Logic Programming with Names and Binding

James Cheney

September 28, 2004

Prologue

Gabbay and Pitts (1999)

- Developed a new theory of names, binding, and α-equivalence based on *swapping* (permutations, FM-set theory)

- *Nominal logic*: variant of first-order logic incorporating these ideas

- I call their approach "nominal abstract syntax" for short.

- Often asked: Why *permutations* instead of good old capture-avoiding substitution?

McKinna and Pollack (1993,1999)

- Formalized reasoning about the λ-calculus in LEGO.

- Along the way:

  – Principle that fresh parameters can always be chosen

  – Inductive definition of α-equivalence

  – Quantifier switching:

  $V closed(\lambda x.t) \iff \forall p.V closed(t[p/x]) \iff \exists p.V closed(t[p/x])$

  – *Invertible renamings* built up out of $\{x/y, y/x\}$

Frege (1879)

- Frege (Begriffsschift 1879) wrote:

  > ... Replacing a German letter [bound name] everywhere in its scope by some other one is, of course, permitted, so long as in places where different letters initially stood different ones also stand afterward. This has no effect on the content.

- Thus, he viewed formulas as invariant under *one-to-one re-namings* (and hence, also *permutations*) of bound names.

My view

Nominal abstract syntax is a *new and simpler way of looking at* reasoning about names and binding. Arguably, the techniques themselves are not new. But they *are* underutilized.

I am interested in applying nominal abstract syntax to real problems in programming and formal reasoning.

Long term goal: Better *logical frameworks* for reasoning about logics and programming languages.

First step: I (and others) have developed αProlog, a logic programming language based on nominal abstract syntax.

Outline

- Overview of nominal logic
- αProlog programming examples
- How it works
- What doesn't work (yet)
- Conclusion

Nominal Logic

Nominal Logic: Syntax

- Names $\mathfrak{a}, \mathfrak{b}$ inhabiting name-sorts $\mathfrak{A}, \mathfrak{A}'$
- Swapping $\mathfrak{a}, \mathfrak{b}, x \mapsto (\mathfrak{a}\ \mathfrak{b}) \cdot x : \mathfrak{A}, \mathfrak{A}, S \to S$ exchanges two names
- Abstraction $\mathfrak{a}, x \mapsto \langle \mathfrak{a} \rangle x : \mathfrak{A}, S \to \langle \mathfrak{A} \rangle S$ used for object-level binding
- Freshness relation $\mathfrak{a} \# x$ means "x does not depend on $\mathfrak{a}$"
- И-quantifier quantifies over fresh names: И$\mathfrak{a}.\phi$ means "for fresh names $\mathfrak{a}$, ϕ holds"

Names: What are they?

- In my approach, names are a new syntactic class, distinct from variables and from function or constant symbols

- Syntactically different names are also semantically distinct

- Names can be used in object terms denoting binding: $\langle \mathfrak{a} \rangle x$, but they can also be "bound" at the metalevel: $\text{И}\mathfrak{a}.\phi$

- $\langle \mathfrak{a} \rangle \mathfrak{a} = \langle \mathfrak{b} \rangle \mathfrak{b}$ is a (true) *formula* of nominal logic, while $\text{И}\mathfrak{a}.p(\mathfrak{a})$ and $\text{И}\mathfrak{b}.p(\mathfrak{b})$ are α-equivalent formulas in the conventional way.

Theory of Swapping and Freshness

- Swapping

$$(\mathfrak{a}\ \mathfrak{b}) \cdot \mathfrak{a} = \mathfrak{b} \quad (\mathfrak{a}\ \mathfrak{a}) \cdot x = x \quad (\mathfrak{a}\ \mathfrak{b}) \cdot (\mathfrak{a}\ \mathfrak{b}) \cdot x = x$$

- Freshness

$$\mathfrak{a} \# \mathfrak{a}' \iff \mathfrak{a} \neq \mathfrak{a}' \quad \mathfrak{a} \# x \wedge \mathfrak{b} \# x \supset (\mathfrak{a}\ \mathfrak{b}) \cdot x = x$$

- Examples

$$\mathfrak{a} \# (\mathfrak{a}\ \mathfrak{b}) \cdot \mathfrak{a} \quad (\mathfrak{a}\ \mathfrak{b}) \cdot f(\mathfrak{a}, \mathfrak{b}, \mathfrak{a}, g(\mathfrak{a})) = f(\mathfrak{b}, \mathfrak{a}, \mathfrak{b}, g(\mathfrak{b}))$$

Theory of Name-Abstraction

- Intuitively, $\langle\mathfrak{a}\rangle x$ is "the value x with a distinguished bound name $\mathfrak{a}$".

