LFM'02

Ambient Calculus and its Logic in the Calculus of Inductive Constructions

Ivan Scagnetto and Marino Miculan

Dipartimento di Matematica e Informatica,
Università di Udine, Italy
scagnett@dimi.uniud.it, miculan@dimi.uniud.it

What's in this talk

A complete case study on

- encoding of Ambient Calculus and its modal logic
- in a type-based logical framework (Coq)
- using Higher Order Abstract Syntax
- and the Theory of Contexts
- and full formalization of most metatheoretic results over the calculus and the logic, as in [4]

Reference paper:

[4] Cardelli, L. and A. D. Gordon, *Logical properties of name restriction*, in: S. Abramsky, editor, *Proc. TLCA 2001*, LNCS **2044** (2001).

Why?

Along the line of previous case studies (λ-calculus, π-calculus, . . .)
BUT:

- Ambients have their own peculiarities (e.g., modal logic, names & variables,. . .)
- Ambients logic is capable to reflect metalogical properties which interact with HOAS (e.g., freshness, equality of names)
- Ambients are fairly new—still in development. This may benefit from systematic analysis of the calculus and its logic.

Why?

Along the line of previous case studies (λ-calculus, π-calculus, . . .)
BUT:

- Ambients have their own peculiarities (e.g., modal logic, names & variables,. . .)
- Ambients logic is capable to reflect metalogical properties which interact with HOAS (e.g., freshness, equality of names)
- Ambients are fairly new—still in development. This may benefit from systematic analysis of the calculus and its logic.

Expected benefits:

- For LF's: it allows to test, refine and compare methodologies for dealing with HOAS (like the Theory of Contexts)
- For Ambients: systematic analysis of many peculiarities, re-design of unpolished notions

Why?

Along the line of previous case studies (λ-calculus, π-calculus, . . .)
BUT:

- Ambients have their own peculiarities (e.g., modal logic, names & variables,. . .)
- Ambients logic is capable to reflect metalogical properties which interact with HOAS (e.g., freshness, equality of names)
- Ambients are fairly new—still in development. This may benefit from systematic analysis of the calculus and its logic.

Expected benefits:

- For LF's: it allows to test, refine and compare methodologies for dealing with HOAS (like the Theory of Contexts)
- For Ambients: systematic analysis of many peculiarities, re-design of unpolished notions

Outline of the talk

- Syntax of Ambient calculus and its logic
- Their representation: names vs. variables
- Semantics of Ambient calculus and its logic
- Their representation
- The Theory of Contexts for Ambients
- Development of (meta)theory
- The $\mathbb{N}$ quantifier
- Conclusions

Ambient Calculus: quick recap

- Ambient calculus = model of agents mobility in a dynamically changing hierarchy of domains [Cardelli, Gordon FOSSACS 98]
- Composed by
  - a *process algebra* with names (much like π-calculus)
  - with reduction operational semantics;
  - a *modal logic* for expressing temporal and spatial properties of agents
  - with satisfaction relation

Ambients processes

Syntactic categories:

- Names: $n \in \Lambda$
- Capabilities ζ: $M ::= n \mid in\ M \mid out\ M \mid open\ M \mid \varepsilon \mid M.M'$
- Processes Π:

$$P, Q, R ::= \mathbf{0} \mid P|Q \mid !P \mid M[P] \mid M.P \mid (\nu n)P \mid (n).P \mid \langle M \rangle$$

Identified up to α-conversion of names.
$P\{n \leftarrow M\}$ denotes usual capture avoiding substitution.

Operational semantics

- A structural equivalence judgment $\equiv \subseteq \Pi \times \Pi$
- A reduction relation $\rightarrow \subseteq \Pi \times \Pi$

Ambients processes

Syntactic categories:

- Names: $n \in \Lambda$
- Capabilities ζ: $M ::= n \mid in\ M \mid out\ M \mid open\ M \mid \varepsilon \mid M.M'$
- Processes Π:

$$P, Q, R ::= \mathbf{0} \mid P|Q \mid !P \mid M[P] \mid M.P \mid (\nu n)P \mid (n).P \mid \langle M \rangle$$

Identified up to α-conversion of names.
$P\{n \leftarrow M\}$ denotes usual capture avoiding substitution.

