Ttulo?

Ernesto Copello Álvaro Tasistro Nora Szasz
Universidad ORT Uruguay
copello,tasistro,szasz@ort.edu.uy

Abstract

We formulate principles of induction and recursion for simply typed lambda calculus with bound names where α-conversion is based upon name swapping as in nominal abstract syntax. The principles allow us to work modulo α-conversion and formally reproduce the Barendregt variable convention. We successfully apply those induction principles to get some fundamental meta-theoretical results, such as the Church-Rosser theorem and the Subject Reduction theorem for the system of assignment of simple types. The whole development has been machine-checked using the system Agda.

Categories and Subject Descriptors CR-number [*subcategory*]: third-level

Keywords Formal Metatheory, Lambda Calculus, Constructive Type Theory.

1. Introduction

There is great interest in the use of proof assistants for formalising definitions of programming languages and checking proofs of their properties. However, despite of the amount of work in this area, the community remains fragmented in the basic issue of how to represent the abstract syntax of programming languages, and in particular, how to represent bound variables. There exists a collection of different name bindings representation techniques, with no one coming up as a clear optimal solution. Aydemir (Aydemir et al. 2005) proposed the POPLMARK challenge, a set of tasks designed to evaluate many of the critical issues that commonly arise in the formalisations of programming languages. They propose three main criteria for evaluating proposed formalisations:

- *formalisation overheads*: operations and proof obligations introduced by the technique and not related to the original problem.
- *transparency*: formal definitions and theorems should not depart radically from the usual informal conventions.
- *cost of entry*: the infrastructure should be usable by someone who is not an expert in theorem prover technology.

Our line of research goes after those criteria, looking for a formalisation as close to pencil-and-paper informal practices as possible while remaning formal. The usual informal procedure in the formalisation of programming languages consists in identify terms up to α-conversion. However, this is not simply carried out when functions are defined by recursion and properties proven by induction over abstract syntax. The problem has to do with the fact that the consideration of the α-equivalence classes is in general actually conducted through the use of convenient representatives thereof. These are chosen by the so-called *Barendregt Variable Convention* (BVC (Barendregt 1985)): each term representing its α-class is assumed to have bound names all different and different from all names free in the current context. This usual informal practice collides with the primitve induction principles automatically derived from the abstract syntax by nowadays proof assistants, which requiries proofs obligations quantified over arbitrary terms, not over α-equivalence classes, or as in the informal practice, over convenient representatives of terms.

There exist formalisations where this problem is avoided, as in the DeBruijn nameless syntax (de Bruijn 1972), or its more up-to-date version *locally nameless* syntax (Aydemir et al. 2008; Charguéraud 2012), which uses names for the free or global variables and indices counting up to the binding abstractor for the occurrences of local parameters. But these methods are not without formalisation overheads in the form of several operations or well-formedness predicates introduced by this representation. As a result, these syntax seriously affects the connection between actual formal procedures and informal parctices.

A continuous line of work during the last decade has been the formalisation of induction/recursion principles over α-equivalence classes of lambda terms, trying to capture the BVC convention.

In one of the first works in this line, Gordon (Gordon 1993) proposes the induction principle shown in figure 1. This principle allows to conveniently choose the abstraction binder a not in a predefined finite set of variables X in the abstraction case of an inductive proof over terms.

$$\frac{\begin{array}{ll}\forall a, P(a) & \wedge \\ \forall M\ N, P(M) \wedge P(N) \Rightarrow P(MN) & \wedge \\ \forall M\ a,\ a \notin X, P(M) \Rightarrow P(\lambda a.M) & \end{array}}{\forall M, P(M)}$$

Figure 1. Alpha induction principle

Although in this principle variables appear in binder positions, Gordon uses a *locally nameless* syntax in his development, managing to hide De Bruijn notation behind some succinct set of lemmas. The main property of this syntax is that alpha convertible terms are syntactically equal. As a consequence of this, in the abstraction case, proposition P can be proved for any representative of the equivalence class. The use of indices for bound variables introduces invalid terms, and a well-formedness predicate is needed to exclude ill terms from the formalisation. Besides, every function introduced must be proved to be closed under well-formed terms, and well-formed hypotheses must be added to all proofs and relations. On the other hand, the main advantage of this mixed strategy is that theorems can be expressed in a conventional form, without showing the De Bruijn encoding, and in spite of this, the renaming of bound variables for fresh ones is still supported in proofs, because syntactical equality is up to alpha conversion. Anyway, when renaming has to be done to pick another representative of an alpha equivalence class, the classical primitive induction hypothesis does not have any information about the new renamed sub-term, becoming necessary in general to do induction over the length of terms. To overcome this overhead, Gordon introduces the presented alpha induction principle for decidable predicates, which, as expected, is proved by induction on the length of De Bruijn's terms, and it is used when renaming is neccesary. As an example of this methodology, some substitution lemmas from Hindley and Seldin's book (Hindley and Seldin 1986) are directly derived using the BVC induction principle, without using theorems about the underlying De Bruijn representation, neither exposing the internal renaming done to select fresh variables.

