambda x.M \rightrightarrows \lambda y.N$ and the α-equivalence obtained above to prove $\lambda x.M \rightrightarrows \lambda x.(x\ y)N$.
- To prove $\lambda y.N \sim_\alpha \lambda x.(x\ y)N$, as x is fresh in $\lambda y.N$, we swap y with x in this term to get the α-equivalent term $\lambda x.(x\ y)N$ (Lemma 4.2).

□

Lemma 4.9 ($\rightrightarrows$ β-inversion)
If $(\lambda x.M)N \rightrightarrows P$ is obtained by application of the $(\rightrightarrows \beta)$ rule in the following way:

$$(\rightrightarrows\beta)\ \frac{\lambda x.M \rightrightarrows \lambda y.M' \quad N \rightrightarrows N' \quad M'[y{:=}N'] \sim_\alpha P}{(\lambda x.M)N \rightrightarrows P}$$

then, $(\exists M'')\ (\lambda x.M \rightrightarrows \lambda x.M'' \wedge\ M''[x{:=}N'] \sim_\alpha P)$.

Proof. We prove that $M'' = (y\ x)M'$ satisfies the thesis.

- $x\#\lambda x.M$ and $\lambda x.M \rightrightarrows \lambda y.M'$ so by Lemma 4.7 $x\#\lambda y.M'$. We can then swap y with x in the last term and obtain the α-equivalent term $\lambda x.(y\ x)M'$, using Lemma 4.2. Then, by Lemma 4.4 we get $\lambda x.M \rightrightarrows \lambda x.(y\ x)M'$.
- For the second condition we reason as follows:

$$\begin{array}{lll} ((y\ x)M')[x{:=}N'] & = & \{\text{swap commutativity}\} \\ ((x\ y)M'))[x{:=}N'] & \sim_\alpha & \{\text{Corollary 2.11}\} \\ M'[y{:=}N'] & \sim_\alpha & \{\text{hypothesis}\} \\ P & & \end{array}$$

□

Theorem 4.10 ($\rightrightarrows$ Substitution Lemma)
$M \rightrightarrows M' \wedge N \rightrightarrows N' \ \Rightarrow\ M[x{:=}N] \rightrightarrows M'[x{:=}N']$.

The Substitution Lemma for parallel reduction is the crux of the Church-Rosser Theorem, and the place in which our α-induction principles in [3] fail to capture the BVC. If we perform induction on the term M, the problem appears in the beta application case, specifically when the term is a redex. We then have $M = (\lambda y.P)Q$, and we need to prove $((\lambda y.P)Q)[x{:=}N] \rightrightarrows R[x{:=}N']$. But, as we are in the application case of the induction, the original α-induction principle gives no freshness information about the binder y. The use of the BVC would allow us to choose y different from x and fresh in N, and with those freshness conditions we could push the substitution inside the abstraction without any variable capture by the use of Lemma 2.8. We next use our strengthened α-induction principle presented in Figure 5 to prove this result.

Proof. Given terms N, N' such that $N \rightrightarrows N'$, and a variable x, we consider the following predicate on terms:

$$\Pi(M) \equiv \ (\forall M')\ (M \rightrightarrows M' \ \Rightarrow \ M[x{:=}N] \rightrightarrows M'[x{:=}N']).$$

Π is α-compatible, which is a direct consequence of both substitution and $\rightrightarrows$ being α-compatible (Lemmas 2.5,4.4,4.5). Then we can use our strengthened α-induction principle to prove Π by induction on the term M, excluding the variable x, and the free variables in terms N and N' from the binders in M. We show the proof of the interesting application and abstraction cases.

