Nominal Isabelle

or, How Not to be Intimidated by the Variable Convention

Christian Urban

King's College London

Variable Convention:

If $M_1, \ldots, M_n$ occur in a certain mathematical context (e.g. definition, proof), then in these terms all bound variables are chosen to be different from the free variables.

Henk Barendregt in "The Lambda-Calculus: Its Syntax and Semantics"

dinner after my PhD examination

- Aim: develop Nominal Isabelle for reasoning formally about programming languages

Variable Convention:
If $M_1, \ldots, M_n$ occur in a certain mathematical context (e.g. definition, proof), then in these terms all bound variables are chosen to be different from the free variables. —Henk Barendregt

- Aim: develop Nominal Isabelle for reasoning formally about programming languages

Variable Convention:
If $M_1, \ldots, M_n$ occur in a certain mathematical context (e.g. definition, proof), then in these terms all bound variables are chosen to be different from the free variables. —Henk Barendregt

- found an error in an ACM journal paper by Bob Harper and Frank Pfenning about LF (and fixed it in three ways)
- found also fixable errors in my Ph.D.-thesis about cut-elimination (examined by Henk Barendregt and Andy Pitts)
- found that the variable convention can in principle be used for proving false

Nominal Techniques

- Andy Pitts showed me that permutations preserve α-equivalence:

$$t_1 \approx_\alpha t_2 \quad \Rightarrow \quad \pi \bullet t_1 \approx_\alpha \pi \bullet t_2$$

- also permutations and substitutions commute, if you suspend permutations in front of variables

$$\pi \bullet \sigma(t) = \sigma(\pi \bullet t)$$

- this allowed us to define as simple Nominal Unification algorithm

$$\nabla \vdash t \approx_\alpha^? t' \qquad\qquad \nabla \vdash a \,\#^?\, t$$

Nominal Isabelle

- a general theory about atoms and permutations
  - sorted atoms and
  - sort-respecting permutations

- support and freshness

$$\mathit{supp}(x) \stackrel{\text{def}}{=} \{a \mid \mathit{infinite}\ \{b \mid (a\,b)\bullet x \neq x\}\}$$

$$a \# x \stackrel{\text{def}}{=} a \notin \mathit{supp}(x)$$

Nominal Isabelle

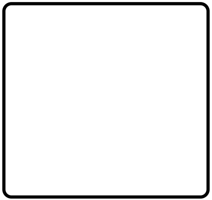
- a general theory about atoms and permutations
  - sorted atoms and
  - sort-respecting permutations
- support and freshness

$$supp(x) \stackrel{\text{def}}{=} \{a \mid infinite\ \{b \mid (a\,b) \bullet x \neq x\}\}$$

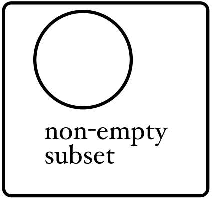
$$a \mathrel{\#} x \stackrel{\text{def}}{=} a \notin supp(x)$$

- allow users to reason about alpha-equivalence classes as if they were syntax trees