- Considered equal up to "safe" renaming:

$$\langle a\rangle x = \langle b\rangle x \iff (a = b \land x = y) \lor (a \# y \land x = (a\ b) \cdot y)$$

- For example,

$$\langle\mathfrak{a}\rangle\mathfrak{a} = \langle\mathfrak{b}\rangle\mathfrak{b} \qquad \langle\mathfrak{a}\rangle(\mathfrak{a},\mathfrak{b}) \neq \langle\mathfrak{b}\rangle(\mathfrak{b},\mathfrak{a})$$

Freshness and Equivariance Principles

- Freshness: Fresh names can always be chosen.

$$\forall \vec{x}.\exists a.a \# \vec{x}$$

- Equivariance: Truth preserved by name-swapping

$$\forall \vec{x}.\forall a, b.p(\vec{x}) \supset p((a\ b) \cdot x)$$

- Also, constants and function symbols preserved by swapping

$$\forall a, b.(a\ b) \cdot c = c \quad \forall \vec{x}.\forall a, b.f((a\ b) \cdot x) = (a\ b) \cdot f(x)$$

$\reflectbox{N}$-Quantifier

- Originally defined as

$$\reflectbox{N}\mathfrak{a}.\phi(\mathfrak{a},\vec{x}) \iff \exists\mathfrak{a}.\mathfrak{a}\ \#\ \vec{x} \wedge \phi(\mathfrak{a},\vec{x})$$

- But equivalent (using freshness, equivariance) to

$$\forall\mathfrak{a}.\mathfrak{a}\ \#\ \vec{x} \supset \phi(\mathfrak{a},\vec{x})$$

- Examples

$$\reflectbox{N}\mathfrak{a},\mathfrak{b}.\mathfrak{a}\ \#\ \mathfrak{b} \qquad \reflectbox{N}\mathfrak{a},\mathfrak{b}.\phi(\mathfrak{a},\mathfrak{b}) \iff \reflectbox{N}\mathfrak{a},\mathfrak{b}.\phi(\mathfrak{b},\mathfrak{a})$$

$$\reflectbox{N}\mathfrak{a}.\phi(\mathfrak{a},\mathfrak{a}) \not\subset\not\supset \reflectbox{N}\mathfrak{a},\mathfrak{b}.\phi(\mathfrak{b},\mathfrak{a})$$

Sequent Calculus

- Judgments use name/variable context Σ expressing both typing and freshness information

$$\Sigma ::= \cdot \mid \Sigma, x : S \mid \Sigma \# \mathfrak{a} : \mathfrak{A}$$

Intuitively, $\Sigma \# \mathfrak{a}$ is equivalent to $\mathfrak{a} \# \vec{x}$ where $\vec{x} = FV(\Sigma)$.

- Freshness principle restated as:

$$\frac{\Sigma \# \mathfrak{a} : \Gamma \Rightarrow C}{\Sigma : \Gamma \Rightarrow C}$$

- Convenient direct proof rules for И:

$$\frac{\Sigma \# \mathfrak{a} : \Gamma \Rightarrow C}{\Sigma : \Gamma \Rightarrow \text{И}\mathfrak{a}.C} \quad \frac{\Sigma \# \mathfrak{a} : \Gamma, A \Rightarrow C}{\Sigma : \Gamma, \text{И}\mathfrak{a}.A \Rightarrow C}$$

Nominal Logic Programming in αProlog

Nominal Logic Programming (Horn clauses)

- Written Prolog-style as

$$A :- B_1, \ldots, B_n.$$

where $A, \vec{B}$ are atomic formulas involving nominal terms.

