Names and Numbers in Binding

Martijn Vermaat

mvermaat@cs.vu.nl
http://www.cs.vu.nl/~mvermaat/

Literature Study
March 13, 2009

Mechanical Reasoning

Shift from on-paper reasoning to mechanical reasoning:

- History of on-paper proofs and ideas
- Informal mechanical implementations of ideas
- Add a scale increase and we have a gap

Mechanical Reasoning

Shift from on-paper reasoning to mechanical reasoning:

- History of on-paper proofs and ideas
- Informal mechanical implementations of ideas
- Add a scale increase and we have a gap

Goal

Let's make rigorous mechanical reasoning possible.

Reasoning about Languages

Reasoning about languages

Often not intrinsically hard, but cumbersome in a mechanical setting.

Why?

Reasoning about Languages

Reasoning about languages

Often not intrinsically hard, but cumbersome in a mechanical setting.

Why?

- Most languages have a notion of binding
- Bindings and bound variables are easy on paper, hard on a computer

So we want to

> *Reason about terms with bindings in tools like Coq, in a way that is close to the on-paper way.*

Mechanical Reasoning about Languages

So we want to

> *Reason about terms with bindings in tools like Coq, in a way that is close to the on-paper way.*

We need a representation for binders and variables.

Representing Bindings

Classical problems related to binders and variables:

- α-equivalence
- α-conversion (e.g. in substitution)

Representing Bindings

Classical problems related to binders and variables:

- α-equivalence
- α-conversion (e.g. in substitution)

Let's look at some representations.

Running example

Substitution in untyped λ-calculus

Traditional Representation 1/2

Traditional representation with named variables:

$$
\begin{aligned}
M ::= {}& x && \text{variable} \\
\mid {}& \lambda x.M && \text{abstraction} \\
\mid {}& M\ M && \text{application}
\end{aligned}
$$

Abstractions introduce names.

Traditional Representation 2/2

- α-equivalent terms are routinely identified
- Substitution $M[N/x]$:

$$\begin{aligned} x[N/x] &= N & & \\ y[N/x] &= y & & x \neq y \\ (\lambda y.M')[N/x] &= \lambda y.M'[N/x] & & x \neq y \text{ and } y \text{ not free in } N \\ (M_1\ M_2)[N/x] &= M_1[N/x]\ M_2[N/x] & & \end{aligned}$$

- Now implement this

Simple Substitution

Use α-conversion to rename bound variables and define substituting N for x in M inductively* on M:

$$
\begin{aligned}
x[N/x] &= N \text{ if } x = y,\ y \text{ otherwise} \\
(\lambda y.M')[N/x] &= \lambda z.M'[z/y][N/x] && z \text{ not free in } N, M' \\
(M_1\ M_2)[N/x] &= M_1[N/x]\ M_2[N/x]
\end{aligned}
$$

Simple Substitution

Use α-conversion to rename bound variables and define substituting N for x in M inductively* on M:

$$
\begin{aligned}
x[N/x] &= N \text{ if } x = y,\ y \text{ otherwise} \\
(\lambda y.M')[N/x] &= \lambda z.M'[z/y][N/x] && z \text{ not free in } N, M' \\
(M_1\ M_2)[N/x] &= M_1[N/x]\ M_2[N/x]
\end{aligned}
$$

Already difficult enough to read, but just what we would intuitively do. So on paper, we can get by with some handwaving.

Named Variables in Coq

Term datatype:

```
Inductive term : Set :=
  | Var : name -> term
  | Abs : name -> term -> term
  | App : term -> term -> term.
```

Substitution in Coq

Simple substitution:

```
Fixpoint subst (t:term) (n:name) (t':term)
  {struct t'} : term :=
  match t' with
  | Var x =>
      if eq_name x n then t else t'
  | Abs x b =>
      let z := fresh_name
                 (n :: (free_vars t) ++ (free_vars b))
      in
      Abs z (subst t n (rename x z b))
  | App f a =>
      App (subst t n f) (subst t n a)
end.
```

Substitution in Coq

Simple substitution:

```
Fixpoint subst (t:term) (n:name) (t':term)
  {struct t'} : term :=
  match t' with
  | Var x =>
      if eq_name x n then t else t'
  | Abs x b =>
      let z := fresh_name
                 (n :: (free_vars t) ++ (free_vars b))
      in
      Abs z (subst t n (rename x z b))
  | App f a =>
      App (subst t n f) (subst t n a)
end.
```

But this is ill-defined.

