A Sequent Calculus for Nominal Logic

Murdoch J. Gabbay

Work with James Cheney

July 15, 2004

Sentiments

LICS'99 in Trento was my first international conference. It left me with a particular affection for LICS. I am happy to be back!

Since then I have met many people. Some invited me to visit in universities all over the world. Thank you.

James Cheney should be here. He has two other papers at this conference.

Q. Which is worse: ignorance or apathy?

A. Who knows, who cares?

Please ask questions. I would.

This will be a simple talk. . .

. . . here it is in one slide

- **Slogan: Names are a datatype.**
- **Question 1: What are the introduction and elimination rules for its elements?**
- **Question 2: What logic programming languages arise from them?**

Roughly speaking, in the literature ‘Nominal’ refers to the Slogan, and ‘Fresh’ refers to Question 1.

Take α-equivalence on names (variable symbols in abstract syntax)

$\lambda f, a.\ f\,a \ \equiv_\alpha\ \lambda a, f.\ a\,f$	Untyped λ-calculus
$\mathsf{fix}\ a.\ S \cup \bigcup a \ \equiv_\alpha\ \mathsf{fix}\ b.\ S \cup \bigcup b$	Typed λ-calculus
$\exists b.\ \forall a.\ a = b \ \equiv_\alpha\ \exists a.\ \forall b.\ b = a$	Logic
$a[b].\overline{b}b \ \equiv_\alpha\ a[c].\overline{c}c$	π-calculus
$a[a \mapsto b] \ \equiv_\alpha\ c[c \mapsto b]$	Explicit substitution (?)

Add substitution and freshness

Operational semantics and logical deduction (and structural congruence, . . .) introduce notions of substitution. . .

$$(\lambda a.s)t \;\rightsquigarrow\; s[a{\mapsto}t] \qquad \overline{a}a \mid a[b].\overline{b}b \;\rightsquigarrow\; \overline{b}b$$

$$\frac{\Gamma, P[a{\mapsto}t] \vdash C}{\Gamma, \forall a.\, P \vdash C}\,(a \text{ fresh})$$

. . . and freshness to the mix:

$$(\lambda b.\lambda a.b)a \;\rightsquigarrow\; \lambda a'.a \qquad \overline{a}a \mid \nu[a]\overline{b}a \;\rightsquigarrow\; \overline{a}a \mid \overline{b}a'$$

$$\frac{x = x \vdash x' = y}{x = x \vdash (\forall x.\, x = y)}$$

Also in tandem, e.g. a reduction in an explicit substitution calculus requiring a capture-avoiding renaming $(\lambda a.b)[b{\mapsto}a] \;\rightsquigarrow\; \lambda a'.a$. Choose your favourite example.

Add unknowns

When we consider Rules, we also need unknowns:

$$\frac{\Gamma \vdash P(a)}{\Gamma \vdash \forall a.\, P(a)} \qquad (a \text{ fresh})$$

$$\frac{C[P] \rightsquigarrow C[P']}{C[\nu[a]P] \rightsquigarrow C[P']} \qquad (a \text{ fresh})$$

$$u[a{\mapsto}X][b{\mapsto}Y] \equiv_\alpha u[b{\mapsto}Y][a{\mapsto}X[b{\mapsto}Y]] \qquad (a \text{ fresh for } Y)$$

$$(\lambda a.X)[b{\mapsto}Y] \rightsquigarrow \lambda a.(X[b{\mapsto}Y]) \qquad (a \text{ fresh})$$

$$(\lambda a.X)[b{\mapsto}Y] \rightsquigarrow \lambda a'.((a'\ a)X[b{\mapsto}Y]) \qquad (a' \text{ fresh})$$

Is this simple? Not really.

Given that **names are a datatype**, that's a, b, c, what do we have?
Consider a canonical problem:

$$(\lambda a.X)[b \mapsto Y] \quad \rightsquigarrow \quad \lambda a'.((a'\ a)X[b \mapsto Y]) \quad (a' \text{ fresh}).$$

Conundrum 1: How is a renamed to a' in the unknown X? Conundrum 2: How is a' fresh for the unknown Y?

We need to answer these questions in order to give a **logic for specifying rules on syntax** in which **names are a datatype**.

This is **Nominal Logic**.

Sorts and terms of (Sequent) Nominal Logic

Assume a sort system τ with function sorts $\tau \longrightarrow \tau$. Assume a sort of atoms $\mathtt{A}$. (**Names are a datatype.**)

Terms are simply-typed λ-terms

$$s, t, a, b, \ldots ::= x \mid \mathsf{c} \mid \lambda x.t \mid t\, t'.$$

Call terms of sort $\mathtt{A}$ atoms and write them a, b, a', f, g, etc.

Constants $\mathtt{c}$ may have any sort. Assume $\mathtt{swap}_\tau : \mathtt{A} \longrightarrow \mathtt{A} \longrightarrow \tau \longrightarrow \tau$ for each sort τ. Abbreviate $\mathtt{swap}_\tau\ a\ a'\ t$ to $(a\ a') \cdot t$ and call it a swapping of a and a' applied to t. That addresses conundrum 1.

The rest is standard. Identify $\equiv_{\beta\eta}$-terms. $V(t)$ is free variables as usual. $t[x \longmapsto s]$ is capture-avoiding as usual.