- We interpret such clauses as NL formulas

$$\text{И}\vec{\mathsf{a}}.\forall\vec{x}.B_1 \wedge \cdots \wedge B_n \supset A$$

- Implementation: αProlog

Some interesting programs I

- Typechecking the λ-calculus

$$\frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau} \quad \frac{\Gamma \vdash e_1 : \sigma \to \tau \quad \Gamma \vdash e_2 : \sigma}{\Gamma \vdash e_1\ e_2 : \tau}$$

$$\frac{\Gamma, x : \tau \vdash e : \sigma \quad (x \notin FV(\Gamma))}{\Gamma \vdash \lambda x.e_1 : \tau \to \sigma}$$

$$\begin{array}{lcl}
tc(G, var(X), T) & :- & mem((X, T), G). \\
tc(G, app(E, E'), T) & :- & tc(G, E, arr(T', T)), tc(G, E', T'). \\
tc(G, lam(\langle \mathfrak{a} \rangle E), arr(T, T')) & :- & \mathfrak{a} \mathrel{\#} G, tc([(\mathfrak{a}, T)|G], E, T').
\end{array}$$

Some interesting programs II

- Substitution in the λ-calculus

$$
\begin{array}{lcll}
x[t/x] & = & t & \\
y[t/x] & = & y & (y \neq x) \\
(e_1\ e_2)[t/x] & = & e_1[t/x]\ e_2[t/x] & \\
(\lambda y.e)[t/x] & = & \lambda y.(e[t/x]) & (y \neq x, y \notin FV(t))
\end{array}
$$

$$
\begin{array}{lcl}
subst(var(\mathfrak{a}), T, \mathfrak{a}) & = & T. \\
subst(var(\mathfrak{b}), T, \mathfrak{a}) & = & var(\mathfrak{b}). \\
subst(app(E_1, E_2), T, \mathfrak{a}) & = & app(subst(E_1', T, \mathfrak{a}), subst(E_2', T, \mathfrak{a})). \\
subst(lam(\langle\mathfrak{b}\rangle E), T, \mathfrak{a}) & = & lam(\langle\mathfrak{b}\rangle subst(E, T, \mathfrak{a})) \\
 & :- & \mathfrak{b} \# T.
\end{array}
$$

Some interesting programs III

- Labeled transitions in the π-calculus (selected transitions)

$$\frac{p \xrightarrow{\alpha} p' \quad bn(\alpha) \cap fn(q) = \emptyset}{p|q \xrightarrow{\alpha} p'} \qquad \frac{}{\bar{x}y.p \xrightarrow{\bar{x}y} p} \qquad \frac{p \xrightarrow{x(a)} p' \quad q \xrightarrow{\bar{x}(a)} q'}{p|q \xrightarrow{\tau} \nu a.(p'|q')}$$

$$\begin{array}{l} step(par(P,Q), A, P') \\ \quad :- \ step(P, A, P'), safe(A, Q). \\ step(out(X, Y, P), fout\_a(X, Y), P). \\ step(par(P, Q), tau\_a, res(\langle \mathfrak{a} \rangle par(P', Q'))) \\ \quad :- \ step(P, in\_a(X, \mathfrak{a}), P'), step(Q, bout\_a(X, \mathfrak{a}), Q'). \end{array}$$

Example queries

- (translated to human readable forms)

- $\cdot \vdash \lambda x.\lambda x.x : T$ solves $T = \alpha \to \beta \to \beta$ (unique answer)

- $(\lambda x.y)[x/y] = \lambda x'.x$ (unique answer modulo α-equiv)

- $p = (\nu y.(\bar{x}y.0))|(x(z).\bar{z}x.0)$ has three transitions:

$$p \xrightarrow{\bar{x}(w)} 0|(x(z).\bar{z}x.0), x \neq w$$
$$p \xrightarrow{x(w)} (\nu y.\bar{x}y.0)|(\bar{w}x.0), x \neq w$$
$$p \xrightarrow{\tau} \nu z.(0|\bar{z}x.0)$$

How it works

How does it work?

- Unification algorithm is modified: *nominal unification* unifies terms modulo equality in NL [UPG03,04]

- Also, freshness constraints must be solved during execution

- Finally, *names* in clauses are freshened prior to unification

- This is justified by the sequent rules.

Nominal unification example

$$\langle \mathfrak{a} \rangle f(X, Y) = \langle \mathfrak{b} \rangle f(\mathfrak{b}, Y)$$

Nominal unification example

$$\langle \mathfrak{a} \rangle f(X, Y) = \langle \mathfrak{b} \rangle f(\mathfrak{b}, Y)$$

$$\Downarrow$$

$$f(X, Y) = (\mathfrak{a}\ \mathfrak{b}) \cdot f(\mathfrak{b}, Y)$$

$$\mathfrak{a} \# f(\mathfrak{b}, Y)$$

Note that $\mathfrak{a} \# f(\mathfrak{b}, Y)$ just reduces to $\mathfrak{a} \# Y$.

