Nominal Datatypes in Isabelle/HOL

Stefan Berghofer
with
Christian Urban, Julien Narboux and Markus Wenzel
Motivation
A paper proof [Barendregt, 1981]

**Substitution lemma:** If $x \neq y$ and $x \notin FV(L)$, then

$$M[x \mapsto N][y \mapsto L] = M[y \mapsto L][x \mapsto N[y \mapsto L]].$$

**Proof:** By induction on the structure of $M$.

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

- **Case 1.1.** $M = x$. Then both sides equal $N[y \mapsto L]$, since $x \neq y$.
- **Case 1.2.** $M = y$. Then both sides equal $L$, for $x \notin FV(L)$ implies $L[x \mapsto \ldots] = L$.
- **Case 1.3.** $M = z \neq x, y$. Then both sides equal $z$.

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

$$\begin{align*}
(\lambda z. M_1)[x \mapsto N][y \mapsto L] &= \lambda z.(M_1[x \mapsto N][y \mapsto L]) \\
&= \lambda z.(M_1[y \mapsto L][x \mapsto N[y \mapsto L]]) \\
&= (\lambda z. M_1)[y \mapsto L][x \mapsto N[y \mapsto L]]
\end{align*}$$

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

\[ \square \]
What the experts say...

“We thank T. Thacher Robinson for showing us on August 19, 1962 by a counterexample the existence of an error in our handling of bound variables.”


“When doing the formalization, I discovered that the core part of the proof ... is fairly straightforward and only requires a good understanding of the paper version. However, in completing the proof I observed that in certain places I had to invest much more work than expected, e.g. proving lemmas about substitution and weakening.”

Thorsten Altenkirch in Proceedings of TLCA, 1993

“Proving theorems about substitutions (and related operations such as \(\alpha\)-conversion) required far more time than any other variety of theorem.”

Myra VanInwegen in her PhD-thesis, 1996

⇒ Better tool support necessary
Our tool: Isabelle

- Developed (since 1986) by Larry Paulson (Cambridge) and Tobias Nipkow
- Interactive theorem prover
- Logical Framework
  Description of various object logics using a meta logic (Isabelle/Pure)
- Most well-developed object logic: Isabelle/HOL
- Design philosophy
  - Inferences may only be performed by a small kernel (“LCF approach”)
  - Definitional theory extension
    New concepts (such as inductive datatypes and predicates) must be defined using already existing, simpler concepts.

“The method of ‘postulating’ what we want has many advantages; they are the same as the advantages of theft over honest toil. Let us leave them to others and proceed with our honest toil.”

Bertrand Russell, Introduction to Mathematical Philosophy
Hierarchy of definitional packages

Nominal Datatype

Datatype

Typedef

Inductive

Rec. Function

Definition

Coinductive

Isabelle/HOL

Isabelle/Pure

Nominal Datatypes in Isabelle/HOL
Existing approaches for reasoning with bound variables

- “Name-carrying” syntax
  + readable
  - \(\alpha\)-equivalence must be formalized explicitly
  - substitution function requires variable renaming

- De Bruijn indices
  + \(\alpha\)-equivalence coincides with syntactic equality
  + simple induction / recursion principles
  - substitution function requires index calculations
  - unreadable

- “Locally nameless” approach
  + \(\alpha\)-equivalence coincides with syntactic equality
  + substitution function does not require index calculations
  - well-formedness must be formalized explicitly

- Higher order abstract syntax
  + abstraction and substitution “for free”
  - exotic terms
Our approach
A more abstract approach

**Problem:** How can we hide details of the representation from the user?

**Possible solution:**
Introduce new type, whose elements correspond to...
- ... the $\alpha$-equivalence classes of “name-carrying”-terms, or
- ... the well-formed “locally nameless” terms.

![Diagram showing lambda terms with green dots and ellipsis]

**Question:** How does abstract “interface” for this type look like? ⇒ **Nominal logic!**
Nominal logic
[A. M. Pitts and M. J. Gabbay, LICS 1999]

- Specific types for names (with infinitely many elements)
- Datatypes with abstractions

\textbf{nominal-datatype} \( \vec{\alpha} \, ty = \cdots \mid C_i \ll a_i^1 \gg \tau_i^1 \cdots \ll a_i^m \gg \tau_i^m \mid \cdots \)

where \( a_i^j \) are lists of atom types and \( \tau_i^j \) are types, possibly containing \( \vec{\alpha} \, ty \)

- Permutations: \( \pi \bullet t \) (bijective)
  where \( \pi = [(a_1, b_1), \ldots, (a_n, b_n)] \), \( a_i \) and \( b_i \) are names

- Support (\( \approx \) set of free variables): \( supp \, t \)
  - nominal datatypes have finite support
  - not all HOL types have finite support
    \( \implies \) use axiomatic type classes to characterize types that have finite support