Recursion on Term Size in Coq 1/3

Using term size as a measure:

```
Fixpoint size (t:term) : nat :=
  match t with
  | Var _    => 0
  | Abs x b => S (size b)
  | App f a => 1 + (size f) + (size a)
end.

Lemma size_rename : forall (n n':name) (t:term),
  size (rename n n' t) = size t.
Proof.
unfold size.
unfold rename.
induction t;
  [ case (eq_name n0 n); intro; trivial  (* Var *)
  | congruence                           (* Abs *)
  | congruence ].                        (* App *)
Qed.
```

Substitution with term size as a recursion measure:

```
Function subst (t:term) (n:name) (t':term)
  {measure size t'} : term :=
  match t' with
  | Var x =>
      if eq_name x n then t else t'
  | Abs x b =>
      let z := fresh_name
                 (n :: (free_vars t) ++ (free_vars b))
      in
      Abs z (subst t n (rename x z b))
  | App f a =>
      App (subst t n f) (subst t n a)
end.
(* Leaves us with 3 obligations. *)
```

Recursion on Term Size in Coq 3/3

Proving termination of `subst`:

```
Proof.
intros.
rewrite size_rename.
auto.

intros.
unfold size.
inversion f; omega.

intros.
unfold size.
inversion a; omega.
Defined.
```

Recursion on Term Size in Coq 3/3

Proving termination of `subst`:

```
Proof.
intros.
rewrite size_rename.
auto.

intros.
unfold size.
inversion f; omega.

intros.
unfold size.
inversion a; omega.
Defined.
```

We really prefer structural recursion.

Simultaneous Substitution

Stoughton suggests the structurally recursive simultaneous substitution $M\sigma$:

$$
\begin{aligned}
x\sigma &= \sigma x \\
(\lambda x.M')\sigma &= \lambda y.(M'\ \sigma[y/x]) \qquad\qquad y \text{ not free in } M', \sigma \\
(M_1\ M_2)\sigma &= M_1\sigma\ M_2\sigma
\end{aligned}
$$

where

$$
\sigma[N/y]\, x = \begin{cases} N & \text{if } x = y, \\ \sigma x & \text{otherwise} \end{cases}
$$

Simultaneous Substitution

Stoughton suggests the structurally recursive simultaneous substitution $M\sigma$:

$$\begin{aligned} x\sigma &= \sigma x \\ (\lambda x.M')\sigma &= \lambda y.(M'\ \sigma[y/x]) && y \text{ not free in } M', \sigma \\ (M_1\ M_2)\sigma &= M_1\sigma\ M_2\sigma \end{aligned}$$

where

$$\sigma[N/y]\, x = \begin{cases} N & \text{if } x = y, \\ \sigma x & \text{otherwise} \end{cases}$$

Substituting N for x in M is now $M\ \iota[N/x]$ with ι the identity substitution.

Simultaneous Substitution in Coq

```
Fixpoint sim_subst (l:list (term*name)) (t:term)
  {struct t} : term :=
  match t with
  | Var x =>
      apply_subst l x
  | Abs x b =>
      let z := fresh_name
                 ((free_vars_sub l) ++ (free_vars b))
      in
      Abs z (sim_subst ((Var z, x)::l) b)
  | App f a =>
      App (sim_subst l f) (sim_subst l a)
end.

Definition subst' (t:term) (n:name) (t':term)
  : term := sim_subst ((t, n) :: nil) t'.
```

de Bruijn Indices 1/2

Represent variable occurrences by number of binders between itself and abstraction:

$$\begin{aligned} M ::= {} & n && \text{variable} \\ \mid {} & \lambda.M && \text{abstraction} \\ \mid {} & M\ M && \text{application} \end{aligned}$$

de Bruijn Indices 1/2

Represent variable occurrences by number of binders between itself and abstraction:

$$
\begin{aligned}
M ::= {} & n && \text{variable} \\
\mid {} & \lambda.M && \text{abstraction} \\
\mid {} & M\ M && \text{application}
\end{aligned}
$$

