The Nominal Datatype Package in Isabelle/HOL

Christian Urban

University of Munich

joint work with Stefan Berghofer, Markus Wenzel, Alexander Krauss. . .

The POPLmark-Challenge

"How close are we to a world where programming language papers are routinely supported by machine-checked metatheory proofs, where full-scale language definitions are expressed in machine-processed mathematics...?"

Obviously we aren't there yet:

- for binders reasonable powerful tools are available: de-Bruijn indices (in Coq, Isabelle,...) or HOAS (mainly in Twelf)
- **but** apart from some theorem-proving experts, nobody seems to use them; non-experts are still routinely do their proofs on paper, only

The POPLmark-Challenge

“How close are we to a world where program-
ming … ted
by m … ere
full-s … d in
mach

The aim of the nominal datatype package is to support the kind of reasoning that is employed on paper. The hope is: if you can do formal proofs on paper, then you can implement them in Isabelle/HOL with ease.

That is not a trivial task.

Obviou

- f
  c …) or
  HOAS (mainly in Twelf)
- **but** apart from some theorem-proving experts, nobody seems to use them; non-experts are still routinely do their proofs on paper, only

Substitution Lemma: If $x \not\equiv y$ and $x \notin FV(L)$, then

$$M[x := N][y := L] \equiv M[y := L][x := N[y := L]].$$

Proof: By induction on the structure of M.

- **Case 1:** M is a variable.

This is a **simple** example illustrating a point. We have already implemented much more complicated proofs, e.g. Church-Rosser, SN, transitivity of subtyping in POPLmark, etc.

$$\equiv \lambda z.(M_1[y := L][x := N[y := L]])$$

$$\equiv (\lambda z.M_1)[y := L][x := N[y := L]].$$

- **Case 3:** $M \equiv M_1 M_2$. The statement follows again from the induction hypothesis. $\square$

Substitution Lemma: If $x \not\equiv y$ and $x \notin FV(L)$, then

$$M[x := N][y := L] \equiv M[y := L][x := N[y := L]].$$

Proof: By induction on the structure of M.

- **Case 1:** M is a variable.

  Case 1.1. $M \equiv x$. Then both sides equal $N[y := L]$ since $x \not\equiv y$.

  Case 1.2. $M \equiv y$. Then both sides equal L, for $x \notin FV(L)$ implies $L[x := \ldots] \equiv L$.

  Case 1.3. $M \equiv z \not\equiv x, y$. Then both sides equal z.

- **Case 2:** $M \equiv \lambda z.M_1$. By the variable convention we may assume that $z \not\equiv x, y$ and z is not free in N, L. Then by induction hypothesis

$$\begin{aligned} & (\lambda z.M_1)[x := N][y := L] \\ \equiv\ & \lambda z.(M_1[x := N][y := L]) \\ \equiv\ & \lambda z.(M_1[y := L][x := N[y := L]]) \\ \equiv\ & (\lambda z.M_1)[y := L][x := N[y := L]]. \end{aligned}$$

- **Case 3:** $M \equiv M_1M_2$. The statement follows again from the induction hypothesis. $\square$

Substitution Lemma: If $x \not\equiv y$ and $x \notin FV(L)$, then

$$M[x := N][y := L] \equiv M[y := L][x := N[y := L]].$$

Proof: By induction on the structure of M.

- **Case 1:** M is a variable.

  Case 1.1. $M \equiv x$. Then both sides equal $N[y := L]$ since $x \not\equiv y$.

  Case 1.2. $M \equiv y$. Then both sides equal L, for $x \notin FV(L)$ implies $L[x := \ldots] \equiv L$.

  Case 1.3. $M \equiv z \not\equiv x, y$. Then both sides equal z.

