Principles of Recursion and Induction for Nominal Lambda Calculus.

Ana Bove [1]

Chalmers University of Technology
Gothenburg, Sweden

Maribel Fernandez [2]

King's College London
London, England

Álvaro Tasistro [3] Nora Szasz [4] Ernesto Copello [5]

Universidad ORT Uruguay
Montevideo, Uruguay

Abstract

We formulate principles of induction and recursion for a variant of lambda calculus with bound names where α-conversion is based upon name swapping as in nominal abstract syntax. The principles allow to work modulo alpha-conversion and apply the Barendregt variable convention. We derive them all from the simple structural induction principle and apply them to get some fundamental meta-theoretical results, such as the substitution lemma for alpha-conversion and the result of substitution composition. The whole work is implemented in Agda.

Keywords: Formal Meta-theory, Lambda Calculus, Constructive Type Theory

0.1 Related Work

There exist a continuous line of works, during approximately a decade, formalising inductive/recursion principles over α-equivalence classes of λ-terms, all of them developed in the Isabelle/HOL proof assistant. In one of the first works in this line, Gordon [4] constructs an induction principle equivalent to one of the developed in this work. Which, in the abstraction case of a proof by induction over α-equivalence

[1] Email: bove@chalmers.se
[2] Email: Maribel.Fernandez@kcl.ac.uk
[3] Email: tasistro@ort.edu.uy
[4] Email: szasz@ort.edu.uy
[5] Email: copello@ort.edu.uy

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

class of terms, as the Barendregt variable convention (BVC), enable us to choose the abstraction variable fresh enough from the context of the proof. That is, we are able to choose a fresh enough citizen from an arbitrary α-equivalence class of terms. Gordon uses a variation of De Bruijn syntax to represent λ-terms. This syntax was already suggested by De Bruijn [1], in which "free variables have names but the bound variables are nameless". The main property of this syntax is that α-convertible terms are syntactically equal. Although, invalid terms appears in this representation, and a well-formed predicate is needed to exclude bad terms from the formalisation. Because of this last issue, every introduced function must be proved to be closed under well-formed terms, and well-formed hypothesis should be added to all proofs. On the other hand, the main advantage of this mixed strategy is that theorems can be expressed in conventional form, without 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 α-conversion. Although, when a renaming has to be done to pick another witnesses of an α-equivalence class, the classical primitive inductive hypothesis does not have any information about the new renamed sub-term, becoming necessary in general to do an induction over the length of terms. In this way, we are able to apply the inductive hypothesis to the renamed sub-term, because its length is strictly decreasing. To overcame this overhead, Gordon introduces a BVC induction principle for decidable predicates, which, as expected, is proved by induction on the length of De Bruijn's terms.

As Gordon point outs, name-carrying syntax up to literal equality would be needed to represent language definitions, such as that of standard ML, for instance, where syntax is not identified up to α-conversion. Although De Bruijn notation has been used to implement several theorem provers, where syntax is internally represented in De Bruijn notation, and for human interacting, a map is given to a more human readable name-carrying syntax, this is different to use this internal notation also at a logic level. In spite of this, Gordon manages to hide De Bruijn notation, behind some succinct set of lemmas. As an example of this, substitution lemmas from sections 1.14 and 1.15 of Hindley and Seldin's book [6] are directly derived using the BVC induction principle, without recourse to theorems about the underlying De Bruijn representation, neither exposing the internal renaming done to select fresh variables.

