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

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

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

□

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."

Stephen C. Kleene in Journal of Symbolic Logic 21(1):11-18, 1962

"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 α-conversion) required far more time than any other variety of theorem."

Myra VanInwegen in her PhD-thesis, 1996

$\Longrightarrow$ **Better tool support necessary**

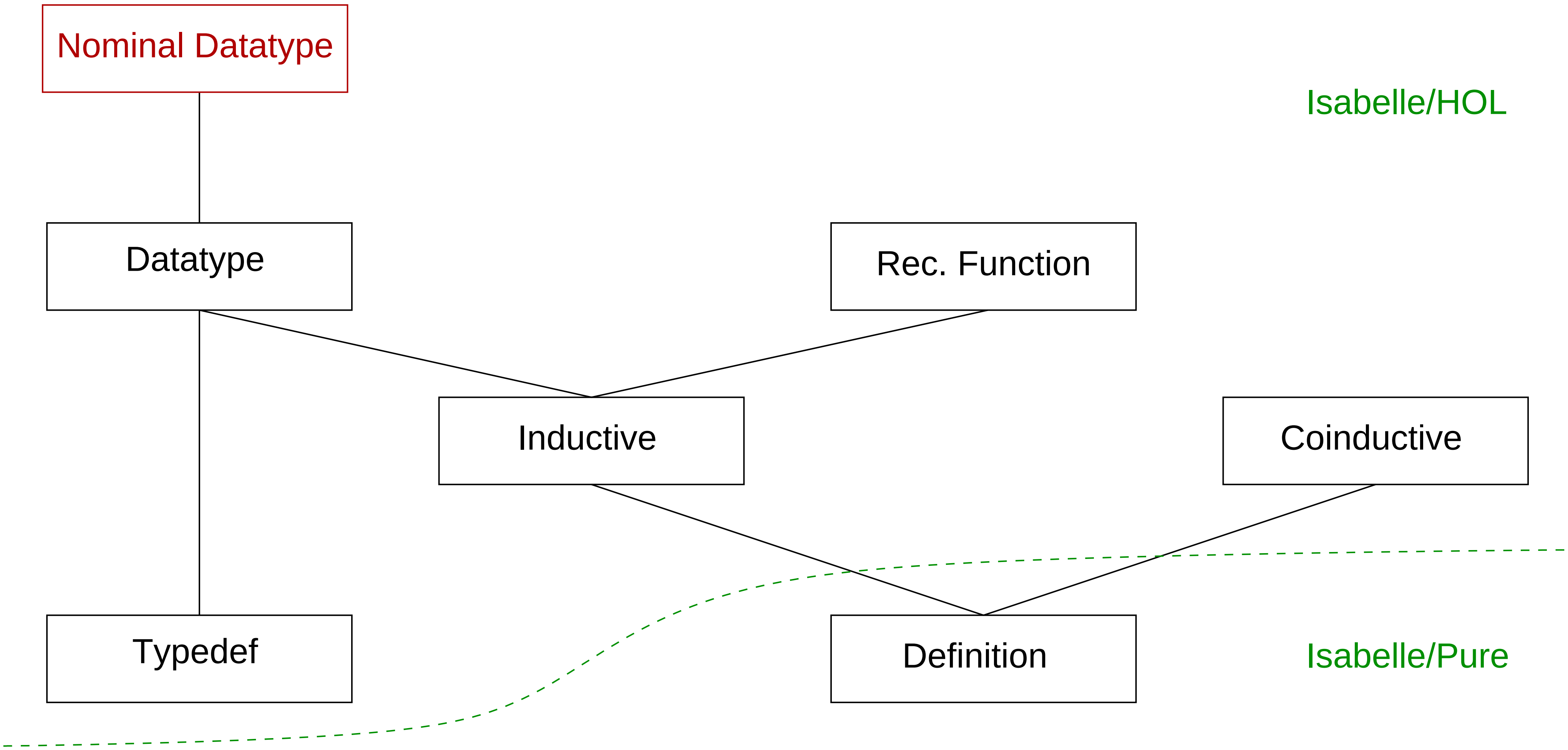
Our tool: Isabelle

- 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



Existing approaches for reasoning with bound variables

- "Name-carrying" syntax
  - \+ readable
  - – α-equivalence must be formalized explicitly
  - – substitution function requires variable renaming
- De Bruijn indices
  - \+ α-equivalence coincides with syntactic equality
  - \+ simple induction / recursion principles
  - – substitution function requires index calculations
  - – unreadable
- "Locally nameless" approach
  - \+ α-equivalence coincides with syntactic equality
  - \+ substitution function does not require index calculations
  - – well-formedness must be formalized explictity
- Higher order abstract syntax
  - \+ abstraction and substitution "for free"
  - – exotic terms

Our approach

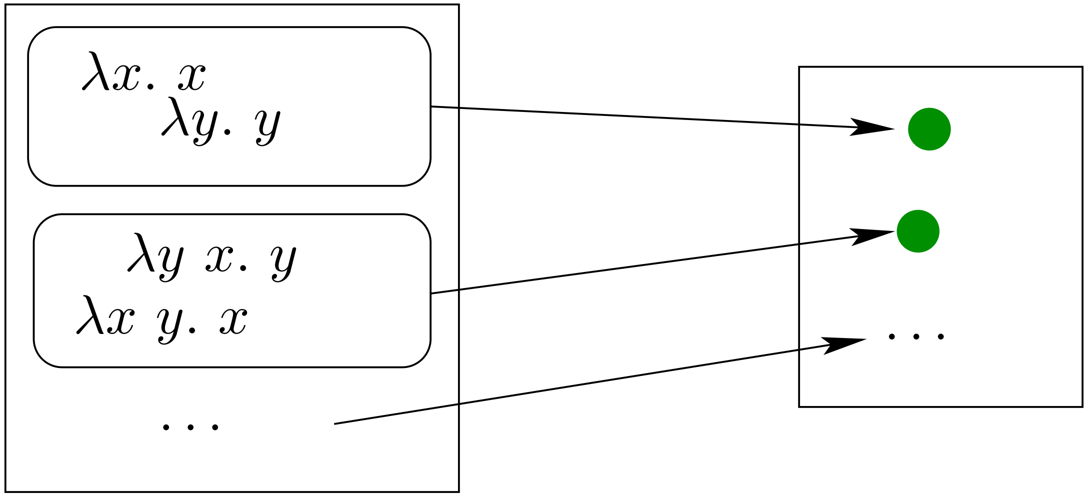
A more abstract approach

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

