ICALP'01

Hersónisos, Crete, July 2001

An axiomatic approach to metareasoning on nominal algebras in HOAS

Marino Miculan

Università di Udine, Italy

miculan@dimi.uniud.it

Joint work with Furio Honsell and Ivan Scagnetto

Research supported by EEC Working Group No.29001, *TYPES*.

Motivations: A common scenario

We want/need to use (semi)automatized tools for dealing with the theory and metatheory of many different calculi involving *names*

- represent formally (*encode*) syntax and semantics of an object language (e.g., λ-, π-, ambient calculus) in some general metalanguage (logical framework) for doing formal (meta)reasoning.

- derive some results interactively in a goal-directed manner, in some general-purpose theorem prover/proof assistant

Problem: how to render binding operators (e.g, λ, ν) efficiently?

In interactive development,

efficiently $\cong$ "formal proofs should look like on paper"

Long standing problem. Many approaches have been proposed, with pros and cons: *de Bruijn indexes, first-order abstract syntax, higher-order abstract syntax* . . . [HHP87,Hue94,DFH95,GM96,MM01,. . .].

First-order approaches

If we follow the rules of the game, we go for a *deep embedding*: all the details have to be spelled out in the framework

- **First-order abstract syntax**

  ```
  nu : Name -> Proc -> Proc
  ```

  Thus, `(nu x 0)` differs from `(nu y 0)`, *a priori*.

  Needs lots of machinery about α-equivalence, substitution, ...

- **de Bruijn indexes**

  ```
  nu : Proc -> Proc
  ```

  Good at α-equivalence (names disappear!), but not immediate to understand and needs even more machinery for capture-avoiding substitution than FOAS

(Weak) Higher-order abstract syntax

Binders are *higher-order* constructors: they take *functions* as arguments

```
nu : (Name -> Proc) -> Proc
```

Thus, $\nu x.\bar{x}y$ is represented as `(nu [x:Name](out x y))`

Objects of type `Name -> Proc` represent *contexts* (terms with *holes*)

♡ many aspects of names management are delegated to the metalanguage (α-conversion, capture-avoiding substitution, generation of fresh names,. . .) $\Rightarrow$ widely used in most logical frameworks

♠ if `Name` is defined as inductive then *exotic terms* (= not corresponding to any real term of the object language) will arise! E.g., if `Name = nat`

```
weird = nu [x:nat](Cases x of 0 => P
                            | _ => P|Q   end).
```

♠ in general, function spaces are not recognized as inductive
$\Rightarrow$ structural induction over higher-order terms is not provided
$\Rightarrow$ metatheoretic analysis is difficult/impossible

The "axiomatic" approach

Basic problem with HOAS: in the usual meaning of `->`, the type `Name -> Proc` contains lots of garbage

$\Rightarrow$ clean up these types by adding (= postulating) to your favourite metalogic a set of properties which characterized the "natural" behaviour of contexts and names. (This set of basic properties is the *Theory of Contexts*.)

Big issues of this approach: soundness? expressivity?

In this talk, the Theory of Contexts will be described in broad generality for a wide range of object logics.

Nominal algebras

A *names set* υ is an infinite enumerable set of different atomic objects, with a decidable equality.

A *names base* is a finite set $V = \{\upsilon_1, \ldots, \upsilon_k\}$ of names sets.

Let $V = \{\upsilon_1, \ldots, \upsilon_k\}$ be a names base, whose elements are ranged over by υ. Let $I = \{\iota_1, \ldots, \iota_m\}$ be a set of *basic types*, ranged over by ι.

A *constructor arity over* V, I *for* ι is a type α of the form $\tau_1 \times \cdots \times \tau_n \to \iota$, where $n \geq 0$ and for $i = 1 \ldots n$, the type τ_i is either in V or it is of the form $\tau_i = \upsilon_{i1} \times \cdots \times \upsilon_{im_i} \to \sigma_i$ where $\upsilon_{ij} \in V$ and $\sigma_i \in I$. If $m_i > 0$ for some i, then α is said to be a *binding arity*, or to *bind* $\upsilon_{i1}, \ldots, \upsilon_{im_i}$ *over* σ_i.