- Application case: we prove $(\forall P, Q)\ ((\forall z \in \{x\} \cup \mathrm{fv}(N) \cup \mathrm{fv}(N'),\ z \notin_b P\ Q)\ \Pi(P) \wedge \Pi(Q) \Rightarrow \Pi(P\ Q))$. We have two subcases according to which rule is used to reduce the application $P\ Q$.
  - ($\rightrightarrows$a) rule subcase: we have that $P \rightrightarrows P'$ and $Q \rightrightarrows Q'$ and we need to prove that $(P\ Q)[x{:=}N] \rightrightarrows (P'\ Q')[x{:=}N']$. The proof is a direct application of the ($\rightrightarrows$a) rule to the induction hypotheses.
  - ($\rightrightarrows\beta$) rule subcase: given $(\lambda y.P)Q \rightrightarrows R$ we must prove $((\lambda y.P)Q)[x{:=}N] \rightrightarrows R[x{:=}N']$. We use the inversion Lemma 4.9 to obtain that there exists P'' such that $\lambda y.P \rightrightarrows \lambda y.P'' \wedge P''[y{:=}Q'] \sim_\alpha R$. Next, as we have assumed the binder y different from x and also fresh in N and N', we can reason as follows:

$$\begin{array}{lll} \lambda y.P \rightrightarrows \lambda y.P'' & \Rightarrow & \{\text{i.h.}\} \\ (\lambda y.P)[x{:=}N] \rightrightarrows (\lambda y.P'')[x{:=}N'] & \Rightarrow & \{\text{Lemma 2.8}\} \\ \lambda y.(P[x{:=}N]) \rightrightarrows \lambda y.(P''[x{:=}N']) & & \end{array}$$

  By the induction hypothesis we know $Q[x{:=}N] \rightrightarrows Q'[x{:=}N']$. So if we prove:

$$P''[x{:=}N'][y{:=}Q'[x{:=}N']] \sim_\alpha R[x{:=}N'] \tag{5}$$

  we will be able to apply the ($\rightrightarrows \beta$) rule and get that $(\lambda y.(P[x{:=}N]))(Q[x{:=}N]) \rightrightarrows R[x{:=}N']$. Then, using the freshness premises, we can pull out the substitution operation on the left side of this parallel reduction, and using the Lemma 4.5, of α-compatibility of $\rightrightarrows$, we finally get the desired result.

  It just remains to prove (5) to end the proof of this subcase. Again, here the classical informal proofs use the BVC. We can also mimic this practice in this case since our induction principle gives us a binder y distinct form x and fresh in N'. Then, we have the freshness premises to successfully apply the substitution composition Lemma 2.9 and conclude this proof in the following steps:

$$\begin{array}{lll} P''[x{:=}N'][y{:=}Q'[x{:=}N']] & \sim_\alpha & \{\text{Lemma 2.9}\} \\ P''[y{:=}Q'][x{:=}N'] & = & \{\text{Lemma 2.5 and } P''[y{:=}Q'] \sim_\alpha R\ \} \\ R[x{:=}N'] & & \end{array}$$

- Abstraction case: we have to prove $(\forall P, y)\ (\forall z \in \{x\} \cup \mathrm{fv}(N) \cup \mathrm{fv}(N'), z \notin_b \lambda y.P) \wedge \Pi(P) \Rightarrow \Pi(\lambda y.P)$. We apply the inversion Lemma 4.9 to the hypothesis $\lambda y.P \rightrightarrows Q$ to get that there exists Q' such that: $P \rightrightarrows Q'$, $\lambda y.P \rightrightarrows \lambda y.Q'$ and $Q \sim_\alpha \lambda y.Q'$. Then, we can conclude the proof in the following way:

$$
\begin{array}{ll}
P \rightrightarrows Q' & \Rightarrow \{\text{ind. hyp.}\} \\
P[x{:=}N] \rightrightarrows Q'[x{:=}N'] & \Rightarrow \{\rightrightarrows \text{ equivariance}\} \\
(y\ z)(P[x{:=}N]) \rightrightarrows (y\ z)(Q'[x{:=}N']) & \Rightarrow \{(\rightrightarrows\lambda) \text{ rule}\} \\
\lambda y.P[x{:=}N] \rightrightarrows \lambda y.Q'[x{:=}N'] & \Rightarrow \{\text{Lemma 2.8}\} \\
(\lambda y.P)[x{:=}N] \rightrightarrows (\lambda y.Q')[x{:=}N'] & \Rightarrow \{\text{Lemma 2.5 and } Q \sim_\alpha \lambda y.Q'\} \\
(\lambda y.P)[x{:=}N] \rightrightarrows Q[x{:=}N'] &
\end{array}
$$

□

In [10], the authors proceed by induction on the relation, so x, N, N' are universally quantified over the β-contraction rule definition, and they are forced to add the same freshness premises that we were able to assume in this proof –by the use of our strengthened α-induction principle– directly into the premises of their modified beta rule of the parallel relation. In contrast, we are performing induction on the term M to prove the predicate Π, and hence we are able to maintain those variables as a fixed context outside the definition of Π. Then by the use of our strengthened α-induction principle we are able to mimic the BVC also in the application case of the proof, specifically in the previously exposed $(\rightrightarrows\beta)$ rule subcase.