Operational semantics

- A structural equivalence judgment $\equiv \subseteq \Pi \times \Pi$
- A reduction relation $\rightarrow \subseteq \Pi \times \Pi$

Ambient logic

Syntax

- Variables $x \in \zeta$
- Formulas Φ:

$$\mathcal{A}, \mathcal{B}, \mathcal{C} ::= \mathbf{T} \mid \neg\mathcal{A} \mid \mathcal{A} \vee \mathcal{B} \mid \mathbf{0} \mid \mathcal{A}|\mathcal{B} \mid \mathcal{A} \triangleright \mathcal{B}$$
$$\mid \eta[\mathcal{A}] \mid \mathcal{A}@\eta \mid \eta \circledR \mathcal{A} \mid \mathcal{A} \oslash \eta \mid \Diamond\mathcal{A} \mid \diamondsuit\mathcal{A} \mid \forall x.\mathcal{A}$$

η may be either a name n or a variable x

Semantics

- *satisfaction* relation $P \models \mathcal{A}$. Defined by clauses.

Ambient logic

Syntax

- Variables $x \in \zeta$
- Formulas Φ:

$$\mathcal{A}, \mathcal{B}, \mathcal{C} ::= \mathbf{T} \mid \neg\mathcal{A} \mid \mathcal{A} \vee \mathcal{B} \mid \mathbf{0} \mid \mathcal{A}|\mathcal{B} \mid \mathcal{A} \rhd \mathcal{B}$$
$$\mid \eta[\mathcal{A}] \mid \mathcal{A}@\eta \mid \eta \circledR \mathcal{A} \mid \mathcal{A} \oslash \eta \mid \Diamond\mathcal{A} \mid \diamondsuit\mathcal{A} \mid \forall x.\mathcal{A}$$

η may be either a name n or a variable x
A first order modal logic. Variables may be replaced by variables or names (which may be replaced by capabilities).

Semantics

- *satisfaction* relation $P \models \mathcal{A}$. Defined by clauses.

Encoding of processes: weak HOAS

```
Variable name : Set.
Inductive proc: Set := nil : proc
  | par : proc -> proc -> proc
  | bang : proc -> proc
  | ambient : cap -> proc -> proc
  | cap_act : cap -> proc -> proc
  | nu : (name -> proc) -> proc
  | in_act : (name -> proc) -> proc
  | out_act : cap -> proc.
```

Encoding of processes: weak HOAS

```
Variable name : Set.
Inductive proc: Set := nil : proc
  | par : proc -> proc -> proc
  | bang : proc -> proc
  | ambient : cap -> proc -> proc
  | cap_act : cap -> proc -> proc
  | nu : (name -> proc) -> proc
  | in_act : (name -> proc) -> proc
  | out_act : cap -> proc.
```

- Object level names = metalanguage variables of type `name`

Encoding of processes: weak HOAS

```
Variable name : Set.
Inductive proc: Set := nil : proc
  | par : proc -> proc -> proc
  | bang : proc -> proc
  | ambient : cap -> proc -> proc
  | cap_act : cap -> proc -> proc
  | nu :  (name -> proc) -> proc
  | in_act :  (name -> proc) -> proc
  | out_act : cap -> proc.
```

- Object level names = metalanguage variables of type `name`
- Binding constructors are represented by 2nd-order term constructors $\Rightarrow$ α-conversion comes for free

$$(n).n[0] \rightsquigarrow \texttt{(in\_act [n:name](ambient n nil))}$$

Encoding of processes: weak HOAS

```
Variable name : Set.
Inductive proc: Set := nil : proc
  | par : proc -> proc -> proc
  | bang : proc -> proc
  | ambient : cap -> proc -> proc
  | cap_act : cap -> proc -> proc
  | nu : (name -> proc) -> proc
  | in_act : (name -> proc) -> proc
  | out_act : cap -> proc.
```

- Object level names = metalanguage variables of type `name`
- Binding constructors are represented by 2nd-order term constructors $\Rightarrow$ α-conversion comes for free
- `name` is not inductive $\Rightarrow$ no exotic terms.
  Required properties will be added later on, as needed.