New Types in HOL

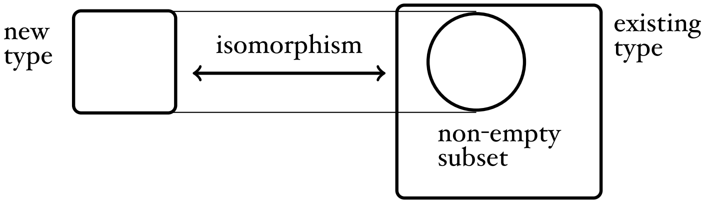


New Types in HOL

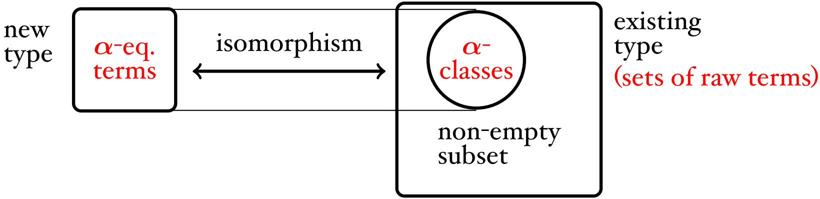


existing
type

New Types in HOL



New Types in HOL

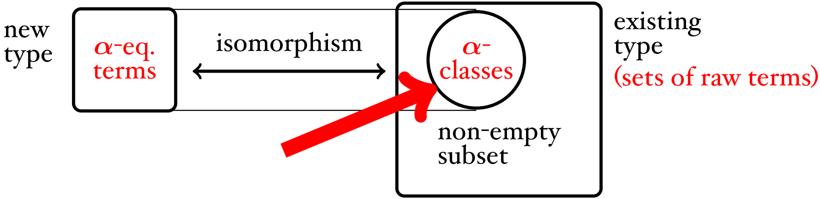


New Types in HOL

new
type

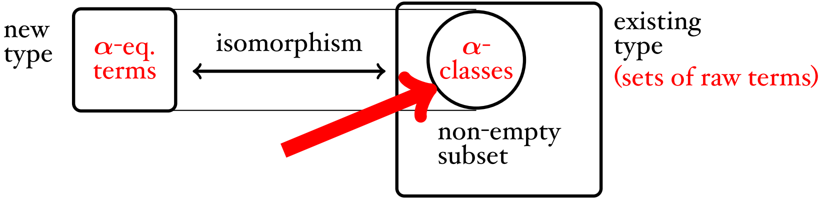
α-eq.
terms

New Types in HOL



define α-equivalence

New Types in HOL



define α-equivalence

The "new types mechanism" is the reason why there is no Nominal Coq.

HOL vs. Nominal

- Nominal logic by Pitts is incompatible with choice principles
- but HOL includes Hilbert's epsilon

HOL vs. Nominal

- Nominal logic by Pitts is incompatible with choice principles
- but HOL includes Hilbert's epsilon
- The solution: Do not require that everything has finite support

$$\mathit{finite}(\mathit{supp}(x)) \quad \Rightarrow \quad a \,\#\, a.x$$

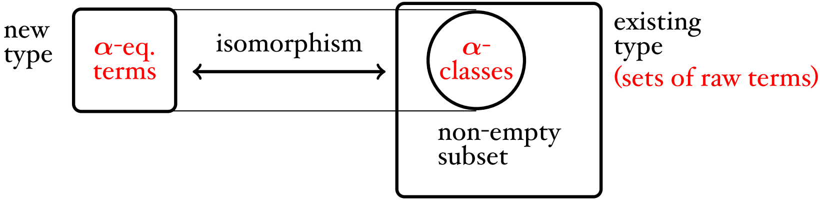
HOL vs. Nominal

- Nominal logic by Pitts is incompatible with choice principles
- but HOL includes Hilbert's epsilon
- The solution: Do not require that everything has finite support

$$a \# a.x$$

Our Work

- define fv and α



Our Work

- define fv and α
- build quotient / new type

new type

α-eq. terms

Our Work

new
type

$\boxed{\begin{array}{c}\alpha\text{-eq.}\\ \text{terms}\end{array}}$

- define fv and α
- build quotient / new type
- derive a reasoning infrastructure (#, distinctness, injectivity, cases,...)

Our Work

new type

α-eq. terms

- define fv and α
- build quotient / new type
- derive a reasoning infrastructure (#, distinctness, injectivity, cases,...)
- derive a **stronger** cases lemma

Our Work

new type $\boxed{\alpha\text{-eq. terms}}$

- define fv and α
- build quotient / new type
- derive a reasoning infrastructure (#, distinctness, injectivity, cases,...)
- derive a **stronger** cases lemma
- from this, a **stronger** induction principle (Barendregt variable convention built in)

$$\text{Foo}\ (\lambda x.\lambda y.t)\ (\lambda u.\lambda v.s)$$

Nominal Isabelle

- Users can for example define lambda-terms as:

atom_decl name

nominal_datatype lam =
  Var name
 | App lam lam
 | Lam x::name t::lam **binds** x **in** t ("Lam _. _")

- These are **named** alpha-equivalence classes, for example

Lam a.(Var a) = Lam b.(Var b)

(Weak) Induction Principles

- The usual induction principle for lambda-terms is as follows:

$$\frac{\begin{array}{l}\forall x.\ P\,x \\ \forall t_1\,t_2.\ P\,t_1 \wedge P\,t_2 \Rightarrow P\,(t_1\ t_2) \\ \forall x\,t.\ P\,t \Rightarrow P\,(\lambda x.t)\end{array}}{P\,t}$$

- It requires us in the lambda-case to show the property P for all binders x.