A *constructor over* V, I *for* ι is a typed constant c^α where α is a constructor arity over V, I. If α is a binding arity, then c is said to be a *binding constructor*, or simply a *binder*.

A *nominal algebra* N is a tuple $\langle V, I, C \rangle$ where V is a set of names sets, I is a set of basic types, and C is a set of constructors over V, I.

Nominal algebras: examples

Many languages can be viewed as nominal algebras.

- Untyped λ-calculus: $N_\lambda = \langle \{\upsilon\}, \{\Lambda\}, \{var^{\upsilon\to\Lambda}, \boldsymbol{\lambda}^{(\upsilon\to\Lambda)\to\Lambda}, app^{\Lambda\times\Lambda\to\Lambda}\}\rangle$

- π-calculus: $N_\pi = \langle \{\upsilon\}, \{\iota\},$
  $\{0^\iota, |^{\iota\times\iota\to\iota}, \tau^{\iota\to\iota}, =^{\upsilon\times\upsilon\times\iota\to\iota}, \nu^{(\upsilon\to\iota)\to\iota}, in^{\upsilon\times(\upsilon\to\iota)\to\iota}, out^{\upsilon\times\upsilon\times\iota\to\iota}\}\rangle$

- Ambient: $N_{Amb} = \langle \{\eta, \upsilon\}, \{C, P, F\},$
  $\{name^{\eta\to C}, in^{C\to C}, out^{C\to C}, open^{C\to C}, \epsilon^C, path^{C\times C\to C},$
  $\nu^{(\eta\to P)\to P}, 0^P, |^{P\times P\to P}, !^{P\to P}, amb^{C\times P\to P}, cap^{C\times P\to P}, in_a^{(\eta\to P)\to P}, out_a^{C\to P}$
  $\mathbf{T}^F, \neg^{F\to F}, \vee^{F\times F\to F}, \mathbf{0}^F, |^{F\times F\to F}, \rhd^{F\times F\to F}, [\cdot]^{\eta\times F\to F}, @^{F\times\eta\to F},$
  $®^{\eta\times F\to F}, \oslash^{F\times\eta\to F}, \Diamond^{F\to F}, \diamondsuit^{F\to F}, \forall^{(\upsilon\to F)\to F}\}$

On the other hand, languages with polyadic binders escape the class of nominal algebras.

The metalanguage Υ

$\Upsilon =$ Simple Theory of Types on a given signature Σ
$+$ Classical Higher Order Logic
$+$ Theory of Contexts
$+$ Higher-Order Induction/Recursion principles

Two kind of judgements:

- Typing judgements have the form $\Gamma \vdash_\Sigma M : \tau$
- Logical derivation judgement $\Gamma; \Delta \vdash_\Sigma p$

where Σ is a signature.

Υ: the Simple Theory of Types

A *type signature* Σ_t is a finite list of atomic type symbols $\sigma_1, \ldots, \sigma_n$.

The *simple types* over a type signature Σ_t are defined as follows:

$$\tau ::= o \mid \sigma \mid \tau \to \tau \quad \text{where } \sigma \in \Sigma_t$$

A *constant signature* Σ_c is a finite list of constant symbols with simple types $c : \tau_1, \ldots, c_m : \tau_m$.

A *signature* Σ consists of a *type signature* Σ_t and a *constant signature* Σ_c.

The *terms* over the signature $\Sigma = \langle \Sigma_c, \Sigma_t \rangle$, ranged over by M, N, P, Q, R, are defined by the following abstract syntax:

$$M ::= x \mid MN \mid \lambda x{:}\tau.M \mid c \mid M \Rightarrow N \mid \forall x{:}\tau.M \quad \text{where } c : \sigma \in \Sigma_c \text{ for some } \sigma$$

As usual, we denote by $M[N/x]$ capture-avoiding substitution. Terms are identified up-to α-conversion.

Υ: typing judgement

(Typing) contexts (ranged over by Γ) are finite sets of typing assertions over distinct variables (e.g. $\{x_1 : \tau_1, x_2 : \tau_2, \ldots, x_n : \tau_n\}$).

