Reasoning about constants in Nominal Isabelle

Formalizing the Second Fixed Point Theorem

Cezary Kaliszyk
University of Innsbruck

Henk Barendregt
Radboud University Nijmegen

24 April, Seminar 3

Outline

Nominal Isabelle

- Framework for constructing α-equated terms

$$t ::= x \mid t\ t \mid \lambda x.\ t$$

- Definitional extension of Isabelle/HOL
- Automatically derives a reasoning infrastructure
    - free variables (support and freshness)
    - renaming
    - strong induction principle

Nominal Isabelle (history)

Nominal has been used successfully in formalisations of:

- π-calculus, ψ-calculus, spi-calculus

  [*BengtsonParow07, BengtsonParow09, KahsaiMiculan*]

- Typed Scheme

  [*TobinHochstadtFelleisen08*]

- Equivalence checking algorithm for LF

  [*UrbanCheneyBerghofer08*]

- Strong normalisation of cut-elimination in classical logic

  [*UrbanZhu08*]

- Formalizations in the locally-nameless approach to binding

  [*SatoPollack10*]

- Compiler Verification

  [*HellerPhd10*]

- Mini X-Query

  [*Cheney11*]

```
nominal_datatype lam =
  Var name
| App lam lam
| Lam x::name l::lam  binds x in l      (λ_. _)

nominal_primrec
  height :: lam ⇒ int
where
  height (Var x) = 1
| height (App t1 t2) = max (height t1) (height t2) + 1
| height (λx. t) = height t + 1

nominal_primrec
  subst :: lam ⇒ name ⇒ lam ⇒ lam              (_ [_ ::= _])
where
  (Var x)[y ::= s] = (if x = y then s else (Var x))
| (App t1 t2)[y ::= s] = App (t1[y ::= s]) (t2[y ::= s])
| atom x ♯ (y, s) ⟹ (λx. t)[y ::= s] = λx. (t[y ::= s])
```

```
nominal_datatype lam =
  Var name
| App lam lam
| Lam x::name l::lam  binds x in l      (λ_. _)

nominal_primrec
  height :: lam ⇒ int
where
  height (Var x) = 1
| height (App t1 t2) = max (height t1) (height t2) + 1
| height (λx. t) = height t + 1

nominal_primrec
  subst :: lam ⇒ name ⇒ lam ⇒ lam            (_ [_ ::= _])
where
  (Var x)[y ::= s] = (if x = y then s else (Var x))
| (App t1 t2)[y ::= s] = App (t1[y ::= s]) (t2[y ::= s])
| atom x ♯ (y, s) ⟹ (λx. t)[y ::= s] = λx. (t[y ::= s])
```

```
nominal_datatype lam =
  Var name
| App lam lam
| Lam x::name l::lam  binds x in l      (λ_. _)

nominal_primrec
  height :: lam ⇒ int
where
  height (Var x) = 1
| height (App t1 t2) = max (height t1) (height t2) + 1
| height (λx. t) = height t + 1

nominal_primrec
  subst :: lam ⇒ name ⇒ lam ⇒ lam              (_ [_ ::= _])
where
  (Var x)[y ::= s] = (if x = y then s else (Var x))
| (App t1 t2)[y ::= s] = App (t1[y ::= s]) (t2[y ::= s])
| atom x ♯ (y, s) ⟹ (λx. t)[y ::= s] = λx. (t[y ::= s])
```

```
lemma height_ge_one: 1 ≤ (height e)
  by (induct e rule: lam.induct) (simp_all)

theorem height (e[x::=e']) ≤ height e - 1 + height e'
proof (nominal_induct e avoiding: x e' rule: lam.strong_induct)
  case (Var y)
  have 1 ≤ height e' using height_ge_one by simp
  then show height (Var y[x::=e']) ≤ height (Var y) - 1 + height e' by simp
next
  case (App e1 e2)
  have ih1: height (e1[x::=e']) ≤ (height e1) - 1 + height e'
  and  ih2: height (e2[x::=e']) ≤ (height e2) - 1 + height e' by fact+
  then show height ((App e1 e2)[x::=e']) ≤ height (App e1 e2) - 1 + height e'
