Reasoning and programming with nominal logic Part I: Nominal logic and abstract syntax

James Cheney

Abstraction, Substitution and Naming in Computer Science
ICMS Workshop
May 27, 2007

How to fail calculus

$$
\begin{aligned}
\int_0^1 \int_0^1 xy \; dx \; dy &= \int_0^1 \int_0^1 xx \; dx \; dx \\
&= \int_0^1 \frac{1}{3} \; dx = \frac{1}{3} \\
&\neq \frac{1}{4} \quad \text{(the real answer)}
\end{aligned}
$$

- We expect freshman calculus students to understand variable binding at an intuitive level
  - often without any explicit explanation or justification.
- And they do.
- But computers don't.

"Without loss of generality" considered harmful

- "Without loss of generality" reasoning about freshness & α-equivalence is common (often implicit) in informal proofs

  > *If e is of the form $\lambda x.M$, where we assume without loss of generality that x does not appear in Γ...*

- Usually means reasoning modulo equivalence relation is being swept under the rug...
- What makes this informal reasoning principle sound?
- Can we capture "without loss of generality" reasoning logically?

What would Church do?

- Church's higher-order logic: define quantifiers as higher-order functions

$$\Pi : (\iota \to o) \to o \qquad \Sigma : (\iota \to o) \to o$$

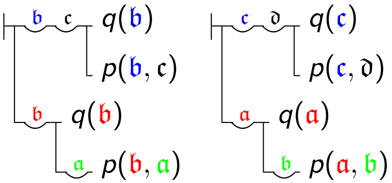
$$\forall x.\phi \equiv \Pi(\lambda x.\phi) \quad \exists x.\phi \equiv \Sigma(\lambda x.\phi)$$

- Forms basis of *higher-order abstract syntax* approach
  - current state of the art (Twelf, λProlog, ∇/Bedwyr...)
- Deal with painful issue of α-equivalence, substitution, etc. once and for all, reuse for each object-language
- Provides elegant apprach to formalizing many systems
- But, names "second-class": name-comparison, generation not well-supported

What would Frege do?

- Quantification theory — introduced by Frege [1879].
- Frege also provided first definition of "α-equivalence":

> *... Replacing a German letter [bound variable] 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. [G. Frege, Begriffsschrift, 1879]*



Gabbay-Pitts approach

- Fast forward 120 years or so...
- [Gabbay, Pitts LICS 1999] formalized names, binding, α-equivalence, fresh name quantification using swapping
  - NB: Injective renamings $\cong$ swappings (Frege was right)
- Formalized within (non-standard) Fraenkel-Mostowski set theory
  - Originally for studying indep. of AC...
  - Created (false) impression that technique requires nonstandard foundations
- [Pitts 2003] introduced elementary (first-order) axiomatization called *nominal logic*

Hold on a minute...

- [Gabbay, Pitts 1999] presented complete picture; but many ingredients already there...
- [Pollack 1994, McKinna-Pollack 1999] investigated defining α-renaming, formalizing PTS using swappings
- [Lescanne, Rouyer-Degli 1995]: explicit substitutions via swapping
- [Odersky 1994] studied "functional" local names via swappings, "finite support"
- [Pitts and Stark 1991, Stark 1998] studied *generativity* using swappings/name-matchings
- [Miller 1991] Injective/bijective renamings in higher-order pattern unification
- ...?

Key idea in one slide

- Consider naive, structurally recursive renamings (of all occurrences) of names:

$$\lambda x.\lambda y.xyz \equiv_\alpha \lambda x'.\lambda y'.x'y'z$$

$$(\lambda x.\lambda y.xyz)[x/y, x/z] = \lambda x.\lambda x.xxx \not\equiv_\alpha (\lambda x'.\lambda y'.x'y'z)[x/y, x/z]$$

  Does not preserve α-equivalence — violates Frege's principle

- But if we restrict to *bijective* renamings, α-equivalence is preserved.

$$(\lambda x.\lambda y.xyz)[x/z, z/x] = \lambda z.\lambda y.zyx \equiv_\alpha (\lambda x'.\lambda y'.x'y'z)[x/z, z/x]$$

  "Naive" bijective renamings are *naturally capture-avoiding*

- We can use this to *define* α-equivalence

What good is nominal logic?

- Names "first-class"; exist as (semantic) values; can be compared
- Binding, free names, and name-generation use same mechanisms
- Induction/recursion principles resemble informal conventions
- Provides a formal foundation for informal, but rigorous idioms
- "Close" to FO logic; many existing techniques can be reused with a token ("nominal"?) amount of work

Outline

- Part I
  - Nominal terms
  - Axiomatization
  - Semantics
- Intermission
- Part II
  - Nominal logic programming
  - Unification/resolution

Ground nominal terms

$$
\begin{array}{rcll}
\mathfrak{a}, \mathfrak{b} & \in & \mathbb{A} & \text{Names} \\
f, g & \in & \mathit{FnSym} & \text{Term symbols} \\
t, u & ::= & \langle\rangle \mid \langle t, u\rangle \mid f(t) & \text{First-order terms} \\
& \mid & \langle\mathfrak{a}\rangle t \mid \mathfrak{a} & \text{Nominal terms} \\
\pi & ::= & (\mathfrak{a}\ \mathfrak{b}) \mid \mathrm{id} \mid \pi \circ \pi' & \text{Permutations} \\
C & ::= & t \approx u \mid \mathfrak{a} \# t & \text{Equality, freshness}
\end{array}
$$

Note: Constants $c = c\ \langle\rangle$, n-ary functions $f(\vec{t}) = f\ \langle t_1, \langle t_2, \cdots\rangle\rangle$

Running example

- Type signature

$$id : name\_type. \quad exp : type$$

- Term signature

$$\begin{array}{rcl} var & : & id \to exp \\ app & : & exp \times exp \to exp \\ lam & : & \langle id \rangle exp \to exp \end{array}$$

- Translation

$$\begin{array}{rcl} \ulcorner \mathfrak{x} \urcorner & = & var(\mathfrak{x}) \\ \ulcorner M\ N \urcorner & = & app(\ulcorner M \urcorner, \ulcorner N \urcorner) \\ \ulcorner \lambda \mathfrak{x}.M \urcorner & = & lam(\langle \mathfrak{x} \rangle \ulcorner M \urcorner) \end{array}$$

Ground swapping

The result of applying a (ground) permutation π to a (ground) term is:

$$\begin{array}{rcl} \pi \cdot \mathfrak{a} & = & \pi(\mathfrak{a}) \\ \pi \cdot \langle\rangle & = & \langle\rangle \\ \pi \cdot \langle t, u\rangle & = & \langle \pi \cdot t, \pi \cdot u\rangle \\ \pi \cdot f(t) & = & f(\pi \cdot t) \\ \pi \cdot \langle \mathfrak{b}\rangle t & = & \langle \pi \cdot \mathfrak{b}\rangle \pi \cdot t \end{array}$$

where

$$\begin{array}{rcl} \mathrm{id}(\mathfrak{a}) & = & \mathfrak{a} \\ \pi \circ \pi'(\mathfrak{a}) & = & \pi(\pi'(\mathfrak{a})) \\ (\mathfrak{a}\ \mathfrak{b})(\mathfrak{c}) & = & \begin{cases} \mathfrak{b} & (\mathfrak{a} = \mathfrak{c}) \\ \mathfrak{a} & (\mathfrak{b} = \mathfrak{c}) \\ \mathfrak{c} & (\mathfrak{a} \neq \mathfrak{c} \neq \mathfrak{b}) \end{cases} \end{array}$$