In (Gordon and Melham 1996), they continue working in this formalisation, and present a way to define functions over lambda terms without any overhead. They do so by introducing an iteration principle over lambda terms. While the previous approach was developed in a *first-order* encoding –in the sense that the variable-binding operations of the embedded syntax is distinct form the meta variable-binding– this work introduces a kind of *second-order* approach, where a typical abstraction expression $Lam\ x\ u$ can be obtained from a meta-level abstraction expression $\lambda y.u[x{:=}y]$. For this, they use a function $abs : (variables \rightarrow terms) \rightarrow terms$, that is, any meta-level function from variables to terms represents a lambda abstraction in the embedded language. The key of their development is a proof of the existence of a theoretical model for the abs function, but they do not give explicitely a computable one. The paper does not give much more insight about the codification of abs in their formalisation. So this requirement is not easy to evaluate, and neither is to deduce how feasible it is to transfer it to a Constructive Type Theory environment.

Gabbay-Pitts (Gabbay and Pitts 2002) introduce a general theory, called *nominal approach*, to deal with issues of bound names and alpha equivalence classes in any abstract syntax. They consider constructions and properties that are invariant with respect to permutation of names, and give principles of recursion/induction over the alpha equivalence classes defined by the abstract syntax with bindings. At the base of their theory is the notion of *finite supported* mathematical objects, which gives a general way of expressing the fact that atoms are fresh for mathematical objects, in terms of name-permutations. This notion allows them to extend the concept of *fresh names* from finite objects (as abstract syntax trees) to infinite ones, as infinite sets and functions.

Continuing Gordon and Melham's work, Norrish (Norrish 2004) introduces a method to define functions in a much more familiar way, approximating it to the classic principles of primitive recursion. For this, he uses some ideas of Gabbay-Pitts nominal approach, introducing the swapping of names operation as a basics for syntax with binders. The swapping operation has nicer properties than the one of renaming as it defines an injection over terms. The introduced iteration principle comes with proof obligations for each auxiliary function used to instantiate its cases. One has to prove that these auxiliary functions are finitely supported, that is, that they do not create too many fresh variables, and that they behave well through the swapping operation (that is, for any auxiliary function f, term M and variables x, y, $swap(x, y, f(M)) \equiv f(swap(x, y, M))$.

Urban and Tasson (Urban and Tasson 2005) use more deeply the Gabbay-Pitts theory to construct an induction principle based on the one introduced by Gordon, introducing the concept of finite support of nominal sets to state the freshness conditions. They also abandon the De Bruijn notation, as in (Gordon and Melham 1996), losing syntactical equality of alpha compatible terms, and need to introduce an alpha conversion relation. They define the quotient set over this relation using a weak HOAS (Despeyroux et al. 1995). Besides, they have to define the substitution operation because weak HOAS does not use meta-level substitution. Anyway, they have an alpha induction principle for "free" over the defined quotient set of terms. They prove the composition lemma for substitutions as an example of use of their induction principle.

As in the previous works, we use lambda calculus as a minimal programming language in which we can validate formal reasoning techniques over abstract syntax trees with name bindings. In our previous work (Copello et al. 2016), we follow Gordon's ideas, but using a named first-order syntax. We presented an alpha induction principle, equivalent to the one defined by Gordon. Our principle enables us to carry out the proof of the lambda-abstraction case by choosing a bound name different from the names in a given list of names, emulating the BVC convention. But, as we do not work over a quotient sets, neither we have syntactical equality of alpha convertible terms, our induction principle requires the property being proved to be α-*compatible*, that is, it must be preserved by the α-conversion relation.

From our induction principle we also directly derive an *alpha iteration* principle. This principle does not have any side-condition properties to be verified, and the result of the iteration is what we call *strong alpha compatible*, that is, the result of applying these principle to alpha convertible terms is syntactically equal. This issue becomes an important one because our approach is computable instead of being a logic framework as (Urban and Tasson 2005). ??

The present work complements and re-validates our previous work, scaling it up to the Church-Rosser and Subject Reduction theorems. The main novelity of this work is the realisation that the alpha induction principle, commonly proposed in the literature to capture the BVC convention fails in its objective when it is used to prove the Church-Rosser theorem, specifically in the proof that the β-reduction relation is preserved by the substitution operation. This happens because when a β-contraction is applied, we have to apply the substitution operation to a contractum $(\lambda y.Q)P$ but in this case we are in the application case of the induction, so the alpha induction principle gives no freshness information about the variable y.

We present a novel principle which succesfully addresses this issue. This is the first detailed publication of a formalistation of the Church-Rosser and Subject Reduction theorems based upon name swapping as in nominal abstract syntax, formally reasembling the BVC convention in the formalisation by the use of an alpha induction principle.

Specifically, concerning β-reduction, we prove three so-called *substitution lemmas* showing that substitution is compatible with α-conversion, parallel β-reduction, and typing in the system of simple type assignment (a.k.a. simply typed Lambda calculus à la Curry).

All the definitions and proofs have been fully formalised in Constructive Type Theory (Martin-Löf and Sambin 1984) and machine-checked employing the system Agda (Norell 2007). The corresponding code is public available at:

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

In the subsequent text we give the proofs in English with a considerable level of detail so that they serve for clarifying their formalisation.

The structure of the paper is as follows: in section 2 we begin by introducing the syntax of the lambda calculus, swapping operation, basic relation, and substitution properties. Section 3 introduces two new alpha induction principles. In Section 4 we introduce 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 (Barendregt 1984; Takahashi 1995). The overall conclusions are exposed in section 5.

2. Preliminaries

In this section we will address some results form our previous work in (Copello et al. 2016) that are necessary for a proper comprehension of the material exposed in present work.

2.1 First Order Nominal Syntax

We define the set terms as usual. The variables belong to a denumerable set of names.

