Nominal Datatypes in Isabelle/HOL

Christian Urban

joint work with Stefan Berghofer and the Isabelle-team in Munich

Formalising Informal Proofs

- Implemented a version of the nominal logic work as a definitional extension of Isabelle/HOL—that is as "syntactic sugar" (early nominal logic work had problems with the axiom of choice).
- Added an infrastructure for defining nominal datatypes—some of the more advanced reasoning infrastructure is however not yet in place. :o(
- My goal with this work is to make formalisations of informal "pencil-and-paper" proofs easy.

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

2.1.12. Convention: Terms that are α-congruent are identified. So now we write $\lambda x.x \equiv \lambda y.y$ etcetera.

2.1.13. 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.

2.1.14. Moral: Using conventions 2.1.12 and 2.1.13 one can work with λ-terms in the naive way.

tion hypothesis.

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. $M \equiv$

  implie

  Case 1.3. $M \equiv$

- **Case 2:** $M \equiv$ … e

  that $z \not\equiv x, y$ … s

  $(\lambda z.M_1)$

  $\equiv \lambda z.(M_1[$

  $\equiv \lambda z.(M_1[$

  $\equiv (\lambda z.M_1)$

- **Case 3:** $M \equiv$ … -

  tion hypothesis. … "

Remember: only if $y \not\equiv x$ and $x \notin FV(N)$ then

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

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

Formal Proof in Isabelle

```
lemma forget:
assumes a: "x # L"
shows "L[x ::= N] = L"
using a by (nominal_induct L avoiding: x N rule: lam_induct)
           (auto simp add: abs_fresh fresh_atm)

lemma fresh_fact:
fixes x :: "name"
assumes a: "x # M" and b: "x # N"
shows "x # M[y ::= N]"
using a b by (nominal_induct M avoiding: x y N rule: lam_induct)
             (auto simp add: abs_fresh fresh_atm)

lemma subst_lemma:
assumes a: "x ≠ 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 "$L[x ::= N] = L$"
using a by (nominal_induct L avoiding: x N rule: lam_induct)
(auto simp add: abs_fresh fresh_atm)

lemma fresh_fact:
fixes x :: "nam
assumes a: "x
shows "x #
using a b by (
(

lemma subst_

abs_fresh:

$$(x \# \mathsf{Lam}\,[y].M) = (x = y \vee x \# M)$$

fresh_atm:

$$(x \# y) = (x \neq y)$$

provided y is an atom

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 "$L[x ::= N] = L$"
using a by (nominal_induct L avoiding: x N rule: lam_induct)
(auto simp add: abs_fresh fresh_atm)

lemma fresh_fact:
fixes x :: "name"
assumes a: "$x \,\#\, M$" and b: "$x \,\#\, N$"
shows "$x \,\#\, M[y ::= N]$"
using a b by (nominal_induct M avoiding: x y N 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)

Weakening Lemma

Weakening Lemma: If $\Gamma_1 \vdash t : \sigma$ and Γ_2 is a valid super-context of Γ_1, then $\Gamma_2 \vdash t : \sigma$.

- **valid**: no variable has two assigned types
- **super-context**: Γ_2 has all the assignments of Γ_1

Proof: The literature is not helpful, because either too complicated[1] (if it is done right, e.g. Gallier, McKinna & Pollack, Pitts) or not helpful[2] (labelled as obvious, routine, trivial, e.g. Hindley).

[1] We can do better.
[2] I am sure they would need lots of fixing, if they were spelled out.

Weakening Lemma

- Declare a type of atoms (for variables in the λ-calculus):

  atom_decl name

- Declare the nominal datatype for α-equated lambda-terms:

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

  We will get equations such as:

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

Weakening Lemma (cont.)

- Declare the (nominal) datatype of types:

```
nominal_datatype ty =
        TVar  "nat"
      | TArr  "ty" "ty" (infix "→" 100)
```

- Define the notion of a valid context:

const valid :: "(name × ty) list ⇒ bool"

$$\frac{}{\text{valid } []} \qquad \frac{\text{valid } \Gamma \quad a \# \Gamma}{\text{valid } (a, \sigma) :: \Gamma}$$

- Define what the super-context relation is:

sc::"(name × ty)list⇒(name × ty)list⇒bool" (infix "≪" 100)

$$\Gamma_1 \ll \Gamma_2 \stackrel{\text{def}}{=} \forall x\, \sigma.\, (x, \sigma) \in \text{set } \Gamma_1 \longrightarrow (x, \sigma) \in \text{set } \Gamma_2$$

Weakening Lemma (cont.)

- Define the typing-judgement:

const typing :: "(name $\times$ ty) list $\Rightarrow$ lam $\Rightarrow$ ty $\Rightarrow$ bool"
("_ $\vdash$ _ : _" [100,100,100] 100)

$$\frac{\text{valid}\ \Gamma \quad (a,\tau) \in \text{set}\ \Gamma}{\Gamma \vdash \text{Var}\ a : \tau}\,\text{t1} \qquad \frac{\Gamma \vdash t_1 : \tau \to \sigma \quad \Gamma \vdash t_2 : \tau}{\Gamma \vdash \text{App}\ t_1\ t_2 : \sigma}\,\text{t2}$$

$$\frac{(a,\tau) :: \Gamma \vdash t : \sigma \quad a \# \Gamma}{\Gamma \vdash \text{Lam}\ [a].t : \tau \to \sigma}\,\text{t3}$$

Formal Proof

lemma weakening_lemma:
assumes a: "$\Gamma_1 \vdash t : \sigma$" and b: "$\Gamma_1 \ll \Gamma_2$" and c: "valid Γ_2"
shows "$\Gamma_2 \vdash t : \sigma$"
using a b c
apply (nominal_induct Γ_1 t σ avoiding: Γ_2 rule: typing_induct)
apply (auto simp add: sc_def)
done

Formal Proof

```
lemma weakening_lemma:
assumes  a: "Γ₁ ⊢ t : σ" and  b: "Γ₁ ≪ Γ₂" and  c: "valid Γ₂"
shows "Γ₂ ⊢ t : σ"
using a b c
proof (nominal_induct Γ₁ t σ avoiding: Γ₂ rule: typing_induct)
  case (t3 a Γ₁ τ σ t) (* lambda-case *)
  have a1: "Γ₁ ≪ Γ₂" by fact
  have a2: "valid Γ₂" by fact
  have ih: "⋀Γ₃. (a,τ)::Γ₁ ≪ Γ₃ ⟹ valid Γ₃ ⟹ Γ₃ ⊢ t : σ" by fact
  have fc: "a # Γ₂" by fact
  have "(a,τ)::Γ₁ ≪ (a,τ)::Γ₂" using a1 sc_def by simp
  moreover
  have "valid (a,τ)::Γ₂" using a2 fc by force
  ultimately have "(a,τ)::Γ₂ ⊢ t : σ" using ih by simp
  with fc show "Γ₂ ⊢ Lam [a].t : τ → σ" by force
qed by (auto simp add: sc_def) (* other cases *)
```

Reducibility Lemma: Let t be any term (not assumed to be reducible), and suppose all the free variables of t are among $x_1,\ldots,x_n$ of types $U_1,\ldots,U_n$. If $u_1,\ldots,u_n$ are reducible terms of types $U_1,\ldots,U_n$ then $t[x_1 := u_1, \ldots, x_n := u_n]$ is reducible.

Proof: By induction on t. We write $t[\underline{x} := \underline{u}]$ for the substitution.

- **Case 1** t is x_i:

  one has to check the tautology "if u_i is reducible, then u_i is reducible"; details are left to the reader.

- **Case 2** t is $w\,v$:

  by induction hypothesis $w[\underline{x} := \underline{u}]$ and $v[\underline{x} := \underline{u}]$ are reducible, and so (by definition) $w[\underline{x} := \underline{u}](v[\underline{x} := \underline{u}])$; but this term is nothing other than $t[\underline{x} := \underline{u}]$.

- **Case 3** t is $\lambda y.w$ of type $V \to W$:

  by induction hypothesis, $w[\underline{x} := \underline{u}, y := v]$ is reducible for all v of type V. Lemma 6.3.2 says that $t[\underline{x} := \underline{u}] = \lambda y.(w[\underline{x} := \underline{u}])$ is reducible. "$\square$"

Reducibility Lemma: Let t be any term (not assumed to be reducible), and suppose all the free variables of t are among $x_1,\ldots,x_n$ of types $U_1,\ldots,U_n$. If $u_1,\ldots,u_n$ are reducible terms of types $U_1,\ldots,U_n$ then $t[x_1 := u_1, \ldots, x_n := u_n]$ is reducible.

Proof: By induction on t. We write $t[\underline{x} := \underline{u}]$ for the substitution.

- **Case 1** t is x_i:

  one has to check the tautology "if u_i is reducible, then u_i is reducible"; details are left to the reader.

- **Case 2** t is $w\,v$:

  by induction hypothesis $w[\underline{x} := \underline{u}]$ and $v[\underline{x} := \underline{u}]$ are reducible, and so (by definition) $w[\underline{x} := \underline{u}](v[\underline{x} := \underline{u}])$; but this term is nothing other than $t[\underline{x} := \underline{u}]$.

- **Case 3** t is $\lambda y.w$ of type $V \to W$:

  by induction hypothesis, $w[\underline{x} := \underline{u}, y := v]$ is reducible for all v of type V. Lemma 6.3.2 says that $t[\underline{x} := \underline{u}] = \lambda y.(w[\underline{x} := \underline{u}])$ is reducible. "$\Box$"

> Lemma 6.3.2: If for all reducible u of type U, $v[x := u]$ is reducible, then so is $\lambda x.v$.

lemma all_Rd:
assumes a: "$\Gamma \vdash t : \tau$"
    and b: "$\forall x\, \sigma.\ (x,\sigma) \in \mathsf{set}\ \Gamma \longrightarrow x \in \mathsf{dom}(\theta) \wedge \theta\langle x\rangle \in Rd_\sigma$"
shows "$t[\theta] \in Rd_\tau$"
using a b proof (nominal_induct t avoiding: $\Gamma\ \tau\ \theta$ rule: lam_induct)
case (Lam a t)
  have ih: "$\bigwedge \Gamma\, \tau\, \theta.\ \Gamma \vdash t : \tau \Longrightarrow$
  $(\forall x\, \sigma.\ (x,\sigma) \in \mathsf{set}\ \Gamma \longrightarrow x \in \mathsf{dom}(\theta) \wedge \theta\langle x\rangle \in Rd_\sigma) \longrightarrow t[\theta] \in Rd_\tau$"

Reducibility Lemma: Let t be any term (not assumed to be reducible), and suppose all the free variables of t are among $x_1,\ldots,x_n$ of types $U_1,\ldots,U_n$. If $u_1,\ldots,u_n$ are reducible terms of types $U_1,\ldots,U_n$ then $t[x_1 := u_1, \ldots, x_n := u_n]$ is reducible.

  hence "$\exists \tau_1 \tau_2.\ \tau = \tau_1 \rightarrow \tau_2 \wedge (a,\tau_1)::\Gamma \vdash t : \tau_2$" using t3_elim fr by simp
  then obtain τ_1 τ_2 where τ_inst: "$\tau = \tau_1 \rightarrow \tau_2$"
        and ty: "$(a,\tau_1)\#\Gamma \vdash t : \tau_2$" by blast
  from ih have "$\forall s \in Rd_{\tau_1}.\, t[\theta][a ::= s] \in Rd_{\tau_2}$" using fr ty θ_cond
    by (force dest: fresh_context simp add: psubts_subst)
  hence "Lam $[a].(t[\theta]) \in Rd_{\tau_1 \rightarrow \tau_2}$" by (simp only: abs_Rd) –"Lem. 6.3.2"
thus "(Lam $[a].t)[\theta] \in Rd_\tau$" using fr τ_inst by simp
qed (force dest!: t1_elim t2_elim)+

