# Nominal Logic and its Isabelle Incarnation

Christian Doczkal

Advisor: Jan Schwinghammer

November 28, 2008

# Motivation

We thank T. Thacher Robinson for showing us on August 19, 1962 by a counterexample the existence of an error in our handling of bound variables.

— S. C. Kleene

# Motivation

Some standard sentences when doing proofs about ASTs

"We identify terms up to $\alpha$-equivalence, i.e. $\lambda x.x = \lambda y.y$"

Barendregt Variable Convention: "Choose a representative parse tree whose bound variables are *fresh*, i.e mutually distinct and distinct from any free variables in the current context "

*Implicit assumption: All constructions and predicates and proofs are independent of the names chosen for bound variables.*

# Motivation

Some standard sentences when doing proofs about ASTs

“We identify terms up to $\alpha$-equivalence, i.e. $\lambda x.x = \lambda y.y$”

Barendregt Variable Convention: “Choose a representative parse tree whose bound variables are *fresh*, i.e mutually distinct and distinct from any free variables in the current context ”

*Implicit assumption: All constructions and predicates and proofs are independent of the names chosen for bound variables.*

# Examples

$$
\begin{aligned}
x[y := t'] &= \mathit{if}\ \ x = y\ \ \mathit{then}\ \ t'\ \ \mathit{else}\ \ x \\
(t_1\ t_2)[y := t'] &= (t_1[y := t'])\,(t_2[y := t']) \\
(\lambda x.t)[y := t'] &= \lambda x.t[y := t'])\ \ \mathit{where}\ \ x \neq y\ \ \mathit{and}\ \ x \notin \mathit{fv}(t')
\end{aligned}
$$

Is total when defined over $\Lambda_{/=_\alpha}$ but partial when defined over $\Lambda$

$$
\begin{aligned}
\mathit{ist}\ x &= \emptyset \\
\mathit{ist}(t_1\ t_2) &= \{t_1, t_2\} \\
\mathit{ist}(\lambda x.t) &= \{t\}
\end{aligned}
$$

Is inconsistent when defined over $\Lambda_{/=_\alpha}$ but fine over $\Lambda$

# Outline

- Common definitions of Nominal Logic
  - Atoms
  - Permutations
  - Support
- Differences between various approaches
  - FO-Nominal Logic / FM-HOL
  - HOL-Nominal
- Specifics of Isabelle/HOL-Nominal
  - Features
  - Limitations

# Atoms

## Definition (Atoms)

We fix some family $(\mathbb{A}_n \mid n \in \mathbb{N})$ of atom sorts where:

$$\forall n, n' : n \neq n' \Rightarrow \mathbb{A}_n \cap \mathbb{A}_{n'} = \emptyset \quad \wedge \quad \forall n : \mathbb{A}_n \cong \mathbb{N}$$

$$\mathbb{A} = \bigcup_{n \in \mathbb{N}} \mathbb{A}_n$$

# Atom Permutations and Actions

## Definition (Perm)

Let *Perm* be the set of all *finite*, *sort respecting*, **atom-permutations**. Thus $(Perm, \circ)$ is a group with unit element $\iota$ generated by the set of all transpositions $(a\, a')$

## Definition (Action)

An **action** of *Perm* on a set $X$ is a function $\cdot \in Perm \times X \longrightarrow X$ satisfying:

$$\iota \cdot x = x$$
$$\pi \cdot (\pi' \cdot x) = (\pi \circ \pi') \cdot x$$

# Some actions of *Perm*

**Lemma**

*Given actions of Perm on $\alpha$ and $\beta$ the following are also actions of Perm.*

$$\begin{aligned}
\mathbb{A} &: \quad \pi \cdot a = \pi\, a \\
\mathit{bool} &: \quad \pi \cdot b = b \\
\mathit{unit} &: \quad \pi \cdot () = () \\
\alpha \times \beta &: \quad \pi \cdot (x_1, x_2) = (\pi \cdot x_1, \pi \cdot x_2) \\
\alpha\ \mathit{set} &: \quad \pi \cdot X = \{\pi \cdot x \mid x \in X\} \\
\alpha \rightarrow \beta &: \quad \pi \cdot f = \lambda x. \pi \cdot (f(\pi^{-1} \cdot x)) \\
\alpha\ \mathit{list} &: \quad \pi \cdot [\,] = [\,] \quad \mathit{and} \quad \pi \cdot (x :: t) = (\pi \cdot x) :: (\pi \cdot t)
\end{aligned}$$