- Freshness: \( a\#t \equiv a \notin supp \, t \)
Nominal datatypes

atom-decl name

nominal-datatype \texttt{term} = \texttt{Var name} | \texttt{Abs \texttt{name} term} | \texttt{App term term}

Permutations

\pi \cdot (\texttt{Var } n) = \texttt{Var } (\pi \cdot n)
\pi \cdot (\texttt{App } t \; u) = \texttt{App } (\pi \cdot t) \; (\pi \cdot u)
\pi \cdot (\texttt{Abs } n \; t) = \texttt{Abs } (\pi \cdot n) \; (\pi \cdot t)

[] \cdot n = n
((a, b) :: \pi) \cdot n = \texttt{swap a b } (\pi \cdot n)

\texttt{swap a b n} = \begin{cases} b & \text{if } n = a \\ a & \text{if } n = b \\ n & \text{otherwise} \end{cases}
Nominal datatypes – $\alpha$-equivalence

Standard datatypes

$\text{Abs } a \ t = \text{Abs } b \ u \iff (a = b \land t = u)$

Nominal datatypes

$\text{Abs } a \ t = \text{Abs } b \ u \iff (a = b \land t = u) \lor (a \neq b \land t = [(a, b)] \bullet u \land a^\# u)$

Example

$\text{Abs } a \ (\text{Var } a) = \text{Abs } b \ (\text{Var } b)$

because

- $a \neq b$
- $\text{Var } a = \text{Var } ([a, b] \bullet b) = [(a, b)] \bullet (\text{Var } b)$
- $a^\# \text{Var } b$
Nominal datatypes – induction

**Weak induction rule**

\[ \forall n. \ P \ (Var \ n) \]
\[ \forall t \ u. \ P \ t \implies P \ u \implies P \ (App \ t \ u) \]
\[ \forall n \ t. \ P \ t \implies P \ (Abs \ n \ t) \]
\[ \forall t. \ P \ t \]

**Strong induction rule**

\[ \forall n \ c. \ P \ c \ (Var \ n) \]
\[ \forall t \ u \ c. \ (\forall d. \ P \ d \ t) \implies (\forall d. \ P \ d \ u) \implies P \ c \ (App \ t \ u) \]
\[ \forall n \ t \ c. \ n\#c \implies (\forall d. \ P \ d \ t) \implies P \ c \ (Abs \ n \ t) \]
\[ \forall t \ c. \ P \ c \ t \]
Deriving the strong from the weak induction rule

Prove \( \forall t \, \pi \, c. \ P \ c \ (\pi \bullet t) \) using weak induction rule

Case \textit{Abs}: Show \( P \ c \ (Abs \ (\pi \bullet n) \ (\pi \bullet t)) \) from

(1) \( \forall \pi \, c. \ P \ c \ (\pi \bullet t) \)
(2) \( \forall n \ t \ c. \ n^\#c \Longrightarrow (\forall d. \ P \ d \ t) \Longrightarrow P \ c \ (Abs \ n \ t) \)

We can find \( n' \) such that

(3) \( n'^\#(c, \pi \bullet n, \pi \bullet t) \)

and hence

(4) \( Abs \ (\pi \bullet n) \ (\pi \bullet t) = Abs \ n' \ (((\pi \bullet n, \ n') \bullet (\pi \bullet t)) = Abs \ n' \ (((\pi \bullet n, \ n') \bullet (\pi \bullet t)) \)

From (1) we know that

(5) \( \forall c. \ P \ c \ (((\pi \bullet n, \ n') \bullet (\pi \bullet t)) \)

Together with (3) and (2), it follows that

(6) \( P \ c \ (Abs \ n' \ (((\pi \bullet n, \ n') \bullet (\pi \bullet t)) \)

Combining this with (4) proves the claim.
Using the strong induction rule

Substitution lemma: If $x \neq y$ and $x \notin FV(L)$, then

$$M[x \mapsto N][y \mapsto L] = M[y \mapsto L][x \mapsto N[y \mapsto L]].$$

Case 2: $M = \lambda z. M_1$.

By the variable convention we may assume that $z \neq x, y$ and $z$ is not free in $N, L$.

$$\forall z \ c. \ P \ c \ (Var \ z)$$

$$\forall M_1 \ M_2 \ c. \ (\forall d. \ P \ d \ M_1) \implies (\forall d. \ P \ d \ M_2) \implies P \ c \ (App \ M_1 \ M_2)$$

$$\forall z \ M_1 \ c. \ z \# c \implies (\forall d. \ P \ d \ M_1) \implies$$

$$P \ c \ (Abs \ z \ M_1)$$

$$\forall M \ c. \ P \ c \ M$$

$P := \lambda(x, y, N, L). \ \lambda M. \ I \ M \ x \ y \ N \ L$

$I \ M \ x \ y \ N \ L \equiv x \neq y \implies x \notin FV(L) \implies M[x \mapsto N][y \mapsto L] = M[y \mapsto L][x \mapsto N[y \mapsto L]]$
Using the strong induction rule

**Substitution lemma:** If $x \neq y$ and $x \notin \text{FV}(L)$, then

$$M[x \mapsto N][y \mapsto L] = M[y \mapsto L][x \mapsto N][y \mapsto L].$$

...