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

We can constructively define the *fresh* relation certifiying a variable (name) does not occur free in a term.

$$\frac{b \neq a}{a\#b} \qquad \frac{a\#M \qquad a\#N}{a\#MN} \qquad \frac{a\#M}{a\#\lambda b.M} \qquad a\#\lambda a.M$$

To reproduce the BVC convention in the alpha induction principle we will present in next section, we need to define when a variable does not occur in a bound position in a term, for this purpose we present the next relation. Its syntax directed scheme will became very usefull in the proof of the presented induction principle.

$$x \notin_b y \qquad \frac{x \notin_b M \qquad x \notin_b N}{x \notin_b MN}$$

$$\frac{x \neq y \qquad x \notin_b M}{x \notin_b \lambda y M}$$

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 swap is first defined on names themselves:

$$(a\ b)\ c\ = \begin{cases} b & \text{if } a = b \\ a & \text{if } b = c \\ c & \text{if } a \neq b \wedge b \neq c \end{cases}$$

The swapping operation is directly extended to terms by swapping all names occuring in a term, including abstraction positions. The *permutation* operation is just the sequential

application of a list of swaps, usually denoted as π, and is denoted by prefixing any term M with a list of swaps, as in πM.

In figure 2 we introduce a syntax-directed definition of α-conversion ($\sim_\alpha$) based on the swapping operation. This definition departs from classic ones only in the abstraction case: for proving that two abstractions are α-equivalent we should be able to prove that the respective bodies are α-equivalent when we rename the bound names to any name not free in both abstractions. The freshness condition on the new name is generalised to "any name not in a given list".This condition is harder to prove, but more convenient to use when we assume $\sim_\alpha$ to hold, which is more often the case in proofs. In our previous work we prove that this is an equivalence relation, preserved by the swapping operation, what it is usually called in nominal literature as *equivariant*.

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

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

Figure 2. Alpha equivalence relation

2.2 Substitution Operation

In (Copello et al. 2016) we derive induction and iteration principles with stronger hypotheses for the lambda abstraction case, namely enabling us to choose a bound name different from the names in a given list xs, as in figure 1. One important point in this implementation is that functions defined usin the iteration principle yield the same result for α-equivalent terms. We call these functions *strongly* α-compatible.

The susbstitution operation is defined using the derived iteration principle, and as a direct consequence of being defined with our iterator the following lemma is automatically derived.

Lemma 1 (Substitution is a strongly α-compatible operation)**.**

$$M \sim_\alpha M' \Rightarrow M[x := N] \equiv M'[x := N]$$

The follwing results are succesfully proved using our induction principles. In general, works on alpha induction principles reache until the next substitution composition lemma.

Lemma 2 (Substitution preserves α-conversion)**.**

$$N \sim_\alpha N' \Rightarrow M[x := N] \sim_\alpha M[x := N']$$

Lemma 3 (Substitution under permutation)**.**

$$\pi\ (M[x := N]) \sim_\alpha (\pi\ M)[\pi\ x := \pi\ N]$$

Next lemma shows that substitution commutes with abstraction without any renaming in a naïve way. This is because the freshness hypothesis grants a sufficient fresh binder.

Lemma 4.
If $x \neq y \wedge x\#L$, then:

$$(\lambda x.M)[y := N] \sim_\alpha \lambda x.(M[y := N])$$

Lemma 5 (Substitution composition)**.**
If $x \neq y \wedge x\#L$, then:

$$M[x := N][y := P] \sim_\alpha M[y := P][x := N[y := P]]$$

From now on the presented material is original.

Lemma 6 (Swapping substitution variable)**.**

$$x\#M \Rightarrow ((x\ y)M)[x := N] \sim_\alpha M[y := N]$$

Proof. We use the classic alpha induction principle presented in figure 1.

First for arbitrary names x, y and term N we define the following predicate over terms:

$$Q(M) \equiv x\#M \Rightarrow (x\ y)M[x := N] \sim_\alpha M[y := N]$$

We prove Q is α-compatible, that is, if $M \sim_\alpha P$ and $Q(M)$, then $Q(P)$. We can prove $x\#P$ as freshness is preserved through $\sim_\alpha$, and then we can prove $Q(P)$ as follows:

$$\begin{array}{rcl} ((x\ y)P)[x := N] & \equiv & \{\text{subst. lemma 1}\} \\ ((x\ y)M)[x := N] & \sim_\alpha & \{Q(M) \text{ and } x\#M\} \\ M[y := N] & \equiv & \{\text{subst. lemma 1}\} \\ P[y := N] & & \end{array}$$

Next, we prove the interesting abstraction case of the induction.

We have $x\#\lambda z.M$ and we can choose $z \notin \{x, y\} \cup fv(N)$.

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:

$$\begin{array}{rcl} ((x\ y)(\lambda z.M))[x := N] & = & \{\text{def. of swap}\} \\ (\lambda((x\ y)z).((x\ y)M))[x := N] & = & \{z \notin \{x, y\}\} \\ (\lambda z.((x\ y)M))[x := N] & \sim_\alpha & \{\text{lemma 4}\} \\ \lambda z.(((x\ y)M)[x := N]) & \sim_\alpha & \{\text{i.h.}\} \\ \lambda z.(M[y := N]) & \sim_\alpha & \{\text{lemma 4}\} \\ (\lambda z.M)[y := N] & & \end{array}$$

□

The previous proof exemplifies the common informal practice in pen-and-paper proofs, which basically uses BVC convention to assume enough fresh binders from some defined context, allowing us to apply the substitution operation in a naïve way without any variable capture problem.