    by simp
next
  case (Lam y e1)
  have ih: height (e1[x::=e']) ≤ height e1 - 1 + height e' by fact
  have vc: atom y ♯ x   atom y ♯ e' by fact+
  show height ((λy. e1)[x::=e']) ≤ height (λy. e1) - 1 + height e'
    using ih vc by simp
qed
```

```
lemma height_ge_one: 1 ≤ (height e)
  by (induct e rule: lam.induct) (simp_all)

theorem height (e[x::=e']) ≤ height e - 1 + height e'
proof (nominal_induct e avoiding: x e' rule: lam.strong_induct)
  case (Var y)
  have 1 ≤ height e' using height_ge_one by simp
  then show height (Var y[x::=e']) ≤ height (Var y) - 1 + height e' by simp
next
  case (App e1 e2)
  have ih1: height (e1[x::=e']) ≤ (height e1) - 1 + height e'
  and  ih2: height (e2[x::=e']) ≤ (height e2) - 1 + height e' by fact+
  then show height ((App e1 e2)[x::=e']) ≤ height (App e1 e2) - 1 + height e'
    by simp
next
  case (Lam y e1)
  have ih: height (e1[x::=e']) ≤ height e1 - 1 + height e' by fact
  have vc: atom y ♯ x   atom y ♯ e' by fact+
  show height ((λy. e1)[x::=e']) ≤ height (λy. e1) - 1 + height e'
    using ih vc by simp
qed
```

```
lemma height_ge_one: 1 ≤ (height e)
  by (induct e rule: lam.induct) (simp_all)

theorem height (e[x::=e']) ≤ height e - 1 + height e'
proof (nominal_induct e avoiding: x e' rule: lam.strong_induct)
  case (Var y)
  have 1 ≤ height e' using height_ge_one by simp
  then show height (Var y[x::=e']) ≤ height (Var y) - 1 + height e' by simp
next
  case (App e1 e2)
  have ih1: height (e1[x::=e']) ≤ (height e1) - 1 + height e'
  and  ih2: height (e2[x::=e']) ≤ (height e2) - 1 + height e' by fact+
  then show height ((App e1 e2)[x::=e']) ≤ height (App e1 e2) - 1 + height e'
    by simp
next
  case (Lam y e1)
  have ih: height (e1[x::=e']) ≤ height e1 - 1 + height e' by fact
  have vc: atom y ♯ x   atom y ♯ e' by fact+
  show height ((λy. e1)[x::=e']) ≤ height (λy. e1) - 1 + height e'
    using ih vc by simp
qed
```

Outline

Defining Constants and Functions

- Nominal Primrec
  - primitive recursion, well understood, no new vars
- Fresh Fun
  - reasoning about the term and freshness together
- Isabelle/HOL function package
  - non-injective datatypes - completeness and compatibility
  - mutually recursive functions, non-primitive-recursive functions, or even functions on datatypes which abstract multiple binders
- Quotients
- Use fixed new names
  - convertibility to statments with assumptions

Outline

- Nominal Logic
  - Nominal Isabelle and Reasoning
- Defining nominal constants and functions
  - Possible approaches
- **Second Fixed Point Theorem**
  - Book Statement
  - Second Fixed Point Theorem
- Conclusion

Scan

6.5.9. SECOND FIXED POINT THEOREM.

$$\forall F\ \exists X \quad F\ulcorner X\urcorner = X.$$

PROOF. By the effectiveness of $\sharp$, there are recursive functions Ap and Num such that Ap $(\sharp M, \sharp N) = \sharp MN$ and Num$(n) = \sharp\ulcorner n\urcorner$. Let Ap and Num be λ-defined by $\mathbf{Ap}$ and $\mathbf{Num} \in \Lambda^0$. Then

$$\mathbf{Ap}\ulcorner M\urcorner\ulcorner N\urcorner = \ulcorner MN\urcorner, \qquad \mathbf{Num}\ulcorner n\urcorner = \ulcorner\ulcorner n\urcorner\urcorner;$$