**Case 2:** $M = \lambda z.M_1$.

By the variable convention we may assume that $z \neq x, y$ and $z$ is not free in $N, L$.

\[
\forall z \ x \ y \ N \ L. \ \mathcal{I} (\text{Var } z) \ x \ y \ N \ L
\]

\[
\forall M_1 \ M_2 \ x \ y \ N \ L.
\quad (\forall x' \ y' \ N' \ L'. \ \mathcal{I} \ M_1 \ x' \ y' \ N' \ L') \implies (\forall x' \ y' \ N' \ L'. \ \mathcal{I} \ M_2 \ x' \ y' \ N' \ L') \implies
\quad \mathcal{I} (\text{App } M_1 \ M_2) \ x \ y \ N \ L
\]

\[
\forall z \ M_1 \ x \ y \ N \ L. \ z\#(x, y, N, L) \implies (\forall x' \ y' \ N' \ L'. \ \mathcal{I} \ M_1 \ x' \ y' \ N' \ L') \implies
\quad \mathcal{I} (\text{Abs } z \ M_1) \ x \ y \ N \ L
\]

\[
\forall M \ x \ y \ N \ L. \ \mathcal{I} \ M \ x \ y \ N \ L
\]

\[P := \lambda (x, y, N, L). \ \lambda M. \ \mathcal{I} \ M \ x \ y \ N \ L\]

\[\mathcal{I} \ M \ x \ y \ N \ L \equiv x \neq y \implies x \notin \text{FV}(L) \implies M[x \mapsto N][y \mapsto L] = M[y \mapsto L][x \mapsto N[y \mapsto L]]\]
Recursive functions on $\alpha$-equivalence classes

Is the following function well-defined?

\[ bvars (Var n) = \{\} \]
\[ bvars (App t u) = bvars t \cup bvars u \]
\[ bvars (Abs n t) = \{n\} \cup bvars t \]

Unproblematic for standard datatypes...

... but not for nominal datatypes:

\[ Abs a (Var a) = Abs b (Var b) \]

but

\[ bvars (Abs a (Var a)) = \{a\} \neq \{b\} = bvars (Abs b (Var b)) \]

\[\implies\] Result of function may not depend on choice of bound variable names!

Nominal Datatypes in Isabelle/HOL
Recursion combinator

Standard datatypes
\[
\begin{align*}
term_{rec} f_1 f_2 f_3 (\text{Var } n) &= f_1 n \\
term_{rec} f_1 f_2 f_3 (\text{App } t u) &= f_2 t u (term_{rec} f_1 f_2 f_3 t) (term_{rec} f_1 f_2 f_3 u) \\
term_{rec} f_1 f_2 f_3 (\text{Abs } n t) &= f_3 n t (term_{rec} f_1 f_2 f_3 t)
\end{align*}
\]

Nominal datatypes
\[
\begin{align*}
\text{freshness condition for binders} \\
n\#(f_1, f_2, f_3) \land (\forall n t r. n\#f_3 \Rightarrow n\#f_3 n t r) \Rightarrow \\
term_{rec} f_1 f_2 f_3 (\text{Abs } n t) &= f_3 n t (term_{rec} f_1 f_2 f_3 t)
\end{align*}
\]

Substitution function
\[
\begin{align*}
(\text{Var } x)[y \mapsto u] &= \text{(if } x = y \text{ then } u \text{ else } (\text{Var } x)) \\
(\text{App } t_1 t_2)[y \mapsto u] &= \text{App } (t_1[y \mapsto u]) (t_2[y \mapsto u]) \\
x\#(y, u) \Rightarrow (\text{Abs } x t)[y \mapsto u] &= \text{Abs } x (t[y \mapsto u])
\end{align*}
\]
Isabelle proof of substitution lemma

lemma substitution-lemma:
    assumes fresh: \( x \neq y \ x \# L \)
    shows \( M[x \mapsto N][y \mapsto L] = M[y \mapsto L][x \mapsto N[y \mapsto L]] \) using fresh