- **Case 2:** $M \equiv \lambda z.M_1$. By the variable convention we may assume that $z \not\equiv x, y$ and z is not free in N, L. Then by induction hypothesis

$$\begin{aligned} & (\lambda z.M_1)[x := N][y := L] \\ \equiv\ & \lambda z.(M_1[x := N][y := L]) \\ \equiv\ & \lambda z.(M_1[y := L][x := N[y := L]]) \\ \equiv\ & (\lambda z.M_1)[y := L][x := N[y := L]]. \end{aligned}$$

- **Case 3:** $M \equiv M_1 M_2$. The statement follows again from the induction hypothesis. $\square$

Substitution Lemma: If $x \not\equiv y$ and $x \notin FV(L)$, then

$$M[x := N][y := L] \equiv M[y := L][x := N[y := L]].$$

Proof: By induction on the structure of M.

- **Case 1:** M is a variable.

  Case 1.1. $M \equiv x$. Then both sides equal $N[y := L]$ since $x \not\equiv y$.

  Case 1.2. $M \equiv y$. Then both sides equal L, for $x \notin FV(L)$ implies $L[x := \ldots] \equiv L$.

  Case 1.3. $M \equiv z \not\equiv x, y$. Then both sides equal z.

- **Case 2:** $M \equiv \lambda z.M_1$. By the variable convention we may assume that $z \not\equiv x, y$ and z is not free in N, L. Then by induction hypothesis

$$\begin{aligned}
& (\lambda z.M_1)[x := N][y := L] \\
\equiv\ & \lambda z.(M_1[x := N][y := L]) \\
\equiv\ & \lambda z.(M_1[y := L][x := N[y := L]]) \\
\equiv\ & (\lambda z.M_1)[y := L][x := N[y := L]].
\end{aligned}$$

- **Case 3:** $M \equiv M_1 M_2$. The statement follows again from the induction hypothesis. $\square$

Substitution Lemma: If $x \not\equiv y$ and $x \notin FV(L)$, then

$$M[x := N][y := L] \equiv M[y := L][x := N[y := L]].$$

Proof: By induction on the structure of M.

- **Case 1:** M is a variable.

  Case 1.1. $M \equiv x$. Then both sides equal $N[y := L]$ since $x \not\equiv y$.

  Case 1.2. $M \equiv y$. Then both sides equal L, for $x \notin FV(L)$ implies $L[x := \ldots] \equiv L$.

  Case 1.3. $M \equiv z \not\equiv x, y$. Then both sides equal z.

- **Case 2:** $M \equiv \lambda z.M_1$. By the variable convention we may assume that $z \not\equiv x, y$ and z is not free in N, L. Then by induction hypothesis

$$\begin{aligned} & (\lambda z.M_1)[x := N][y := L] \\ \equiv\ & \lambda z.(M_1[x := N][y := L]) \\ \equiv\ & \lambda z.(M_1[y := L][x := N[y := L]]) \\ \equiv\ & (\lambda z.M_1)[y := L][x := N[y := L]]. \end{aligned}$$

- **Case 3:** $M \equiv M_1 M_2$. The statement follows again from the induction hypothesis. $\square$

Substitution Lemma: If $x \not\equiv y$ and $x \notin FV(L)$, then

$$M[x := N][y := L] \equiv M[y := L][x := N[y := L]].$$

Proof: By induction on the structure of M.

- **Case 1:** M is a variable.

  Case 1.1. $M \equiv x$. Then both sides equal $N[y := L]$ since $x \not\equiv y$.

  Case 1.2. $M \equiv y$. Then both sides equal L, for $x \notin FV(L)$ implies $L[x := \ldots] \equiv L$.

  Case 1.3. $M \equiv z \not\equiv x, y$. Then both sides equal z.

- **Case 2:** $M \equiv \lambda z.M_1$. By the variable convention we may assume that $z \not\equiv x, y$ and z is not free in N, L. Then by induction hypothesis

$$\begin{aligned} & (\lambda z.M_1)[x := N][y := L] \\ \equiv\ & \lambda z.(M_1[x := N][y := L]) \\ \equiv\ & \lambda z.(M_1[y := L][x := N[y := L]]) \\ \equiv\ & (\lambda z.M_1)[y := L][x := N[y := L]]. \end{aligned}$$

- **Case 3:** $M \equiv M_1 M_2$. The statement follows again from the induction hypothesis. □