  (This nearly always requires renamings and they can be tricky to automate.)

Strong Induction Principles

- Therefore we introduced the following strong induction principle:

$$\frac{\begin{array}{l}\forall x\, c.\ P\, c\, x\\ \forall t_1\, t_2\, c.\ (\forall d.\ P\, d\, t_1) \wedge (\forall d. P\, d\, t_2) \Rightarrow P\, c\, (t_1\, t_2)\\ \forall x\, t\, c.\ x \# c \wedge (\forall d. P\, d\, t) \Rightarrow P\, c\, (\lambda x. t)\end{array}}{P\ c\ t}$$

Strong Induction Principles

- Therefore we introduced the following strong induction principle:

$$\dfrac{\begin{array}{l}\forall x\, c.\ P\, c\, x \\ \forall t_1\, t_2\, c.\ (\forall d.\ P\, d\, t_1) \wedge (\forall d. P\, d\, t_2) \Rightarrow P\, c\, (t_1\, t_2) \\ \forall x\, t\, c.\ x \# c \wedge (\forall d. P\, d\, t) \Rightarrow P\, c\, (\lambda x.t)\end{array}}{P\ c\ t}$$

The variable over which the induction proceeds:

"...By induction over the structure of M..."

Strong Induction Principles

- Therefore we introduced the following strong induction principle:

$$\frac{\begin{array}{l}\forall x\, c.\ P\, c\, x \\ \forall t_1\, t_2\, c.\ (\forall d.\ P\, d\, t_1) \wedge (\forall d. P\, d\, t_2) \Rightarrow P\, c\, (t_1\, t_2) \\ \forall x\, t\, c.\ x \ \#\ c \wedge (\forall d. P\, d\, t) \Rightarrow P\, c\, (\lambda x. t)\end{array}}{P\ c\ t}$$

The **context** of the induction; i.e. what the binder should be fresh for $\Rightarrow (x, y, N, L)$:

"...By the variable convention we can assume $z \not\equiv x, y$ and z not free in N, L..."

Strong Induction Principles

- Therefore we introduced the following strong induction principle:

$$\frac{\begin{array}{l}\forall x\, c.\ P\, c\, x \\ \forall t_1\, t_2\, c.\ (\forall d.\ P\, d\, t_1) \wedge (\forall d. P\, d\, t_2) \Rightarrow P\, c\, (t_1\, t_2) \\ \forall x\, t\, c.\ x \,\#\, c \wedge (\forall d. P\, d\, t) \Rightarrow P\, c\, (\lambda x. t)\end{array}}{P\ \ c\ t}$$

The property to be proved by induction:

$$\begin{array}{l}\lambda(x,y,N,L).\,\lambda M.\ x \neq y \wedge x \,\#\, L \Rightarrow \\ \qquad M[x:=N][y:=L] = M[y:=L][x:=N[y:=L]]\end{array}$$

Binding Sets of Names

- binding sets of names has some interesting properties:

$$\forall\{x, y\}.\, x \rightarrow y \quad \approx_\alpha \quad \forall\{y, x\}.\, y \rightarrow x$$

* x, y, z are assumed to be distinct

Binding Sets of Names

- binding sets of names has some interesting properties:

$$\forall\{x,y\}.\, x \rightarrow y \quad \approx_\alpha \quad \forall\{y,x\}.\, y \rightarrow x$$

$$\forall\{x,y\}.\, x \rightarrow y \quad \not\approx_\alpha \quad \forall\{z\}.\, z \rightarrow z$$

* x, y, z are assumed to be distinct

Binding Sets of Names

- binding sets of names has some interesting properties:

$$\forall\{x, y\}.\, x \rightarrow y \quad \approx_\alpha \quad \forall\{y, x\}.\, y \rightarrow x$$

$$\forall\{x, y\}.\, x \rightarrow y \quad \not\approx_\alpha \quad \forall\{z\}.\, z \rightarrow z$$

$$\forall\{x\}.\, x \rightarrow y \quad \approx_\alpha \quad \forall\{x, z\}.\, x \rightarrow y$$

provided z is fresh for the type

* x, y, z are assumed to be distinct

Binding Sets of Names

- bind ... prop...