proof (nominal-induct \( M \) avoiding: \( x \ y \ N \ L \) rule: lam.induct)
  case (Abs \( z \) \( M_1 \))
  have \( (Abs \ z \ M_1)[x \mapsto N][y \mapsto L] = Abs \ z \ (M_1[x \mapsto N][y \mapsto L]) \)
    using \( \langle z \# x \rangle \langle z \# y \rangle \langle z \# N \rangle \langle z \# L \rangle \) by simp
  also from Abs have \( \ldots \) = \( Abs \ z \ (M_1[y \mapsto L][x \mapsto N[y \mapsto L]]) \)
    using \( \langle x \neq y \rangle \langle x \# L \rangle \) by simp
  also have \( \ldots \) = \( (Abs \ z \ (M_1[y \mapsto L]))[x \mapsto N[y \mapsto L]] \)
    using \( \langle z \# x \rangle \langle z \# N \rangle \langle z \# L \rangle \) by \( (\text{simp add: fresh-fact}) \)
  also have \( \ldots \) = \( (Abs \ z \ M_1)[y \mapsto L][x \mapsto N[y \mapsto L]] \)
    using \( \langle z \# y \rangle \langle z \# L \rangle \) by simp
  finally show \( (Abs \ z \ M_1)[x \mapsto N][y \mapsto L] = (Abs \ z \ M_1)[y \mapsto L][x \mapsto N[y \mapsto L]] \).

next
  \ldots

qed
Inductive predicates involving nominal datatypes

\[
\begin{align*}
\text{valid}(\Gamma) \quad (x:T) \in \Gamma &\quad \frac{}{\Gamma \vdash \text{Var } x : T} \quad \text{Var } T \\
\Gamma \vdash M : T_1 \rightarrow T_2 \quad \Gamma \vdash N : T_1 &\quad \frac{}{\Gamma \vdash \text{App } M \ N : T_2} \quad \text{App } T \\
\frac{x \notin \text{dom}(\Gamma)}{\Gamma \vdash \text{Abs } x \ M : T_1 \rightarrow T_2} \quad \text{Abs } T
\end{align*}
\]

An informal proof by rule induction

**Weakening lemma:** If \( \Gamma \vdash M : T \) is derivable, and \( \Gamma \subseteq \Gamma' \) with \( \Gamma' \) valid, then \( \Gamma' \vdash M : T \) is also derivable.

**Proof:** By rule induction over \( \Gamma \vdash M : T \) showing that \( \Gamma' \vdash M : T \) holds for all \( \Gamma' \) with \( \Gamma \subseteq \Gamma' \) and \( \Gamma' \) being valid.

\[
\text{Case } \text{Abs}: \Gamma \vdash M : T \text{ is } \Gamma \vdash \text{Abs } x \ M_1 : T_1 \rightarrow T_2. \text{ Using the variable convention we assume that } x \# \Gamma'. \text{ Then we know that } ((x:T_1):::\Gamma') \text{ is valid and hence that} \\
((x:T_1):::\Gamma') \vdash M_1 : T_2 \text{ holds. Thus, we can conclude that } \Gamma' \vdash \text{Abs } x \ M_1 : T_1 \rightarrow T_2 \text{ holds using rule Abs } T.
\]

Nominal Datatypes in Isabelle/HOL
Rule induction principle

Standard rule induction

\[ \Gamma \vdash M : T \]

\[ \ldots \]

\[ \forall x \Gamma T_1 M T_2. \ x \notin \text{dom}(\Gamma) \implies P ((x : T_1)::\Gamma) M T_2 \implies \]

\[ (x : T_1)::\Gamma \vdash M : T_2 \implies P \Gamma (\text{Abs } x M) (T_1 \rightarrow T_2) \]

\[ \frac{P \Gamma M T}{P \Gamma M T} \]

Strengthened rule induction

\[ \Gamma \vdash M : T \]

\[ \ldots \]

\[ \forall x \Gamma T_1 M T_2 \ c. \ x \# c \implies x \notin \text{dom}(\Gamma) \implies (\forall d. \ P d ((x : T_1)::\Gamma) M T_2) \implies \]

\[ (x : T_1)::\Gamma \vdash M : T_2 \implies P \ c \ \Gamma (\text{Abs } x M) (T_1 \rightarrow T_2) \]

\[ \frac{P \ c \ \Gamma M T}{P \ c \ \Gamma M T} \]
Conclusion
Conclusion

• **Bad news:** proofs about calculi with variable binding are inherently complicated
• **Good news:** some of the complexity can be hidden inside nominal datatype package
• Implementation is contained in Isabelle Development Snapshot

**Applications**
- Pi-calculus [Bengtson, Parrow]
- Metatheory of $F_<$: [Urban, Weirich, Zdancewic]
- Strong normalization of simply-typed lambda-calculus [Urban]
- Chapter about logical relations from book by B. Pierce [Narboux, Urban]
- Other applications are being developed . . . (see the mailing list!)

**Further work**
- Strengthened inversion principles
- More general binding constructs
  - let $p = t$ in $u$
- Generation of code from specifications involving nominal datatypes
- Support for more general recursion schemes


See the web site: isabelle.in.tum.de/nominal

Thanks for your attention!