Ground freshness theory

$$\frac{(\mathfrak{a} \neq \mathfrak{b})}{\mathfrak{a} \# \mathfrak{b}}$$ Distinct names fresh

$$\frac{}{\mathfrak{a} \# \langle\rangle}$$ Anything fresh for unit

$$\frac{\mathfrak{a} \# t}{\mathfrak{a} \# f(t)}$$ Freshness ignores function symbols

$$\frac{\mathfrak{a} \# t \quad \mathfrak{a} \# u}{\mathfrak{a} \# \langle t, u\rangle}$$ Freshness ignores pairs

$$\frac{}{\mathfrak{a} \# \langle\mathfrak{a}\rangle t}$$ Fresh for abs. if bound

$$\frac{(\mathfrak{a} \neq \mathfrak{b}) \quad \mathfrak{a} \# t}{\mathfrak{a} \# \langle\mathfrak{b}\rangle t}$$ Fresh for abs. if fresh for body

Ground equational theory

$$\left.\begin{array}{c}
\dfrac{}{\mathfrak{a} \approx \mathfrak{a}} \\[2ex]
\dfrac{}{\langle\rangle \approx \langle\rangle} \\[2ex]
\dfrac{t_1 \approx u_1 \quad t_2 \approx u_2}{\langle t_1, t_2\rangle \approx \langle u_1, u_2\rangle} \\[2ex]
\dfrac{t \approx u}{f(t) \approx f(u)} \\[2ex]
\dfrac{t \approx u}{\langle\mathfrak{a}\rangle t \approx \langle\mathfrak{a}\rangle u}
\end{array}\right\} \text{Standard equational rules}$$

$$\dfrac{\mathfrak{a} \# u \quad t \approx (\mathfrak{a}\ \mathfrak{b}) \cdot u}{\langle\mathfrak{a}\rangle t \approx \langle\mathfrak{b}\rangle u} \qquad \alpha\text{-equivalence for abstractions}$$

Examples

- Swapping

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

$$(\mathfrak{a}\ \mathfrak{b}) \cdot \langle \mathfrak{b} \rangle f(\mathfrak{a}, c) = \langle \mathfrak{a} \rangle f(\mathfrak{b}, c)$$

$$(\mathfrak{a}\ \mathfrak{b}) \cdot \langle \mathfrak{a} \rangle \langle \mathfrak{b} \rangle f(\mathfrak{a}, c) = \langle \mathfrak{b} \rangle \langle \mathfrak{a} \rangle f(\mathfrak{b}, c)$$

Examples

- Freshness

$$\dfrac{\dfrac{\mathfrak{a} \neq \mathfrak{b}}{\mathfrak{a} \# \mathfrak{b}} \quad \dfrac{}{\mathfrak{a} \# c}}{\mathfrak{a} \# f(\mathfrak{b}, c)}$$

$$\dfrac{}{\mathfrak{a} \# \langle\mathfrak{a}\rangle f(\mathfrak{a}, c)}$$

$$\dfrac{\mathfrak{a} \# \langle\mathfrak{a}\rangle f(\mathfrak{a}, c) \quad \dfrac{}{\mathfrak{a} \# \mathfrak{b}}}{\mathfrak{a} \# g(\langle\mathfrak{a}\rangle f(\mathfrak{a}, c), \mathfrak{b})}$$

Examples

- Equality

$$\frac{\mathfrak{a} \approx \mathfrak{a} \quad c \approx c}{f(\mathfrak{a}, c) \approx f(\mathfrak{a}, c)}$$

$$\frac{\mathfrak{a} = \mathfrak{a} \quad f(\mathfrak{a}, c) \approx f(\mathfrak{a}, c)}{\langle \mathfrak{a} \rangle f(\mathfrak{a}, c) \approx \langle \mathfrak{a} \rangle f(\mathfrak{a}, c)}$$

$$\frac{\mathfrak{a} \# \langle \mathfrak{a} \rangle f(\mathfrak{a}, c) \quad \langle \mathfrak{b} \rangle f(\mathfrak{b}, c) \approx (\mathfrak{a}\ \mathfrak{b}) \cdot \langle \mathfrak{a} \rangle f(\mathfrak{a}, c)}{\langle \mathfrak{a} \rangle \langle \mathfrak{a} \rangle f(\mathfrak{a}, c) \approx \langle \mathfrak{b} \rangle \langle \mathfrak{b} \rangle f(\mathfrak{b}, c)}$$

since $(\mathfrak{a}\ \mathfrak{b}) \cdot \langle \mathfrak{a} \rangle f(\mathfrak{a}, c) = \langle \mathfrak{b} \rangle f(\mathfrak{b}, c)$

FAQ's (0)

- Why is this a correct definition of α-equivalence?
- The following rules all generate $\equiv_\alpha$:

$$\frac{y \notin FV(M)}{\lambda x.M \equiv_\alpha \lambda y.M[y/x]}$$

$$\frac{y \notin FV(M) \quad M[y/x] \equiv_\alpha N}{\lambda x.M \equiv_\alpha \lambda y.N}$$

$$\frac{y \notin FV(M) \quad M[y/x, x/y] \equiv_\alpha N}{\lambda x.M \equiv_\alpha \lambda y.N}$$

since $M[y/x] = M[y/x, x/y]$ if $y \notin FV(M)$.

Adequacy

- What is the relationship between nominal operations on ground *exp*-terms and operations on λ-terms?
- Swapping = simultaneous capture-avoiding renaming

$$(\mathfrak{x}\ \mathfrak{y}) \cdot \ulcorner M \urcorner = \ulcorner M[\mathfrak{x}/\mathfrak{y}, \mathfrak{y}/\mathfrak{x}] \urcorner$$

- Freshness = "not among free variables of"

$$\mathfrak{x} \# \ulcorner M \urcorner \iff \mathfrak{x} \notin FV(M)$$

- Equality = α-equivalence

$$\ulcorner M \urcorner \approx \ulcorner N \urcorner \iff M \equiv_\alpha N$$

FAQ's (I)

- *Why isn't there a symmetric freshness constraint in the α-rule?*

$$\frac{\mathfrak{a} \# u \quad \mathfrak{b} \# t \quad t \approx (\mathfrak{a}\ \mathfrak{b}) \cdot u}{\langle \mathfrak{a} \rangle t \approx \langle \mathfrak{b} \rangle u}$$

- Answer: If $\mathfrak{a} \# u$ and $t \approx (\mathfrak{a}\ \mathfrak{b}) \cdot u$ hold, then

$$\mathfrak{b} \approx (\mathfrak{a}\ \mathfrak{b}) \cdot \mathfrak{a} \# (\mathfrak{a}\ \mathfrak{b}) \cdot u \approx t$$

  holds.
- Key step: $\mathfrak{a} \# u$ implies $(\mathfrak{a}\ \mathfrak{b}) \cdot \mathfrak{a} \# (\mathfrak{a}\ \mathfrak{b}) \cdot u$
- Instance of *equivariance*

Equivariance

- What is “equivariance”?
  - Value x is equivariant if
    $$\pi \cdot x \approx x$$
    for any permutation π.
  - Function F is equivariant if
    $$\pi \cdot F(\vec{x}) \approx F(\pi \cdot \vec{x})$$
    for any $\vec{x}$ and any permutation π.
  - Relation R is equivariant if
    $$R(\vec{x}) \iff R(\pi \cdot \vec{x})$$
    for any $\vec{x}$ and any permutation π.
- Fact: All function symbols, $\approx$ and $\#$ are equivariant (easy inductions).

FAQ's (II)

- *The asymmetric α-equivalence law still bothers me...*
- OK then...

$$\frac{\text{И}\mathfrak{c}.(\mathfrak{a}\ \mathfrak{c}) \cdot t \approx (\mathfrak{b}\ \mathfrak{c}) \cdot u}{\langle \mathfrak{a} \rangle t \approx \langle \mathfrak{b} \rangle u}$$

where $\text{И}\mathfrak{a}.\phi$ quantifies over *fresh names only*.