Possible solution:

Introduce new type, whose elements correspond to...

- ... the α-equivalence classes of "name-carrying"-terms, or
- ... the well-formed "locally nameless" terms.



Question: How does abstract "interface" for this type look like? $\Longrightarrow$ **Nominal logic!**

Nominal logic
[A. M. Pitts and M. J. Gabbay, LICS 1999]

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

  nominal-datatype $\vec{\alpha}\ ty\ =\ \cdots \mid C_i \ll\overrightarrow{a_i^1}\gg\tau_i^1\ \cdots\ \ll\overrightarrow{a_i^{m_i}}\gg\tau_i^{m_i} \mid \cdots$

  where $\overrightarrow{a_i^j}$ are lists of atom types and τ_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
    $\Longrightarrow$ use axiomatic type classes to characterize types that have finite support
- Freshness: $a \sharp t \equiv a \notin supp\ t$

Nominal datatypes

atom-decl $name$

nominal-datatype $term \;=\; Var\ name \mid Abs\ \ll name \gg term \mid App\ term\ term$

Permutations

$$
\begin{array}{lcl}
\pi \bullet (Var\ n) & = & Var\ (\pi \bullet n) \\
\pi \bullet (App\ t\ u) & = & App\ (\pi \bullet t)\ (\pi \bullet u) \\
\pi \bullet (Abs\ n\ t) & = & Abs\ (\pi \bullet n)\ (\pi \bullet t)
\end{array}
$$

$$
\begin{array}{lcl}
[] \bullet n & = & n \\
((a,b) :: \pi) \bullet n & = & swap\ a\ b\ (\pi \bullet n)
\end{array}
$$

$$
swap\ a\ b\ n = \begin{cases} b & \text{if } n = a \\ a & \text{if } n = b \\ n & \text{otherwise} \end{cases}
$$