Next result is a direct consequence of previous lemma.

Corollary 1.

$$x\#\lambda y.M \Rightarrow ((x\ y)M)[x := N] \sim_\alpha M[y := N]$$

3. Alpha Induction Principles

In this section we introduce two new alpha induction principles. The first one in figure 3 is a renaming version of the previously presented one, where in the hypothesis of the abstraction case a term permutation is allowed. This principle is derived in a similar way to previous one, but in this case we allow some extra explicit renaming in addition to internal one automatically done to exclude not enough fresh binders. This variation is useful to do induction over terms and deal with relations definitions which make use of the permutation operation in its premises. We will show an example of this situation in the lemma 10 in next section.

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

Figure 3. Alpha permutation induction principle

As in BVC convention, the second induction principle in figure 4 enables us to assume bound variables fresh enough from a given finite list of variables xs through the entire induction, and not only for the abstraction case as in classic induction principles.

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

Figure 4. New alpha induction principle

To derive this principle we introduce the $fresh$ function that, given a list of names and a term, returns a term alpha-compatible with the original one, and satsifying that all elements of the given list do not ocurr bound in it. To prove $P(M)$ for any term M, our alpha induction principle works as follows, given a list of variables xs, an alpha compatible predicate P, and the following proofs:

$$\begin{array}{l} \forall a, P(a) \\ \forall M\ N, (\forall b \in xs, b \notin_b MN) \wedge P(M) \wedge P(N) \Rightarrow P(MN) \\ \forall M\ a, (\forall b \in xs, b \notin_b \lambda a.M) \wedge P(M) \Rightarrow P(\lambda a.M) \end{array} \quad (1)$$

it proves by primitive induction on terms the following Q predicate:

$$Q(M) = ((\forall x, x \in xs \Rightarrow x \notin_b M) \Rightarrow P(M)) \quad (2)$$

Then, it applies the proof of predicate Q to the term $fresh(xs, M)$ and the proof that this term have no bound variables in xs to get $P(fresh(xs, M))$. Finally, as P is alpha-compatible and $fresh(xs, M) \sim_\alpha M$ we can get that $P(M)$ holds for any M.

Next, we do primitive induction on term M to prove Q predicate.

Proof.

- var. case: Direct.
- app. case: We need to prove $Q(MN)$ for any M, N, such that $Q(M)$ and $Q(N)$ hold. So expanding $Q(MN)$ definition (2), we have to prove $P(MN)$, given that any x variable in xs satisfies that $x \notin_b MN$. Then by the syntax directed definition of $\notin_b$ relation, we directly derive that $x \notin_b M$ and $x \notin_b N$. We can use those results with the induction hypothesis $Q(M)$ and $Q(N)$ to get $P(M)$ and $P(N)$ hold. We have all the premises in the second predicate in (1), so finally its conclusion $P(MN)$ is valid.
- abs. case: We must prove $Q(\lambda y.M)$, that is, we need to prove $P(\lambda y.M)$ knowing that for any variable x in xs satisfies that $x \notin_b \lambda yM$, then by $\notin_b$ definition $x \neq y$ and $x \notin_b M$. We apply the last result to the inductive hypothesis $Q(M)$ to get $P(M)$. Finally, we get the desired result using the third assertion in (1).

□

4. Parallel Beta-reduction

β-reduction can be defined as the reflexive, transitive and compatible with the syntactic constructors, clousure of the β-contraction $(\lambda x.M)N \triangleright M[x := N]$. The classic 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 the definition in figure 5.

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

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

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

Figure 5. Parallel reduction relation

The first three rules correspond to the ones of the α-conversion relation presented in figure 2, and evidence we are defining a relation over the alpha equivalence classes of terms. The β-rule needs an α-conversion in its premises because our substitution operation normalises terms, as a consequence of being defined with our alpha iteration principle. Next we prove the parallel reduction relation is equivariant.

This is a basic property in nominal settings, establishing every "well defined" concept should not depend in any particular choice of names (names should be interchangable).

Lemma 7 (Parallel Relation is equivariant)**.**

$$M \rightrightarrows N \Rightarrow \pi M \rightrightarrows \pi N$$

Proof. By induction on the parallel relation. The variable and application rules of the parallel relation are direct. For the abstraction rule, we have to prove $\lambda(\pi\ x).(\pi\ M) \rightrightarrows \lambda(\pi\ y).(\pi\ N)$ with $\lambda x.M \rightrightarrows \lambda y.N$ as hypothesis. The only rule with an abstraction as an outer constructor in its conclusions is $(\rightrightarrows \lambda)$, so its premise should hold for any z not in some list of variables xs. We can also exclude the domain of the permutation π from the range of variable z, and reason as follows:

$$\begin{array}{ll}
(x\ z)M \rightrightarrows (y\ z)N & \Rightarrow \{\text{i.h.}\} \\
\pi((x\ z)M) \rightrightarrows \pi((y\ z)N) & \Rightarrow \\
((\pi\ x)\ (\pi\ z))(\pi\ M) \rightrightarrows ((\pi\ y)\ (\pi\ z))(\pi\ N) & \Rightarrow \{z \notin dom(\pi)\} \\
((\pi\ x)\ z)(\pi\ M) \rightrightarrows ((\pi\ y)\ z)(\pi\ N) & \Rightarrow \{\rightrightarrows \lambda \text{ rule}\} \\
\lambda(\pi\ x).(\pi\ M) \rightrightarrows \lambda(\pi\ y).(\pi\ N) &
\end{array}$$