The $\reflectbox{N}$-quantifier

- What does $\reflectbox{N}\mathfrak{a}.\phi$ mean?
- Intuitively: "for arbitrary fresh names $\mathfrak{a}$, $\phi(\mathfrak{a})$ holds"
- Note that
  - Syntax trees are finite (& mention finitely many names)
  - Set of names is infinite
  - Sets of names fresh for (finitely many) syntax trees are *cofinite*
  - Fresh names are "indistinguishable"
- So, we say that $\reflectbox{N}\mathfrak{a}.\phi$ holds if $\{\mathfrak{a} \mid \phi(\mathfrak{a})\}$ is *cofinite*.
- (Think "almost everywhere" in measure theory)

Self-duality of И

- Fact: $\neg\text{И}\mathfrak{a}.\phi \iff \text{И}\mathfrak{a}.\neg\phi$
- Intuition: "not mostly ϕ = mostly not ϕ"
- Proof (sketch):

$$\begin{array}{rcl} \neg\text{И}\mathfrak{a}.\phi & \iff & \{\mathfrak{a} \mid \phi(\mathfrak{a})\} \text{ not cofinite} \\ & \iff & \{\mathfrak{a} \mid \phi(\mathfrak{a})\} \text{ finite} \\ & \iff & \{\mathfrak{a} \mid \neg\phi(\mathfrak{a})\} \text{ cofinite} \\ & \iff & \text{И}\mathfrak{a}.\neg\phi \end{array}$$

- NB: Not cofinite implies finite since ϕ mentions/depends on only finitely many names
- *Some/any reasoning*
  - "Freshness principle": we can never run out of fresh names
  - "Equivariance principle": any two fresh names have the same properties

Nominal logic: axiomatization

- Goal: Capture valid reasoning about nominal terms in a logic.
- Take *sorted, first-order logic* as a starting point
- Axioms for equality, freshness, И
- Gentzen-style sequent calculus

Nominal logic basics

Extends (sorted, $=$) first-order logic with syntax/axioms for

- *names* $\mathfrak{a}, \mathfrak{b}$ inhabiting *name-sorts* $A, B, \ldots$
- a *name-swapping* function symbol $(- \ -) \cdot - : A \times A \times S \to S$ for each name-sort A and sort S
- a *name-binding* or *abstraction* function symbol/sort $\langle - \rangle - : A \times S \to \langle A \rangle S$
- a *freshness* relation $- \# - : A \times S$ relating names $\mathfrak{a}$ and terms t with no free occurrences of $\mathfrak{a}$
- a *fresh-name quantifier* $\reflectbox{N}\mathfrak{a}.\phi$.

Let Ω be a *signature* listing basic data-sorts & name-sorts, and assigning sorts to function & relation symbols

Axioms for swapping

$$\begin{array}{lrl} (S1) & \forall a{:}A, x{:}S. & (a\ a) \cdot x \approx x \\ (S2) & \forall a, b{:}A, x{:}S. & (a\ b) \cdot (a\ b) \cdot x \approx x \\ (S3) & \forall a, b{:}A. & (a\ b) \cdot a \approx b \end{array}$$

Note: Can derive $(a\ b) \cdot b \approx a$:

$$\begin{array}{rcll} (a\ b) \cdot a & \approx & b & (S3) \\ (a\ b) \cdot (a\ b) \cdot a & \approx & (a\ b) \cdot b & (=) \\ a & \approx & (a\ b) \cdot b & (S2) \end{array}$$

Axioms for freshness

$$
\begin{array}{lrl}
(F1) & \forall a, b{:}A.\forall x{:}S. & a \# x \wedge b \# x \supset (a\ b) \cdot x = x \\
(F2) & \forall a, b{:}A. & a \# b \iff a \neq b \\
(F3) & \forall a{:}A, b{:}A'. & a \# b \\
(F4) & \forall \vec{x}{:}\vec{S}.\exists a. & a \# \vec{x}
\end{array}
$$

Freshness principle (F4): Can never run out of fresh names. NB: Can derive previous laws, e.g.

$$\mathfrak{a} \# x \supset \mathfrak{a} \# f(x) \quad \mathfrak{a} \# \langle \mathfrak{a} \rangle x$$

Axioms for equivariance

$$
\begin{array}{lrl}
(E1) & \forall a,a'{:}A, b,b'{:}A', x{:}S. & (a\ a')\cdot(b\ b')\cdot x \\
 & & \approx ((a\ a')\cdot b\ (a\ a')\cdot b')\cdot(a\ a')\cdot x \\
(E2) & \forall a,a'{:}A, b{:}A', x{:}S. & b \# x \supset (a\ a')\cdot b \# (a\ a')\cdot x \\
(E3) & \forall a,a'{:}A, \vec{x}{:}\vec{S}. & (a\ a')\cdot f(\vec{x}) \approx f((a\ a')\cdot\vec{x}) \\
(E4) & \forall a,a'{:}A, \vec{x}{:}\vec{S}. & R(\vec{x}) \supset R((a\ a')\cdot\vec{x})
\end{array}
$$

Equivariance principle: all constant/function/relation symbols are equivariant.

Proposition

For any $\phi(\vec{x})$, we have $NL \vdash \phi(\vec{x}) \iff \phi((a\ b)\cdot\vec{x})$.

Axioms for abstraction

$$
\begin{array}{lll}
(A1) & \forall a, b{:}A, x, y{:}S. & \langle a\rangle x \approx \langle b\rangle y \iff \\
 & & (a \approx b \wedge x \approx y) \\
 & & \vee(a \# y \wedge x \approx (a\ b) \cdot y) \\
(A2) & \forall y{:}\langle A\rangle S.\exists a{:}A, x{:}S. & y \approx \langle a\rangle x \\
(E5) & \forall a, a'{:}A, b{:}A', x{:}S. & (a\ a') \cdot \langle b\rangle x \approx \langle (a\ a') \cdot b\rangle (a\ a') \cdot x
\end{array}
$$

(A1) defines α-equivalence
(A2) is a *surjectivity* property for abstraction.
Fact: (A2) equivalent to

$$\forall y{:}\langle A\rangle S.\text{И}a{:}A.\exists x{:}S.y \approx \langle a\rangle x$$

Axiomatizing N

- We can axiomatize the N-quantifier as follows:

$$(Q) \quad \mathsf{N}\mathfrak{a}.\phi(\mathfrak{a}, \vec{X}) \iff \exists a.a \# \vec{x} \wedge \phi(a, \vec{x})$$

  where $\phi(\mathfrak{a}, \vec{x})$ indicates that $\mathfrak{a}, \vec{x}$ list all the free variables of ϕ.

- Fact: Equivalent to "universal" characterization

Proposition

$NL \vdash \exists a.a \# \vec{x} \wedge \phi(a, \vec{x}) \iff \forall a.a \# \vec{x} \supset \phi(a, \vec{x})$

Proof.

Need both equivariance (Prop 1) and freshness (F4). $\square$

Aside: Proof theory of NL

- Sequent/ND proof systems for NL have also been studied.
- One idea: build freshness information into variable contexts:

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

- И-rules:

$$\frac{\Sigma\#\mathfrak{a}{:}A : \Gamma, \phi \Rightarrow \psi}{\Sigma : \Gamma, \text{И}\mathfrak{a}{:}A.\phi \Rightarrow \psi}\ \text{И}L \qquad \frac{\Sigma\#\mathfrak{a}{:}A : \Gamma \Rightarrow \phi}{\Sigma : \Gamma \Rightarrow \text{И}\mathfrak{a}{:}A.\phi}\ \text{И}R \quad (\mathfrak{a} \notin \Sigma)$$

- Embed $\approx$/#-axioms as additional rules (c.f. *Structural Proof Theory* [Negri and van Plato 2001])
- Don't have time for more on this...