Finally, we prove the diamond property of the parallel reduction. Instead of directly proving it by induction on terms (which can easily be done), we will follow the shorter method by Takahashi [9]. For this we first define the "star" operation (Figure 8), such that for any λ-term M, M^* is the result of contracting all the β-redexes existing in M simultaneously. Then we prove that for any terms M, N, if $M \rightrightarrows N$, then $N \rightrightarrows M^*$ (Lemma 4.11). Finally, the diamond property of $\rightrightarrows$ follows directly as a corollary of this result.

$$
\begin{array}{lcl}
x^* & = & x \\
(\lambda x.M)^* & = & \lambda x.M^* \\
(\quad x \qquad M)^* & = & x\ M^* \\
((M_1 M_2)\ M_3)^* & = & (M_1 M_2)^* M_3^* \\
((\lambda x.M_1)\ M_2)^* & = & M_1^*[x := M_2^*]
\end{array}
$$

Fig. 8. Takahashi's star function

Lemma 4.11 (Star property)
$M \rightrightarrows N \ \Rightarrow\ N \rightrightarrows M^*$.

Proof. By structural induction on M. We show the interesting application and abstraction cases.

- Abstraction case: we have to prove that $N \rightrightarrows (\lambda x.M)^* = \lambda x.M^*$, knowing that $\lambda x.M \rightrightarrows N$ holds.
  We can use the inversion Lemma 4.8 on the latter to obtain the existence of the term N' such that: $N \sim_\alpha \lambda x.N'$ and $M \rightrightarrows N'$. We can now apply the induction hypothesis, and then Corollary 4.3 to $M \rightrightarrows N'$, and obtain $\lambda x.N' \rightrightarrows \lambda x.(M^*)$.

This last result directly gives us the desired result by Lemma 4.5 since we know that $N \sim_\alpha \lambda x.N'$.

- Application case: we have three subcases.
  The first two correspond to the third and fourth lines of the star operation definition, and are directly derived from the induction hypotheses.
  Finally, the redex case can be subdivided accordingly to which rule, ($\rightrightarrows$a) or ($\rightrightarrows\beta$), is used in the last step of its parallel reduction:
  · ($\rightrightarrows$a) rule subcase: we have that $\lambda x.M \rightrightarrows N$ and $M' \rightrightarrows N'$, and we need to prove that $NN' \rightrightarrows ((\lambda x.M)M')^* = M^*[x := M'^*]$.
  We begin applying the inversion Lemma 4.8 to the hypothesis $\lambda x.M \rightrightarrows N$ to get that there exists N'' such that $N \sim_\alpha \lambda x.N''$ and $M \rightrightarrows N''$. We can now apply the induction hypothesis to the latter, and then Corollary 4.3 to conclude $\lambda x.N'' \rightrightarrows \lambda x.M^*$. Besides, we can also apply the induction hypothesis to the premise $M' \rightrightarrows N'$ to get $N' \rightrightarrows M'^*$. We can combine the last two inferred parallel reductions, using the ($\rightrightarrows\beta$) rule, and derive that $(\lambda x.N'')N' \rightrightarrows M^*[x := M'^*]$ holds. From this result we directly get the desired result just noticing that $N\,N' \sim_\alpha (\lambda x.N'')N'$, because $N \sim_\alpha \lambda x.N''$ and $N' \sim_\alpha N'$. Hence, by left α-compatibility of the parallel relation (Lemma 4.5) we finish this subcase.
  · ($\rightrightarrows\beta$) rule subcase: we have the following hypotheses: $\lambda x.M \rightrightarrows \lambda y N$, $M' \rightrightarrows N'$ and $N[y := N'] \sim_\alpha P$, and we need to prove that $P \rightrightarrows M^*[x := M'^*]$ holds.
  We proceed analogously to the previous subcase getting that there exists N'' such that $\lambda y.N \sim_\alpha \lambda x.N''$, $N'' \rightrightarrows M^*$ and $N' \rightrightarrows M'^*$. Then we apply Substitution Lemma for $\rightrightarrows$ (Lemma 4.10) to obtain $N''[x := N'] \rightrightarrows M^*[x := M'^*]$, and we can use left α-compatibility Lemma 4.5 to finish the proof if we prove that $P \sim_\alpha N''[x := N']$.
  Finally, we prove that this last alpha equivalence holds:

$$\begin{array}{lll} P & \sim_\alpha & \{\text{hypothesis}\} \\ N[y := N'] & \sim_\alpha & \{\text{by Lemma 2.11 as } x\#\lambda y.N\} \\ ((x\ y)N)[x := N'] & = & \{\text{by Lemma 2.5 as } (x\ y)N \sim_\alpha N''\} \\ N''[x := N'] & & \end{array}$$

  In the previous derivation we used the freshness condition $x\#\lambda y.N$, which follows from $\lambda y.N \sim_\alpha \lambda x.N''$, $x\#\lambda x.N''$, and the fact that freshness is preserved under α-conversion.

□

As a direct consequence of the previous lemma, we have the following result:

Lemma 4.12 (Diamond property of $\rightrightarrows$)
$M \rightrightarrows N \wedge M \rightrightarrows P \;\Rightarrow\; \exists Q, N \rightrightarrows Q \wedge P \rightrightarrows Q.$

Definition 4.13 [Confluence]
A relation is *confluent* if its reflexive and transitive closure has the diamond property.

We omit the proof details of the next results because they do no deal with λ-terms. They are proved in a direct way as in the classical literature.

Lemma 4.14 *If a relation R has the diamond property then it is confluent.*

Lemma 4.15 *If a reduction relation R is confluent, then so is its reflexive and transitive closure R^*.*

As a direct application of the preceding two lemmas, we obtain:

Lemma 4.16 (Confluence of $\rightrightarrows$)
$\rightrightarrows^*$ is confluent.

If we now consider the β-reduction $\rightarrow_\beta$, we have:

Lemma 4.17
$(\rightarrow_\beta \cup \sim_\alpha)^* = \rightrightarrows^*$

Proof. We prove the double inclusion.
To prove $(\rightarrow_\beta \cup \sim_\alpha)^* \subseteq \rightrightarrows^*$, it is enough to prove $(\rightarrow_\beta \cup \sim_\alpha) \subseteq \rightrightarrows$. By Lemma 4.6 we know $\sim_\alpha \subseteq \rightrightarrows$, and $\rightarrow_\beta \subseteq \rightrightarrows$ can be proved by a direct induction on the $\rightarrow_\beta$ reduction relation.
Finally, to prove $\rightrightarrows^* \subseteq (\rightarrow_\beta \cup \sim_\alpha)^*$ we first prove $\rightrightarrows \subseteq (\rightarrow_\beta \cup \sim_\alpha)^*$ by a direct induction on $\rightrightarrows$. Then, by monotonicity of * over $\subseteq$, we get $\rightrightarrows^* \subseteq ((\rightarrow_\beta \cup \sim_\alpha)^*)^*$, and the desired result follows from idempotence of *. □

Using the last two lemmas we finally arrive at the Church-Rosser Theorem.

Theorem 4.18 (Church-Rosser)
The relation $(\rightarrow_\beta \cup \sim_\alpha)$ is confluent.