- Harder to read
- α-equivalence is term equivalence
- Substitution is structurally recursive
- No renaming (but lifting)
- Mechanically less tedious

de Bruijn Indices 2/2

Substitution can be defined as:

$$
\begin{aligned}
n[N/n] &= N \\
m[N/n] &= m && m \neq n \\
(\lambda.M')[N/n] &= \lambda.M'[\uparrow N/n{+}1] \\
(M_1\ M_2)[N/n] &= M_1[N/n]\ M_2[N/n]
\end{aligned}
$$

where $\uparrow M$ is M with all free variables incremented.

Term datatype:

```
Inductive term : Set :=
  | Var : nat -> term
  | Abs : term -> term
  | App : term -> term -> term.
```

Substitution in Coq

Substitution:

```
Fixpoint lift (l:nat) (t:term) {struct t} : term :=
  match t with
  | Var n   => Var (if le_lt_dec l n then (S n)
                      else n)
  | Abs u   => Abs (lift (S l) u)
  | App u v => App (lift l u) (lift l v)
end.

Fixpoint subst (t:term) (n:nat) (t':term)
  {struct t'} : term :=
  match t' with
  | Var m   => if eq_nat_dec n m then t else t'
  | Abs u   => Abs (subst (lift 0 t) (S n) u)
  | App u v => App (subst t n u) (subst t n v)
end.
```

Locally Nameless Representation 1/2

Combine names and numbers – names for free variables, de Bruijn indices for bound variables:

$M ::=$	x	free variable
$\mid$	n	bound variable
$\mid$	$\lambda.M$	abstraction
$\mid$	$M\ M$	application

Locally Nameless Representation 1/2

Combine names and numbers – names for free variables, de Bruijn indices for bound variables:

$$
\begin{array}{rll}
M ::= & x & \text{free variable} \\
\mid & n & \text{bound variable} \\
\mid & \lambda.M & \text{abstraction} \\
\mid & M\ M & \text{application}
\end{array}
$$

- α-equivalence is term equivalence
- No renaming, no lifting (but freshening)
- Substitution is structurally recursive

Locally Nameless Representation 2/2

Two substitution operations:

- Substitute a term for a named variable
- Substitute a term for a de Bruijn index

Locally Nameless Representation 2/2

Two substitution operations:

- Substitute a term for a named variable
- Substitute a term for a de Bruijn index

Named variable substitution:

$$
\begin{aligned}
x[N/x] &= N \\
y[N/x] &= y && x \neq y \\
n[N/x] &= n \\
(\lambda.M')[N/x] &= \lambda.M'[N/x] \\
(M_1\ M_2)[N/x] &= M_1[N/x]\ M_2[N/x]
\end{aligned}
$$

No renaming.

Locally Nameless Representation 2/2

Two substitution operations:

- Substitute a term for a named variable
- Substitute a term for a de Bruijn index

de Bruijn substitution:

$$\begin{aligned}
x[N/n] &= x \\
n[N/n] &= N \\
m[N/n] &= m && m \neq n \\
(\lambda.M')[N/n] &= \lambda.M'[N/n{+}1] \\
(M_1\ M_2)[N/n] &= M_1[N/n]\ M_2[N/n]
\end{aligned}$$

No lifting.