Nominal logic: semantics

- Nominal logic was *inspired by* nominal terms, which can mention/depend on only *finitely many* names
- And sound for reasoning about such models... but incomplete
- We now consider the semantics of nominal logic in general
  - Finitely-supported nominal sets — incomplete
  - *Ideal-supported* nominal sets & completeness
  - Herbrand models & completeness for nominal-universal theories

Finitely-supported nominal sets

Definition

A *finitely-supported nominal set* is a structure

$$(X, \cdot_X : \mathit{Perm}(\mathbb{A}) \times X \to X, \mathrm{supp} : X \to \mathcal{P}_{\mathsf{fin}}(\mathbb{A}))$$

such that:

- $\cdot_X$ is a *group action* of $\mathit{Perm}(\mathbb{A})$ on X
- each $x \in X$ has a $\subseteq$-minimum, finite *support* $\mathrm{supp}(x)$ such that:
$$\forall a, b \notin \mathrm{supp}(x).(a\ b) \cdot x = x$$

Basic nominal sets

- Any "ordinary" set ($\mathbb{N}$, $\mathbb{R}$, etc.) can be viewed as a nominal set:

$$\begin{aligned}(\mathfrak{a}\ \mathfrak{b}) \cdot x &= x \\ \mathrm{supp}(x) &= \varnothing\end{aligned}$$

- The set of names $\mathbb{A}$ is a nominal set:

$$\begin{aligned}(\mathfrak{a}\ \mathfrak{b}) \cdot x &= \begin{cases} \mathfrak{a} & x = \mathfrak{b} \\ \mathfrak{b} & x = \mathfrak{a} \\ \mathfrak{x} & \mathfrak{a} \neq x \neq \mathfrak{b} \end{cases} \\ \mathrm{supp}(\mathfrak{a}) &= \{\mathfrak{a}\}\end{aligned}$$

Nominal set constructions

- Unit $1 = \{\star\}$:

$$\begin{aligned} \pi \cdot \star &= \star \\ \mathrm{supp}(\star) &= \varnothing \end{aligned}$$

- Cartesian products $X \times Y$:

$$\begin{aligned} \pi \cdot (x, y) &= (\pi \cdot x, \pi \cdot y) \\ \mathrm{supp}(x, y) &= \mathrm{supp}(x) \cup \mathrm{supp}(y) \end{aligned}$$

- Disjoint unions $X + Y$:

$$\begin{aligned} \pi \cdot (\iota_i(x)) &= \iota_i(\pi \cdot x) \\ \mathrm{supp}(\iota_i(x)) &= \mathrm{supp}(x) \end{aligned}$$

Nominal set constructions

- Function spaces $X \to Y$ (finitely-supported):

$$(\pi \cdot f)(x) \quad = \quad \pi \cdot (f(\pi^{-1} \cdot x))$$

- Powersets $\mathcal{P}(X)$ (finitely-supported):

$$\pi \cdot S \quad = \quad \{\pi \cdot x \mid x \in S\}$$

- Quotients $X/_{\equiv}$ ($\equiv$ equivariant):

$$\pi \cdot [x]_{\equiv} \quad = \quad \{\pi \cdot y \mid x \equiv y\} = [\pi \cdot x]_{\equiv}$$

- Nominal sets & equivariant functions have topos structure ($\cong$ "Schanuel topos").

Name-abstraction construction

- Consider the equivalence relation on $\mathbb{A} \times X$ generated by:

$$\text{И}\mathfrak{c}.(\mathfrak{a}\ \mathfrak{c}) \cdot x = (\mathfrak{b}\ \mathfrak{c}) \cdot y \supset (\mathfrak{a}, x) \equiv_\alpha (\mathfrak{b}, y)$$

- Define the *name-abstraction construction* as:

$$\begin{aligned} \langle\langle \mathbb{A} \rangle\rangle X &\triangleq (\mathbb{A} \times X)/_{\equiv_\alpha} \\ \langle\langle a \rangle\rangle x &\triangleq (a, x)_{\equiv_\alpha} \end{aligned}$$

- Facts:

$$\begin{aligned} \pi \cdot (\langle\langle a \rangle\rangle x) &= \langle\langle \pi \cdot a \rangle\rangle \pi \cdot x \\ \mathrm{supp}(\langle\langle a \rangle\rangle x) &= \mathrm{supp}(x) - \{a\} \end{aligned}$$

Soundness

We can (soundly) interpret...

- $A^{\mathcal{M}} = \mathbb{A}$; $(\langle A \rangle S)^{\mathcal{M}} = \langle\langle \mathbb{A} \rangle\rangle S^{\mathcal{M}}$
- Other sorts S as nominal sets $S^{\mathcal{M}}$
- $\mathfrak{a}^{\mathcal{M}} = \mathfrak{a}$
- $(\langle a \rangle t)^{\mathcal{M}} = \langle\langle a^{\mathcal{M}} \rangle\rangle t^{\mathcal{M}}$
- $- \approx -$ as “real” equality
- $- \# -$ as $- \notin \mathrm{supp}(-)$ (“real” freshness)
- Other constant, function, predicate symbols as equivariant values, functions, relations

Soundness and (in)completeness

Theorem (Soundness)

$\Gamma \vdash \phi$ *implies* $\Gamma \vDash_{\mathsf{FS}} \phi$

- Unfortunately, finite-support semantics is incomplete.
- Compactness fails: all finite subsets of

$$\Gamma = \{\neg(a_i \# x) \mid i \in \omega\} \cup \{a_i \# a_j \mid i \neq j \in \omega\}$$

  are satisfiable in FS-models, but Γ is not.
- Idea: Γ says "x has infinitely many different names in its support"
- And noncompactness implies incompleteness

Recovering completeness

- Obviously, NL is complete w.r.t. all first-order models, but this is unhelpful...
- What do the FO models of NL look like?
- Answer: *ideal-supported models* in which supports form a proper ideal.
  - NB: Already known in study of $\neg AC$ set theory...
- Ideal properties ensure that we can *never run out* of fresh names even if some values have *infinite support*!
- Thus, it is also possible to use NL to reason about *infinite objects with infinite support*.
- NB: Even if $\mathbb{A}$ countable! (by Löwenheim-Skolem)

Ideal-supported nominal sets

- Let $\mathcal{I} \subseteq \mathcal{P}(\mathbb{A})$ be a proper ideal containing $\mathcal{P}_{\mathsf{fin}}(\mathbb{A})$.

Definition

An *$\mathcal{I}$-nominal set* is an algebraic structure

$$(X, \cdot_X : \mathit{Perm}(\mathbb{A}) \times X \to X, \mathrm{supp} : X \to \mathcal{I})$$

such that:

- $\cdot_X$ is a *group action* of $\mathit{Perm}(\mathbb{A})$ on X
- each elt of X has a $\subseteq$-minimum *support* $\mathrm{supp}(x) \in \mathcal{I}$

- All constructions for finitely-supported nominal sets still work.

Theorem (Soundness & Completeness)

$\Gamma \vDash \phi$ *over ideal-supported structures iff* $\Gamma \vdash \phi$

To be continued...

- Part I
  - Nominal terms
  - Axiomatization
  - Semantics
- Intermission
- Part II
  - Nominal logic programming
  - Unification/resolution

Reasoning and programming with nominal logic
Part II: Nominal logic programming

James Cheney

Abstraction, Substitution and Naming in Computer Science
ICMS Workshop
May 27, 2007

Outline

- Part I
  - Nominal terms
  - Axiomatization
  - Semantics
- Intermission
- Part II
  - Nominal logic programming
  - Unification/resolution

Nominal logic programming

- An application: logic programming in NL ("αProlog")
- Goal: Translate informal "paper" inference rule definitions to executable, yet formally transparent language
- Use nominal features to handle generativity, α-equivalence
- Equivariance ensures cannot "get at" bound names
- No help with substitution
  - But no longer "hard" to implement declaratively (no need for *gensym*)

Example (I)

- A warm-up (warmed-over?) example: typechecking.