Nominal datatypes – α-equivalence

Standard datatypes

$$Abs\ a\ t = Abs\ b\ u \Longleftrightarrow (a = b \wedge t = u)$$

Nominal datatypes

$$Abs\ a\ t = Abs\ b\ u \Longleftrightarrow (a = b \wedge t = u) \vee (a \neq b \wedge t = [(a,b)] \bullet u \wedge a \sharp u)$$

Example

$$Abs\ a\ (Var\ a) = Abs\ b\ (Var\ b)$$

because

- $a \neq b$
- $Var\ a = Var\ ([(a,b)] \bullet b) = [(a,b)] \bullet (Var\ b)$
- $a \sharp Var\ b$

Nominal datatypes – induction

Weak induction rule

$$\frac{\begin{array}{l}\forall n.\ P\ (Var\ n)\\ \forall t\ u.\ P\ t \Longrightarrow P\ u \Longrightarrow P\ (App\ t\ u)\\ \forall n\ t.\ P\ t \Longrightarrow P\ (Abs\ n\ t)\end{array}}{\forall t.\ P\ t}$$

Strong induction rule

$$\frac{\begin{array}{l}\forall n\ c.\ P\ c\ (Var\ n)\\ \forall t\ u\ c.\ (\forall d.\ P\ d\ t) \Longrightarrow (\forall d.\ P\ d\ u) \Longrightarrow P\ c\ (App\ t\ u)\\ \forall n\ t\ c.\ n \sharp c \Longrightarrow (\forall d.\ P\ d\ t) \Longrightarrow P\ c\ (Abs\ n\ t)\end{array}}{\forall t\ c.\ P\ c\ t}$$

Deriving the strong from the weak induction rule

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

Case Abs**:** Show $P\ c\ (Abs\ (\pi \bullet n)\ (\pi \bullet t))$ from

(1) $\forall \pi\ c.\ P\ c\ (\pi \bullet t)$

(2) $\forall n\ t\ c.\ n \sharp c \Longrightarrow (\forall d.\ P\ d\ t) \Longrightarrow P\ c\ (Abs\ n\ t)$

We can find n' such that

(3) $n' \sharp (c,\ \pi \bullet n,\ \pi \bullet t)$

and hence

(4) $Abs\ (\pi \bullet n)\ (\pi \bullet t) = Abs\ n'\ ([(\pi \bullet n,\ n')] \bullet (\pi \bullet t)) = Abs\ n'\ (([(\pi \bullet n,\ n')]\ @\ \pi) \bullet t)$

From **(1)** we know that

(5) $\forall c.\ P\ c\ (([(\pi \bullet n,\ n')]\ @\ \pi) \bullet t)$

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

(6) $P\ c\ (Abs\ n'\ (([(\pi \bullet n,\ n')]\ @\ \pi) \bullet t))$

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.

$$\frac{\begin{array}{l} \forall z\ c.\ P\ c\ (Var\ z) \\[1em] \forall M_1\ M_2\ c. \\ \qquad (\forall d.\ P\ d\ M_1) \Longrightarrow (\forall d.\ P\ d\ M_2) \Longrightarrow \\ \qquad P\ c\ (App\ M_1\ M_2) \\[1em] \forall z\ M_1\ c.\ z \sharp c \Longrightarrow (\forall d.\ P\ d\ M_1) \Longrightarrow \\ \qquad P\ c\ (Abs\ z\ M_1) \end{array}}{\forall M\ c.\ P\ c\ M}$$

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

$\mathcal{I}\ M\ x\ y\ N\ L \equiv x \neq y \Longrightarrow x \notin FV(L) \Longrightarrow M[x \mapsto N][y \mapsto L] = M[y \mapsto L][x \mapsto N[y \mapsto L]]$