Nominal unification example

$$f(X, Y) = (\mathfrak{a}\ \mathfrak{b}) \cdot f(\mathfrak{b}, Y)$$

Nominal unification example

$$f(X, Y) = (\mathfrak{a}\ \mathfrak{b}) \cdot f(\mathfrak{b}, Y)$$

$$\Downarrow$$

$$f(X, Y) = f(\mathfrak{a}, (\mathfrak{a}\ \mathfrak{b}) \cdot Y)$$

Nominal unification example

$$f(X, Y) = f(\mathfrak{a}, (\mathfrak{a}\ \mathfrak{b}) \cdot Y)$$

Nominal unification example

$$f(X, Y) = f(\mathfrak{a}, (\mathfrak{a}\ \mathfrak{b}) \cdot Y)$$

$$\Downarrow$$

$$X = \mathfrak{a}$$

$$Y = (\mathfrak{a}\ \mathfrak{b}) \cdot Y$$

Nominal unification example

$$Y = (\mathfrak{a}\ \mathfrak{b}) \cdot Y$$

Nominal unification example

$$Y = (\mathfrak{a}\ \mathfrak{b}) \cdot Y$$

$$\Downarrow$$

$$\mathfrak{a} \# Y, \mathfrak{b} \# Y$$

Answer: $\langle\mathfrak{a}\rangle f(X, Y) = \langle\mathfrak{b}\rangle f(\mathfrak{b}, Y)$ whenever

$$X = \mathfrak{a}, \mathfrak{a} \# Y, \mathfrak{b} \# Y$$

Freshness constraint solving example

$$\mathfrak{a} \# f(X, \langle\mathfrak{a}\rangle Y)$$

$$\Downarrow$$

$$\mathfrak{a} \# X, \mathfrak{a} \# \langle\mathfrak{a}\rangle Y$$

$$\Downarrow$$

$$\mathfrak{a} \# X$$

What doesn’t work (yet)

What doesn't work

- Unfortunately, the proof search technique I've outlined is incomplete!

- Why?

- Search for a proof of $\text{И}\mathfrak{a}.p(\mathfrak{a}) \Rightarrow \text{И}\mathfrak{a}.p(\mathfrak{a})$ fails after reducing to $\mathfrak{a} \# \mathfrak{a}' : p(\mathfrak{a}') \Rightarrow p(\mathfrak{a})$

- Problem: equivariance not taken into account, needed here to swap $\mathfrak{a}$ for $\mathfrak{a}'$ in goal.

Option 1: Ignore the problem

- Actually lots of interesting programs that work without equivariance (including the ones in this talk)

- And we know how to identify them (that's another talk...)

- But there are also lots of interesting programs that require equivariance

  - automata constructions, type inference, higher-order unification, etc...

Option 2: Find an efficient algorithm

- Also a nonstarter.
- Instead, I found a reduction from Graph 3-Colorability.
- So probably no such algorithm exists.
- Currently working on an exponential (but at least terminating) algorithm

Another problem

- There are only two equivariant binary relations on names: equality and freshness.

- Ergo, there is no equivariant proper linear ordering on names.

- Orderings are needed for efficient implementations of most data structures.

- New ideas are needed.

Conclusions

Related work

- Huge literature on programming and formalizing languages with names and binding: cannot be summarized in one slide

- Closest in spirit: logical frameworks [HHP91 LF, Twelf, etc], λProlog

- Closest in theory: FreshML [SPG03], dependently typed theory of names & binding [SS04]

Future work

- Solve the problems! (ev unification, ordering names)
- More advanced nominal equational reasoning (e.g. π-calculus structural equivalence as a theory of nominal logic)
- Formalization of λ-calculus using nominal logic/abstract syntax in e.g. HOL [Urban]
- Nominal logical frameworks?

Conclusions

- Nominal abstract syntax is a new way of looking at the very important phenomena of names and binding.

- In particular, it can be used to write logic programs that are direct translations from ordinary informal presentations.

- Future: can this approach be used to make formal reasoning about PLs/logics more practical?