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T} \quad \frac{\Gamma \vdash M : T \to U \quad \Gamma \vdash N : T}{\Gamma \vdash M\ N : U} \quad \frac{(x \notin \Gamma) \quad \Gamma, x : T \vdash M : U}{\Gamma \vdash \lambda x.M : T \to U}$$

$$\begin{array}{lcl}
tc(G, var(X), T) & :- & mem((X, T), G). \\
tc(G, app(M, N), U) & :- & tc(G, M, arr(T, U)), tc(G, N, T). \\
tc(G, lam(\langle \mathfrak{x} \rangle M), arr(T, U)) & :- & \mathfrak{x} \# G, tc([(\mathfrak{x}, T)|G], M, U).
\end{array}$$

Example (I)

- A warm-up (warmed-over?) example: typechecking.

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T} \quad \frac{\Gamma \vdash M : T \to U \quad \Gamma \vdash N : T}{\Gamma \vdash M\ N : U} \quad \frac{(x \notin \Gamma) \quad \Gamma, x : T \vdash M : U}{\Gamma \vdash \lambda x.M : T \to U}$$

$$\begin{array}{lll} tc(G, var(X), T) & :- & mem((X, T), G). \\ tc(G, app(M, N), U) & :- & tc(G, M, arr(T, U)), tc(G, N, T). \\ tc(G, lam(\langle \mathfrak{x} \rangle M), arr(T, U)) & :- & \mathfrak{x} \# G, tc([(\mathfrak{x}, T)|G], M, U). \end{array}$$

- Note that clauses and subgoals correspond closely
  - Read $\mathfrak{x} \# G$ as $x \notin \Gamma$
  - Read $lam(\langle \mathfrak{x} \rangle M)$ as $\lambda x.M$

Example (II)

- Alpha-*in*equivalence.
- Non-immediate in a higher-order setting (fun exercise!).

$$
\begin{array}{lcl}
neq(var(X), var(Y)) & :- & X \# Y. \\
neq(app(M, N), app(M', N')) & :- & neq(M, M'). \\
neq(app(M, N), app(M', N')) & :- & neq(N, N'). \\
neq(lam(\langle\mathfrak{x}\rangle M), lam(\langle\mathfrak{x}\rangle N)) & :- & neq(M, N). \\
neq(var(\_), app(\_, \_)). & & neq(var(\_), lam(\_)). \\
neq(app(\_, \_), var(\_)). & & neq(app(\_, \_), lam(\_)). \\
neq(lam(\_), app(\_, \_)). & & neq(lam(\_), var(\_)).
\end{array}
$$

- Here, $X \# Y$ just means $X \neq Y$.
- NB: This also works for π-calc mismatch, explicit substitutions, ...

Example (III)

- Large-step semantics for ML-like references:

$$\frac{(a \in Lab)}{\langle \sigma, a \rangle \rightarrow \langle \sigma, a \rangle} \qquad \frac{\langle \sigma, M_1 \rangle \rightarrow \langle \sigma', a \rangle \quad \langle \sigma', M_2 \rangle \rightarrow \langle \sigma'', V \rangle}{\langle \sigma, M_1 := M_2 \rangle \rightarrow \langle \sigma''[a := V], () \rangle}$$

$$\frac{\langle \sigma, M \rangle \rightarrow \langle \sigma', a \rangle}{\langle \sigma, !M \rangle \rightarrow \langle \sigma', \sigma'(a) \rangle} \qquad \frac{\langle \sigma, M \rangle \rightarrow \langle \sigma', V \rangle \quad (a \notin dom(\sigma'))}{\langle \sigma, \mathrm{ref}\ M \rangle \rightarrow \langle \sigma'[a := V], a \rangle}$$

- Interesting part: last rule requires *fresh* label for new memory cell

Example (III)

- Large-step semantics for ML-like references:

$$
\begin{array}{lcl}
(S, \mathit{lab}(A)) \Longrightarrow (S, \mathit{lab}(A)). & & \\
(S, \mathit{assign}(M_1, M_2)) \Longrightarrow (S_3, \mathit{unit}) & :- & (S, M_1) \Longrightarrow (S_1, \mathit{lab}(A)), \\
 & & (S_1, M_2) \Longrightarrow (S_2, V), \\
 & & \mathit{update}((A, V), S_2, S_3). \\
(S, \mathit{deref}(M)) \Longrightarrow (S', V) & :- & (S, M) \Longrightarrow (S', \mathit{lab}(A)), \\
 & & \mathit{mem}((A, V), S'). \\
(S, \mathit{ref}(M)) \Longrightarrow ([(\mathfrak{a}, V)|S'], \mathit{lab}(\mathfrak{a})) & :- & (S, M) \Longrightarrow (S', V), \\
 & & \mathfrak{a} \# S'.
\end{array}
$$

- Interesting part: in last rule, name $\mathfrak{a}$ is constrained to be sufficiently fresh

Logical foundations

- Logic programming languages derive much of their power from logical foundations
- Prolog
  - semantics based on first-order logic
  - implementation based on first-order unification
- λProlog
  - semantics based on higher-order logic
  - implementation based on higher-order unification
- αProlog
  - semantics based on nominal logic?
  - implementation based on ??

Nominal logic programs

- We consider *goals* and *program clauses* in NL defined as follows:

$$
\begin{array}{rcl}
G & ::= & \top \mid p(\vec{t}) \mid G \wedge G' \mid G \vee G' \mid \exists X.G \mid t \approx u \mid t \# u \mid \reflectbox{N}\mathfrak{a}.G \\
D & ::= & \top \mid p(\vec{t}) \mid D \wedge D' \mid G \supset D \mid \forall X.D \mid \reflectbox{N}\mathfrak{a}.D
\end{array}
$$

- Example: "Open" program clause

$$tc(G, lam(\langle\mathfrak{x}\rangle M), arr(T, U)) :- \mathfrak{x} \# G, tc([(\mathfrak{x}, T)|G], M, U).$$

logically interpreted as "closed" formula

$$
\begin{array}{l}
\reflectbox{N}\mathfrak{x}.\forall G, M, T, U. \\
\mathfrak{x} \# G \wedge tc([(\mathfrak{x}, T)|G], M, U) \supset tc(G, lam(\langle\mathfrak{x}\rangle M), arr(T, U))
\end{array}
$$

Semantics

- Traditional approaches to the semantics of LP can be adapted to nominal logic programs:
    - Least fixed point models
    - Least Herbrand models
    - Proof-theoretic semantics (uniform proofs)
    - Operational semantics (idealized, nondeterministic)

  Moreover all of the semantics are equivalent

Herbrand models

- As a starting point, consider *Herbrand models* of NL.
- Define *Herbrand base* $\mathcal{B}_\Omega$ as set of all atomic (non-constraint) formulas over signature Ω.
- A *Herbrand model* is an *equivariant* subset $\mathcal{H} \subseteq \mathcal{B}_\Omega$
- Given such a model, interpret
  - term symbols as themselves ($c^\mathcal{H} = c$, $f^\mathcal{H}(\vec{t}) = f(\vec{t})$)
  - sorts S as sets ($S^\mathcal{H} = \{t : S \mid t \text{ ground}\}$)
  - predicates R as relations ($R^\mathcal{H} = \{\vec{t} \mid R(\vec{t}) \in \mathcal{H}\}$)
- Write $\vDash^\mathcal{H} \phi$ if ϕ holds in $\mathcal{H}$; etc.
- Write $\Sigma : \nabla \vDash C$ for *constraint entailment* ($\forall \theta : \Sigma.\theta \vDash^\mathcal{H} \nabla$ implies $\theta \vDash^\mathcal{H} C$)

Herbrand's theorem for NL

- Full NL *sound* but *incomplete* with respect to "intended models" over nominal terms.
- But complete for "nominal universal" ($\reflectbox{N}\forall$) theories

$$
\begin{array}{rcl}
\phi_0 & ::= & R(\vec{t}) \mid \phi_0 \wedge \psi_0 \mid \phi_0 \vee \psi_0 \mid \neg\phi_0 \\
\phi & ::= & \phi_0 \mid a \# t \supset \phi \mid t \approx u \supset \phi \mid \forall X{:}S.\phi \mid \reflectbox{N}\mathsf{a}{:}A.\phi
\end{array}
$$

Theorem (Nominal Herbrand theorem)

A $\reflectbox{N}\forall$-theory Γ is satisfiable iff it has a nominal term model.