In [5] Gordon and Melham continues working in previous formalisation problems, and present a way to define functions over λ-terms without any overhead. They do so introducing an iteration principle over λ-terms. Previous approach is *first-order* in the sense that the variable-binding operations of the embedded syntax is distinct form the meta variable-binding, at the host proof assistant language level. This work began to explore a kind of *second-order* approach, where a typical abstraction expression $Lam\ x\ u$ can be obtained from a meta-level abstraction expression $\lambda y.u[x := var\ y]$. For this, they use a function $abs : (variables\ names \to terms) \to terms$, that is, any meta-level function from variables to terms represents a λ-abstraction in the embedded language. In theirs iteration principle, to define a function f over the case of a typical abstraction of the form $Lam\ x\ u$, the value of $f(Lam\ x\ u)$ can be determined using the meta-abstraction $\lambda y.f(u[x := var\ y])$, not the classic primitive call $f(u)$. If it was the latter, that will allow us to distin-

guish α-equivalent terms in the definition of a function, so theirs iteration principle defines well behaved functions in the sense α-equivalent terms should return equal results. The key of their development is the function *abs*, for which they present a model that involves the iteration over the infinite set of possible variables, so they prove the existence of a theoretical model for *abs* function, but they do not give explicitely a computable one. We do not know in deep the Isabelle/HOL proof assistant capabilities, and the paper does not give much more information about *abs* codification in their formalisation. So this requirement is not easily to evaluate or deduce how feasible is to transfer to a Constructive Type Theory environment.

Gabbay-Pitts [3] introduces a general theory, called *nominal approach*, to deal with issues of bound names and α-equivalence classes in any abstract syntax. They considering constructions an properties that are invariant with respect to permutating names, and gives for them principles of recursion/induction over the α-equivalence classes defined by the abstract syntax bindings. At the base of their theory is the notion of *finite supported* mathematical objects, which gives a well-behaved way, in terms of name-permutations, of expressing the fact that atoms are fresh for mathematical objects. This notion enable us 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 [7] try to introduce a method to define functions in a much more familiar way, approximating it to the classics 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 resulting iteration principle have complicated side-conditions to prove about the functions used to instantiate this iteration principle. To use his principle we have to prove, for each function used to define the cases of the λ-terms iteration, that are finited supported, that is, do not create too many fresh variables, and that behave in a linear way through the swapping operation, that is, for any auxiliary function f used, term M and x, y variables $(swap(x, y, f(M)) \equiv f(swap(x, y, M))$.

Urban and Tasson [8] uses more in deep the Gabbay-Pitts theory to construct an induction principle similar to first introduced by Gordon, but using the concept of finite support of nominal sets, and not the free variables function over terms to state the freshness conditions. They abandon the De Bruijn notation to explore a weak HOAS [2], which uses functions at the meta-level to encode λ-abstractions. As a consequence they loose syntactical equality in α-compatible terms, and they have to introduce an α-compatible relation. Besides, they have to define the substitution operation because weak HOAS does not use meta-level substitution one. Although, they have a induction principle for "free" over theirs syntax. They prove the composition of substitution lemma (lemma 1.15 (c) of [6]) as an example of use of theirs induction principle. Our work is much more in the line of this one, we do not have syntactical equality of α-compatible terms, but we keep using a name-carrying syntax and not a variation of HOAS as them.

References

[1] N.G de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the church-rosser theorem. *Indagationes Mathematicae (Proceedings)*, 75(5):381 – 392, 1972.

[2] Jolle Despeyroux, Amy P. Felty, and Andr Hirschowitz. Higher-order abstract syntax in coq. In Mariangiola Dezani-Ciancaglini and Gordon D. Plotkin, editors, *TLCA*, volume 902 of *Lecture Notes in Computer Science*, pages 124–138. Springer, 1995.

[3] Murdoch J. Gabbay and Andrew M. Pitts. A New Approach to Abstract Syntax with Variable Binding. *Formal Aspects of Computing*, 13(3-5):341–363, 2002.

[4] Andrew 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.

[5] Andrew D. Gordon and Thomas F. Melham. Five axioms of alpha-conversion. In Joakim von Wright, Jim Grundy, and John 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.

[6] J. Roger Hindley and Jonathan P. Seldin. *Introduction to Combinators and Lambda-Calculus*. Cambridge University Press, 1986.

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

[8] Christian Urban and Christine Tasson. Nominal techniques in isabelle/hol. In Robert Nieuwenhuis, editor, *Automated Deduction CADE-20*, volume 3632 of *Lecture Notes in Computer Science*, pages 38–53. Springer Berlin Heidelberg, 2005.