Locally Nameless Representation in Coq

Term datatype:

```
Inductive term : Set :=
  | FreeVar  : name -> term
  | BoundVar : nat -> term
  | Abs      : term -> term
  | App      : term -> term -> term.
```

Named Variable Substitution in Coq

Substitute a term for a named variable:

```
Fixpoint subst (t:term) (x:name) (t':term)
  {struct t'} : term :=
  match t' with
  | FreeVar y  => if eq_name x y then t else t'
  | BoundVar n => t'
  | Abs b      => Abs (subst t x b)
  | App f a    => App (subst t x f) (subst t x a)
end.
```

de Bruijn Substitution in Coq

Substitute a term for a de Bruijn index:

```
Fixpoint subst (t:term) (n:nat) (t':term)
  {struct t'} : term :=
  match t' with
  | FreeVar x  => t'
  | BoundVar m => if eq_nat_dec m n then t else t'
  | Abs b      => Abs (subst t (S n) b)
  | App f a    => App (subst t n f) (subst t n a)
end.
```

The POPLmark Challenge

Mechanized metatheory for the masses:

> *Your average POPL paper should include machine-checked proofs*

Set of benchmarks for measuring progress:

- Based on metatheory of System $F_{<:}$
- Binding issues are a central aspect
- 15 (partial) solutions
- Part 1a: transitivity of subtyping

Syntax of System $F_{<:}$

Part 1a considers just the type language of $F_{<:}$.

$T ::= X$	type variable
$\mid \texttt{Top}$	maximum type
$\mid T \to T$	type of functions
$\mid \forall X{<:}T.T$	universal type

$\Gamma ::= \emptyset$	empty type environment
$\mid \Gamma, X{<:}T$	type variable binding

Subtyping Rules of System $F_{<:}$

$$\frac{}{\Gamma \vdash S <: \mathtt{Top}} \text{ SA-Top} \qquad \frac{}{\Gamma \vdash X <: X} \text{ SA-Refl-TVar}$$

$$\frac{X <: U \in \Gamma \qquad \Gamma \vdash U <: T}{\Gamma \vdash X <: T} \text{ SA-Trans-TVar}$$

$$\frac{\Gamma \vdash T_1 <: S_1 \qquad \Gamma \vdash S_2 <: T_2}{\Gamma \vdash S_1 \to S_2 <: T_1 \to T_2} \text{ SA-Arrow}$$

$$\frac{\Gamma \vdash T_1 <: S_1 \qquad \Gamma, X <: T_1 \vdash S_2 <: T_2}{\Gamma \vdash \forall X <: S_1 . S_2 <: \forall X <: T_1 . T_2} \text{ SA-All}$$

POPLmark 1a: transitivity of $<:$

Solutions in Coq

Named variables:

- Stump

de Bruijn indices:

- Vouillon
- Sallinens
- Chargéraud

Locally nameless:

- Leroy
- Chlipala
- Charguéraud

Nested datatypes:

- Hirschowitz and Maggesi

Solutions in Coq

Named variables:

- **Stump** (7641)

de Bruijn indices:

- **Vouillon** (5443)
- Sallinens (unavailable)
- Chargéraud (3727)

Locally nameless:

- **Leroy** (1081+5414=6495)
- Chlipala (2650+2400=5050)
- Charguéraud (803+3533=4336)

Nested datatypes:

- Hirschowitz and Maggesi (2757)

Stump – Named Variables

Two main techniques to avoid difficulties with named variables:

1. Free and bound variables are disjoint
   - Substitution is just grafting

Stump – Named Variables

Two main techniques to avoid difficulties with named variables:

1. Free and bound variables are disjoint
   - Substitution is just grafting
2. Use common bound variable in the bodies of the SA-All rule
   - Avoid α-equivalence issues
   - Original rule:

$$\frac{\Gamma \vdash T_1 <: S_1 \qquad \Gamma, X <: T_1 \vdash S_2 <: T_2}{\Gamma \vdash \forall X <: S_1 . S_2 <: \forall X <: T_1 . T_2} \text{ SA-All}$$

Stump – Named Variables

Two main techniques to avoid difficulties with named variables:

1. Free and bound variables are disjoint
   - Substitution is just grafting
2. Use common bound variable in the bodies of the SA-All rule
   - Avoid α-equivalence issues
   - Original rule:

$$\frac{\Gamma \vdash T_1 <: S_1 \qquad \Gamma, X <: T_1 \vdash S_2 <: T_2}{\Gamma \vdash \forall X <: S_1 . S_2 <: \forall X <: T_1 . T_2} \text{ SA-All}$$