Using the strong induction rule

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

$$\frac{\begin{array}{l} \forall z\ x\ y\ N\ L.\ \mathcal{I}\ (Var\ z)\ x\ y\ N\ L \\[1em] \forall M_1\ M_2\ x\ y\ N\ L. \\ \quad (\forall x'\ y'\ N'\ L'.\ \mathcal{I}\ M_1\ x'\ y'\ N'\ L') \Longrightarrow (\forall x'\ y'\ N'\ L'.\ \mathcal{I}\ M_2\ x'\ y'\ N'\ L') \Longrightarrow \\ \quad \mathcal{I}\ (App\ M_1\ M_2)\ x\ y\ N\ L \\[1em] \forall z\ M_1\ x\ y\ N\ L.\ z\sharp(x,\ y,\ N,\ L) \Longrightarrow (\forall x'\ y'\ N'\ L'.\ \mathcal{I}\ M_1\ x'\ y'\ N'\ L') \Longrightarrow \\ \quad \mathcal{I}\ (Abs\ z\ M_1)\ x\ y\ N\ L \end{array}}{\forall M\ x\ y\ N\ L.\ \mathcal{I}\ M\ x\ y\ N\ L}$$

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

$\mathcal{I}\ M\ x\ y\ N\ L \equiv x \neq y \Longrightarrow x \notin FV(L) \Longrightarrow M[x \mapsto N][y \mapsto L] = M[y \mapsto L][x \mapsto N[y \mapsto L]]$

Recursive functions on α-equivalence classes

Is the following function well-defined?

$$\begin{array}{lcl} bvars\ (Var\ n) & = & \{\} \\ bvars\ (App\ t\ u) & = & bvars\ t \cup bvars\ u \\ bvars\ (Abs\ n\ t) & = & \{n\} \cup bvars\ t \end{array}$$

Unproblematic for standard datatypes...
... but not for nominal datatypes:

$$Abs\ a\ (Var\ a) = Abs\ b\ (Var\ b)$$

but

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

$\Longrightarrow$ **Result of function may not depend on choice of bound variable names!**

Recursion combinator

Standard datatypes

$$\begin{array}{lcl} term\_rec\ f_1\ f_2\ f_3\ (Var\ n) & = & f_1\ n \\ term\_rec\ f_1\ f_2\ f_3\ (App\ t\ u) & = & f_2\ t\ u\ (term\_rec\ f_1\ f_2\ f_3\ t)\ (term\_rec\ f_1\ f_2\ f_3\ u) \\ term\_rec\ f_1\ f_2\ f_3\ (Abs\ n\ t) & = & f_3\ n\ t\ (term\_rec\ f_1\ f_2\ f_3\ t) \end{array}$$

Nominal datatypes

$$n\#(f_1, f_2, f_3) \wedge \overbrace{(\forall n\ t\ r.\ n\#f_3 \Longrightarrow n\#f_3\ n\ t\ r)}^{\text{freshness condition for binders}} \Longrightarrow$$
$$term\_rec\ f_1\ f_2\ f_3\ (Abs\ n\ t) = f_3\ n\ t\ (term\_rec\ f_1\ f_2\ f_3\ t)$$

Substitution function

$$\begin{array}{rcl} (Var\ x)[y \mapsto u] & = & (\text{if } x = y \text{ then } u \text{ else } (Var\ x)) \\ (App\ t_1\ t_2)[y \mapsto u] & = & App\ (t_1[y \mapsto u])\ (t_2[y \mapsto u]) \\ x\sharp(y, u) \Longrightarrow (Abs\ x\ t)[y \mapsto u] & = & Abs\ x\ (t[y \mapsto u]) \end{array}$$

