Nominal Reasoning Techniques in Coq

(Work in Progress)

Brian Aydemir
Aaron Bohannon   Stephanie Weirich

University of Pennsylvania

16 August 2006

What is nominal reasoning (in Coq)?

- Using names for both bound and free variables

λx. x y $\rightarrow$ `lam (app 0 1)` ✖
`lam (app 0 (var y))` ✖
`lam x (app (var x) (var y))` ✔

- Using "built-in" equality to represent α-equality

1 + 1 = 2   `lam x (var x) = lam y (var y)`

- Minimizing the need to rename bound variables

How to implement this in Coq?

- `lam` is not injective!

$$\texttt{lam}\ x\ (\texttt{var}\ x) = \texttt{lam}\ y\ (\texttt{var}\ y) \not\to x = y$$

- Therefore, can't use native inductive datatypes.

```
Inductive tm : Set :=
  | var : tmvar → tm
  | app : tm → tm → tm
  | lam : tmvar → tm → tm.
```

Our solution

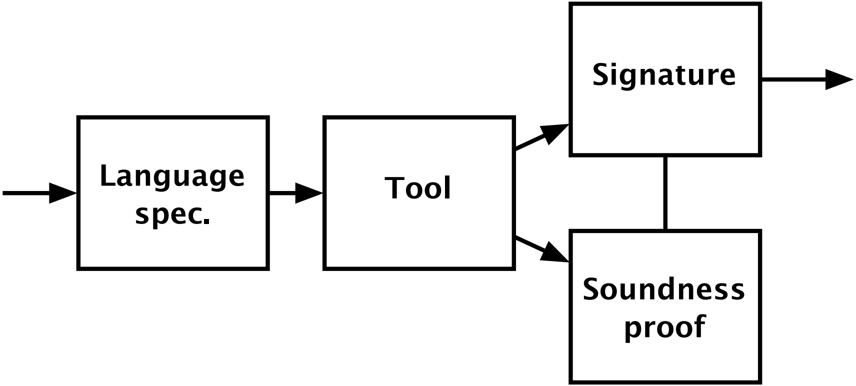
- Axiomatize everything.
- Similar in spirit to Gordon-Melham axioms.

- Types and constructors:

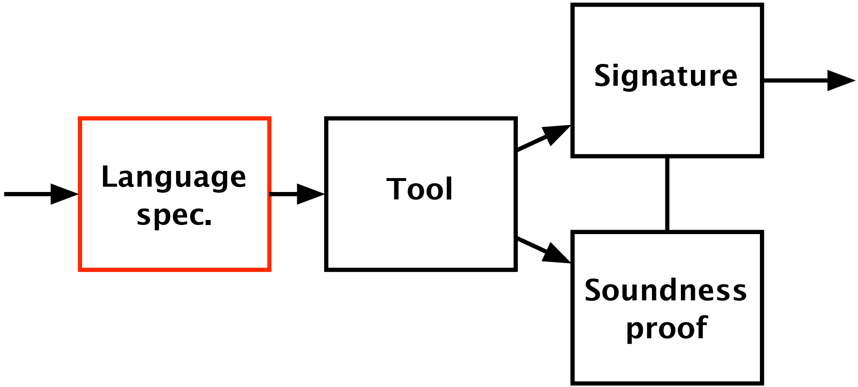
```
Parameter tmvar : AtomT.
Parameter tm : Set.

Parameter var : tmvar → tm.
Parameter app : tm → tm → tm.
Parameter lam : tmvar → tm → tm.
```

System description

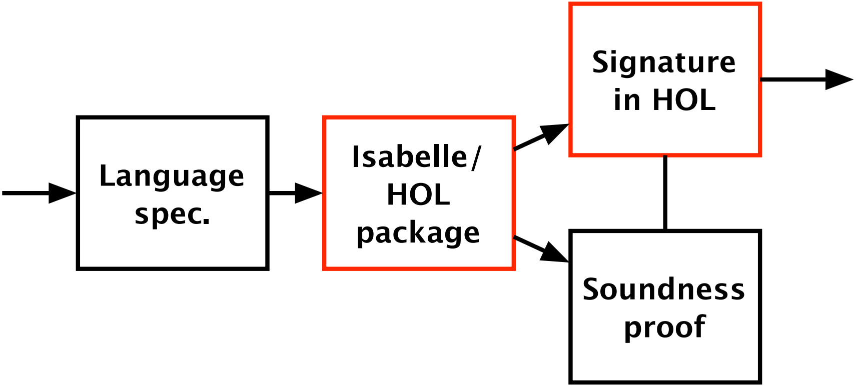


System description



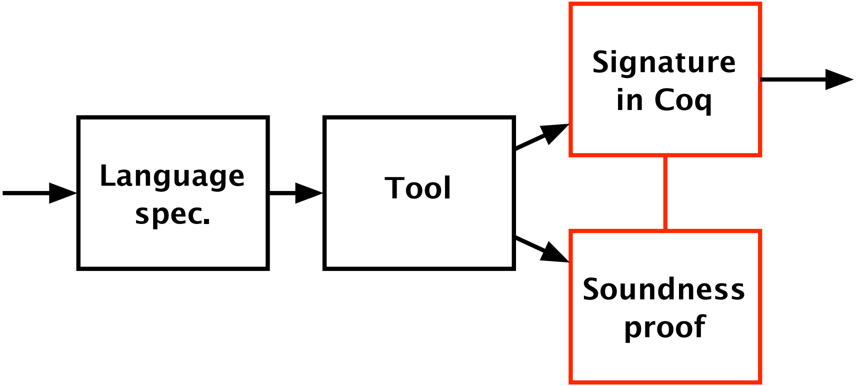
High-level description language may be similar to Fresh O'Caml, Cαml, Isabelle/HOL-Nominal