Typing judgements have the form $\Gamma \vdash_\Sigma M : \tau$. Rules:

$$\frac{-}{\Gamma, x : \tau \vdash_\Sigma x : \tau} \quad \text{(VAR)} \qquad \frac{-}{\Gamma \vdash_\Sigma c : \tau}(c{:}\tau) \in \Sigma_c \quad \text{(CONST)}$$

$$\frac{\Gamma \vdash_\Sigma M : \tau' \to \tau \quad \Gamma \vdash_\Sigma N : \tau'}{\Gamma \vdash_\Sigma MN : \tau} \quad \text{(APP)} \qquad \frac{\Gamma \vdash_\Sigma M : o \quad \Gamma \vdash_\Sigma N : o}{\Gamma \vdash_\Sigma M \Rightarrow N : o} \quad \text{(IMP)}$$

$$\frac{\Gamma, x : \tau' \vdash_\Sigma M : \tau}{\Gamma \vdash_\Sigma \lambda x{:}\tau'.M : \tau' \to \tau} \quad \text{(ABS)} \qquad \frac{\Gamma, x : \tau \vdash_\Sigma M : o}{\Gamma \vdash_\Sigma \forall x{:}\tau.M : o} \quad \text{(FORALL)}$$

Terms of type o are the *propositions* of our logic. Terms of type $\tau \to o$ are called *predicates (over τ)*.

As usual in HOL, all logical connectives can be defined in terms of $\forall$ and $\Rightarrow$.

All usual properties of simply typed λ-calculi are satisfied: uniqueness of type, subject reduction, normal form, Church-Rosser, ...

Encoding nominal algebras in Υ

Υ is expressive enough to represent faithfully any nominal algebra, via HOAS:

1. object level names are represented by metalanguage variables;
2. contexts are represented by higher-order terms, *i.e.* functions;
3. binders are represented by constructors which take functions as arguments;
4. contexts instantiation and capture-avoiding substitution are meta-level applications; hence, α-conversion is immediately inherited from the metalanguage.

Let $N = \langle V, I, C\rangle$ be a nominal algebra. The *signature for* N, $\Sigma(N)$, is defined as $\Sigma(N) \triangleq \langle V \cup I, \{c : \tau \mid c^\tau \in C\}\rangle$.

Theorem 1 *Let X be a stage in V, and let $\Gamma(X) \triangleq \{x : \upsilon_i \mid x \in X_i, i = 1 \ldots n\}$. For each type $\iota \in I$, there exists a bijection between $\mathcal{L}^\iota_X$ and the set of terms in $\beta\eta$-normal form of type ι in the context $\Gamma(X)$.*

Encodings in Υ: Examples

- λ-calculus:

$$\Sigma(N_\lambda)_t = \upsilon, \Lambda$$
$$\Sigma(N_\lambda)_c = var : \upsilon \to \Lambda, \boldsymbol{\lambda} : (\upsilon \to \Lambda) \to \Lambda, app : \Lambda \to \Lambda \to \Lambda$$

  For instance, $\boldsymbol{\lambda}x(xx) \rightsquigarrow \boldsymbol{\lambda}\lambda x{:}\upsilon.(app\ (var\ x)\ (var\ x))$.

- π-calculus:

$$\Sigma(N_\pi)_t = \upsilon, \iota$$
$$\Sigma(N_\pi)_c = 0 : \iota, \tau : \iota \to \iota, | : \iota \to \iota \to \iota, =: \upsilon \to \upsilon \to \iota \to \iota, \nu : (\upsilon \to \iota) \to \iota,$$
$$in : \upsilon \to (\upsilon \to \iota) \to \iota, out : \upsilon \to \upsilon \to \iota \to \iota$$

Υ: logical judgement

The *logical derivation* judgement "$\Gamma; \Delta \vdash_\Sigma p$" means "$p$ derives from the set of propositions Δ in context Γ".

Logical derivation system = natural deduction style system for classical higher-order logic, with $\beta\eta\xi$-equality
+ non-occurrence predicates
+ a set of axioms for the *Theory of Contexts*.