- Non-immediate due to $\exists$ axioms in NL (F4,A2)

Least Herbrand models

- Fact: Every NL program (set of D-formulas) is (equivalent to) a $\reflectbox{N}\forall$-theory.

$$
\begin{array}{lcll}
(\reflectbox{N}\mathsf{a}.G) \supset D & \Longrightarrow & \reflectbox{N}\mathsf{a}.(G \supset D) & (\mathsf{a} \notin FN(D)) \\
(\reflectbox{N}\mathsf{a}.G) \wedge G' & \Longrightarrow & \reflectbox{N}\mathsf{a}.(G \wedge G') & (\mathsf{a} \notin FN(G')) \\
 & \vdots & &
\end{array}
$$

- Fact: Herbrand models closed under intersection (proof similar to FO case)
- Conclusion: Least Herbrand model $\mathcal{H}_{\mathcal{P}}$ exists for any NL program $\mathcal{P}$.

Computing least models

- Can we "compute" (recursively enumerate) least Herbrand model?
- In FO case, yes; $\mathcal{H}_{\mathcal{P}} = LFP(T_{\mathcal{P}})$, for $T_{\mathcal{P}}$ a continuous operator on Herbrand models
- Can extend this proof to NL
- Key fact 1: continuity in presence of Ꙩ-quantifier
- Key fact 2: $T_{\mathcal{P}}$ must also be *equivariant* to ensure that $LFP(T_{\mathcal{P}})$ is equivariant

Computing least models

- Define one-step deduction operator (by induction on D-formulas):

$$
\begin{array}{rcl}
T_{\top}(\mathcal{H}) & = & \mathcal{H} \\
T_{A}(\mathcal{H}) & = & \mathcal{H} \cup \{A\} \\
T_{D_1 \wedge D_2}(\mathcal{H}) & = & T_{D_1}(\mathcal{H}) \cup T_{D_2}(\mathcal{H}) \\
T_{G \supset D}(\mathcal{H}) & = & \begin{cases} T_D(\mathcal{H}) & \text{if } \mathcal{H} \vDash G \\ \mathcal{H} & \text{otherwise} \end{cases} \\
T_{\forall X:S.D}(\mathcal{H}) & = & \bigcup_{t:S} T_{D[t/X]}(\mathcal{H}) \\
T_{\text{И}\mathfrak{a}:A.D}(\mathcal{H}) & = & \bigcup_{\mathfrak{b}:A \notin FN(\text{И}\mathfrak{a}.D)} T_{D[\mathfrak{b}/\mathfrak{a}]}(\mathcal{H})
\end{array}
$$

- Define $T_{\mathcal{P}}$ as $T_{\bigwedge_i D_i}$ if $\mathcal{P} = \{D_1, \ldots, D_n\}$.
- Continuity easy; only И is nonstandard
- Equivariance: must generalize to $\pi \cdot T_D(S) = T_{\pi \cdot D}(\pi \cdot S)$.

Aside: Proof-theoretic semantics

- Using proof theory of NL, can provide proof-theoretic semantics
- Explain behavior of $\reflectbox{N}$ in terms of proof-search behavior
- For goals, $\reflectbox{N}\mathfrak{a}.G$ means "generate fresh name $\mathfrak{a}$ and solve G"

$$\frac{\Sigma : \nabla \vDash \reflectbox{N}\mathfrak{a}{:}A.C \quad \Sigma\#\mathfrak{a}{:}A : \Delta; \nabla, C \Longrightarrow G}{\Sigma : \Delta; \nabla \Longrightarrow \reflectbox{N}\mathfrak{a}{:}A.G} \; \reflectbox{N}R$$

- Similarly, for program clauses $\reflectbox{N}\mathfrak{a}.D$ means "generate fresh name $\mathfrak{a}$ and proceed using D"

$$\frac{\Sigma : \nabla \vDash \reflectbox{N}\mathfrak{a}{:}A.C \quad \Sigma\#\mathfrak{a}{:}A : \Delta; \nabla, C \xrightarrow{D} A}{\Sigma : \Delta; \nabla \xrightarrow{\reflectbox{N}\mathfrak{a}{:}A.D} A} \; \reflectbox{N}L$$

Implementation

- Traditional approaches to implementing LP can be adapted:
  - SLD-resolution: depth-first proof search
  - backtracking via continuations
  - unification/constraint solving
- Key differences: need to
  - solve equations/freshness constraints over nominal terms
  - deal with nominal resolution correctly (modulo equivariance)
- These lead to nontrivial unification/resolution problems
- Overview in rest of talk

Nominal Unification

- Urban, Pitts, Gabbay [CSL 2003, TCS 2004]: developed an algorithm for
  - unifying nominal terms
  - solving freshness constraints

  with $O(n^2)$ complexity.
- BUT, nontrivial restrictions:

$$\mathfrak{a} \# t \quad (\mathfrak{a}\ \mathfrak{b}) \cdot t \quad \langle \mathfrak{a} \rangle t$$

  only ground names $\mathfrak{a}$ may appear in marked positions (not variables)
- What about the general case?

Full nominal unification

- Full nominal unification: allow name-variables anywhere.

$$
\begin{array}{rcl}
a, b, t, u & ::= & X \mid \langle\rangle \mid \langle t, u\rangle \mid f(t) \mid \langle a\rangle t \mid \Pi \cdot t \mid \mathfrak{a} \\
\Pi & ::= & (a\ b) \mid \mathrm{id} \mid \Pi \circ \Pi' \\
C & ::= & t \approx u \mid a \# t
\end{array}
$$

- *NP*-complete because guessing is needed to deal with swapping
- Reduction from GRAPH 3-COLORABILITY [C, ICALP 2004]

Equivariance

- In nominal logic, *truth is preserved by name-swapping*
- Two atomic formulas $R(\vec{t}), R(\vec{u})$ can be *logically equivalent* yet $\vec{t}, \vec{u}$ not *equal* as nominal terms.
- Example:

$$R(\mathfrak{a}) \iff R((\mathfrak{a}\ \mathfrak{b}) \cdot \mathfrak{a}) \approx R(\mathfrak{b}) \quad \text{but} \quad \mathfrak{a} \not\approx \mathfrak{b}$$

- Proof search based on $\approx$-unification is *incomplete*
- This can happen even for programs satisfying UPG name-groundness restriction
- In particular, needed to program generative relations.

Why is this hard?

- Let's take a little quiz.
- Satisfiable or not?

$$R((\mathsf{a}\ \mathsf{b})\cdot X, X, (\mathsf{b}\ \mathsf{c})\cdot Y, Y) \iff R(\mathsf{a}', \mathsf{b}', \mathsf{c}', \mathsf{d}')$$

- Satisfiable or not?

$$R((\mathsf{a}\ \mathsf{b})\cdot X, X, (\mathsf{c}\ \mathsf{d})\cdot Y, Y) \iff R(\mathsf{a}', \mathsf{b}', \mathsf{c}', \mathsf{d}')$$

Why is this hard?

- Let's take a little quiz.
- Satisfiable or not?

$$R((\mathfrak{a}\ \mathfrak{b}) \cdot X, X, (\mathfrak{b}\ \mathfrak{c}) \cdot Y, Y) \iff R(\mathfrak{a}', \mathfrak{b}', \mathfrak{c}', \mathfrak{d}')$$

  - No!
- Satisfiable or not?

$$R((\mathfrak{a}\ \mathfrak{b}) \cdot X, X, (\mathfrak{c}\ \mathfrak{d}) \cdot Y, Y) \iff R(\mathfrak{a}', \mathfrak{b}', \mathfrak{c}', \mathfrak{d}')$$

  - Yes: $X = \mathfrak{b}, Y = \mathfrak{d}$, permutation $(\mathfrak{a}\ \mathfrak{a}')(\mathfrak{b}\ \mathfrak{b}')(\mathfrak{c}\ \mathfrak{c}')(\mathfrak{d}\ \mathfrak{d}')$
  - Wasn't that easy?