System description



Nominal datatype package for Isabelle/HOL
[Berghofer and Urban, 2006]

System description



Today: Nominal reasoning in Coq

Signature in Coq

- Types and constructors:

```
Parameter tmvar : AtomT.
Parameter tm : Set.

Parameter var : tmvar → tm.
Parameter app : tm → tm → tm.
Parameter lam : tmvar → tm → tm.
```

- Axioms for discrimination:

  $\forall$ x s t, var x $\neq$ app s t

- Axioms for injectivity:

  $\forall$ x x′, var x = var x′ $\rightarrow$ x = x′

Properties of `lam`

- Alpha-equivalence:
  $\forall$ x y t, y $\notin$ fvar t $\rightarrow$
  lam x t = lam y ((y, x) $\bullet$ t)

- Eliminating an equality:
  $\forall$ x x′ t t′, lam x t = lam x′ t′ $\rightarrow$
  (x = x′ $\wedge$ t = t′) $\vee$
  (x $\neq$ x′ $\wedge$ x $\notin$ fvar t′ $\wedge$ t = (x, x′) $\bullet$ t′)

- (y, x) $\bullet$ t denotes a swap, which we take from Nominal Logic.

Properties of `lam` (cont.)

- Free variables:
  $\forall$ x t, fvar (lam x t) = (fvar t) \ {x}

- Swapping:
  $\forall$ a b x t,
  (a, b) • (lam x t) = lam ((a, b) • x) ((a, b) • t)

Structural induction

```
∀ (P : tm → Prop) (F : aset tmvar),
(∀ x, P (var x)) →
(∀ t u, P t → P u → P (app t u)) →
(∀ x t, x ∉ F → P t → P (lam x t)) →
∀ t, P t.
```

- In the `lam` case, we only need to consider suitably fresh names x.
- This is equivalent to the principle that omits F.

Using the signature

- Proofs using this signature seem natural.
- We can use our induction principle to prove:
  `∀ y x t, y ∉ fvar t → t [y := s] = t`
- Proof: By induction on `t`.
  Choose "F" to be {y} ∪ `fvar s`.

Example proof: Property about substitution

In the `lam` case:

$$\frac{\begin{array}{l}\texttt{y} \notin \texttt{fvar (lam x t)} \\ \texttt{x} \notin \{\texttt{y}\} \cup \texttt{fvar s} \\ \texttt{y} \notin \texttt{fvar t} \longrightarrow \texttt{t [y := s] = t}\end{array}}{\texttt{(lam x t) [y := s]} = \texttt{lam x t}}$$

Example proof: Property about substitution

In the `lam` case:

$$\frac{\begin{array}{l}\texttt{y} \notin \texttt{fvar (lam x t)} \\ \color{red}{\texttt{x} \neq \texttt{y} \wedge \texttt{x} \notin \texttt{fvar s}} \\ \texttt{y} \notin \texttt{fvar t} \rightarrow \texttt{t [y := s] = t}\end{array}}{\texttt{(lam x t) [y := s] = lam x t}}$$

Next, since:

$\forall$ `x y t s, x` $\neq$ `y` $\rightarrow$ `x` $\notin$ `fvar s` $\rightarrow$
`(lam x t) [y := s] = lam x (t [y := s])`

Example proof: Property about substitution

In the `lam` case:

$$\frac{\begin{array}{l}\texttt{y} \notin \texttt{fvar (lam x t)} \\ \texttt{x} \neq \texttt{y} \wedge \texttt{x} \notin \texttt{fvar s} \\ \texttt{y} \notin \texttt{fvar t} \rightarrow \texttt{t [y := s] = t}\end{array}}{\texttt{lam x (t [y := s]) = lam x t}}$$

Next, recalling that:

$\forall$ x t, fvar (lam x t) $=$ (fvar t) $\setminus$ $\{$x$\}$

Example proof: Property about substitution

In the `lam` case:

$$\frac{\begin{array}{l} \texttt{y} \notin (\texttt{fvar t}) \setminus \{\texttt{x}\} \\ \texttt{x} \neq \texttt{y} \wedge \texttt{x} \notin \texttt{fvar s} \\ \texttt{y} \notin \texttt{fvar t} \longrightarrow \texttt{t [y := s] = t} \end{array}}{\texttt{lam x (t [y := s]) = lam x t}}$$