Hence in particular

$$\mathbf{Num}\ulcorner M\urcorner = \ulcorner\ulcorner M\urcorner\urcorner.$$

Now define

$$W \equiv \lambda x. F(\mathbf{Ap}\, x(\mathbf{Num}\, x)), \qquad X \equiv W\ulcorner W\urcorner.$$

Then

$$X \equiv W\ulcorner W\urcorner = F(\mathbf{Ap}\ulcorner W\urcorner(\mathbf{Num}\ulcorner W\urcorner))$$

$$= F\ulcorner W\ulcorner W\urcorner\urcorner \equiv F\ulcorner X\urcorner. \quad \square$$

Second Fixed Point Theorem

Notations:

nominal_datatype t =
  $\overline{v}$
| t · t
| λx. t   **bind** x **in** t

Term encoding (Böhm trees):

$$\ulcorner t \urcorner$$

Substitution and Convertibility

Substituting a variable y for a term S in term M is defined by:

$$\mathsf{T}\,[\mathsf{y} := \mathsf{S}] = \begin{cases} \text{if } \mathsf{x} = \mathsf{y} \text{ then } \mathsf{S} \text{ else } \bar{\mathsf{x}} & \text{if } \mathsf{T} = \bar{\mathsf{x}} \\ (\mathsf{T}_1\,[\mathsf{y} := \mathsf{S}]) \cdot (\mathsf{T}_2\,[\mathsf{y} := \mathsf{S}]) & \text{if } \mathsf{T} = \mathsf{T}_1 \cdot \mathsf{T}_2 \\ \lambda \mathsf{x}.\ \mathsf{U}\,[\mathsf{y} := \mathsf{S}] & \text{if } \mathsf{T} = \lambda \mathsf{x}.\ \mathsf{U} \\ & \text{and } \mathsf{x} \# (\mathsf{y}, \mathsf{S}) \end{cases}$$

Convertibility is an inductively defined relation axiomatized by the following (note: no =):

$$\begin{aligned} &(\lambda x.\ M) \cdot N \approx M\,[x := N] \\ &M \approx M \\ &M \approx N \Longrightarrow N \approx M \\ &M \approx N \Longrightarrow N \approx L \Longrightarrow M \approx L \\ &M \approx N \Longrightarrow Z \cdot M \approx Z \cdot N \\ &M \approx N \Longrightarrow M \cdot Z \approx N \cdot Z \\ &M \approx N \Longrightarrow (\lambda x.\ M) \approx (\lambda x.\ N) \end{aligned}$$

Initial Functions

Assuming $\mathsf{x} \neq \mathsf{y}$, $\mathsf{y} \neq \mathsf{z}$ and $\mathsf{z} \neq \mathsf{x}$:

$$\begin{aligned} U^2_0 &= \lambda \mathsf{x}.\ \lambda \mathsf{y}.\ \lambda \mathsf{z}.\ \overline{\mathsf{z}} \\ U^2_1 &= \lambda \mathsf{x}.\ \lambda \mathsf{y}.\ \lambda \mathsf{z}.\ \overline{\mathsf{y}} \\ U^2_2 &= \lambda \mathsf{x}.\ \lambda \mathsf{y}.\ \lambda \mathsf{z}.\ \overline{\mathsf{x}} \end{aligned}$$