Equivariant unification

- Idea: Reduce $R(\vec{t}) \iff R(\vec{u})$ to $\exists \pi . \pi \cdot \vec{t} = \vec{u}$.
- Allow permutation variables & inverses

$$
\begin{array}{rcl}
a, b, t, u & ::= & X \mid \langle\rangle \mid \langle t, u\rangle \mid f(t) \mid \langle a \rangle t \mid \Pi \cdot t \mid \mathfrak{a} \\
\Pi & ::= & (a\ b) \mid \mathrm{id} \mid \Pi \circ \Pi' \mid \Pi^{-1} \mid P \\
C & ::= & t \approx u \mid a \# t
\end{array}
$$

- t and u unify "up to a permutation" if $P \cdot t \approx u$ is satisfiable.
- Also *NP*-hard [C ICALP 2004]
- Rest of talk: show *NP* algorithm for EV unification

Our approach

- Phase 0: Always push permutation applications down to names/variables
- Phase I: Get rid of term symbols (unit, pair, functions, abstractions)
- Phase II: Get rid of permutation operations (id, inverse, composition, swapping)
- This leaves problems of the form $P \cdot a \approx b$, $P \cdot a \# b$ only.
- Phase III: Solve remaining problems using *permutation graphs*

Our approach (I)

- First, get rid of unit, pair, function symbols and abstractions:

$$
\begin{array}{rcl}
a, b, t, u & ::= & X \mid \langle\rangle \mid \langle t, u\rangle \mid f(t) \mid \langle a\rangle t \mid \Pi \cdot t \mid \mathfrak{a} \\
\Pi & ::= & (a\ b) \mid \mathrm{id} \mid \Pi \circ \Pi' \mid \Pi^{-1} \mid P \\
C & ::= & t \approx u \mid a \# t
\end{array}
$$

Our approach (I)

- Reduction rules for phase I:

$$
\begin{array}{lrcl}
(\#?_{\mathbf{1}}) & S, a \mathrel{\#?} \langle\rangle & \rightarrow_1 & S \\
(\#?_{\times}) & S, a \mathrel{\#?} \langle u_1, u_2 \rangle & \rightarrow_1 & S, a \mathrel{\#?} u_1, a \mathrel{\#?} u_2 \\
(\#?_{f}) & S, a \mathrel{\#?} f(u) & \rightarrow_1 & S, a \mathrel{\#?} u \\
(\#?_{abs}) & S, a \mathrel{\#?} \langle b \rangle u & \rightarrow_1 & \left\{ \begin{array}{c} S, a \approx? b \\ \vee\ S, a \mathrel{\#?} u \end{array} \right\}
\end{array}
$$

- Note the 2-way choice point in rule for abstraction
- Otherwise, rules similar to UPG algorithm

Our approach (I)

- Reduction rules for phase I:

$$
\begin{array}{lrcl}
(\approx?_{\mathbf{1}}) & S, \langle\rangle \approx? \langle\rangle & \rightarrow_1 & S \\
(\approx?_{\times}) & S, \langle t_1, t_2\rangle \approx? \langle u_1, u_2\rangle & \rightarrow_1 & S, t_1 \approx? u_1, t_2 \approx? u_2 \\
(\approx?_{f}) & S, f(t) \approx? f(u) & \rightarrow_1 & S, t \approx? u \\
(\approx?_{abs}) & S, \langle a\rangle t \approx? \langle b\rangle u & \rightarrow_1 & \left\{ \begin{array}{c} S, a \approx? b, t \approx? u \\ \vee\ S, a \#? u, t \approx? (a\ b)\cdot u \end{array} \right\} \\
(\approx?_{var}) & S, \Pi \cdot X \approx? t & \rightarrow_1 & S[X := \Pi^{-1}\cdot t], X \approx? \Pi^{-1}\cdot t \\
 & & & (\text{where } X \notin FV(t), X \in FV(S))
\end{array}
$$

- Note the 2-way choice point in rule for abstraction
- Otherwise, rules similar to UPG algorithm

Our approach (II)

- Next, get rid of complex permutation terms:

$$
\begin{array}{rcl}
a, b, t, u & ::= & X \mid \langle\rangle \mid \langle t, u\rangle \mid f(t) \mid \langle a\rangle t \mid \Pi \cdot t \mid \mathfrak{a} \\
\Pi & ::= & (a\ b) \mid \mathrm{id} \mid \Pi \circ \Pi' \mid \Pi^{-1} \mid P \\
C & ::= & t \approx u \mid a \# t
\end{array}
$$

Our approach (II)

- Reduction rules, phase II:

$$
\begin{array}{llcl}
(id) & S[\mathrm{id} \cdot v] & \rightarrow_2 & S[v] \\
(inv) & S[\Pi^{-1} \cdot v] & \rightarrow_2 & \exists X.S[X], \Pi \cdot X \approx v \\
(comp) & S[\Pi \circ \Pi' \cdot v] & \rightarrow_2 & \exists X.S[\Pi \cdot X], \Pi' \cdot v \approx X) \\
(swap) & S[(a\ a') \cdot v] & \rightarrow_2 & \left\{ \begin{array}{c} S[a], a' \approx v \\ \vee\ S[a'], a \approx v \\ \vee\ \exists X.S[X], v \approx X, a \mathrel{\#} X, a' \mathrel{\#} X \end{array} \right\} \\
(\#_Q) & S, Q \cdot v \mathrel{\#} w & \rightarrow_2 & \exists X.S, Q \cdot v \approx X, X \mathrel{\#} w
\end{array}
$$

- Note the 3-way choice point in rule for swapping

Our approach (III)

- The remaining constraints involve only names, variables, and permutation variables.

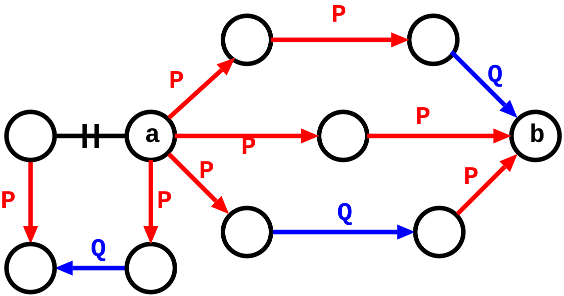
$$
\begin{array}{rcl}
a, b, t, u & ::= & X \mid \langle\rangle \mid \langle t, u\rangle \mid f(t) \mid \langle a\rangle t \mid \Pi \cdot t \mid \mathfrak{a} \\
\Pi & ::= & (a\ b) \mid \mathrm{id} \mid \Pi \circ \Pi' \mid \Pi^{-1} \mid P \\
C & ::= & t \approx u \mid a \# t
\end{array}
$$

- Conjunctive satisfiability for problems of this form can be solved by graph reduction in polynomial time.
- Idea: Build a graph with "equality", "freshness", and "permutation" edges; reduce using permutation laws

An example

- Here's how to reduce a permutation graph corresponding to:

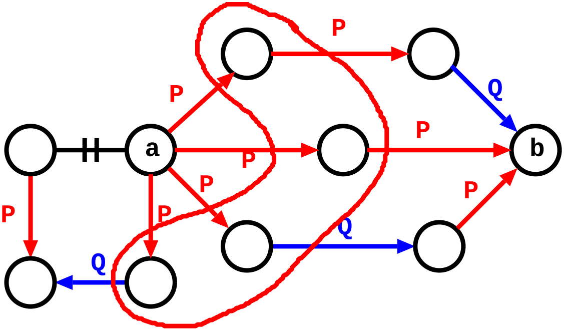
$$QPP\mathfrak{a} \approx \mathfrak{b} \qquad PQP\mathfrak{a} \approx \mathfrak{b} \qquad PP\mathfrak{a} \approx \mathfrak{b} \qquad P^{-1}QP\mathfrak{a} \# \mathfrak{a}$$