Example proof: Property about substitution

In the `lam` case:

$$\frac{\begin{array}{l} \mathtt{y} = \mathtt{x} \vee \mathtt{y} \notin \mathtt{fvar\ s} \\ \mathtt{x} \neq \mathtt{y} \wedge \mathtt{x} \notin \mathtt{fvar\ s} \\ \mathtt{y} \notin \mathtt{fvar\ t} \longrightarrow \mathtt{t\ [y\ :=\ s]} = \mathtt{t} \end{array}}{\mathtt{lam\ x\ (t\ [y\ :=\ s])} = \mathtt{lam\ x\ t}}$$

Example proof: Property about substitution

In the `lam` case:

$$\frac{\begin{array}{l} y = x \\ x \neq y \wedge x \notin \texttt{fvar}\ s \\ y \notin \texttt{fvar}\ t \longrightarrow t\ [y := s] = t \end{array}}{\texttt{lam}\ x\ (t\ [y := s]) = \texttt{lam}\ x\ t}$$

$$\frac{\begin{array}{l} y \notin \texttt{fvar}\ t \\ x \neq y \wedge x \notin \texttt{fvar}\ s \\ y \notin \texttt{fvar}\ t \longrightarrow t\ [y := s] = t \end{array}}{\texttt{lam}\ x\ (t\ [y := s]) = \texttt{lam}\ x\ t}$$

Some questions

Given our signature for the untyped λ-calculus:

1. Is this signature sound?
2. How do we define functions over terms?
3. What should be in this signature?

Is our signature sound?

- We model our signature using a locally nameless representation for terms.

- We do require two axioms.
  1. Proof irrelevance
  2. Extensional equality on functions

An operator for primitive recursion

```
Parameter tm_rec :
  ∀ R : Set,
  ∀ fv : tmvar → R,
  ∀ fa : tm → R → tm → R → R,
  ∀ fl : tmvar → tm → R → R,
  ∀ F : aset tmvar,
  (supports F (fv, fa, fl)) →
  (∃ b, (b ∉ F ∧
          ∀ x y, b ♯ (fl b x y))) →
  (tm → R).
```

Return type of the function being constructed.

An operator for primitive recursion

```
Parameter tm_rec :
  ∀ R : Set,
  ∀ fv : tmvar → R,
  ∀ fa : tm → R → tm → R → R,
  ∀ fl : tmvar → tm → R → R,
  ∀ F : aset tmvar,
  (supports F (fv, fa, fl)) →
  (∃ b, (b ∉ F ∧
         ∀ x y, b ♯ (fl b x y))) →
  (tm → R).
```

Functions for each case.

An operator for primitive recursion

```
Parameter tm_rec :
  ∀ R : Set,
  ∀ fv : tmvar → R,
  ∀ fa : tm → R → tm → R → R,
  ∀ fl : tmvar → tm → R → R,
  ∀ F : aset tmvar,
  (supports F (fv, fa, fl)) →
  (∃ b, (b ∉ F ∧
          ∀ x y, b ♯ (fl b x y))) →
  (tm → R).
```

Side conditions about names. [Pitts, 2006]

An operator for primitive recursion

```
Parameter tm_rec :
  ∀ R : Set,
  ∀ fv : tmvar → R,
  ∀ fa : tm → R → tm → R → R,
  ∀ fl : tmvar → tm → R → R,
  ∀ F : aset tmvar,
  (supports F (fv, fa, fl)) →
  (∃ b, (b ∉ F ∧
         ∀ x y, b ♯ (fl b x y))) →
  (tm → R).
```

Final result: A non-dependent function.

An operator for primitive recursion (cont.)

Key property:

```
∀ R fv fa fl F H J,
let g := (tm_rec R  fv fa fl  F H J) in
  ∀ x t, x ∉ F →
  g (lam x t) = fl x t (g t).
```

We can always swap names to make this rule apply.

Example: Substitution

Defining (_ [y := s]):

- Take fl to be (fun x t r ⇒ lam x r).
- Take F to be $\{y\} \cup$ fvar s.

Then

```
∀ x t, x ∉ F →
g (lam x t) = fl x t (g t).
```

becomes

```
∀ x t, x ∉ {y} ∪ fvar s →
(lam x t) [y := s] = lam x (t [y := s]).
```

Example: Substitution

Defining (_ [y := s]):

- Take fl to be (fun x t r ⇒ lam x r).
- Take F to be $\{y\} \cup$ fvar s.

Then

```
∀ x t, x ∉ F →
g (lam x t) = fl x t (g t).
```

becomes

```
∀ x t, x ≠ y → x ∉ fvar s →
(lam x t) [y := s] = lam x (t [y := s]).
```

What should be in our signature?

- We need the following:
  - Types and constructors
  - Injection and discrimination theorems
  - Alpha-equivalence
  - Free variables and swapping
  - Induction principle
  - Recursion operator
- Also include functions like substitution.
- We'll want to automatically generate more.
  - Specialized induction principles
  - Inversion principles for relations

Conclusions

- We've shown how "nominal reasoning" can work in Coq.
  - Using names for bound and free variables
  - No separate α-equivalence relation
  - Minimal need for name swapping
- Definitions and proofs follow informal practice.
- Future work: tool support, dependent swapping