```
lemma all_Rd:
assumes a: "Γ ⊢ t : τ"
    and b: "∀x σ. (x,σ) ∈ set Γ ⟶ x ∈ dom(θ) ∧ θ⟨x⟩ ∈ Rd_σ"
shows "t[θ] ∈ Rd_τ"
using a b proof (nominal_induct t avoiding: Γ τ θ  rule:  lam_induct)
case (Lam  a  t)
  have ih: "⋀ Γ τ θ. Γ ⊢ t : τ ⟹
  (∀x σ. (x,σ) ∈ set Γ ⟶ x ∈ dom(θ) ∧ θ⟨x⟩ ∈ Rd_σ) ⟹ t[θ] ∈ Rd_τ"
  and θ_cond: "∀x σ. (x,σ) ∈ set Γ ⟶ x ∈ dom(θ) ∧ θ⟨x⟩ ∈ Rd_σ"
  and fr: "a # Γ" "a # θ"
  and "Γ ⊢ Lam [a].t : τ" by fact
  hence "∃τ1 τ2. τ = τ1→τ2 ∧ (a,τ1)::Γ ⊢ t : τ2" using t3_elim fr by simp
  then obtain τ1 τ2 where τ_inst: "τ = τ1→τ2"
                      and ty: "(a,τ1)#Γ ⊢ t : τ2" by blast
  from ih have "∀s ∈ Rd_τ1 .t[θ][a ::= s] ∈ Rd_τ2" using fr ty θ_cond
    by (force dest: fresh_context simp add: psubts_subst)
  hence "Lam [a].(t[θ]) ∈ Rd_τ1→τ2" by (simp only: abs_Rd) –"Lem. 6.3.2"
thus "(Lam [a].t)[θ] ∈ Rd_τ" using fr τ_inst by simp
qed (force dest!: t1_elim t2_elim)+
```

What We Can Do

Good news:

- CR and SN from the books
- transitivity and narrowing of subtyping for System $F_{<:}$ (first part of the POPLmark challenge)
- SN as in my PhD, etc.

Bad news:

- it all looks easy, but some expertise in theorem provers is still a prerequisite (like $\LaTeX$)
- formalising things will always take more time than with pencil and paper
- there are still some gaps in the reasoning infrastructure of the nominal datatype package (I cheated a bit in the weakening example)