System for Classical HOL: a standard one

$$\frac{\Gamma; \Delta, p \vdash_\Sigma q}{\Gamma; \Delta \vdash_\Sigma p \Rightarrow q} \ (\Rightarrow\text{-I}) \qquad \frac{\Gamma \vdash_\Sigma p : o}{\Gamma; \Delta \vdash_\Sigma p \vee \neg p} \ (\text{LEM})$$

$$\frac{\Gamma; \Delta \vdash_\Sigma p \Rightarrow q \quad \Gamma; \Delta \vdash_\Sigma p}{\Gamma; \Delta \vdash_\Sigma q} \ (\Rightarrow\text{-E}) \qquad \frac{\Gamma, x : \tau \vdash_\Sigma M : \sigma \quad \Gamma \vdash_\Sigma N : \tau}{\Gamma; \Delta \vdash_\Sigma (\lambda x{:}\tau.M)N =^\sigma M[N/x]} \ (\beta)$$

$$\frac{\Gamma, x : \tau; \Delta \vdash_\Sigma p}{\Gamma; \Delta \vdash_\Sigma \forall x{:}\tau.p} x \notin FV(\Delta) \ (\forall\text{-I}) \qquad \frac{\Gamma \vdash_\Sigma M : \tau \to \sigma}{\Gamma; \Delta \vdash_\Sigma \lambda x{:}\tau.Mx =^{\tau\to\sigma} M} x \notin FV(M) \ (\eta)$$

$$\frac{\Gamma; \Delta \vdash_\Sigma \forall x{:}\tau.p \quad \Gamma \vdash_\Sigma M : \tau}{\Gamma; \Delta \vdash_\Sigma p[M/x]} \ (\forall\text{-E}) \qquad \frac{\Gamma, x : \sigma; \Delta \vdash_\Sigma M =^\tau N}{\Gamma; \Delta \vdash_\Sigma \lambda x{:}\sigma.M =^{\sigma\to\tau} \lambda x{:}\sigma.N} \ (\xi)$$

Υ: Non-occurrence predicates

For each $\upsilon \in V$ and $\iota \in I$, we define a predicate $\not\in^{\iota}_{\upsilon}$: $\upsilon \to \iota \to o$.

"$x \not\in^{\iota}_{\upsilon} M$" $\cong$ "the name x (of type υ) does not appear free in M (of type ι)."

Rules for deriving $x \not\in^{\iota}_{\upsilon} M$ are mechanically defined from the signature: i.e., for each constructor c, there is a rule as follows

$$\frac{H_1 \quad \ldots \quad H_n}{\Gamma; \Delta \vdash_\Sigma x \not\in^{\iota}_{\upsilon} (c\ M_1 \ldots M_n)} c^{\tau_1 \times \cdots \times \tau_n \to \iota} \in C \qquad (\text{Notin}_c)$$

where $H_i = \begin{cases} \Gamma; \Delta \vdash_\Sigma \neg(x =^{\upsilon} M_i) & \text{if } \tau_i = \upsilon \\ \Gamma, \Gamma_i; \Delta, \Delta_i \vdash_\Sigma x \not\in^{\iota'}_{\upsilon} (M_i\ y_1 \ldots y_{m_i}) & \text{if } \tau_i = \upsilon_{i1} \times \cdots \times \upsilon_{im_i} \to \iota' \end{cases}$

$\Gamma_i = y_1 : \upsilon_{i1}, \ldots, y_{m_i} : \upsilon_{im_i} \qquad \Delta_i = \{\neg(x =^{\upsilon} y_j) \mid \upsilon_j = \upsilon, j = 1 \ldots m_i\}$

Proposition 1 *For all Γ contexts, $(x : \upsilon) \in \Gamma$ and M such that $\Gamma \vdash_\Sigma M : \iota$, we have: $\Gamma; \emptyset \vdash_\Sigma x \not\in^{\tau}_{\upsilon} M$ iff $x \notin FV(M)$*

Non-occurrence predicates can be lifted to contexts:

$$x \not\in^{\upsilon \to \tau}_{\upsilon} M \triangleq \forall y{:}\upsilon.\neg(x =^{\upsilon} y) \Rightarrow x \not\in^{\tau}_{\upsilon} (M\ y)$$
$$x \not\in^{\upsilon' \to \tau}_{\upsilon} M \triangleq \forall y{:}\upsilon'.x \not\in^{\tau}_{\upsilon} (M\ y) \qquad (\upsilon \neq \upsilon')$$

The Theory of Contexts

A set of axiom schemata, which reflect at the theory level some fundamental properties of the intuitive notion of "context" and "occurrence" of variables. Their informal meaning is the following:

Unsaturability of variables: no term can contain all variables; i.e., there exists always a variable which does not occur free in a given term;

Extensionality of contexts: two contexts are equal if they are equal on a fresh variable; that is, if $M(x) = N(x)$ and $x \notin M(\cdot), N(\cdot)$, then $M = N$.

