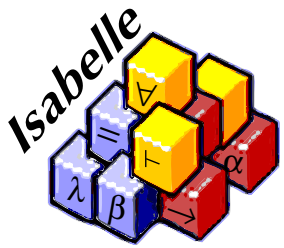
Nominal Inversion Principles

Stefan Berghofer and Christian Urban

Institut für Informatik
TU München



Roadmap

1. Nominal datatypes
2. Simply-typed lambda calculus
3. Inversion
4. Type preservation
5. Strong inversion rules
6. Conclusion

Nominal Datatypes

Datatypes with binders

nominal-datatype $lam = Var\ name \mid App\ lam\ lam \mid Lam\ \ll name \gg lam$

Equality modulo α

$$(Lam\ a\ t = Lam\ a'\ t') = (a = a' \wedge t = t' \vee a \neq a' \wedge t = [(a,\ a')] \cdot t' \wedge a \#\ t')$$

Variable renaming via permutations: $\pi \cdot t$, where $\pi = [(a_1,\ b_1),\ \ldots,\ (a_n,\ b_n)]$

Freshness: $a \#\ x \equiv a \notin supp\ x$

Strong induction principles with built-in variable convention

> names of bound variables are chosen in such a way that they are distinct and do not clash with other names used elsewhere in the proof

Permutations

$$\pi \cdot Var\ x = Var\ (\pi \cdot x)$$
$$\pi \cdot App\ t\ u = App\ (\pi \cdot t)\ (\pi \cdot u)$$
$$\pi \cdot Lam\ x\ t = Lam\ (\pi \cdot x)\ (\pi \cdot t)$$

Swapping

$$((a,\ b) :: \pi) \cdot x = swap\ (a,\ b)\ (\pi \cdot x)$$
$$[] \cdot x = x$$

$$swap\ (a,\ b)\ c = (if\ a = c\ then\ b\ else\ if\ b = c\ then\ a\ else\ c)$$

Equivariance

A function f is called equivariant, if

$$\pi \cdot f\ x_1\ \ldots\ x_n = f\ (\pi \cdot x_1)\ \ldots\ (\pi \cdot x_n)$$

For example, $\pi \cdot t[x{:=}u] = (\pi \cdot t)[(\pi \cdot x){:=}(\pi \cdot u)]$

Simply-typed lambda calculus

$$\frac{valid\ \Gamma \qquad (x,\ T) \in set\ \Gamma}{\Gamma \vdash Var\ x : T}$$

$$\frac{\Gamma \vdash t_1 : T_1 \to T_2 \qquad \Gamma \vdash t_2 : T_1}{\Gamma \vdash App\ t_1\ t_2 : T_2} \qquad \frac{(x,\ T_1) :: \Gamma \vdash t : T_2}{\Gamma \vdash Lam\ x\ t : T_1 \to T_2}$$

Beta reduction

$$\frac{}{App\ (Lam\ x\ s_1)\ s_2 \longrightarrow_\beta s_1[x{:=}s_2]}$$

$$\frac{s_1 \longrightarrow_\beta s_2}{App\ s_1\ t \longrightarrow_\beta App\ s_2\ t} \qquad \frac{s_1 \longrightarrow_\beta s_2}{App\ t\ s_1 \longrightarrow_\beta App\ t\ s_2} \qquad \frac{s_1 \longrightarrow_\beta s_2}{Lam\ x\ s_1 \longrightarrow_\beta Lam\ x\ s_2}$$

Inversion

Studied by

- Cristina Cornes and Delphine Terrasse (for Coq)
- Conor McBride (for LEGO)
- Larry Paulson (for Isabelle)

Idea

- Any inhabitant of an inductively defined predicate must have been derived by at least one of its introduction rules
- Inverting a hypothesis yields the premises (plus extra equational constraints) that have been used to derive the hypothesis via a particular rule
- Many proofs require an induction followed by an inversion

Problem

Solving equational constraints is problematic in the presence of non-injective constructors such as Lam.