   - Adapted rule:

$$\frac{\Gamma \vdash T_1 <: S_1 \qquad \Gamma, X <: T_1 \vdash S_2[X/X_1] <: T_2[X/X_2]}{\Gamma \vdash \forall X_1 <: S_1 . S_2 <: \forall X_2 <: T_1 . T_2} \text{ SA-All}$$

Vouillon – de Bruijn Indices

- Very clear implementation, even suggested as baseline by POPLmark team
- A lot of code deals with shifting (but straightforward)
- Proofs not by structural induction, but by induction on the size of types
- Narrowing and transitivity are proved separately
- Unfortunately no accompanying paper

Leroy – Locally Nameless Representation

- Two substitution operations, no renaming or lifting
- Considering abstraction bodies, freshening is needed
- Proofs not by structural induction, but by induction on the size of types
- A lot of code deals with swaps (used for equivariance proofs)
- Room for improvements, some implemented by Charguéraud
  - Useless case in de Bruijn substitution
  - Treat well-formed typing environments as sets
  - Cofinite quantification of free variable in SA-All
  - Proofs by induction on well-formdness derivation instead of size

Engineering Formal Metatheory

Aydemir et al, POPL'08: *Engineering Formal Metatheory*

Complete style for formalizing language metatheory:

- Building on experience from POPLmark solutions
- Locally nameless representation
- Cofinite quantification of free variables in inductive definitions of relations on terms

Implemented in this style:

- Parts of POPLmark challenge
- Type soundness for core ML
- Subject reduction for Calculus of Constructions
- Several small developments

LNgen

Aydemir and Weirich, this Wednesday:

> *LNgen: Tool Support for Locally Nameless Representations*

Building on *Engineering Formal Metatheory*:

- Takes Ott-like specifications
- Generates locally nameless infrastructure for Coq

Names and Numbers in Binding

Related Work

Other Representations

Nominal Representation

- Based on nominal logic (Pitts, Gabbay)
- Names for variables
- Swapping as primitive
- Urban in Isabelle/HOL

Higher Order Abstract Syntax

- Meta-variables for variables
- Meta-functions for functions
- α-equivalence for free
- No renaming needed
- Leads to quite unusual formulations

Conclusion 1/2

Named variables:

- Closest to paper, but too much trouble mechanically

Conclusion 1/2

Named variables:

- Closest to paper, but too much trouble mechanically

de Bruijn indices:

- Good candidate for mechanical developments
- Involves a lot of ‘easy’ work
- Not so easy to read

Conclusion 1/2

Named variables:

- Closest to paper, but too much trouble mechanically

de Bruijn indices:

- Good candidate for mechanical developments
- Involves a lot of ‘easy’ work
- Not so easy to read

Locally nameless representation:

- Improves on de Bruijn
- Still quite some boilerplate code
- Tools like LNgen might make it feasible

Conclusion 1/2

Named variables:

- Closest to paper, but too much trouble mechanically

de Bruijn indices:

- Good candidate for mechanical developments
- Involves a lot of ‘easy’ work
- Not so easy to read

Locally nameless representation:

- Improves on de Bruijn
- Still quite some boilerplate code
- Tools like LNgen might make it feasible

Nominal approaches are promising

Conclusion 2/2

Of course, it also depends on your goal:

- A language implementation might not need to be close to its concrete syntax
- Metatheory for many users on the other hand probably does

. . . and on your environment:

- We focussed on Coq
- There are other tools

Questions and Further Reading

Questions?

Literature

- de Bruijn, 1972: *λ-calculus with nameless dummies*
- Stoughton, 1988: *Substitution revisited*
- McBride and McKinna, 2004: *I am not a number – I am a free variable*
- Aydemir et al, 2005: *The POPLmark challenge*
- Pollack, 2006: *Reasoning about languages with binding*
- Aydemir et al, 2008: *Engineering formal metatheory*