Substitution Len

$M[x := N$

Remember: only if $y \neq x$ and $x \notin FV(N)$ then

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

$$\begin{aligned}
& (\lambda z.M_1)[x := N][y := L] & \\
\equiv\ & (\lambda z.(M_1[x := N]))[y := L] & \overset{1}{\leftarrow} \\
\equiv\ & \lambda z.(M_1[x := N][y := L]) & \overset{2}{\leftarrow} \\
\equiv\ & \lambda z.(M_1[y := L][x := N[y := L]]) & \text{IH} \\
\equiv\ & (\lambda z.(M_1[y := L]))[x := N[y := L]]) & \overset{2}{\rightarrow}\ ! \\
\equiv\ & (\lambda z.M_1)[y := L][x := N[y := L]]. & \overset{1}{\rightarrow}
\end{aligned}$$

Proof: By inducti

- **Case 1:** M is

  Case 1.1. $M \equiv$

  Case 1.2. $M \equiv$

  implie

  Case 1.3. $M \equiv$

- **Case 2:** $M \equiv$ e
  that $z \not\equiv x, y$ s

$$\begin{aligned}
& (\lambda z.M_1)[x := N][y := L] \\
\equiv\ & \lambda z.(M_1[x := N][y := L]) \\
\equiv\ & \lambda z.(M_1[y := L][x := N[y := L]]) \\
\equiv\ & (\lambda z.M_1)[y := L][x := N[y := L]].
\end{aligned}$$

- **Case 3:** $M \equiv M_1 M_2$. The statement follows again from the induction hypothesis. $\square$

Substitution Lemma: If $x \not\equiv y$ and $x \notin FV(L)$, then

$$M[x := N][y := L] \equiv M[y := L][x := N[y := L]].$$

Proof: By induction on the structure of M.

- **Case 1:** M is a variable.

  Case 1.1. $M \equiv x$. Then both sides equal $N[y := L]$ since $x \not\equiv y$.

  Case 1.2. $M \equiv y$. Then both sides equal L, for $x \notin FV(L)$ implies $L[x := \ldots] \equiv L$.

  Case 1.3. $M \equiv z \not\equiv x, y$. Then both sides equal z.

- **Case 2:** $M \equiv \lambda z.M_1$. By the variable convention we may assume that $z \not\equiv x, y$ and z is not free in N, L. Then by induction hypothesis

$$\begin{aligned}
&(\lambda z.M_1)[x := N][y := L] \\
\equiv\ & \lambda z.(M_1[x := N][y := L]) \\
\equiv\ & \lambda z.(M_1[y := L][x := N[y := L]]) \\
\equiv\ & (\lambda z.M_1)[y := L][x := N[y := L]].
\end{aligned}$$

- **Case 3:** $M \equiv M_1 M_2$. The statement follows again from the induction hypothesis. $\square$

Formal Proof in Isabelle

```
lemma forget:
assumes a: "$x \ \# \ L$"
shows "$L[x::=P] = L$"
using a by (nominal_induct $L$ avoiding: $x$ $P$ rule: lam.induct)
           (auto simp add: abs_fresh fresh_atm)

lemma fresh_fact:
fixes $z$ :: "name"
assumes a: "$z \ \# \ N$" and b: "$z \ \# \ L$"
shows "$z \ \# \ N[y::=L]$"
using a b by (nominal_induct $N$ avoiding: $z$ $y$ $L$ rule: lam.induct)
             (auto simp add: abs_fresh fresh_atm)

lemma subst_lemma:
assumes a: "$x \neq y$" and b: "$x \ \# \ L$"
shows "$M[x::=N][y::=L] = M[y::=L][x::=N[y::=L]]$"
using a b by (nominal_induct $M$ avoiding: $x$ $y$ $N$ $L$ rule: lam.induct)
             (auto simp add: forget fresh_fact)
```

Formal Proof in Isabelle

lemma forget:
assumes a: "$x \ \# \ L$"
shows "$L[x ::= P] = L$"
using a by (nominal_induct L avoid
(auto simp add: abs_fres