# Some actions of *Perm*

**Lemma**

*Given actions of Perm on $\alpha$ and $\beta$ the following are also actions of Perm.*

$$\begin{aligned}
\mathbb{A}: &\quad \pi \cdot a = \pi\, a \\
bool: &\quad \pi \cdot b = b \\
unit: &\quad \pi \cdot () = () \\
\alpha \times \beta: &\quad \pi \cdot (x_1, x_2) = (\pi \cdot x_1, \pi \cdot x_2) \\
\alpha\ set: &\quad \pi \cdot X = \{\pi \cdot x \mid x \in X\} \\
\alpha \rightarrow \beta: &\quad \pi \cdot (f\ x) = (\pi \cdot f)\ (\pi \cdot x) \\
\alpha\ list: &\quad \pi \cdot [] = [] \quad and \quad \pi \cdot (x :: t) = (\pi \cdot x) :: (\pi \cdot t)
\end{aligned}$$

# Example - $\lambda$-calculus

$$x \in \mathbb{A}_0$$
$$t ::= x \mid t\,t \mid \lambda x.t$$

$$\pi \cdot x = \pi\, x$$
$$\pi \cdot (s\,t) = (\pi \cdot s)\,(\pi \cdot t)$$
$$\pi \cdot (\lambda x.t) = \lambda(\pi \cdot x).\pi \cdot t$$

# Support and Freshness

## Definition (Support)

The support of $x$ is defined as:

$$\mathrm{supp}(x) \equiv \{a \mid \mathrm{infinite}\{b \mid (a\, b) \cdot x \neq x\}\}$$

## Definition (Freshness)

$a \sharp x \equiv a \notin supp(x)$

# Example - $\lambda$-calculus

$$\mathrm{supp}(x) \equiv \{a \mid \mathrm{infinite}\{b \mid (a\,b) \cdot x \neq x\}\}$$

$$\begin{aligned}
\pi \cdot x &= \pi\, x \\
\pi \cdot (s\, t) &= (\pi \cdot s)\,(\pi \cdot t) \\
\pi \cdot (\lambda x.t) &= \lambda(\pi \cdot x).\pi \cdot t
\end{aligned}$$

$$\begin{aligned}
\mathrm{supp}(x) &= \{x\} \\
\mathrm{supp}(s\, t) &= \mathrm{supp}(s) \cup \mathrm{supp}(t) \\
\mathrm{supp}(\lambda x.t) &= \mathrm{supp}(t) \cup \{x\} \;\; \textit{for}\;\; \Lambda
\end{aligned}$$

# Example - $\lambda$-calculus

$$\mathrm{supp}(x) \equiv \{a \mid \mathrm{infinite}\{b \mid (a\,b) \cdot x \neq x\}\}$$

$$\begin{aligned} \pi \cdot x &= \pi\, x \\ \pi \cdot (s\, t) &= (\pi \cdot s)\,(\pi \cdot t) \\ \pi \cdot (\lambda x.t) &= \lambda(\pi \cdot x).\pi \cdot t \end{aligned}$$

$$\begin{aligned} \mathrm{supp}(x) &= \{x\} \\ \mathrm{supp}(s\, t) &= \mathrm{supp}(s) \cup \mathrm{supp}(t) \\ \mathrm{supp}(\lambda x.t) &= \mathrm{supp}(t) - \{x\} \;\; \textit{for}\;\; \Lambda_{/=_\alpha} \end{aligned}$$