Assuming $\mathsf{x} \neq \mathsf{y}$, $\mathsf{x} \neq \mathsf{e}$ and $\mathsf{y} \neq \mathsf{e}$:

$$\begin{aligned} \mathsf{Var} &= \lambda \mathsf{x}.\ \lambda \mathsf{e}.\ \overline{\mathsf{e}} \cdot U^2_2 \cdot \overline{\mathsf{x}} \cdot \overline{\mathsf{e}} \\ \mathsf{App} &= \lambda \mathsf{x}.\ \lambda \mathsf{y}.\ \lambda \mathsf{e}.\ \overline{\mathsf{e}} \cdot U^2_1 \cdot \overline{\mathsf{x}} \cdot \overline{\mathsf{y}} \cdot \overline{\mathsf{e}} \\ \mathsf{Abs} &= \lambda \mathsf{x}.\ \lambda \mathsf{e}.\ \overline{\mathsf{e}} \cdot U^2_0 \cdot \overline{\mathsf{x}} \cdot \overline{\mathsf{e}} \end{aligned}$$

Böhm Encoding (1/2)

For a given λ-term t, its Böhm encoding $\ulcorner \mathsf{t} \urcorner$ is is defined by:

$$\ulcorner \mathsf{t} \urcorner = \begin{cases} \mathsf{Var} \cdot \overline{\mathsf{x}} & \text{provided } \mathsf{t} = \overline{\mathsf{x}} \\ \mathsf{App} \cdot \ulcorner \mathsf{M} \urcorner \cdot \ulcorner \mathsf{N} \urcorner & \text{provided } \mathsf{t} = \mathsf{M} \cdot \mathsf{N} \\ \mathsf{Abs} \cdot (\lambda \mathsf{x}.\ \ulcorner \mathsf{M} \urcorner) & \text{provided } \mathsf{t} = (\lambda \mathsf{x}.\ \mathsf{M}) \end{cases}$$

Böhm Encoding (1/2)

For a given λ-term t, its Böhm encoding $\ulcorner \mathsf{t} \urcorner$ is is defined by:

$$\ulcorner \mathsf{t} \urcorner = \begin{cases} \mathsf{Var} \cdot \overline{\mathsf{x}} & \text{provided } \mathsf{t} = \overline{\mathsf{x}} \\ \mathsf{App} \cdot \ulcorner \mathsf{M} \urcorner \cdot \ulcorner \mathsf{N} \urcorner & \text{provided } \mathsf{t} = \mathsf{M} \cdot \mathsf{N} \\ \mathsf{Abs} \cdot (\lambda \mathsf{x}.\ \ulcorner \mathsf{M} \urcorner) & \text{provided } \mathsf{t} = (\lambda \mathsf{x}.\ \mathsf{M}) \end{cases}$$

But we also need a λ-term that represents the Böhm encoding!

Böhm Encoding (2/2)

Don't try to understand!

Assuming $a \neq b$, $b \neq c$ and $c \neq a$:

$$
\begin{aligned}
F_1 &= (\lambda a.\ \mathsf{App} \cdot \ulcorner \mathsf{Var} \urcorner \cdot (\mathsf{Var} \cdot \bar{a})) \\
F_2 &= (\lambda a.\ \lambda b.\ \lambda c.\ \mathsf{App} \cdot (\mathsf{App} \cdot \ulcorner \mathsf{App} \urcorner \cdot (\bar{c} \cdot \bar{a})) \cdot (\bar{c} \cdot \bar{b})) \\
F_3 &= (\lambda a.\ \lambda b.\ \mathsf{App} \cdot \ulcorner \mathsf{Abs} \urcorner \cdot (\mathsf{Abs} \cdot (\lambda c.\ \bar{b} \cdot (\bar{a} \cdot \bar{c})))) \\
A_1 &= (\lambda a.\ \lambda b.\ F_1 \cdot \bar{a}) \\
A_2 &= (\lambda a.\ \lambda b.\ \lambda c.\ F_2 \cdot \bar{a} \cdot \bar{b} \cdot [\bar{c}]) \\
A_3 &= (\lambda a.\ \lambda b.\ F_3 \cdot \bar{a} \cdot [\bar{b}]) \\
&[M] \cdot N \approx N \cdot M \\
&[M, N, P] \cdot R \approx R \cdot M \cdot N \cdot P \\
\mathsf{NUM} &\stackrel{\text{def}}{=} [[A_1, A_2, A_3]]
\end{aligned}
$$

```
lemma NUM · ⌜M⌝ ≈ ⌜⌜M⌝⌝ proof (induct M)
  case n̄
  have NUM · ⌜(n̄)⌝ = NUM · (Var · n̄) by simp
  also have ... = [[A₁, A₂, A₃]] · (Var · n̄) by simp
  also have ... ≈ Var · n̄ · [A₁, A₂, A₃] using 8 .
  also have ... ≈ [A₁, A₂, A₃] · U²₂ · n̄ · [A₁, A₂, A₃] using 5 .
  also have ... ≈ A₁ · n̄ · [A₁, A₂, A₃] using 9 by simp
  also have ... ≈ F₁ · n̄ using 13 .
  also have ... ≈ App · ⌜Var⌝ · (Var · n̄) using 10 .
  also have ... = ⌜⌜(n̄)⌝⌝ by simp
  finally show NUM · ⌜(n̄)⌝ ≈ ⌜⌜(n̄)⌝⌝ .
next case M · N
  assume IH: NUM · ⌜M⌝ ≈ ⌜⌜M⌝⌝  NUM · ⌜N⌝ ≈ ⌜⌜N⌝⌝
  have NUM · ⌜(M · N)⌝ = NUM · (App · ⌜M⌝ · ⌜N⌝) by simp
  also have ... = [[A₁, A₂, A₃]] · (App · ⌜M⌝ · ⌜N⌝) by simp
  also have ... ≈ App · ⌜M⌝ · ⌜N⌝ · [A₁, A₂, A₃] using 8 .
  also have ... ≈ [A₁, A₂, A₃] · U²₁ · ⌜M⌝ · ⌜N⌝ · [A₁, A₂, A₃] using 6 .
  also have ... ≈ A₂ · ⌜M⌝ · ⌜N⌝ · [A₁, A₂, A₃] using 9 by simp
  also have ... ≈ F₂ · ⌜M⌝ · ⌜N⌝ · NUM using 14 by simp
  also have ... ≈ App · (App · ⌜App⌝ · (NUM · ⌜M⌝)) · (NUM · ⌜N⌝) using 11 .
  also have ... ≈ App · (App · ⌜App⌝ · ⌜⌜M⌝⌝) · (NUM · ⌜N⌝) using IH by simp
  also have ... ≈ ⌜⌜(M · N)⌝⌝ using IH by simp
  finally show NUM · ⌜(M · N)⌝ ≈ ⌜⌜(M · N)⌝⌝ .
next case λx. P
  assume IH: NUM · ⌜P⌝ ≈ ⌜⌜P⌝⌝
  have NUM · ⌜(λx. P)⌝ = NUM · (Abs · (λx. ⌜P⌝)) by simp
  also have ... = [[A₁, A₂, A₃]] · (Abs · (λx. ⌜P⌝)) by simp
  also have ... ≈ Abs · (λx. ⌜P⌝) · [A₁, A₂, A₃] using 8 .
  also have ... ≈ [A₁, A₂, A₃] · U²₀ · (λx. ⌜P⌝) · [A₁, A₂, A₃] using 7 .
  also have ... ≈ A₃ · (λx. ⌜P⌝) · [A₁, A₂, A₃] using 9 by simp
  also have ... ≈ F₃ · (λx. ⌜P⌝) · [[A₁, A₂, A₃]] using 15 .
  also have ... = F₃ · (λx. ⌜P⌝) · NUM by simp
  also have ... ≈ App · ⌜Abs⌝ · (Abs · (λx. NUM · ((λx. ⌜P⌝) · x̄))) by (rule 12) simp_all
  also have ... ≈ App · ⌜Abs⌝ · (Abs · (λx. NUM · ⌜P⌝)) using 4 by simp
  also have ... ≈ App · ⌜Abs⌝ · (Abs · (λx. ⌜⌜P⌝⌝)) using IH by simp
  also have ... = ⌜⌜(λx. P)⌝⌝ by simp
```