Encoding of formulas: full HOAS

```
Inductive form: Set :=  T: form
  | neg: form -> form
  | Or: form -> form -> form
  | zero: form
...
  | rev: name -> form -> form
  | rev_adj: form -> name -> form
  | sometime: form -> form
  | somewhere: form -> form
  | forall: (name -> form) -> form.
```

- no need of a separate type for variables
- α-conversion and capture-avoiding substitution are inherited
- no exotic terms either (name is not inductive)

Names = Variables?

Object level names = metalevel variables of type `name`
Object level variables = metalevel variables of type `name`

Names = Variables?

Object level names = metalevel variables of type `name`
Object level variables = metalevel variables of type `name`

- Names can be replaced — and variables too...
- Names can be bound — and variables too...
- Processes are up-to α-conversion of names — and formulas are up-to α-conversion of variables...

Names = Variables?

Object level names = metalevel variables of type `name`
Object level variables = metalevel variables of type `name`

- Names can be replaced — and variables too...
- Names can be bound — and variables too...
- Processes are up-to α-conversion of names — and formulas are up-to α-conversion of variables...
- But different names are different, different variables may be not!

Names = Variables?

Object level names = metalevel variables of type `name`
Object level variables = metalevel variables of type `name`

- Names can be replaced — and variables too...
- Names can be bound — and variables too...
- Processes are up-to α-conversion of names — and formulas are up-to α-conversion of variables...
- But different names are different, different variables may be not!

Thus, what's in a name?

Names = Variables?

Object level names = metalevel variables of type `name`
Object level variables = metalevel variables of type `name`

- Names can be replaced — and variables too. . .
- Names can be bound — and variables too. . .
- Processes are up-to α-conversion of names — and formulas are up-to α-conversion of variables. . .
- But different names are different, different variables may be not!

Thus, what's in a name? Apartness!

A name is a variable whose possible values are restricted.

Representing Apartness

- Apartness can be represented by inequalities assumptions.
- Given $n_1, \ldots, n_k$ names and $x_1, \ldots, x_h$ variables, these are represented by the context

  $\mathsf{n}_1 : \mathsf{name}, \ldots, \mathsf{n}_k : \mathsf{name},\ \mathsf{x}_1 : \mathsf{name}, \ldots, \mathsf{x}_h : \mathsf{name},$
  $\mathsf{d}_{ij} : \mathsf{n}_i \neq \mathsf{n}_j$

  where $(1 \leq i < j \leq k)$

Representing Apartness

- Apartness can be represented by inequalities assumptions.
- Given $n_1, \ldots, n_k$ names and $x_1, \ldots, x_h$ variables, these are represented by the context

  $\mathsf{n}_1 : \mathsf{name}, \ldots, \mathsf{n}_k : \mathsf{name},\ \mathsf{x}_1 : \mathsf{name}, \ldots, \mathsf{x}_h : \mathsf{name},$
  $\mathsf{d}_{ij} : \mathsf{n}_i \neq \mathsf{n}_j$

  where $(1 \leq i < j \leq k)$
- For the semantic-aware: inequalities represent the tensor product

$$Name \otimes \cdots \otimes Name \times Name \times \cdots \times Name$$

in the category Set^I.