# Support and Freshness

## Definition (Supports)

$S$ supports $x \equiv \forall a, a' \notin S : (a\,a') \cdot x = x$

## Lemma

- $\text{finite}(\text{supp}(x)) \Rightarrow \exists a : a \sharp x$
- $\forall a, a' : a \sharp x \wedge a' \sharp x \Rightarrow (a\,a') \cdot x = x$

  $\Rightarrow$ $\text{supp}(x)$ supports $x$
- $\text{finite}\, S \wedge S$ supports $x \Rightarrow \text{supp}(x) \subseteq S$

# Support and Freshness

## Definition (Supports)

$S$ supports $x \equiv \forall a, a' \notin S : (a\,a') \cdot x = x$

## Lemma

- $\text{finite}(\text{supp}(x)) \Rightarrow \exists a : a \sharp x$
- $\forall a, a' : a \sharp x \wedge a' \sharp x \Rightarrow (a\,a') \cdot x = x$

  $\Rightarrow$ $\text{supp}(x)$ supports $x$
- $S$ supports $x \not\Rightarrow \text{supp}(x) \subseteq S$

# Example Support

Want to show: $S$ supports $x \not\Rightarrow \mathrm{supp}(x) \subseteq S$
Remember:

$$\forall n : \mathbb{A}_n \cong \mathbb{N}$$

Consider some $\mathbb{A}_n = EVEN \uplus ODD$ using the isomorphism to $\mathbb{N}$

We have $ODD$ supports $EVEN$

We also have $\mathrm{supp}(EVEN) = \mathbb{A}_n$

Thus $ODD$ supports $EVEN$ but $\mathrm{supp}(EVEN) = \mathbb{A}_n \not\subseteq ODD$

# Example Support

Want to show: $S$ supports $x \not\Rightarrow \text{supp}(x) \subseteq S$
Remember:

$$S \text{ supports } x \equiv \forall a, a' \notin S : (a\,a') \cdot x = x$$

Consider some $\mathbb{A}_n = EVEN \uplus ODD$ using the isomorphism to $\mathbb{N}$

We have $ODD$ supports $EVEN$

We also have $\text{supp}(EVEN) = \mathbb{A}_n$

Thus $ODD$ supports $EVEN$ but $\text{supp}(EVEN) = \mathbb{A}_n \not\subseteq ODD$

# Example Support

Want to show: $S$ supports $x \not\Rightarrow \mathrm{supp}(x) \subseteq S$
Remember:

$$\mathrm{supp}(z) \equiv \{a \mid \mathrm{infinite}\{b \mid (a\,b) \cdot z \neq z\}\}$$

Consider some $\mathbb{A}_n = EVEN \uplus ODD$ using the isomorphism to $\mathbb{N}$

We have $ODD$ supports $EVEN$

We also have $\mathrm{supp}(EVEN) = \mathbb{A}_n$

Thus $ODD$ supports $EVEN$ but $\mathrm{supp}(EVEN) = \mathbb{A}_n \not\subseteq ODD$

# Example Support

Want to show: $S$ supports $x \not\Rightarrow \text{supp}(x) \subseteq S$
Remember:

$$\text{supp}(z) \equiv \{a \mid \text{infinite}\{b \mid (a\, b) \cdot z \neq z\}\}$$

Consider some $\mathbb{A}_n = \mathit{EVEN} \uplus \mathit{ODD}$ using the isomorphism to $\mathbb{N}$

We have $\mathit{ODD}$ supports $\mathit{EVEN}$

We also have $\text{supp}(\mathit{EVEN}) = \mathbb{A}_n$

Thus $\mathit{ODD}$ supports $\mathit{EVEN}$ but $\text{supp}(\mathit{EVEN}) = \mathbb{A}_n \not\subseteq \mathit{ODD}$

# Nominal Sets

## Definition

A *nominal set* is a set $X$ together with an action of *Perm* such that

$$\forall x \in X : \text{finite}(\text{supp}\, x))$$

## Lemma

*Given nominal sets* $\alpha$ *and* $\beta$ *then* $\alpha$ *list*, $\alpha \times \beta$, $\mathbb{A}$, *bool and unit are also nominal sets using the actions defined previously*