In the $(\rightrightarrows\beta)$ case we must prove:

$$(\lambda(\pi\ x).(\pi\ M))(\pi\ N) \rightrightarrows \pi\ P$$

The hypothesis are: $\lambda x.M \rightrightarrows \lambda y.P'$, $N \rightrightarrows P''$ and $P \sim_\alpha P'[y := P'']$. By direct applicaton of the induction hypothesis to the first two premises we directly get:

$$\begin{array}{r}
\lambda(\pi\ x).(\pi\ M) \rightrightarrows \lambda(\pi\ y).(\pi\ P') \\
\pi\ N \rightrightarrows \pi\ P''
\end{array} \tag{3}$$

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

$$\begin{array}{ll}
P \sim_\alpha P'[y := P''] & \Rightarrow \{\sim_\alpha \text{ equivariant}\} \\
\pi\ P \sim_\alpha \pi\ (P'[y := P'']) & \Rightarrow \{\text{lemma 3}\} \\
\pi\ P \sim_\alpha (\pi\ P')[(\pi\ y) := (\pi\ P'')] & \Rightarrow \{\sim_\alpha \text{ symmetric}\} \\
(\pi\ P')[(\pi\ y) := (\pi\ P'')] \sim_\alpha \pi\ P &
\end{array}$$

We obtain the desired result using the $(\rightrightarrows\beta)$ rule with (3) and the last result as premises. □

HASTA ACA LLEGUE

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 8 ($\rightrightarrows$ is right α-equivalent)**.**

$$M \rightrightarrows N \wedge N \sim_\alpha P \Rightarrow M \rightrightarrows P$$

Lemma 9 ($\rightrightarrows$ is left α-equivalent)**.**

$$M \sim_\alpha N \wedge N \rightrightarrows P \Rightarrow M \rightrightarrows P$$

As the parallel reduction relation basically apply β-contractions, so no free variable should be introduced at each relation step, therefore freshness should be preserved.

Lemma 10 ($\rightrightarrows$ preserves freshness)**.**

$$x\#M \wedge M \rightrightarrows N \Rightarrow x\#N$$

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

$$P(M) = \forall N, x\#M \wedge M \rightrightarrows N \Rightarrow x\#N$$

As the fresh and parallel relations are α-compatible the P predicate is also α-compatible. Now we do the induction on term M, only showing the interesting abstraction case. We have that $x\#\lambda y.M \wedge \lambda y.M \rightrightarrows \lambda z.N$ and we must prove $x\#\lambda z.N$. The parallel relation hypothesis, by its syntax directed definition, must be the result of an application of the $(\rightrightarrows\lambda)$ rule, so we get its premise $\forall w, w \notin xs, (y\ w)M \rightrightarrows (z\ w)N$. The alpha induction principle allow us to exclude some variables for the abstraction case, so we can also assume $x \neq y$. Using this inequality and the hypothesis $x\#\lambda y.M$ we get by definition that $x\#M$ holds. Now let be a variable u such that $u\#N, u \notin xs$ and $u \neq x$, then $x\#(y\ u)M$ because $x \neq y, u$ and $x\#M$. We can apply the premise of $(\rightrightarrows\lambda)$ rule with u, as $u \notin xs$, and 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 α-conversion relation preserves freshness we end the proof. □

We have proved our relation is well beahaved with respect alpha conversion and the permutation operation, we now introduce next lemmas to prove that our parallel relation implies the next more classical one.

$$\rightrightarrows\text{v}\ \frac{}{x \rightrightarrows x} \qquad \rightrightarrows\text{a}\ \frac{M \rightrightarrows M' \qquad N \rightrightarrows N'}{MN \rightrightarrows M'N'}$$

$$\rightrightarrows\lambda\ \frac{M \rightrightarrows M'}{\lambda x.M \rightrightarrows \lambda x.M'}$$

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

Lemma 11 ($\rightrightarrows$ λ-elimination)**.**

$$\lambda xM \rightrightarrows M' \Rightarrow \begin{array}{l} \exists M'', M \rightrightarrows M'', \\ M' \sim_\alpha \lambda xM'', \lambda xM \rightrightarrows \lambda xM'' \end{array}$$

Proof. We prove exists $M'' = (y\ x)M'$ that satisfies the thesis. As $\lambda x.M \rightrightarrows M'$, we have that M' is in an abstraction $\lambda y.M'$, and exists a list of variables xs such that $\forall z \notin xs, (x\ z)M \rightrightarrows (y\ z)M'$. Next we prove M'' satifies the three thesis conjunctions:

- Be z such that $z \notin xs$ and $z\#\lambda y.M'$. Trivially $x\#\lambda xM$ so as parallel relation preserves freshness $x\#\lambda y.M'$,

then $(x\ z)(y\ z)M' \sim_\alpha (x\ y)M'$. We can now follow next deduction:

$$\begin{array}{lll} (x\ z)M \rightrightarrows (y\ z)M' & \Rightarrow & \{\rightrightarrows\text{equivariant}\} \\ (x\ z)(x\ z)M \rightrightarrows (x\ z)(y\ z)M' & \Rightarrow & \{\text{swap idempotent}\} \\ M \rightrightarrows (x\ z)(y\ z)M' & \Rightarrow & \{\text{lemma 8}\} \\ M \rightrightarrows (x\ y)M' & & \end{array}$$

- As x is fresh in $\lambda y.M'$, we can swap y by x in this term and get the alpha equivalent $\lambda x.(x\ y)M'$ term.
- We can apply lemma 8 with $\lambda x.M \rightrightarrows \lambda y.M'$ hypothesis and previous α-equivalence assertion to prove the last conjunction.