Type Preservation

$If\ \Gamma \vdash u : U\ \ and\ \ u \longrightarrow_\beta u'\ \ then\ \Gamma \vdash u' : U.$

Proof: By induction on $\Gamma \vdash u : U$. We get three subgoals:

$$\begin{array}{r} (i)\ \ Var\ x \longrightarrow_\beta u' \Longrightarrow \ldots \Longrightarrow \Gamma \vdash u' : T \\ (ii)\ \ App\ t_1\ t_2 \longrightarrow_\beta u' \Longrightarrow \ldots \Longrightarrow \Gamma \vdash u' : T_2 \\ (iii)\ \ Lam\ x\ t \longrightarrow_\beta u' \Longrightarrow \ldots \Longrightarrow \Gamma \vdash u' : T_1 \to T_2 \end{array}$$

Proof of *(ii)* by Inversion

$$\frac{s_1 \longrightarrow_\beta s_2}{App\ s_1\ t \longrightarrow_\beta App\ s_2\ t} \qquad \longmapsto \qquad \begin{array}{l} s_1 = t_1 \\ t = t_2 \\ u' = App\ s_2\ t_2 \\ t_1 \longrightarrow_\beta s_2 \end{array}$$

$$App\ t_1\ t_2 \longrightarrow_\beta u' \quad (=^?)$$

By induction hypothesis: $\Gamma \vdash s_2 : T_1 \to T_2$ and $\Gamma \vdash t_2 : T_1$
Therefore, $\Gamma \vdash u' : T_2$.

Type Preservation

$$\frac{}{App\ (Lam\ x\ s_1)\ s_2 \longrightarrow_\beta s_1[x{:=}s_2]} \qquad \longmapsto \qquad \begin{array}{l} t_1 = Lam\ x\ s_1 \\ s_2 = t_2 \\ u' = s_1[x{:=}t_2] \end{array}$$

$$=^?$$

$$App\ t_1\ t_2 \longrightarrow_\beta u'$$

By induction hypothesis: $\Gamma \vdash Lam\ x\ s_1 : T_1 \to T_2$ and $\Gamma \vdash t_2 : T_1$

$$\frac{(x,\ T_1) :: \Gamma \vdash t : T_2}{\Gamma \vdash Lam\ x\ t : T_1 \to T_2} \qquad \longmapsto \qquad \begin{array}{l} t = s_1 \\ (x,\ T_1) :: \Gamma \vdash s_1 : T_2 \end{array}$$

$$=^?$$

$$\Gamma \vdash Lam\ x\ s_1 : T_1 \to T_2$$

By the substitution lemma

If $(x,\ T_1) :: \Gamma \vdash s_1 : T_2$ *and* $\Gamma \vdash t_2 : T_1$ *then* $\Gamma \vdash s_1[x{:=}t_2] : T_2$.

we get $\Gamma \vdash u' : T_2$.

Standard Inversion Rules

$$\frac{\begin{array}{l}\bigwedge x\ s_2\ s_1.\ u_1 = App\ (Lam\ x\ s_1)\ s_2 \Longrightarrow u_2 = s_1[x{:=}s_2] \Longrightarrow P \\ \bigwedge s_1\ s_2\ t.\ u_1 = App\ s_1\ t \Longrightarrow u_2 = App\ s_2\ t \Longrightarrow s_1 \longrightarrow_\beta s_2 \Longrightarrow P \\ \bigwedge s_1\ s_2\ t.\ u_1 = App\ t\ s_1 \Longrightarrow u_2 = App\ t\ s_2 \Longrightarrow s_1 \longrightarrow_\beta s_2 \Longrightarrow P \\ \bigwedge s_1\ s_2\ x.\ u_1 = Lam\ x\ s_1 \Longrightarrow u_2 = Lam\ x\ s_2 \Longrightarrow s_1 \longrightarrow_\beta s_2 \Longrightarrow P\end{array}}{u_1 \longrightarrow_\beta u_2 \Longrightarrow P}$$

$$\frac{\begin{array}{l}\bigwedge \Gamma\ x\ T.\ \Delta = \Gamma \Longrightarrow u = Var\ x \Longrightarrow U = T \Longrightarrow \\ \quad valid\ \Gamma \Longrightarrow (x,\ T) \in set\ \Gamma \Longrightarrow P \\ \bigwedge t_1\ T_1\ T_2\ t_2.\ \Delta = \Gamma \Longrightarrow u = App\ t_1\ t_2 \Longrightarrow U = T_2 \Longrightarrow \\ \quad \Gamma \vdash t_1 : T_1 \to T_2 \Longrightarrow \Gamma \vdash t_2 : T_1 \Longrightarrow P \\ \bigwedge x\ T_1\ \Gamma\ t\ T_2.\ \Delta = \Gamma \Longrightarrow u = Lam\ x\ t \Longrightarrow U = T_1 \to T_2 \Longrightarrow \\ \quad (x,\ T_1) :: \Gamma \vdash t : T_2 \Longrightarrow P\end{array}}{\Delta \vdash u : U \Longrightarrow P}$$