Second Fixed Point Theorem

theorem
  fixes $\mathsf{F} :: \mathsf{t}$
  shows $\exists \mathsf{X}.\ \mathsf{X} \approx \mathsf{F} \cdot \ulcorner \mathsf{X} \urcorner$
proof -

  def $\mathsf{W} \stackrel{\text{def}}{=} \lambda x.\ \mathsf{F} \cdot (\mathsf{App} \cdot \bar{x} \cdot (\mathsf{NUM} \cdot \bar{x}))$
  def $\mathsf{X} \stackrel{\text{def}}{=} \mathsf{W} \cdot \ulcorner \mathsf{W} \urcorner$
  have a: $\mathsf{X} = \mathsf{W} \cdot \ulcorner \mathsf{W} \urcorner$ **unfolding** X_def ..
  also have $\ldots = (\lambda x.\ \mathsf{F} \cdot (\mathsf{App} \cdot \bar{x} \cdot (\mathsf{NUM} \cdot \bar{x}))) \cdot \ulcorner \mathsf{W} \urcorner$ ..
  also have $\ldots \approx \mathsf{F} \cdot (\mathsf{App} \cdot \ulcorner \mathsf{W} \urcorner \cdot (\mathsf{NUM} \cdot \ulcorner \mathsf{W} \urcorner))$ **by** simp
  also have $\ldots \approx \mathsf{F} \cdot (\mathsf{App} \cdot \ulcorner \mathsf{W} \urcorner \cdot \ulcorner \ulcorner \mathsf{W} \urcorner \urcorner)$ **by** simp
  also have $\ldots \approx \mathsf{F} \cdot \ulcorner (\mathsf{W} \cdot \ulcorner \mathsf{W} \urcorner) \urcorner$ **by** simp
  also have $\ldots = \mathsf{F} \cdot \ulcorner \mathsf{X} \urcorner$ **unfolding** X_def ..
  finally show $\mathsf{X} \approx \mathsf{F} \cdot \ulcorner \mathsf{X} \urcorner$ ..
qed

Second Fixed Point Theorem

theorem
  fixes F :: t
  shows $\exists$X. X $\approx$ F $\cdot$ ⌜X⌝
proof -
  obtain x :: var **where** x # F **using** obtain_fresh **by** blast
  def W $\stackrel{\text{def}}{=}$ λx. F $\cdot$ (App $\cdot$ $\bar{x}$ $\cdot$ (NUM $\cdot$ $\bar{x}$))
  def X $\stackrel{\text{def}}{=}$ W $\cdot$ ⌜W⌝
  have a: X = W $\cdot$ ⌜W⌝ **unfolding** X_def ..
  also have ... = (λx. F $\cdot$ (App $\cdot$ $\bar{x}$ $\cdot$ (NUM $\cdot$ $\bar{x}$))) $\cdot$ ⌜W⌝ ..
  also have ... $\approx$ F $\cdot$ (App $\cdot$ ⌜W⌝ $\cdot$ (NUM $\cdot$ ⌜W⌝)) **by** simp
  also have ... $\approx$ F $\cdot$ (App $\cdot$ ⌜W⌝ $\cdot$ ⌜⌜W⌝⌝) **by** simp
  also have ... $\approx$ F $\cdot$ ⌜(W $\cdot$ ⌜W⌝)⌝ **by** simp
  also have ... = F $\cdot$ ⌜X⌝ **unfolding** X_def ..
  finally show X $\approx$ F $\cdot$ ⌜X⌝ ..
qed

Outline

Conclusion

- Constants and Functions in Nominal Isabelle
  - Nominal primrec / Function package
  - CPS
- Formalizing λ-calculus
- More use of quotients