□

Lemma 12 ($\rightrightarrows$ β-elimination)**.**

$$(\rightrightarrows \beta)\ \frac{\begin{array}{c}\lambda x M \rightrightarrows \lambda y M', \\ N \rightrightarrows N', \\ M'[y := N'] \sim_\alpha P\end{array}}{(\lambda x M)N \rightrightarrows P} \Rightarrow \begin{array}{l}\exists M'', \\ \lambda x M \rightrightarrows \lambda x M'', \\ M''[x := N'] \sim_\alpha P)\end{array}$$

Proof. We prove $M'' = (y\ x)M'$ satisfies the existential conjunction condition.

- $x\#\lambda x.M$ and $\lambda x M \rightrightarrows \lambda y M'$ so by lemma 10 $x\#\lambda y M'$, we can then swap y by x in last term, and obtain the next alpha equivalent term $\lambda x.(y\ x)M'$. Appling lemma 8 we get the first condition.
- For the last conjunction condition we reason as follows:

$$\begin{array}{ll} ((y\ x)M')[x := N'] & = \{\text{swap conmutative}\} \\ ((x\ y)M'))[x := N'] & \sim_\alpha \{\text{corollary 1}\} \\ M'[y := N'] & \sim_\alpha \{\text{hypothesis}\} \\ P & \end{array}$$

□

Theorem 1 ($\rightrightarrows$ substitution lemma)**.**

$$M \rightrightarrows M' \wedge N \rightrightarrows N' \Rightarrow M[x := N] \rightrightarrows M'[x := N']$$

The parallel substitution theorem 1 is the crux of Church-Rosser theorem, and where the classic alpha induction principles fails in capturing the BVC convention. The problem appears in the beta application case of an induction over term M. For example, we need to prove $((\lambda y P)Q)[x := N] \rightrightarrows R[x := N']$, but as we are in the application case, not the abstraction one, the usual alpha induction principle gives no freshness information about y variable. While the common use of BVC would allow us to choose y not equal x and fresh in N to be able to push the substitution inside the abstraction without any variable capture. Besides the following proof also exemplifies the use of previous elimination lemmas. Next, we use our propused alpha induction principle to prove this result.

Proof. Given some N, N' terms such that $N \rightrightarrows N'$, and x atom, we define the following predicate over terms:

$$P(M) \equiv \forall M', M \rightrightarrows M' \Rightarrow M[x := N] \rightrightarrows M'[x := N']$$

We must prove that P is α-compatible, that is, $P(M) \wedge M \sim_\alpha N \Rightarrow P(N)$, which is direct substitution operation is strong α-compatible and the parallel relation is α-compatible .

We can then use the alpha induction principle presented in 4 to prove P by induction over term M, excluding the x variable, and the free variables in N and N', from the binders in M term.

We prove next the interesting application and abstraction cases.

- app. case: we have to prove the application case of our induction principle: $\forall P\ Q, (\forall b \in \{x\} \cup fv(N) \cup fv(N'), b \notin_b PQ) \wedge P(P) \wedge P(Q) \Rightarrow P(PQ)$. We have two subcases according to what parallel rule it is used.
  - ($\rightrightarrows$a) rule subcase: we have that $P \rightrightarrows P'$ and $Q \rightrightarrows Q'$ and we have to prove that $(PQ)[x := N] \rightrightarrows (P'Q')[x := N']$. The proof is a direct application of $(\rightrightarrows a)$ rule to the induction hypothesis results.
  - ($\rightrightarrows\beta$) rule subcase: given $(\lambda y P)Q \rightrightarrows R$ we must prove $((\lambda y.P)Q)[x := N] \rightrightarrows R[x := N']$. We use lemma 12 to get that exists P'' such that $\lambda y.P \rightrightarrows \lambda y.P'' \wedge P''[y := Q'] \sim_\alpha R$. Next, like when we use the BVC convention, we can asume the binder y different fromm x and also fresh in N and N', allowing us to reason as follows:

$$\begin{array}{ll} \lambda y.P \rightrightarrows \lambda y.P'' & \Rightarrow \{\text{ind. hyp.}\} \\ (\lambda y.P)[x := N] \rightrightarrows (\lambda y.P'')[x := N'] & \Rightarrow \{\text{lemma 4}\} \\ \lambda y.P[x := N] \rightrightarrows \lambda y.P''[x := N'] & \end{array}$$

By the induction hypothesis we also know:

$$Q[x := N] \rightrightarrows Q'[x := N']$$

So if we prove that:

$$P''[x := N'][y := Q'[x := N']] \sim_\alpha R[x := N'] \tag{4}$$

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 we can pull out the substitution operation in the left side of this relation, and using the lemma 9 we will get the desired result.