5 Conclusions

We have introduced principles of induction on terms of the Lambda Calculus that allow us to reason on the abstract terms that arise by identifying α-equivalent concrete ones. The principles work for α-compatible predicates, i.e. properties preserved by α-conversion, and allow us to carry out the corresponding proofs by choosing convenient representatives of the abstract terms in question, namely by avoiding names in binding positions belonging to explicitly provided finite lists. We have derived these principles ultimately from the ordinary structural induction on (concrete) terms. Therefore we have provided a full justification of this form of the Barendregt Variable Convention (BVC). The whole work has been carried out in Constructive Type Theory —and machine-checked using the system Agda— without modifying the ordinary definitional equality of terms or formulating any kind of quotient construction. For the whole implementation to work it has proven essential to define α-conversion in terms of a fundamental operation of name swapping, as in Pitts and Gabbay's Nominal Techniques.

That the method of formalisation can be useful is maybe illustrated by our full formal proof in Agda of the Church-Rosser Theorem. In this paper we have given a summarized description in English language of such proof, although showing

the details we believe essential for the reconstruction of the completely formal version, which is publicly available at https://github.com/ernius/formalmetatheory-nominal-Church-Rosser.

In our development, the definition of the parallel reduction relation has to be formulated in such a way as to ensure that the relation is α-compatible. Because of this, it looks more concrete than the classical one, as presented by Barendregt [1] or Takahasi [9]. However, we are able to prove inversion lemmas that allow us to recover the original parallel reduction definition, and from them we are able to reproduce Takahashi's proof of the diamond property.

In a similar work, Urban and Norrish [10] also have to modify the parallel reduction relation in order to derive an ad-hoc induction principle on the parallel reduction to successfully prove the Substitution Lemma for this relation. However, they do not have to ensure the α-compatibility of the parallel reduction because in their formalisation α-convertible terms are syntactically equal, since they work at the level of terms quotiented by α-equivalence, as explained in [6]. We believe our approach is more direct and general, since we derive an induction principle on simple terms, and not on the more complex relations over them. As shown in the beta subcase of the proof of the Substitution Theorem, we are able to derive the freshness conditions for the binders directly from our α-induction principle, as in the BVC, and not explicitly imposing them in the definition of the parallel reduction.

As another application of our method, we have been able to formalise a proof of the Subject Reduction Theorem for the system of assignment of simple types in a quite direct way, which is also publicly available at the aforementioned site. Finally, we have generalised the techniques here exposed to a framework of regular trees with binders, thereby obtaining pleasant treatments of e.g. metatheory of the System F alongside that of the pure Lambda Calculus by way of instantiation. Report on this work can be found in the first author's PhD thesis [2].

References

[1] Hendrik Barendregt. *The λ-calculus Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North Holland, revised edition, 1984.

[2] Ernesto Copello. *On the Formalisation of the Metatheory of the Lambda Calculus and Languages with Binders*. PhD thesis, PEDECIBA Informática, Uruguay, August 2017.

[3] Ernesto Copello, Álvaro Tasistro, Nora Szasz, Ana Bove, and Maribel Fernández. Alpha-structural induction and recursion for the λ-calculus in constructive type theory. *Electronic Notes in Theoretical Computer Science*, 323:109 – 124, 2016.

[4] Per Martin-Löf. *Intuitionistic type theory*, volume 1 of *Studies in Proof Theory. Lecture Notes*. Bibliopolis, Naples, 1984. Notes by Giovanni Sambin.

[5] Ulf Norell. *Towards a Practical Programming Language Based on Dependent Type Theory*. PhD thesis, Department of Computer Science and Engineering, Chalmers University of Technology, September 2007.

[6] Michael Norrish. Mechanising λ-calculus using a classical first order theory of terms with permutations. *Higher-Order and Symbolic Computation*, 19(2):169–195, 2006.

[7] Andrew M. Pitts. Nominal Logic, a First 0rder Theory of Names and Binding. *Information and Computation*, 186(2):165–193, 2003.

[8] Andrew M. Pitts. Alpha-Structural Recursion and Induction. *Journal of the ACM*, 53(3):459–506, May 2006.

[9] M. Takahashi. Parallel Reductions in λ-Calculus. *Information and Computation*, 118(1):120 – 127, 1995.

[10] Christian Urban and Michael Norrish. A formal treatment of the Barendregt variable convention in rule inductions. In *Proceedings of the 3rd ACM SIGPLAN Workshop on Mechanized Reasoning About Languages with Variable Binding*, MERLIN '05, pages 25–32, New York, NY, USA, 2005. ACM.