Representing Apartness

- Apartness can be represented by inequalities assumptions.
- Given $n_1, \ldots, n_k$ names and $x_1, \ldots, x_h$ variables, these are represented by the context

  $\mathtt{n}_1\mathtt{:name}, \ldots, \mathtt{n}_k\mathtt{:name},\ \mathtt{x}_1\mathtt{:name}, \ldots, \mathtt{x}_h\mathtt{:name},$
  $\mathtt{d}_{ij}\mathtt{:n}_i \neq \mathtt{n}_j$

  where $(1 \leq i < j \leq k)$
- Inequalities can be used in proving *non-occurrences judgments*
  - `(notin_cap x M)` holds iff `x` does not occur in `M`;
  - `(notin_proc x P)` holds iff `x` does not occur in `P`;
  - `(notin_form x A)` holds iff `x` does not occur in `A`.

  Inductively defined.

Operational semantics: reduction

$$\frac{}{n[in\ m.P|Q]|m[R] \rightarrow m[n[P|Q]|R]} \quad \text{(Red In)}$$

$$\frac{P \rightarrow Q}{(\nu n)P \rightarrow (\nu n)Q} \quad \text{(Red Res)}$$

$$\frac{}{m[n[out\ m.P|Q]|R] \rightarrow n[P|Q]|m[R]} \quad \text{(Red Out)}$$

$$\frac{P \rightarrow Q}{P|R \rightarrow Q|R} \quad \text{(Red Par)}$$

$$\frac{P' \equiv P, \quad P \rightarrow Q, \quad Q \equiv Q'}{P' \rightarrow Q'} \quad \text{(Red } \equiv\text{)}$$

$$\frac{P \rightarrow Q}{n[P] \rightarrow n[Q]} \quad \text{(Red Amb)}$$

$$\frac{}{(n).P|\langle M\rangle \rightarrow P\{n \leftarrow M\}} \quad \text{(Red Comm)}$$

$$\frac{}{open\ n.P|n[Q] \rightarrow P|Q} \quad \text{(Red Open)}$$

Encoding of reduction

```
Inductive red: proc -> proc -> Prop :=
...
 | red_comm : (P:name->proc)(M:cap)(P':proc)
              (subst_proc M P P') ->
              (red (par (in_act P) (out_act M)) P')
 | red_res  : (P,Q:name->proc)(l:Nlist)
              ((n:name)(Nlist_notin n l) ->
                       (notin_proc n (nu P)) ->
                       (notin_proc n (nu Q)) ->
                       (red (P n) (Q n))
              ) -> (red (nu P) (nu Q))
...
```

- "Fresh" names come with extra assumptions yielding apartness.
- Explicit substitution relations are needed (cf. rule `red_comm`).

Substitution

- Substitution of capabilities for names in capabilities and processes cannot be delegated to the metalanguage (type mismatch proc $\neq$ `name` $\neq$ `cap`)
- Substitution must be represented explicitly by two judgments

```
subst_cap  : cap -> (name->cap) -> cap

subst_proc : cap -> (name->proc) -> proc
```

`(subst_proc M P P')` means

`P'` is the result of "filling the hole" in `P` with `M`.

- Syntax-driven derivations, though.

Satisfaction clauses (sample)

$$P \models \mathbf{T}$$

$$P \models \mathbf{0} \iff P \equiv \mathbf{0}$$

$$P \models \neg\mathcal{A} \iff \text{not } P \models \mathcal{A}$$

$$P \models \mathcal{A} \oslash n \iff (\nu n)P \models \mathcal{A}$$

$$P \models \mathcal{A}@n \iff n[P] \models \mathcal{A}$$

$$P \models \mathcal{A} \rhd \mathcal{B} \iff \text{for all } P' \in \Pi, P' \models \mathcal{A} \text{ implies } P|P' \models \mathcal{B}$$

$$P \models n[\mathcal{A}] \iff \text{there exists } P' \in \Pi \text{ such that } P \equiv n[P'] \text{ and } P' \models \mathcal{A}$$

$$P \models \Diamond\mathcal{A} \iff \text{there exists } P' \in \Pi \text{ such that } P \rightarrow^* P' \text{ and } P' \models \mathcal{A}$$

$$P \models \forall x\mathcal{A} \iff \text{for all } m \in \Lambda, P \models \mathcal{A}\{x \leftarrow m\}$$