Type Preservation – more formally

Case $(i)\ Var\ x \longrightarrow_\beta u' \Longrightarrow \ldots \Longrightarrow \Gamma \vdash u' : T$

$$\bigwedge x'\ s_2\ s_1.\ Var\ x = App\ (Lam\ x'\ s_1)\ s_2 \Longrightarrow u' = s_1[x' := s_2] \Longrightarrow \ldots$$
$$\bigwedge s_1\ s_2\ t.\ Var\ x = App\ s_1\ t \Longrightarrow u' = App\ s_2\ t \Longrightarrow s_1 \longrightarrow_\beta s_2 \Longrightarrow \ldots$$
$$\bigwedge s_1\ s_2\ t.\ Var\ x = App\ t\ s_1 \Longrightarrow u' = App\ t\ s_2 \Longrightarrow s_1 \longrightarrow_\beta s_2 \Longrightarrow \ldots$$
$$\bigwedge s_1\ s_2\ x'.\ Var\ x = Lam\ x'\ s_1 \Longrightarrow u' = Lam\ x'\ s_2 \Longrightarrow s_1 \longrightarrow_\beta s_2 \Longrightarrow \ldots$$

Type Preservation – more formally

Case $(ii)\ App\ t_1\ t_2 \longrightarrow_\beta u' \Longrightarrow \ldots \Longrightarrow \Gamma \vdash u' : T_2$

$$\bigwedge x\ s_2\ s_1.\ App\ t_1\ t_2 = App\ (Lam\ x\ s_1)\ s_2 \Longrightarrow u' = s_1[x::=s_2] \Longrightarrow \ldots$$
$$\bigwedge s_1\ s_2\ t.\ App\ t_1\ t_2 = App\ s_1\ t \Longrightarrow u' = App\ s_2\ t \Longrightarrow s_1 \longrightarrow_\beta s_2 \Longrightarrow \ldots$$
$$\bigwedge s_1\ s_2\ t.\ App\ t_1\ t_2 = App\ t\ s_1 \Longrightarrow u' = App\ t\ s_2 \Longrightarrow s_1 \longrightarrow_\beta s_2 \Longrightarrow \ldots$$
$$\bigwedge s_1\ s_2\ x.\ App\ t_1\ t_2 = Lam\ x\ s_1 \Longrightarrow u' = Lam\ x\ s_2 \Longrightarrow s_1 \longrightarrow_\beta s_2 \Longrightarrow \ldots$$

Case $(iii)\ Lam\ x\ t \longrightarrow_\beta u' \Longrightarrow \ldots \Longrightarrow \Gamma \vdash u' : T_1 \to T_2$

$$\bigwedge x'\ s_2\ s_1.\ Lam\ x\ t = App\ (Lam\ x'\ s_1)\ s_2 \Longrightarrow u' = s_1[x'::=s_2] \Longrightarrow \ldots$$
$$\bigwedge s_1\ s_2\ t.\ Lam\ x\ t = App\ s_1\ t \Longrightarrow u' = App\ s_2\ t \Longrightarrow s_1 \longrightarrow_\beta s_2 \Longrightarrow \ldots$$
$$\bigwedge s_1\ s_2\ t.\ Lam\ x\ t = App\ t\ s_1 \Longrightarrow u' = App\ t\ s_2 \Longrightarrow s_1 \longrightarrow_\beta s_2 \Longrightarrow \ldots$$
$$\bigwedge s_1\ s_2\ x'.\ Lam\ x\ t = Lam\ x'\ s_1 \Longrightarrow u' = Lam\ x'\ s_2 \Longrightarrow s_1 \longrightarrow_\beta s_2 \Longrightarrow \ldots$$

If $Lam\ x\ t = Lam\ x'\ s_1$ then
(i) $x = x'$ and $t = s_1$ or
(ii) $x \neq x'$ and $t = [(x, x')] \cdot s_1$ and $x \# s_1$

Naive inversion is dangerous

$$\frac{}{Var\ x \hookrightarrow Var\ x} \qquad \frac{}{App\ t_1\ t_2 \hookrightarrow App\ t_1\ t_2} \qquad \frac{t \hookrightarrow t'}{Lam\ x\ t \hookrightarrow t'}$$

Proof (faulty)

Choose x and y with $x \neq y$. By the above rules, we have $Lam\ x\ (Var\ x) \hookrightarrow Var\ x$.
Due to α-conversion, $Lam\ x\ (Var\ x) = Lam\ y\ (Var\ y)$ and therefore
$Lam\ y\ (Var\ y) \hookrightarrow Var\ x$. By inversion, $Var\ y \hookrightarrow Var\ x$.
By another application of inversion, we get $x = y$, which contradicts $x \neq y$.