β-expansion: given a term M and a variable x, there is a context $C_M(\cdot)$, obtained by abstracting M over x

Υ: the Theory of Contexts

$$\frac{\Gamma \vdash_\Sigma P : \iota}{\Gamma; \Delta \vdash_\Sigma \exists x{:}\upsilon.x \notin P} \qquad (Unsat_\iota^\upsilon)$$

$$\frac{\Gamma \vdash_\Sigma P : \upsilon \to \tau \quad \Gamma \vdash_\Sigma Q : \upsilon \to \tau \quad \Gamma \vdash_\Sigma x : \upsilon}{\Gamma; \Delta, x \not\in^{\upsilon\to\tau} P, x \not\in^{\upsilon\to\tau} Q, (P\ x) =^\tau (Q\ x) \vdash_\Sigma P =^{\upsilon\to\tau} Q} \qquad (Ext_\upsilon^\tau)$$

$$\frac{\Gamma \vdash_\Sigma P : \tau \quad \Gamma \vdash_\Sigma x : \upsilon}{\Gamma; \Delta \vdash_\Sigma \exists Q{:}\upsilon \to \tau.x \not\in^{\upsilon\to\tau} Q \wedge P =^\tau (Q\ x)} \qquad (\beta\_exp_\upsilon^\tau)$$

where $\tau = \upsilon_{i_1} \to \cdots \to \upsilon_{i_k} \to \iota$

Properties of Υ

Proposition 2 (Hof99) *The Axiom of Unique Choice*

$$\frac{\Gamma \vdash R : \sigma \to \tau \to o \quad \Gamma, a : \sigma; \Delta \vdash \exists! b : \tau.(R\ a\ b)}{\Gamma; \Delta \vdash \exists f : \sigma \to \tau. \forall a : \sigma.(R\ a\ (f\ a))} \qquad (\mathsf{AC!}_{\sigma,\tau})$$

is inconsistent with the Theory of Contexts.

Consequences:

- in toposes, AC! always holds $\Rightarrow$ topos logic is not enough $\Rightarrow$ soundness of the Theory of Contexts is not so trivial
- relations are more expressive than functions: there are functional relations whose characteristic functions cannot be defined
  $\Rightarrow$ often, one has to use functional relations in place of functions

Theorem 2 *For all nominal algebras N: Υ over the signature $\Sigma(N)$ is sound.*

Idea: build a model (close to Schanuel topos) using a tripos ove *functor categories*. The index category is the category of permutations over finite sets of atoms. See [BHHMS01] for details.

Properties of Υ (cont.)

Let $\Gamma \vdash_\Sigma p : \upsilon \to o$; consider the rules

$$\frac{\Gamma;\Delta \vdash_\Sigma \forall y{:}\upsilon.y \not\in^{\upsilon\to o} p \Rightarrow (p\ y)}{\Gamma;\Delta \vdash_\Sigma \exists y{:}\upsilon.y \not\in^{\upsilon\to o} p \wedge (p\ y)}\ (\forall\exists) \qquad \frac{\Gamma;\Delta \vdash_\Sigma \exists y{:}\upsilon.y \not\in^{\upsilon\to o} p \wedge (p\ y)}{\Gamma;\Delta \vdash_\Sigma \forall y{:}\upsilon.y \not\in^{\upsilon\to o} p \Rightarrow (p\ y)}\ (\exists\forall)$$

These rules capture the idea that freshness has both an "existential" and a "universal" flavour. Indeed in Υ we have that

Theorem 3 *$\forall\exists$ is derivable, and $\exists\forall$ is admissible.*

In fact also the following bindable name renaming rule

$$\frac{\Gamma, x : \upsilon; \Delta, x \not\in^{\upsilon\to o} p \vdash_\Sigma (p\ x)}{\Gamma, y : \upsilon; \Delta, y \not\in^{\upsilon\to o} p \vdash_\Sigma (p\ y)} x, y \not\in FV(\Delta) \qquad \text{(Ren)}$$

is admissible in our system.