Only remains to prove 4 to end the proof. Again, here the classic informal proofs use the BVC convention. We can mimic this common practice as our induction principle gives us a binder y distinct form x and fresh

in N', so we can succesfully apply the substitution composition lemma 5 in the following steps.

$$\begin{array}{ll} P''[x := N'][y := Q'[x := N']] & \sim_\alpha \{\text{lemma 5}\} \\ P''[y := Q'][x := N'] & \equiv \{\text{lemma 1}\} \\ R[x := N'] & \end{array}$$

- abs. case: we must prove $\forall M\ a,\ (\forall b \in \{x\} \cup fv(N) \cup fv(N'), b \not\in_b \lambda aM) \wedge PM \Rightarrow P(\lambda aM)$. We begin appling lemma 12 to $\lambda yP \rightrightarrows Q$ hypothesis to get that exists Q' such that: $P \rightrightarrows Q'$, $\lambda yP \rightrightarrows \lambda yQ'$ and $Q \sim_\alpha \lambda yQ'$. Then we can end the proof in the following way:

$$\begin{array}{ll} P \rightrightarrows Q' & \Rightarrow \{\text{ind. hyp.}\} \\ P[x := N] \rightrightarrows Q'[x := N'] & \Rightarrow \{\rightrightarrows\text{equiv.}\} \\ (x\ y)(P[x := N]) \rightrightarrows (x\ y)(Q'[x := N']) & \Rightarrow \{(\rightrightarrows\lambda)\text{ rule}\} \\ \lambda y.P[x := N] \rightrightarrows \lambda y.Q'[x := N'] & \Rightarrow \{\text{lemma 4}\} \\ (\lambda yP)[x := N] \rightrightarrows (\lambda y.Q')[x := N'] & \Rightarrow \{\text{lemma 1}\} \\ (\lambda yP)[x := N] \rightrightarrows Q[x := N'] & \end{array}$$

□

Finally, we can prove the diamond property of parallel reduction by a simple induction on term.

Lemma 13 (Diamond property of parallel relation)**.**

$$M \rightrightarrows N \wedge M \rightrightarrows P \Rightarrow \exists Q, N \rightrightarrows Q \wedge P \rightrightarrows Q$$

Proof. By induction on term M. We show next the more interesting case, the rest are direct or similar to the detailed case.

- app. case: We do case analysis in both parallel relations, the interesting subcase is the combination of $(\rightrightarrows\ a)$ and $(\rightrightarrows\ \beta)$ rules, so our hypothesis are:

$$\rightrightarrows\text{a}\ \frac{\lambda xM \rightrightarrows N' \qquad M' \rightrightarrows N''}{(\lambda xM)M' \rightrightarrows N'N''}$$

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

And we have to prove: $\exists Q, N'N'' \rightrightarrows Q \wedge P \rightrightarrows Q$. We can now apply the abstraction elimination lemma 11 to $\lambda xM \rightrightarrows N'$, and the beta elimination 12 to $(\lambda xM)M' \rightrightarrows P$ to get the following results:

$$\begin{array}{l} \exists N''', M \rightrightarrows N''' \wedge \lambda xM \rightrightarrows \lambda xN''' \wedge N' \sim_\alpha \lambda xN''' \\ \exists P''', \lambda xM \rightrightarrows \lambda xP''' \wedge P'''[x := P''] \sim_\alpha P \end{array}$$

As $\lambda xM \rightrightarrows \lambda xN'''$ and $\lambda xM \rightrightarrows \lambda xP'''$ by inductive hypothesis we get:

$$\exists Q, \lambda xN''' \rightrightarrows Q \wedge \lambda xP''' \rightrightarrows Q$$

Next results are results of the direct applcation of the parallel abstraction elimination lemma 11 to each element of previous conjunctions.

$$\begin{array}{l} \exists S', N''' \rightrightarrows S' \wedge \lambda xN''' \rightrightarrows \lambda xS' \wedge Q \sim_\alpha \lambda xS' \\ \exists S'', P''' \rightrightarrows S'' \wedge \lambda xP''' \rightrightarrows \lambda xS'' \wedge Q \sim_\alpha \lambda xS'' \end{array}$$

By symmetry and transitive properties of $\sim_\alpha$ we get $\lambda xS' \sim_\alpha \lambda xS''$ from what it can be derived that $S'' \sim_\alpha S'$.
We can use the remaining inductive hypothesis with $M' \rightrightarrows P'$ and $M' \rightrightarrows P''$ and get:

$$\exists R, P' \rightrightarrows R \wedge P'' \rightrightarrows R$$

Next we prove that $S'[x := R]$ is the confluent term we are looking for, that is, it satifies the following assertions:

$$N'N'' \rightrightarrows S'[x := R] \tag{5}$$

$$P \rightrightarrows S'[x := R] \tag{6}$$

For (5), we have that $\lambda x.N'' \rightrightarrows \lambda x.S'$ and $N'' \rightrightarrows R$, and by the reflexivity of $\sim_\alpha$ relation $S'[x := R] \sim_\alpha S'[x := R]$, we can apply $(\rightrightarrows\ \beta)$ rule to get $(\lambda xN''')N'' \rightrightarrows S'[x := R]$, and using $N' \sim_\alpha \lambda xN'''$ we get the desired result.
Finally for (6), we know $P''' \rightrightarrows S''$ and $S'' \sim_\alpha S'$ then $P''' \rightrightarrows S'$, we can use the preservation of parallel relation under substitution, crucial lemma 1, to get $P'''[x := P''] \rightrightarrows S'[x := R]$, which as $P \sim_\alpha P[x := P'']$ ends the proof.

□

Lemma 14. *If a reduction relation R is confluent, then so is its reflexive and transitive closure R^*.*

The proof is standard, by a double induction.