Satisfaction clauses (sample)

$$P \models \mathbf{T}$$

$$P \models \mathbf{0} \iff P \equiv \mathbf{0}$$

$$P \models \neg\mathcal{A} \iff \text{not } P \models \mathcal{A}$$

$$P \models \mathcal{A} \oslash n \iff (\nu n)P \models \mathcal{A}$$

$$P \models \mathcal{A}@n \iff n[P] \models \mathcal{A}$$

$$P \models \mathcal{A} \triangleright \mathcal{B} \iff \text{for all } P' \in \Pi, P' \models \mathcal{A} \text{ implies } P|P' \models \mathcal{B}$$

$$P \models n[\mathcal{A}] \iff \text{there exists } P' \in \Pi \text{ such that } P \equiv n[P'] \text{ and } P' \models \mathcal{A}$$

$$P \models \Diamond\mathcal{A} \iff \text{there exists } P' \in \Pi \text{ such that } P \rightarrow^* P' \text{ and } P' \models \mathcal{A}$$

$$P \models \forall x\mathcal{A} \iff \text{for all } m \in \Lambda, P \models \mathcal{A}\{x \leftarrow m\}$$

Notice: in some clauses, satisfaction occurs in negative position.

Encoding of satisfaction (1)

- Inductive definition is not possible (negative occurrences)
- Actually, clauses specify a translation of satisfaction judgments in the metalogic $\Rightarrow$ $\models : \Pi \rightarrow \Phi \rightarrow Prop$ is encoded as a function recursively defined on the syntax of formulas:

```
Fixpoint satF [P:proc;A:form]: Prop:=
<Prop>Cases A of T => True
| (neg B)          => (satF P B) -> False
| (Or A1 A2)       => (satF P A1) \/ (satF P A2)
| (comp_adj A1 A2) => (P':proc)(satF P' A1) ->
                                 (satF (par P P') A2)
| (forall B)       => ((m:name)(satF P (B m)))
...
end.
```

- A goal `(satF P A)` can be automatically `Simp`lified to the corresponding metalogic proposition

Encoding of satisfaction (2)

- A true Natural Deduction proof system with two mutally defined judgments

$$\models_i, \not\models_i : \Pi \to \Phi \to Prop$$

dual of each other

- Negative occurrences of $\models$ are replaced by (positive) $\not\models$

$$\frac{P \not\models_i \mathcal{A}}{P \models_i \neg\mathcal{A}} \qquad \frac{P \models_i \mathcal{A}}{P \not\models_i \neg\mathcal{A}}$$

$$\frac{\text{for all } P'.P' \not\models_i \mathcal{A} \text{ or } P|P' \models \mathcal{B}}{P \models_i \mathcal{A} \rhd \mathcal{B}} \qquad \frac{\text{for some } P'.P \models_i \mathcal{A} \text{ and } \mathcal{P}|\mathcal{P}' \not\models \mathcal{B}}{P \not\models_i \mathcal{A} \rhd \mathcal{B}}$$

- Easily encoded in CIC (Mutual Inductive)
- Useful for proof-theoretical investigations

Ambient Calculus (Meta)theory

- Many properties in [4] deal with names and contexts. E.g.

  For all closed formulas $\mathcal{A}$, processes P, and names m, m', if $m' \notin fn(P) \cup fn(\mathcal{A})$ then $P \models \mathcal{A}$ iff $P\{m \leftarrow m'\} \models \mathcal{A}\{m \leftarrow m'\}$.

Ambient Calculus (Meta)theory

- Many properties in [4] deal with names and contexts. E.g.

  For all closed formulas $\mathcal{A}$, processes P, and names m, m', if $m' \notin fn(P) \cup fn(\mathcal{A})$ then $P \models \mathcal{A}$ iff $P\{m \leftarrow m'\} \models \mathcal{A}\{m \leftarrow m'\}$.