For most specific predicates of interest —e.g., strong (late) bisimilarity and operational semantics of π-calculus, typing system of λ-calculus, etc.— the rule schema $\exists\forall$ is *derivable* in Υ using Ext^τ_υ and $\beta\_exp^\tau_\upsilon$.

Higher-order Induction and Recursion

The tripos model justifies also *recursion and induction principles* over higher-order types
$\Rightarrow$ we can reason by structural induction, and define function by structural recursion, over contexts.

The general schemata, parametric in a given nominal algebra, don't fit easily into a slide — see paper on proceedings.

Example: induction principle over contexts of λ-calculus:

$$\frac{\begin{array}{l}\Gamma \vdash P : (\upsilon \to \Lambda) \to o \\ \Gamma, x_1 : \upsilon; \Delta \vdash (P\ \lambda x{:}\upsilon.(var\ x_1)) \\ \Gamma, x_1 : \upsilon; \Delta \vdash (P\ \lambda x{:}\upsilon.(var\ x)) \\ \Gamma, M_1{:}\upsilon \to \Lambda, M_2{:}\upsilon \to \Lambda; \Delta, (P\ M_1), (P\ M_2) \vdash (P\ \lambda x{:}\upsilon.(app\ (M_1\ x)\ (M_2\ x))) \\ \Gamma, M_1 : \upsilon \to \upsilon \to \Lambda; \Delta, \forall y_1{:}\upsilon.(P\ \lambda x{:}\upsilon.(M_1\ x\ y_1)) \vdash (P\ \lambda x{:}\upsilon.(\boldsymbol{\lambda}(M_1\ x)))\end{array}}{\Gamma; \Delta \vdash \forall M{:}\upsilon \to \Lambda.(P\ M)}$$

This principle is strictly stronger than the one provided, e.g., by the Calculus of Inductive Constructions or Isabelle/HOL. These systems do not recognize that $(M_1\ x)$ is structurally smaller than $\lambda x : \upsilon.(\boldsymbol{\lambda}\ (M_1\ x))$.

Case studies

Expressivity and easiness of use of the Theory of Contexts should be tested via case studies.

The Theory of Contexts has been used for developing non trivial metatheories of several calculi:

- π-calculus: among others, most of the "algebraic laws" of strong late bisimilarity in [Milner et al., 1992]

- untyped and simply typed λ-calculus: functionality of substitution, generation lemmata, confluence of evaluation, equivalence of big-step and small-step semantics, preservation of types under renaming of variables, and under substitution, subject reductions,...

- in progress: Abramsky applicative bisimulation, Ambient calculus, ...

These examples shows that we got a low mathematical and logical overhead: "proofs looks almost like on the paper". Almost, because many functions must be represented as functional relations.

How much classical logic is needed?

In fact, full classical logic is not strictly needed. We could drop axiom LEM, and simply assume that

- either equality over names is decidable

$$\frac{\Gamma \vdash_\Sigma x : \upsilon \quad \Gamma \vdash_\Sigma y : \upsilon}{\Gamma; \Delta \vdash_\Sigma x =^\upsilon y \vee x \neq^\upsilon y} \qquad (\mathsf{LEM}_{=^\upsilon})$$

- or occurrence predicates of names in terms are decidable

$$\frac{\Gamma \vdash_\Sigma x : \upsilon \quad \Gamma \vdash_\Sigma P : \iota}{\Gamma; \Delta \vdash_\Sigma x \notin^\iota_\upsilon P \vee \neg(x \notin^\iota_\upsilon P)} \qquad (\mathsf{LEM}_{\notin^\iota_\upsilon})$$

$\mathsf{LEM}_{\notin^\iota_\upsilon} \Rightarrow \mathsf{LEM}_{=^\upsilon}$ directly.