# Approaches

There are (at least) two approaches to dealing with finite support

- Build a new logic and axiomatize everything to have finite support
  - Nominal Logic: A First Order Theory of Names and Binding [Pitts 2001]
  - FM-HOL, A Higher Order Theory of Names [Gabbay 2002]
  - Models in the FM set theory
- Work in ordinary HOL and prove finite support whenever needed.
  - Alpha Structural Recursion and Induction [Pitts 2006]
  - Nominal Techniques in Isabelle/HOL [Urban 2007]

# Approaches

Pitts' axioms for FO-Nominal Logic include equivariance:

$$(\forall a, a' : A)(\forall \vec{x} : \vec{S})\ (a\, a') \cdot f(\vec{x}) = f((a\, a') \cdot \vec{x})$$

$$(\forall a, a' : A)(\forall \vec{x} : \vec{S})\ R(\vec{x}) \Rightarrow R((a\, a') \cdot \vec{x})$$

**Theorem (Finite Support Principle)**

*Any function or relation that is defined from finitely supported functions and relations using higher-order logic is itself finitely supported.*

# Approaches

There are (at least) two approaches to dealing with finite support

- Build a new logic and axiomatize everything to have finite support
  - Nominal Logic: A First Order Theory of Names and Binding [Pitts 2001]
  - FM-HOL, A Higher Order Theory of Names [Gabbay 2002]
  - Models in the FM set theory
- Work in ordinary HOL and prove finite support whenever needed.
  - Alpha Structural Recursion and Induction [Pitts 2006]
  - Nominal Techniques in Isabelle/HOL [Urban 2007]

# Approaches

There is no finitely supported function $choose : (\mathbb{A} \rightarrow_{fs} bool) \rightarrow \mathbb{A}$ satisfying

$$\exists a.f(a) \Rightarrow f(choose(f))$$

# Approaches

There are (at least) two approaches to dealing with finite support

- Build a new logic and axiomatize everything to have finite support
    - Nominal Logic: A First Order Theory of Names and Binding [Pitts 2001]
    - FM-HOL, A Higher Order Theory of Names [Gabbay 2002]
    - Models in the FM set theory
- Work in ordinary HOL and prove finite support whenever needed.
    - Alpha Structural Recursion and Induction [Pitts 2006]
    - Nominal Techniques in Isabelle/HOL [Urban 2007]

# HOL-Nominal

- Work in ordinary higher-order logic
- Make only definitional extensions to HOL
  $\Rightarrow$ no soundness argument required
- Compatible with choice
- Implementation provides `nominal_datatype` declaration with
  - Built in $\alpha$-equivalence
  - Permutation operation - finite support
  - Strong induction principles
  - Primitive recursion operators - with freshness conditions

# Type Classes and Finite Support

Assume we have: `atom_decl name`

HOL-Nominal provides ...

- a **type class** `pt_name` of permutation types
  - types with an action of perm.
- a **type class** `fs_name` of finitely supported types
  - representing nominal sets.
- **instance declarations** of all types obtainable by the lemmas above including types declared by `nominal_datatype`

# Nominal Datatypes

```
atom_decl name
nominal_datatype lam =
    Var "name"
  | App "lam" "lam"
  | Lam "<<name>>lam"
```

# Nominal Datatypes

```
atom_decl name
datatype plam =
    PVar "name"
  | PApp "plam" "plam"
  | PLam "name => plam option"
```

Restrict to:

$$[a].t \equiv \lambda b.if\ \ a = b\ then\ Some(t)$$
$$else\ \ if\ \ b \sharp t\ then\ Some((a\,b)\cdot t)\ else\ None$$

# Nominal Datatypes

```
atom_decl name
datatype plam =
    PVar "name"
  | PApp "plam" "plam"
  | PLam "name => plam option"
```

Restrict to:

$$[a].t \equiv \lambda b. \mathit{if}\ \ a = b\ \mathit{then}\ \mathit{Some}(t)$$
$$\mathit{else}\ \mathit{if}\ \ b \sharp t\ \mathit{then}\ \mathit{Some}((a\, b) \cdot t)\ \mathit{else}\ \mathit{None}$$

representing $\alpha$-equivalence classes

# Nominal Datatypes

```
atom_decl name
datatype plam =
    PVar "name"
  | PApp "plam" "plam"
  | PLam "name => plam option"
```