- The theory is too weak
  - we need properties about names and contexts (nothing is known about `name`).
  - inductive reasoning on processes and formulas is problematic (usual induction principle is too weak)

Ambient Calculus (Meta)theory

- Many properties in [4] deal with names and contexts. E.g.

  For all closed formulas $\mathcal{A}$, processes P, and names m, m', if $m' \notin fn(P) \cup fn(\mathcal{A})$ then $P \models \mathcal{A}$ iff $P\{m \leftarrow m'\} \models \mathcal{A}\{m \leftarrow m'\}$.

- The theory is too weak
  - we need properties about names and contexts (nothing is known about `name`).
  - inductive reasoning on processes and formulas is problematic (usual induction principle is too weak)
- Add the Theory of Contexts [HMS01]:

  A set of axiom schemata, which reflect at the theory level some fundamental properties of the intuitive notion of "context" and "occurrence" of variables.

- applicable to any HOAS encoding

The Theory of Contexts

- Decidability of occurrence: every variable either occurs or does not occur free in a term (generalizes decidability of equality on $\mathtt{Var}$). Unnecessary if we are in a classical setting;

The Theory of Contexts

- Decidability of occurrence: every variable either occurs or does not occur free in a term (generalizes decidability of equality on $\mathtt{Var}$). Unnecessary if we are in a classical setting;
- Unsaturability of variables: there exists always a variable which does not occur free in a given term;

The Theory of Contexts

- Decidability of occurrence: every variable either occurs or does not occur free in a term (generalizes decidability of equality on $\mathtt{Var}$). Unnecessary if we are in a classical setting;
- Unsaturability of variables: 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 $M = N$.

The Theory of Contexts

- Decidability of occurrence: every variable either occurs or does not occur free in a term (generalizes decidability of equality on $\mathtt{Var}$). Unnecessary if we are in a classical setting;
- Unsaturability of variables: 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 (i.e., such that $C_M(x) = M$)

The Theory of Contexts for Ambients

```
Axiom dec_name: (x,y:name)x=y \/ ~x=y.

Axiom unsat: (P:proc)(Ex [n:name](notin_proc n P)).

Axiom proc_ext: (P,Q:name->proc)(x:name)
                (notin_proc x (nu P)) ->
                (notin_proc x (nu Q)) ->
                (P x)=(Q x) -> P=Q.

Axiom proc_exp: (P:proc)(n:name)
          (Ex [P':name->proc](notin_proc n (nu P'))
                             /\ P=(P' n)).
```

(Higher order) induction principles

- Induction principles over HOAS datatypes can be derived
- More generally, higher order induction principles over types $\texttt{name}^n\texttt{->proc}$ (for all n) are derivable.
- Stronger than usual ones:

```
Lemma PROC_IND:
  (P:proc -> Prop)
  (P nil) ->
  ...
  ((Q:name->proc)((y:Var)(P (Q y))) -> (P (nu Q))) ->
  (Q:proc)(P Q).
```

- complete induction over size of terms, using β-expansion and extensionality for lifting structural informations from `proc` to `name->proc`

Fresh renaming properties

- Many properties in [4] are “renaming properties”
- All instances of the same pattern:

$$\frac{\text{for some } x \notin \bigcup_{i=1}^{n} fn(C_i[\cdot]) : \mathcal{R}(C_1[x], \ldots, C_n[x])}{\text{for all } y \notin \bigcup_{i=1}^{n} fn(C_i[\cdot]) : \mathcal{R}(C_1[y], \ldots, C_n[y])}$$

  where $\mathcal{R}$ is a given n-ary relation (e.g., structural congruence, capture-avoiding substitution, reduction relation etc.)
- Usually proved by induction either on the derivation of the premise $\mathcal{R}(C_1[x], \ldots, C_n[x])$ or on one of the arguments $C_i[x]$
- A general proof strategy has been streamlined for proving this kind of properties
- β-expansion and extensionality are used for lifting structural information at the higher types