Isabelle proof of substitution lemma

lemma *substitution-lemma*:
  assumes *fresh*: $x \neq y$ $x \sharp L$
  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 $\langle z \sharp x \rangle$ $\langle z \sharp y \rangle$ $\langle z \sharp N \rangle$ $\langle z \sharp L \rangle$ **by** *simp*
  also from *Abs* **have** $\ldots = Abs\ z\ (M_1[y \mapsto L][x \mapsto N[y \mapsto L]])$
    using $\langle x \neq y \rangle$ $\langle x \sharp L \rangle$ **by** *simp*
  also have $\ldots = (Abs\ z\ (M_1[y \mapsto L]))[x \mapsto N[y \mapsto L]]$
    using $\langle z \sharp x \rangle$ $\langle z \sharp N \rangle$ $\langle z \sharp L \rangle$ **by** (*simp add*: *fresh-fact*)
  also have $\ldots = (Abs\ z\ M_1)[y \mapsto L][x \mapsto N[y \mapsto L]]$
    using $\langle z \sharp y \rangle$ $\langle z \sharp L \rangle$ **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

$$\frac{valid(\Gamma) \quad (x\!:\!T) \in \Gamma}{\Gamma \vdash Var\ x : T}\ VarT \qquad \frac{\Gamma \vdash M : T_1 \to T_2 \quad \Gamma \vdash N : T_1}{\Gamma \vdash App\ M\ N : T_2}\ AppT$$

$$\frac{x \notin dom(\Gamma) \quad (x\!:\!T_1)\!::\!\Gamma \vdash M : T_2}{\Gamma \vdash Abs\ x\ M : T_1 \to T_2}\ AbsT$$

An informal proof by rule induction

Weakening lemma: If $\Gamma \vdash M : T$ is derivable, and $\Gamma \subseteq \Gamma'$ with Γ' 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 Γ' with $\Gamma \subseteq \Gamma'$ and Γ' being valid.

...

Case Abs: $\Gamma \vdash M : T$ is $\Gamma \vdash Abs\ x\ M_1 : T_1 \to T_2$. Using the variable convention we assume that $x \sharp \Gamma'$. Then we know that $((x\!:\!T_1)\!::\!\Gamma')$ is valid and hence that $((x\!:\!T_1)\!::\!\Gamma') \vdash M_1 : T_2$ holds. Thus, we can conclude that $\Gamma' \vdash Abs\ x\ M_1 : T_1 \to T_2$ holds using rule $AbsT$.

Rule induction principle

Standard rule induction

$$\frac{\begin{array}{l}\Gamma \vdash M : T \\ \ldots \\ \forall x\ \Gamma\ T_1\ M\ T_2.\ x \notin dom(\Gamma) \Longrightarrow P\ ((x{:}T_1){::}\Gamma)\ M\ T_2 \Longrightarrow \\ \qquad (x{:}T_1){::}\Gamma \vdash M : T_2 \Longrightarrow P\ \Gamma\ (Abs\ x\ M)\ (T_1 \rightarrow T_2)\end{array}}{P\ \Gamma\ M\ T}$$

Strengthened rule induction

$$\frac{\begin{array}{l}\Gamma \vdash M : T \\ \ldots \\ \forall x\ \Gamma\ T_1\ M\ T_2\ c.\ x \sharp c \Longrightarrow x \notin dom(\Gamma) \Longrightarrow (\forall d.\ P\ d\ ((x{:}T_1){::}\Gamma)\ M\ T_2) \Longrightarrow \\ \qquad (x{:}T_1){::}\Gamma \vdash M : T_2 \Longrightarrow P\ c\ \Gamma\ (Abs\ x\ M)\ (T_1 \rightarrow T_2)\end{array}}{P\ c\ \Gamma\ M\ T}$$

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

Literature

C. Urban and C. Tasson *Nominal techniques in Isabelle/HOL.* In Proceedings of the 20th International Conference on Automated Deduction (CADE), volume 3632 of LNCS, Springer-Verlag 2005.

C. Urban and S. Berghofer *A Recursion Combinator for Nominal Datatypes Implemented in Isabelle/HOL.* In U. Furbach and N. Shankar, editors, Third International Joint Conference on Automated Reasoning (IJCAR 2006), LNCS 4130, Springer-Verlag 2006.

C. Urban, S. Berghofer, and M. Norrish *Barendregt's Variable Convention in Rule Inductions.* To appear in proceedings of CADE 2007.

J. Bengtson and J. Parrow *Formalising the pi-calculus using nominal logic.* In Proceedings of FOSSACS 2007, LNCS, Springer-Verlag 2007.

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

Thanks for your attention!