- stands for $x \notin FV(L)$
- reads as "x fresh for L"
- is a polymorphic construction from the Nominal Logic Work by Pitts

lemma fresh_fact:
fixes z :: "name"
assumes a: "$z \ \# \ N$" and b: "$z \ \#$
shows "$z \ \# \ N[y ::= L]$"
using a b by (nominal_induct N avoiding: z y L rule: lam.induct)
(auto simp add: abs_fresh fresh_atm)

lemma subst_lemma:
assumes a: "$x \neq y$" and b: "$x \ \# \ L$"
shows "$M[x ::= N][y ::= L] = M[y ::= L][x ::= N[y ::= L]]$"
using a b by (nominal_induct M avoiding: x y N L rule: lam.induct)
(auto simp add: forget fresh_fact)

Formal Proof in Isabelle

lemma forget:
assumes a: "$x \;\#\; L$"
shows "$L[x ::= P] = L$"
using a by (nominal_induct L avoiding: x P rule: lam.induct)
          (auto simp add: abs_fresh fresh_atm)

lemma fresh_fact:
fixes z :: "name"
assumes a: "$z \;\#\; N$" and b: "$z \;\#\; L$"
shows "$z \;\#\; N[y ::= L]$"
using a b by (nominal_induct N avoiding: z y L rule: lam.induct)
          (auto simp add: abs_fresh fresh_atm)

lemma subst_lemma:
assumes a: "$x \neq y$" and b: "$x \;\#\; L$"
shows "$M[x ::= N][y ::= L] = M[y ::= L][x ::= N[y ::= L]]$"
using a b by (nominal_induct M avoiding: x y N L rule: lam.induct)
          (auto simp add: forget fresh_fact)

Crucial Points

The nominal datatype package generates the α-equivalence classes as a **type** in Isabelle/HOL.

```
atom_decl name

nominal_datatype lam =
          Var  "name"
        | App  "lam" "lam"
        | Lam  "«name»lam"  ("Lam [_]._" [100,100] 100)
```

The type lam is defined so that we have **equations**

$$\text{Lam } [a].(\text{Var } a) = \text{Lam } [b].(\text{Var } b)$$

which do **not** hold for "normal" datatypes.

Structural Induction

Then automatically generated is a structural induction principle that has Barendregt's convention already build in:

$$\frac{\begin{array}{l}\forall a\, x.\ P\, x\, (\mathsf{Var}\, a) \\ \forall t_1\, t_2\, x.\ (\forall z.\, P\, z\, t_1) \wedge (\forall z.\, P\, z\, t_2) \Rightarrow P\, x\, (\mathsf{App}\, t_1\, t_2) \\ \forall a\, t\, x.\ a \,\#\, x \wedge (\forall z.\, P\, z\, t) \Rightarrow P\, x\, (\mathsf{Lam}\, [a].t)\end{array}}{P\, x\, t}$$

Structural Induction

Then automatically generated is a structural induction principle that has Barendregt's convention already build in:

$$\frac{\begin{array}{l}\forall a\, x.\ P\, x\, (\mathsf{Var}\, a)\\ \forall t_1\, t_2\, x.\ (\forall z.\, P\, z\, t_1) \wedge (\forall z.\, P\, z\, t_2) \Rightarrow P\, x\, (\mathsf{App}\, t_1\, t_2)\\ \forall a\, t\, x.\ a \# x \wedge (\forall z.\, P\, z\, t) \Rightarrow P\, x\, (\mathsf{Lam}\, [a].t)\end{array}}{P\, x\, t}$$

the variable over which the induction proceeds:

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

Structural Induction

Then automatically generated is a structural induction principle that has Barendregt's convention already build in:

$$\dfrac{\begin{array}{l}\forall a\,x.\ P\,x\,(\mathsf{Var}\,a)\\ \forall t_1\,t_2\,x.\ (\forall z.\,P\,z\,t_1) \wedge (\forall z.\,P\,z\,t_2) \Rightarrow P\,x\,(\mathsf{App}\,t_1\,t_2)\\ \forall a\,t\,x.\ a\ \#\ x \wedge (\forall z.\,P\,z\,t) \Rightarrow P\,x\,(\mathsf{Lam}\,[a].t)\end{array}}{P\,x\,t}$$

the context of the induction; for which the binder should be fresh $\Rightarrow (x, y, N, L)$:

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

Structural Induction

Then automatically generated is a structural induction principle that has Barendregt's convention already build in:

$$\dfrac{\begin{array}{l}\forall a\, x.\ P\, x\, (\mathsf{Var}\, a) \\ \forall t_1\, t_2\, x.\ (\forall z.\, P\, z\, t_1) \wedge (\forall z.\, P\, z\, t_2) \Rightarrow P\, x\, (\mathsf{App}\, t_1\, t_2) \\ \forall a\, t\, x.\ a \mathrel{\#} x \wedge (\forall z.\, P\, z\, t) \Rightarrow P\, x\, (\mathsf{Lam}\, [a].t)\end{array}}{P\, x\, t}$$

the property to be proved by induction:

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

Structural Induction

Then automatically generated is a structural induction principle that has Barendregt's convention already build in:

$$\frac{\begin{array}{l}\forall a\,x.\ P\,x\,(\mathsf{Var}\,a)\\ \forall t_1\,t_2\,x.\ (\forall z.\,P\,z\,t_1) \wedge (\forall z.\,P\,z\,t_2) \Rightarrow P\,x\,(\mathsf{App}\,t_1\,t_2)\\ \forall a\,t\,x.\ a \# x \wedge (\forall z.\,P\,z\,t) \Rightarrow P\,x\,(\mathsf{Lam}\,[a].t)\end{array}}{P\,x\,t}$$

One only has to write (more in the talk of Markus Wenzel):

by (nominal_induct M avoiding: $x\ y\ N\ L$ rule: lam.induct)

Structural Induction

Then automatically generated is a structural induction principle that has Barendregt's convention already build in:

$$\frac{\begin{array}{l}\forall a\, x.\ P\, x\, (\mathsf{Var}\, a)\\ \forall t_1\, t_2\, x.\ (\forall z.\, P\, z\, t_1) \wedge (\forall z.\, P\, z\, t_2) \Rightarrow P\, x\, (\mathsf{App}\, t_1\, t_2)\\ \forall a\, t\, x.\ a \# x \wedge (\forall z.\, P\, z\, t) \Rightarrow P\, x\, (\mathsf{Lam}\, [a].t)\end{array}}{P\, x\, t}$$

The lambda-case amounts to:

$$\frac{\begin{array}{l}z \# (x, y, N, L) \quad \color{red}{!!}\\ \forall xyNL.\ x \neq y \wedge x \# L \Rightarrow\\ \qquad M[x ::= N][y ::= L] = M[y ::= L][x ::= N[y ::= L]]\\ x \neq y, x \# L\end{array}}{\begin{array}{l}(\mathsf{Lam}\, [z].M)[x ::= N][y ::= L] =\\ \qquad (\mathsf{Lam}\, [z].M)[y ::= L][x ::= N[y ::= L]]\end{array}}$$

Structural Induction

Then automatically generated is a structural induction principle that has Barendregt's convention already build in:

$$\frac{\begin{array}{l}\forall a\,x.\ P\,x\,(\mathsf{Var}\,a)\\ \forall t_1\,t_2\,x.\ (\forall z.\,P\,z\,t_1) \wedge (\forall z.\,P\,z\,t_2) \Rightarrow P\,x\,(\mathsf{App}\,t_1\,t_2)\\ \forall a\,t\,x.\ a \# x \wedge (\forall z.\,P\,z\,t) \Rightarrow P\,x\,(\mathsf{Lam}\,[a].t)\end{array}}{P\,x\,t}$$

By the way: There is a condition for when Barendregt's variable convention is applicable—it is almost always satisfied, but not always:

x needs to be finitely supported (is not allowed to mention all names as free)

Conclusion

- the nominal datatype package is still work in progress
- already quite usable for the lambda-calculus
  - Church-Rosser
  - strong normalisation using candidates
  - weakening
  - (transitivity of subtyping, π-calc.)
- mailing list and download

  nominal-isabelle@mailbroy.informatik.tu-muenchen.de

  http://isabelle.in.tum.de/nominal/