Corollary 2. *$\rightrightarrows^*$ is confluent.*

If we now write $\twoheadrightarrow$ for the relation of beta-reduction, we have:

Lemma 15. *$\twoheadrightarrow = \rightrightarrows^*$*

from which we finally arrive at:

Theorem 2 (Church-Rosser)**.** *Beta-reduction is confluent.*

5. Conclusion

We show the classic alpha induction principle fails in emulating BVC in the proof of parallel reduction relation is preserved by substitution operation. We propouse a novel induction principle that sucesfully overcomes this problem. We remain in a nominal firs-order syntax.

Acknowledgments

The first author work was partially supported by a PhD. scholarship granted by ANII.

References

B. Aydemir, A. Charguéraud, B. C. Pierce, R. Pollack, and S. Weirich. Engineering formal metatheory. In *Proceedings of the 35th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '08, pages 3–15, New York, NY, USA, 2008. ACM. ISBN 978-1-59593-689-9. doi: 10.1145/1328438.1328443. URL http://doi.acm.org/10.1145/1328438.1328443.

B. E. Aydemir, A. Bohannon, M. Fairbairn, J. N. Foster, B. C. Pierce, P. Sewell, D. Vytiniotis, G. Washburn, S. Weirich, and S. Zdancewic. *Mechanized Metatheory for the Masses: The PoplMark Challenge*, pages 50–65. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005. ISBN 978-3-540-31820-0. doi: 10.1007/11541868_4. URL http://dx.doi.org/10.1007/11541868_4.

H. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. Studies in Logic and the Foundations of Mathematics. Elsevier Science, 1985. ISBN 9780080933757.

H. P. Barendregt. *The Lambda Calculus Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North Holland, revised edition, 1984. ISBN 9780080933757.

A. Charguéraud. The locally nameless representation. *J. Autom. Reasoning*, 49(3):363–408, 2012. doi: 10.1007/s10817-011-9225-2. URL http://dx.doi.org/10.1007/s10817-011-9225-2.

E. Copello, A. Tasistro, N. Szasz, A. Bove, and M. Fernández. Alpha-structural induction and recursion for the lambda calculus in constructive type theory. *Electr. Notes Theor. Comput. Sci.*, 323:109–124, 2016. doi: 10.1016/j.entcs.2016.06.008. URL http://dx.doi.org/10.1016/j.entcs.2016.06.008.

N. G. de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with applications to the church-rosser theorem. *Indagationes Mathematicae (Koninglijke Nederlandse Akademie van Wetenschappen)*, 34(5):381–392, 1972. http://www.win.tue.nl/automath/archive/pdf/aut029.pdfElectronic Edition.

J. Despeyroux, A. P. Felty, and A. Hirschowitz. Higher-order abstract syntax in coq. In M. Dezani-Ciancaglini and G. D. Plotkin, editors, *TLCA*, volume 902 of *Lecture Notes in Computer Science*, pages 124–138. Springer, 1995. ISBN 3-540-59048-X. URL http://dblp.uni-trier.de/db/conf/tlca/tlca95.html#DespeyrouxFH95.

M. J. Gabbay and A. M. Pitts. A New Approach to Abstract Syntax with Variable Binding. *Formal Aspects of Computing*, 13(3-5):341–363, 2002. ISSN 0934-5043. doi: 10.1007/s001650200016. URL http://dx.doi.org/10.1007/s001650200016.

A. D. Gordon. A Mechanisation of Name Carrying Syntax up to Alpha Conversion. In *Proceedings of Higher Order Logic Theorem Proving and its Applications*, Lecture Notes in Computer Science, pages 414–426, 1993.

A. D. Gordon and T. F. Melham. Five axioms of alpha-conversion. In J. von Wright, J. Grundy, and J. Harrison, editors, *Theorem Proving in Higher Order Logics, 9th International Conference, TPHOLs'96, Turku, Finland, August 26-30, 1996, Proceedings*, volume 1125 of *Lecture Notes in Computer Science*, pages 173–190. Springer, 1996. ISBN 3-540-61587-3. doi: 10.1007/BFb0105404. URL http://dx.doi.org/10.1007/BFb0105404.

J. R. Hindley and J. P. Seldin. *Introduction to Combinators and Lambda-Calculus*. Cambridge University Press, 1986.

P. Martin-Löf and G. Sambin. *Intuitionistic type theory*. Studies in Proof Theory. Bibliopolis, 1984. URL http://books.google.com.uy/books?id=_D0ZAQAAIAAJ.

U. Norell. *Towards a practical programming language based on dependent type theory*. PhD thesis, Department of Computer Science and Engineering, Chalmers University of Technology, SE-412 96 Göteborg, Sweden, September 2007.

M. Norrish. Recursive function definition for types with binders. In *In Seventeenth International Conference on Theorem Proving in Higher Order Logics*, pages 241–256, 2004.

M. Takahashi. Parallel reductions in -calculus. *Information and Computation*, 118(1):120 – 127, 1995. ISSN 0890-5401.

C. Urban and C. Tasson. Nominal techniques in isabelle/hol. In R. Nieuwenhuis, editor, *Automated Deduction CADE-20*, volume 3632 of *Lecture Notes in Computer Science*, pages 38–53. Springer Berlin Heidelberg, 2005. ISBN 978-3-540-28005-7. doi: 10.1007/11532231_4. URL http://dx.doi.org/10.1007/11532231_4.