Variable convention compatibility condition needed to avoid faulty reasoning

Strong Inversion Rules

Introduction Rules

$$\frac{B[as_1;\ xs_1]}{R\ ts_1[as_1;\ xs_1]} \qquad \cdots \qquad \frac{B[as_n;\ xs_n]}{R\ ts_n[as_n;\ xs_n]}$$

Weak Inversion Rule

$$\frac{\begin{array}{c}\bigwedge as_1\ xs_1.\ zs = ts_1[as_1;\ xs_1] \Longrightarrow B[as_1;\ xs_1] \Longrightarrow P \\ \vdots \\ \bigwedge as_n\ xs_n.\ zs = ts_n[as_n;\ xs_n] \Longrightarrow B[as_n;\ xs_n] \Longrightarrow P\end{array}}{R\ zs \Longrightarrow P}$$

Strong Inversion Rule

$$\frac{\begin{array}{c}\bigwedge xs_1.\ (bs_1\ \#\ zs \Longrightarrow distinct(bs_1) \Longrightarrow zs = ts_1[bs_1;\ xs_1] \wedge B[bs_1;\ xs_1]) \Longrightarrow P \\ \vdots \\ \bigwedge xs_n.\ (bs_n\ \#\ zs \Longrightarrow distinct(bs_n) \Longrightarrow zs = ts_n[bs_n;\ xs_n] \wedge B[bs_n;\ xs_n]) \Longrightarrow P\end{array}}{R\ zs \Longrightarrow P}$$

provided that $B[bs_i;\ xs_i] \Longrightarrow bs_i\ \#\ ts_i[bs_i;\ xs_i] \wedge distinct(bs_i)$ for all i

Proof of the Strong Inversion Rule

We can use the premises of the strong inversion rule

$$\bigwedge xs_i.\ (bs_i \# zs \Longrightarrow distinct(bs_i) \Longrightarrow zs = ts_i[bs_i;\ xs_i] \wedge B[bs_i;\ xs_i]) \Longrightarrow P \qquad (1)$$

Using the weak inversion rule, we have to show P using

$$zs = ts_i[as_i;\ xs_i] \qquad \text{and} \qquad B[as_i;\ xs_i] \qquad (2)$$

By vc-compatibility, we have

$$(a)\ as_i \# zs \qquad \text{and} \qquad (b)\ distinct(as_i) \qquad (3)$$

Choose names cs_i such that

$$(a)\ cs_i \# zs \quad (b)\ cs_i \neq as_i \quad (c)\ cs_i \neq bs_i \quad (d)\ distinct(cs_i) \qquad (4)$$

Let

$$\pi = [(c_n,\ b_n),\ \ldots,\ (c_1,\ b_1),\ (a_n,\ c_n),\ \ldots,\ (a_1,\ c_1)]$$

Proof of the Strong Inversion Rule – continued

Using the instance of (1)

$$(bs_i \# zs \Longrightarrow distinct(bs_i) \Longrightarrow zs = ts_i[bs_i;\, \pi \cdot xs_i] \wedge B[bs_i;\, \pi \cdot xs_i]) \Longrightarrow P$$

to show P, it suffices to prove

$$zs = ts_i[bs_i;\, \pi \cdot xs_i] \wedge B_i[bs_i;\, \pi \cdot xs_i] \tag{5}$$

under the assumptions

$$(a)\ bs_i \# zs \qquad \text{and} \qquad (b)\ distinct(bs_i) \tag{6}$$

Note that (2) implies

$$\pi \cdot zs = \pi \cdot ts_i[as_i;\, xs_i] \qquad \text{and} \qquad \pi \cdot B[as_i;\, xs_i] \tag{7}$$

from which (5) follows from (3), (4), (6), and equivariance.

Strong Inversion Rules – Beta Reduction

$$\frac{\begin{array}{l}\bigwedge s_2\ s_1.\ (y \# u_1 \Longrightarrow y \# u_2 \Longrightarrow \\ \quad u_1 = App\ (Lam\ y\ s_1)\ s_2 \wedge u_2 = s_1[y{:=}s_2] \wedge y \# s_2) \Longrightarrow P \\ \bigwedge s_1\ s_2\ t.\ u_1 = App\ s_1\ t \Longrightarrow u_2 = App\ s_2\ t \Longrightarrow s_1 \longrightarrow_\beta s_2 \Longrightarrow P \\ \bigwedge s_1\ s_2\ t.\ u_1 = App\ t\ s_1 \Longrightarrow u_2 = App\ t\ s_2 \Longrightarrow s_1 \longrightarrow_\beta s_2 \Longrightarrow P \\ \bigwedge s_1\ s_2.\ (x \# u_1 \Longrightarrow x \# u_2 \Longrightarrow \\ \quad u_1 = Lam\ x\ s_1 \wedge u_2 = Lam\ x\ s_2 \wedge s_1 \longrightarrow_\beta s_2) \Longrightarrow P\end{array}}{u_1 \longrightarrow_\beta u_2 \Longrightarrow P}$$