For type-schemes the order of bound names does not matter, and α-equivalence is preserved under vacuous binders.

$$\forall\{x, y\}.\, x \to y \quad \not\approx_\alpha \quad \forall\{z\}.\, z \to z$$

$$\forall\{x\}.\, x \to y \quad \approx_\alpha \quad \forall\{x, z\}.\, x \to y$$

provided z is fresh for the type

* x, y, z are assumed to be distinct

Other Binding Modes

- alpha-equivalence being preserved under vacuous binders is <u>not</u> always wanted:

$$\text{let } x = 3 \text{ and } y = 2 \text{ in } x - y \text{ end}$$

Other Binding Modes

- alpha-equivalence being preserved under vacuous binders is not always wanted:

$$\text{let } x = 3 \text{ and } y = 2 \text{ in } x - y \text{ end}$$
$$\approx_\alpha \text{let } y = 2 \text{ and } x = 3 \text{ in } x - y \text{ end}$$

Other Binding Modes

- alpha-equivalence being preserved under vacuous binders is <u>not</u> always wanted:

$$\mathrm{let}\ x = 3\ \mathrm{and}\ y = 2\ \mathrm{in}\ x - y\ \mathrm{end}$$

$$\not\approx_\alpha \mathrm{let}\ y = 2\ \mathrm{and}\ x = 3\ \mathrm{and}\ z = \mathrm{loop}\ \mathrm{in}\ x - y\ \mathrm{end}$$

Even Another Binding Mode

- sometimes one wants to abstract more than one name, but the order <u>does</u> matter

$$\mathrm{let}\ (\boldsymbol{x},\boldsymbol{y}) = (3,2)\ \mathrm{in}\ \boldsymbol{x} - \boldsymbol{y}\ \mathrm{end}$$

$$\not\approx_\alpha \mathrm{let}\ (\boldsymbol{y},\boldsymbol{x}) = (3,2)\ \mathrm{in}\ \boldsymbol{x} - \boldsymbol{y}\ \mathrm{end}$$

Specification of Binding

```
nominal_datatype trm =
    Var name
  | App trm trm
  | Lam x::name t::trm    bind x in t
  | Let as::assns t::trm  bind bn(as) in t
and assns =
    ANil
  | ACons name trm assns
```

Specification of Binding

```
nominal_datatype trm =
    Var name
  | App trm trm
  | Lam x::name t::trm     bind x in t
  | Let as::assns t::trm   bind bn(as) in t
and assns =
    ANil
  | ACons name trm assns
binder bn where
    bn(ANil) = []
  | bn(ACons a t as) = [a] @ bn(as)
```

So Far So Good

- A Faulty Lemma with the Variable Convention?

> **Variable Convention:**
>
> If $M_1, \ldots, M_n$ occur in a certain mathematical context (e.g. definition, proof), then in these terms all bound variables are chosen to be different from the free variables.
>
> Barendregt in "The Lambda-Calculus: Its Syntax and Semantics"

Inductive Definitions:

$$\frac{\text{prem}_1 \ldots \text{prem}_n \ \ \text{scs}}{\text{concl}}$$

Rule Inductions:

1.) Assume the property for the premises. Assume the side-conditions.

2.) Show the property for the conclusion.

Faulty Reasoning

- Consider the two-place relation foo:

$$\frac{}{x \mapsto x} \qquad \frac{}{t_1\ t_2 \mapsto t_1\ t_2} \qquad \frac{t \mapsto t'}{\lambda x.t \mapsto t'}$$

Faulty Reasoning

- Consider the two-place relation foo:

$$\frac{}{x \mapsto x} \qquad \frac{}{t_1\ t_2 \mapsto t_1\ t_2} \qquad \frac{t \mapsto t'}{\lambda x.t \mapsto t'}$$

- The lemma we going to prove:

  Let $t \mapsto t'$. If $y \# t$ then $y \# t'$.