Restrict to:

$$[a].t \equiv \lambda b. \mathit{if}\ \ a = b\ \ \mathit{then}\ \ \mathit{Some}(t)$$
$$\mathit{else}\ \ \mathit{if}\ \ b \sharp t\ \ \mathit{then}\ \ \mathit{Some}((a\,b)\cdot t)\ \ \mathit{else}\ \ \mathit{None}$$

$$[a].s = [b].t \iff a = b \wedge s = t \quad \vee \quad a \neq b \wedge s = (a\,b)\cdot t \wedge a \sharp t$$

# Strong Induction

The `nominal_datatype` declaration provides:

$$\frac{\begin{array}{c}\forall c\ a.\ P\ (Var\ a)\ c \\ \forall c\ s\ t.\ (\forall d. P\ s\ d) \wedge (\forall d. P\ t\ d) \Rightarrow P\ (App\ s\ t)\ c \\ \forall c\ a\ t.\ a \sharp c \wedge (\forall d. P\ t\ d) \Rightarrow P\ (Lam\ a\ t)\ c\end{array}}{P\ t\ c}$$

where $a :: name$, $s, t :: lam$ and $c :: \alpha :: fs\_name$

Common instantiation: P is the theorem to prove with all free variables (except t) abstracted into c

# Strong Induction

The nominal_datatype declaration provides:

$$\frac{\begin{array}{c}\forall c\ a.\ P\ (Var\ a)\ c \\ \forall c\ s\ t.\ (\forall d.P\ s\ d) \wedge (\forall d.P\ t\ d) \Rightarrow P\ (App\ s\ t)\ c \\ \forall c\ a\ t.\ a \sharp c \wedge (\forall d.P\ t\ d) \Rightarrow P\ (Lam\ a\ t)\ c\end{array}}{P\ t\ c}$$

where $a :: name$, $s, t :: lam$ and $c :: \alpha :: fs\_name$

```
proof (nominal_induct t avoiding: x t'
       rule: lam.strong_induct)
```

# Restrictions

- no function types in `nominal_datatype` declarations
- only one type of atom abstraction is allowed
- no nested recursion - has to be unwinded "by hand"
- no support for non-primitive recursion - one needs to prove pattern completeness, functionality, and termination "by hand"

# References

- C. Urban, Nominal Techniques in Isabelle/HOL, Journal of Automatic Reasoning, Vol. 40(4), pap: 327-356, 2008
- A. Pitts, Nominal Logic: A First Order Theory of Names and Binding, LNCS Vol. 2215, pp: 219-242, Springer Verlag, 2001
- A. Pitts Alpha-Structural Recursion and Induction. Journal of the ACM, Volume 53, Issue 3, pp: 459 - 506, 2006

# Thank You!

# Example - Weakening

Weakening : $\Gamma \vdash [t]_\alpha : \tau \Rightarrow \forall a' \notin \operatorname{dom}\Gamma.\ \Gamma, a' : \tau' \vdash [t]_\alpha : \tau$

Proof by 'rule induction' - case: $\dfrac{\Gamma, a : \tau_1 \vdash [t]_\alpha : \tau_2 \qquad a \notin \operatorname{dom}\Gamma}{\Gamma \vdash [\lambda a.t]_\alpha : \tau_1 \to \tau_2}$

Given weakening on premise $\Gamma, a : \tau_1 \vdash [t]_\alpha : \tau_2$ show:

For *all* $a' \notin \operatorname{dom}\Gamma$ we have $\Gamma, a' : \tau' \vdash [\lambda a.t]_\alpha : \tau_1 \to \tau_2$

Problematic case $a = a'$ : Cannot weaken the premise to $\Gamma, a : \tau_1, a' : \tau' \vdash [t]_\alpha : \tau_2$ without renaming beforehand.

Need *equivariance* of the typing relation