$\mathsf{LEM}_{=^\upsilon} \Rightarrow \mathsf{LEM}_{\notin^\iota_\upsilon}$ using $Unsat^\upsilon_\iota$ and induction both over plain terms and over contexts (Ind^ι and $Ind^{\upsilon\to\iota}$).

Thus, the Theory of Contexts can be added also to intuitionistic metalogics (like, e.g., Calculus of Inductive Constructions in Coq).

Related work

- Models of HOAS (4 LICS papers [FPT99,GP99,Hof99,FT01]) and of the Theory of Contexts [BHHMS01]

- Pitt's *Nominal Logic* [Pitts01]: a first-order logic for properties whose validity is invariant under *bindable name swapping*, with a special quantifier $\mathbb{N}$ for expressing freshness of names.

  $$\mathbb{N}y.p \cong \text{“}p \text{ holds for } y \text{ a fresh name”}.$$

  $\mathbb{N}$ resembles both $\forall$ and $\exists$, and it satisfies the rules:

  $$\frac{\Gamma, y\#\vec{x} \vdash p}{\Gamma \vdash \mathbb{N}y.p} \qquad \frac{\Gamma \vdash \mathbb{N}y.p \quad \Gamma, p, y\#\vec{x} \vdash q}{\Gamma \vdash q}$$

  where $\vec{x}$ is the “support” of p. In the Theory of Contexts, $\mathbb{N}y.p$ and $y\#\vec{x}$ can be encoded as follows:

  $$\mathbb{N}y.p \triangleq \forall y{:}\upsilon.y \not\in^{\upsilon\to o} (\lambda y{:}\upsilon.p) \Rightarrow p \qquad y\#\vec{x} \triangleq y \not\in^{o} p$$

Conclusions

Main features of the Theory of Contexts:

- ♡ it can be used safely in most Classical and Intuitionistic HOLs (which do not entail the Axiom of Unique Choice, AC!)
  $\Rightarrow$ you do not have to change your favourite metalanguage.
- ♡ general: it applies to a wide range of object logics (*nominal algebras*)
- ♡ it allows for induction and recursion principles over higher-order datatypes
- ♠ it is not compatible with the AC! $\Rightarrow$ expressive power of functions is stricly less than that of relations
- ♠ complex (i.e., non-standard) model

Future work:

- dependent types (for dealing with, e.g., Natural Deduction style systems)
- programming language for dealing with higher-order terms

Proof.

- $AC!$ allows to derive the characteristic function of the equality over names $eq : \upsilon \to \upsilon \to nat$ (defined by $\forall x, y : \upsilon.\ x = y \Leftrightarrow eq(x, y) = 1$, where $=$ is Leibniz equality);

- $Q \stackrel{\text{def}}{=} \lambda x^{\upsilon}.\ if\ eq(x, y)\ then\ p\ else\ q$ (where $y : \upsilon$ e $p, q : \iota$);

- using $Ext^{\upsilon \to \iota}$ one can prove that $Q =^{\upsilon \to \iota} \lambda x^{\upsilon}.\ q$;

- hence it is possible to show that all processes are syntactically equal (absurd).

In the π-calculus encoding, define the term

$$R \triangleq \lambda u : \upsilon.\lambda q : \iota.\lambda x : \upsilon.\lambda p : \iota.(x =^{\upsilon} u \wedge p =^{\iota} 0) \vee (\neg x =^{\upsilon} u \wedge p =^{\iota} q).$$

Let u' a fresh name; for all $p' : \iota$, $(R\ u'\ p') : \upsilon \to \iota \to o$ is a functional binary relation. From AC!$_{\upsilon,\iota}$, there exists a function $f : \upsilon \to \iota$ such that, for all $x : \upsilon$, $((R\ u'\ p)\ x\ (f\ x))$ holds. Hence, by Ext^{ι}_{υ}, we can prove that $f =^{\upsilon\to\iota} \lambda x : \upsilon.p$ because for any fresh name w we have that $(f\ w) =^{\iota} p$. Then we have that, for all names y, $(f\ y) =^{\iota} ((\lambda x : \upsilon.p)\ y) =^{\iota} p$. Since $(f\ u') = 0$, we have that $\forall p : \iota.p =^{\iota} 0$ holds—which is absurd.