An example

- Here's how to reduce a permutation graph corresponding to:

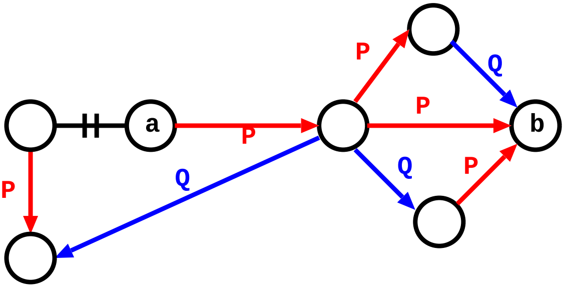
$$QPP\mathfrak{a} \approx \mathfrak{b} \qquad PQP\mathfrak{a} \approx \mathfrak{b} \qquad PP\mathfrak{a} \approx \mathfrak{b} \qquad P^{-1}QP\mathfrak{a} \# \mathfrak{a}$$



An example

- Here's how to reduce a permutation graph corresponding to:

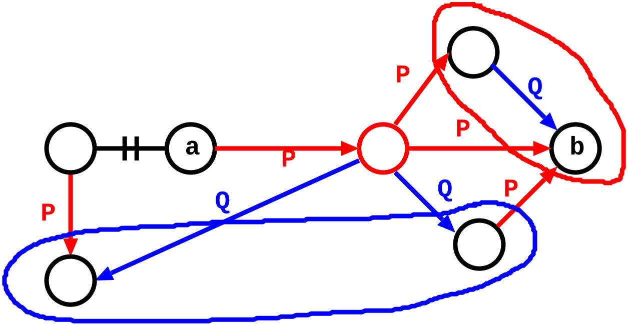
$$QP P\mathfrak{a} \approx \mathfrak{b} \qquad PQP\mathfrak{a} \approx \mathfrak{b} \qquad PP\mathfrak{a} \approx \mathfrak{b} \qquad P^{-1}QP\mathfrak{a} \# \mathfrak{a}$$



An example

- Here's how to reduce a permutation graph corresponding to:

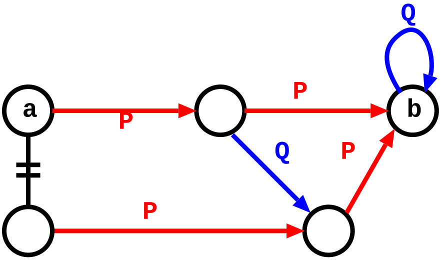
$$QPP\mathfrak{a} \approx \mathfrak{b} \qquad PQP\mathfrak{a} \approx \mathfrak{b} \qquad PP\mathfrak{a} \approx \mathfrak{b} \qquad P^{-1}QP\mathfrak{a} \# \mathfrak{a}$$



An example

- Here's how to reduce a permutation graph corresponding to:

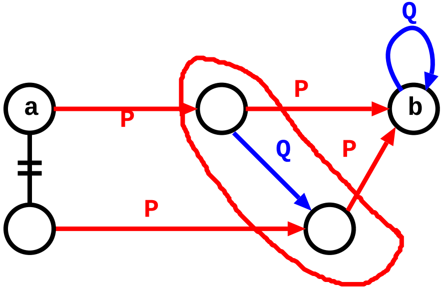
$$QPP\mathfrak{a} \approx \mathfrak{b} \qquad PQP\mathfrak{a} \approx \mathfrak{b} \qquad PP\mathfrak{a} \approx \mathfrak{b} \qquad P^{-1}QP\mathfrak{a} \# \mathfrak{a}$$



An example

- Here's how to reduce a permutation graph corresponding to:

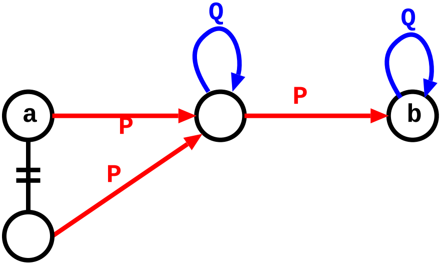
$$QP P \mathfrak{a} \approx \mathfrak{b} \qquad PQP\mathfrak{a} \approx \mathfrak{b} \qquad PP\mathfrak{a} \approx \mathfrak{b} \qquad P^{-1}QP\mathfrak{a} \# \mathfrak{a}$$



An example

- Here's how to reduce a permutation graph corresponding to:

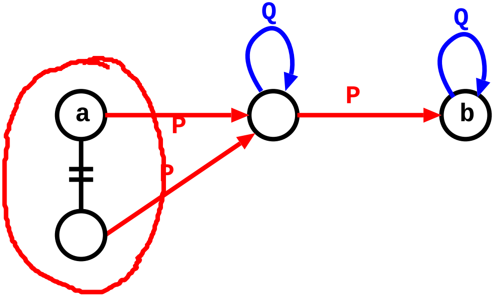
$$QPP\mathfrak{a} \approx \mathfrak{b} \qquad PQP\mathfrak{a} \approx \mathfrak{b} \qquad PP\mathfrak{a} \approx \mathfrak{b} \qquad P^{-1}QP\mathfrak{a} \# \mathfrak{a}$$



An example

- Here's how to reduce a permutation graph corresponding to:

$$QP P\mathfrak{a} \approx \mathfrak{b} \qquad PQP\mathfrak{a} \approx \mathfrak{b} \qquad PP\mathfrak{a} \approx \mathfrak{b} \qquad P^{-1}QP\mathfrak{a} \# \mathfrak{a}$$



- Unsatisfiable because $Q\mathfrak{a} \# \mathfrak{a}$ and $Q\mathfrak{a} \approx \mathfrak{a}$

Results

- Phase I (term reduction): *NP* time, finitary
- Phase II (permutation reduction): *NP* time, finitary
- Phase III (graph reduction): *P* time, unitary.
- Overall: *NP* time, finitely many answers.
- Variant of Phase I that is *P* time and unitary, but introduces lots of swappings

So this is a little depressing...

- At least it's decidable! (wasn't obvious)
- In practice many programs/clauses don't require full EVU
- Many *NP*-hard constraint problems have good behavior in common cases
  - programmers tend not to encode hard combinatorial problems as constraints
  - fixed parameter tractability?
- Seems painful to implement; unifiers complex
- There may be tractable special cases that permit some reasoning about generativity (my hope)
  - Do we really need equivariance?

Further reading

Survey/column (more expository):

- Nominal logic and abstract syntax, C, SIGACT News 2005

Primary sources:

- A new approach to abstract syntax involving binders, Gabbay & Pitts, Formal Aspects of Computing 2002
- Nominal logic, Pitts, Inf. Comput. 2003
- Nominal logic programming, C, thesis 2004
- Completeness and Herbrand theorems for nominal logic, C, JSL 2006

Future work/open problems

- Efficient implementation (perhaps via translation to CLP?)
- Variants of NL (e.g. equivariance vs. orderings)
- Proof theory of NL and nominal type theory
- Beyond binding: practical reasoning with generativity (states in automata, nonces in security, ids in program analysis)
- Beyond names: practical support for reasoning modulo structural congruences (e.g. π-calculus)

Conclusions

- Nominal abstract syntax is a new way of reasoning about and programming with names and binding.
  - $+$ Nominal logic programs frequently direct translations from ordinary informal presentations.
  - $+$ Can be used to reason about generativity, name-inequality
  - $+/-$ Names kept "abstract" via equivariance; complicates resolution
  - $-$ Lacks HOAS-style built-in substitution