The “new” quantifier

- In [4], Ambient Logics is extended with $\mathbb{N}$ quantifer, defined as a syntactic shorthand

$$\mathbb{N}x.\mathcal{A} \triangleq \exists x.x\#(fnv(\mathcal{A}) \setminus \{x\}) \wedge x \circledR \mathbf{T} \wedge \mathcal{A},$$

- Not directly representable: function fnv is not definable (recursion over HOAS datatypes)
- Represented as a term constructor `new:(name->form)->form` Semantics is easily extended:

```
Fixpoint satF [P:proc;A:form]: Prop:=
<Prop>Cases A of
...
| (new B) => (Ex [m:name](notin_proc m P)
                  /\ (notin_form m (forall B))
                  /\ (satF P (B m)))
end.
```

Properties of “new”

- Most properties of $\mathbb{N}$ have been formalized and proved. For instance:

$$P \models \mathbb{N}x.\mathcal{A} \iff \exists m \in \Lambda.m \notin fn(P,\mathcal{A}) \text{ and } P \models \mathcal{A}\{x \leftarrow m\}$$
$$\iff \forall x \in \Lambda.m \notin fn(P,\mathcal{A}) \text{ implies } P \models \mathcal{A}\{x \leftarrow m\}$$

$$P \models \neg\mathbb{N}x.\mathcal{A} \iff P \models \mathbb{N}x.\neg\mathcal{A}$$

$$P \models \mathbb{N}x.(\mathcal{A}|\mathcal{B}) \iff P \models (\mathbb{N}x.\mathcal{A})|(\mathbb{N}x.\mathcal{B})$$

- last one is said in [4] “of particular interest (and difficulty)”; in this encoding proof is quite simple (a few lines of tactics)

Conclusions

- First implementation of Ambient Calculus and its Logic in a LF
- Most of the theory and the metatheory in [4] (including $\not\mid$) has been formally proved using the Theory of Contexts.
- Benefits for
  - the calculus: new proof system, clarification of the rôle of names and variables, . . .
  - the framework: derivation of properties originally taken as axioms (e.g., induction principles over HOAS datatype), development of a general strategy for renaming properties. . .

Conclusions (really)

Pros and cons of the Theory of Contexts

- **low overhead:** smooth handling of schemata in HOAS, no exotic terms to rule out explicitly. Proofs look *almost* like on the paper.
- **expressive:** induction and recursion principles also over higher-order datatypes. $\not\Vdash$ is rendered faithfully
- but **incompatible** with the Axiom of Unique Choice $\Rightarrow$ expressive power of functions is stricly less than that of relations. Some functions must be then represented by relations.

Conclusions (really)

Pros and cons of the Theory of Contexts

- **low overhead:** smooth handling of schemata in HOAS, no exotic terms to rule out explicitly. Proofs look *almost* like on the paper.
- **expressive:** induction and recursion principles also over higher-order datatypes. $\mathbb{N}$ is rendered faithfully
- but **incompatible** with the Axiom of Unique Choice $\Rightarrow$ expressive power of functions is stricly less than that of relations. Some functions must be then represented by relations.

Theory of Contexts = steroids for weak HOAS

The Axiom of Unique Choice

Proposition [Hof99] The Axiom of Unique Choice

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

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

Soundness

Theorem HOL extended with the Theory of Contexts is sound.

Idea: build a model (close to Schanuel topos) using a tripos ove *functor categories*.

$$\mathcal{F} \xleftarrow{in} I$$

$$Set^{\mathcal{F}} \underset{in_*}{\overset{in^*}{\rightleftarrows}} Set^{I} \overset{\mathbf{a}}{\underset{}{\rightleftarrows}} Sh_{\neg\neg}(I)$$

The index categories are the category of substitutions ($\mathcal{F}$) and injective substitutions (I) over finite sets of atoms. See [BHHMS01] for details.