Variable convention compatibility condition

$$\frac{x \# s_2}{App\ (Lam\ x\ s_1)\ s_2 \longrightarrow_\beta s_1[x{:=}s_2]}$$

$$x \# s_2 \Longrightarrow \begin{array}{l} x \# App\ (Lam\ x\ s_1)\ s_2 \\ x \# s_1[x{:=}s_2]\end{array}$$

$$\frac{s_1 \longrightarrow_\beta s_2}{Lam\ x\ s_1 \longrightarrow_\beta Lam\ x\ s_2}$$

$$s_1 \longrightarrow_\beta s_2 \Longrightarrow \begin{array}{l} x \# Lam\ x\ s_1 \\ x \# Lam\ x\ s_2\end{array}$$

Strong Inversion Rules – Typing Relation

$$\bigwedge \Gamma\ x\ T.\ \Delta = \Gamma \Longrightarrow u = Var\ x \Longrightarrow U = T \Longrightarrow \\ valid\ \Gamma \Longrightarrow (x,\ T) \in set\ \Gamma \Longrightarrow P$$

$$\bigwedge t_1\ T_1\ T_2\ t_2.\ \Delta = \Gamma \Longrightarrow u = App\ t_1\ t_2 \Longrightarrow U = T_2 \Longrightarrow \\ \Gamma \vdash t_1 : T_1 \to T_2 \Longrightarrow \Gamma \vdash t_2 : T_1 \Longrightarrow P$$

$$\frac{\bigwedge T_1\ \Gamma\ t\ T_2.\ (x \# \Delta \Longrightarrow x \# u \Longrightarrow x \# U \Longrightarrow \\ \Delta = \Gamma \wedge u = Lam\ x\ t \wedge U = T_1 \to T_2 \wedge (x,\ T_1) :: \Gamma \vdash t : T_2) \Longrightarrow P}{\Delta \vdash u : U \Longrightarrow P}$$

Variable convention compatibility condition

$$\frac{(x,\ T_1) :: \Gamma \vdash t : T_2}{\Gamma \vdash Lam\ x\ t : T_1 \to T_2} \qquad (x,\ T_1) :: \Gamma \vdash t : T_2 \Longrightarrow \begin{array}{l} x \# \Gamma \\ x \# Lam\ x\ t \\ x \# T_1 \to T_2 \end{array}$$

Proof of Type Preservation in Isabelle

theorem *type-preservation*:
  assumes *ty*: $\Gamma \vdash u : U$ **and** *red*: $u \longrightarrow_\beta u'$
  shows $\Gamma \vdash u' : U$ **using** *ty red*
proof (*nominal-induct avoiding*: u' *rule*: *typing.strong-induct*)
  case (*ty-Var* Γ x T) **from** ‹$Var\ x \longrightarrow_\beta u'$› **show** $\Gamma \vdash u' : T$ **by** *cases simp-all*
next
  case (*ty-App* Γ t_1 T_1 T_2 t_2) ...
next
  case (*ty-Lam* x T_1 Γ t T_2)
  from ‹$Lam\ x\ t \longrightarrow_\beta u'$› ‹$x \# u'$› **obtain** s_2
    where *t-red*: $t \longrightarrow_\beta s_2$ **and** u'*-eq*: $u' = Lam\ x\ s_2$
    by (*cases rule*: *Beta.strong-cases*) (*force simp add*: *alpha*)+
  from *t-red* **have** $(x,\ T_1) :: \Gamma \vdash s_2 : T_2$ **by** (*rule ty-Lam*)
  then have $\Gamma \vdash Lam\ x\ s_2 : T_1 \to T_2$ **by** (*rule typing.ty-Lam*)
  with u'*-eq* **show** $\Gamma \vdash u' : T_1 \to T_2$ **by** *simp*
qed

Conclusion

- Standard inversion rules do not work well with non-injective constructors
- Strong inversion rules allow reasoning "like on paper"
- Variable convention compatibility condition for justifying the admissibility of strong inversion is the same as the one for justifying strong induction rules
- Strong inversion rules are derived automatically by the current version of the Nominal Package

Come to the Nominal Isabelle tutorial at IJCAR'08 in Sydney (11th August)

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

Thanks for your attention!