Faulty Reasoning

- Consider the two-place relation foo:

$$\frac{}{x \mapsto x} \qquad \frac{}{t_1\ t_2 \mapsto t_1\ t_2} \qquad \frac{t \mapsto t'}{\lambda x.t \mapsto t'}$$

- The lemma we going to prove:

  Let $t \mapsto t'$. If $y \# t$ then $y \# t'$.

- Cases 1 and 2 are trivial:
  - If $y \# x$ then $y \# x$.
  - If $y \# t_1\ t_2$ then $y \# t_1\ t_2$.

Faulty Reasoning

- Consider the two-place relation foo:

$$\frac{}{x \mapsto x} \qquad \frac{}{t_1\ t_2 \mapsto t_1\ t_2} \qquad \frac{t \mapsto t'}{\lambda x.t \mapsto t'}$$

- The lemma we going to prove:

  Let $t \mapsto t'$. If $y \# t$ then $y \# t'$.

- Case 3:
  - We know $y \# \lambda x.t$. We have to show $y \# t'$.
  - The IH says: if $y \# t$ then $y \# t'$.

Variable Convention:

If $M_1, \ldots, M_n$ occur in a certain mathematical context (e.g. definition, proof), then in these terms all bound variables are chosen to be different from the free variables.

In our case:

The free variables are y and t'; the bound one is x.

By the variable convention we conclude that $x \neq y$.

- Case 3:
  - We know $y \# \lambda x.t$. We have to show $y \# t'$.
  - The IH says: if $y \# t$ then $y \# t'$.

Variable Convention:

If $M_1, \ldots, M_n$ occur in a certain mathematical context (e.g. definition, proof), then in these terms all bound variables are chosen to be different from the free variables.

In our case:

The free variables are y and t'; the bound one is x.

By the variable convention we conclude that $x \neq y$.

$$y \not\in \mathrm{fv}(\lambda x.t) \iff y \not\in \mathrm{fv}(t) - \{x\} \overset{x \neq y}{\iff} y \not\in \mathrm{fv}(t)$$

- Case 3:
  - We know $y \;\#\; \lambda x.t$. We have to show $y \;\#\; t'$.
  - The IH says: if $y \;\#\; t$ then $y \;\#\; t'$.

Variable Convention:

If $M_1, \ldots, M_n$ occur in a certain mathematical context (e.g. definition, proof), then in these terms all bound variables are chosen to be different from the free variables.

In our case:

The free variables are y and t'; the bound one is x.

By the variable convention we conclude that $x \neq y$.

$$y \notin \mathrm{fv}(\lambda x.t) \iff y \notin \mathrm{fv}(t) - \{x\} \overset{x \neq y}{\iff} y \notin \mathrm{fv}(t)$$

- Case 3:
  - We know $y \# \lambda x.t$. We have to show $y \# t'$.
  - The IH says: if $y \# t$ then $y \# t'$.
  - So we have $y \# t$. Hence $y \# t'$ by IH. Done!

Faulty Reasoning

- Consider the two-place relation foo:

$$\frac{}{x \mapsto x} \qquad \frac{}{t_1\ t_2 \mapsto t_1\ t_2} \qquad \frac{t \mapsto t'}{\lambda x.t \mapsto t'}$$

- The lemma we going to prove:

  Let $t \mapsto t'$. If $y \# t$ then $y \# t'$.

- Case 3:
  - We know $y \# \lambda x.t$. We have to show $y \# t'$.
  - The IH says: if $y \# t$ then $y \# t'$.
  - So we have $y \# t$. Hence $y \# t'$ by IH. Done!

Conclusions

- The user does not see anything of the “raw” level.
- The Nominal Isabelle automatically derives the strong structural induction principle for <u>**all**</u> nominal datatypes (not just the lambda-calculus)
- They are easy to use: you just have to think carefully what the variable convention should be.
- We can explore the dark corners of the variable convention: when and where it can be used safely.

Conclusions

- The user does not see anything of the "raw" level.
- The Nominal Isabelle automatically derives the strong structural induction principle for **<u>all</u>** nominal datatypes (not just the lambda-calculus)
- They are easy to use: you just have to think carefully what the variable convention should be.
- We can explore the dark corners of the variable convention: when and where it can be used safely.
- **Main Point:** Actually these proofs using the variable convention are all trivial / obvious / routine...**provided** you use Nominal Isabelle. ;o)

Thank you very much!
Questions?