Expressivity

We can express object-level syntax using terms. E.g. assume sort Λ and constructors $\mathtt{Var} : \mathtt{A} \to \Lambda$, $\mathtt{App} : \Lambda \to \Lambda \to \Lambda$, $\mathtt{Lam} : \mathtt{A} \to \Lambda$. Then

- $\mathtt{Var}a$ is 'the variable a'.
- $\mathtt{App}xx'$ is 'the unknown λ-term x applied to the unknown λ-term x''.
- $\mathtt{Lam}f\mathtt{Lam}a\mathtt{App}(\mathtt{Var}f)(\mathtt{Var}a)$ is '$\lambda f, a.fa$'.
- $(a\ b) \cdot x$ is 'swap a and b in the unknown λ-term x'.

Propositions or formulae

$$\begin{array}{rcl} P & ::= & p(ts) \mid P \wedge P \mid P \vee P \mid P \Rightarrow P \\ & \mid & \top \mid \bot \mid \forall x.\, P \mid \exists x.\, P \mid И n.\, P. \end{array}$$

p, q, r are predicate constant symbols each with an arity $\tau_1 \cdots \tau_n$ (i.e. the list of sorts of arguments). Assume equality $= : \tau\tau$ and freshness $\# : \mathtt{A}\tau$.

The rest is standard. $V(P)$ and $P[x \mapsto s]$ as usual. Equate α-equivalent formulae as usual.

The intuition of $И n.\, P(n)$ is 'P holds of a fresh n'. The semantics is 'P holds of cofinitely many n', that addresses conundrum 2.

Also a propos conundrum 2, we can state individual freshnesses using the predicate constant $a \# x$.

Expressivity

Recall the sort Λ. Abuse notation and write $\mathtt{Var}a$ as a, $\mathtt{App}tt'$ as tt', and $\mathtt{Lam}at$ as $\lambda a.t$. We can now express alpha-equivalence as a relation $\underset{\alpha}{\equiv} : \Lambda\Lambda$ given by formulae

$$a \underset{\alpha}{\equiv} a$$

$$t \underset{\alpha}{\equiv} t' \ \wedge \ s \underset{\alpha}{\equiv} s' \implies st \underset{\alpha}{\equiv} s't'$$

$$\left(\mathit{И}n.\ (n\ a) \cdot t = (n\ a') \cdot t'\right) \implies \lambda a.t \underset{\alpha}{\equiv} \lambda a'.t'.$$

Expressivity

Introduce $\mathtt{Sub} : \Lambda \to \mathtt{A} \to \Lambda \to \Lambda$ and sugar it to $t[a \mapsto t']$. We can express operational semantics $\rightsquigarrow : \Lambda\Lambda$:

$$(\lambda x.s)t \rightsquigarrow s[x \mapsto t] \qquad (ss')[x \mapsto t] \rightsquigarrow s[x \mapsto t]s'[x \mapsto t]$$

$$И n.\left((\lambda a.u)[x \mapsto t] \rightsquigarrow \lambda n.((n\ a) \cdot u)[x \mapsto t]\right)$$

Freshness and New

We can reexpress using $\#$:

$$a \equiv_\alpha a \qquad t \equiv_\alpha t' \wedge s \equiv_\alpha s' \implies st \equiv_\alpha s't'$$

$$n\#t, t' \wedge (n\ a) \cdot t = (n\ a') \cdot t' \implies \lambda a.t \equiv_\alpha \lambda a'.t'$$

$$(\lambda x.s)t \rightsquigarrow s[x \mapsto t] \qquad (ss')[x \mapsto t] \rightsquigarrow s[x \mapsto t]s'[x \mapsto t]$$

$$n\#a, u, x, t \implies (\lambda a.u)[x \mapsto t] \rightsquigarrow \lambda n.((n\ a) \cdot u)[x \mapsto t]$$

Notice that these are all Horn.

Relation of И and $\#$

The following are theorems of Nominal Logic:

$$\forall x.\ И n.\ n\#x$$

$$И n.\ P(n, xs) \iff \forall n.\ n\#xs \Rightarrow P(n, xs)$$

$$И n.\ P(n, xs) \iff \exists n.\ n\#xs \wedge P(n, xs).$$

Here $V(P) = \{n, xs\}$ and $n\#xs$ denotes a conjunction of freshness assertions.

These say “You can always generate a fresh atom” and “If something holds of *some* fresh atom, it holds of *all* fresh atoms”. Useful for proofs, see ‘equivariance’ in the next slide.

Taking stock

We have a logic with built-in names $a, b, c : \mathtt{A}$, renaming $(a\ b)$, freshness $\#$, and new name $И$. It has useful meta-theoretic properties:

- If $\Gamma \vdash C$ then $\Gamma[x{\mapsto}t] \vdash C[x{\mapsto}t]$. (Substitution Lemma)
- Cut is an admissible rule. (Cut-elimination)
- If $\Gamma \vdash C$ then $\Gamma \vdash (a\ b) \cdot C$. (Equivariance)

The last rule means that if you know C (an inductive hypothesis, say), then it is safe to suppose also $(a\ b) \cdot C$. Suppose $C(a)$ is '$\lambda a.a$ is a normal form'. Then we know '$\lambda b.b$ is a normal form'.

It is an axiom that if $\Gamma \vdash a\#x$ and $\Gamma \vdash b\#x$ then $\Gamma \vdash x = (a\ b) \cdot x$. Therefore if we can prove a and b fresh for parameters in C, we can freely deduce $C(b)$ from $C(a)$. This is 'we assume a is fresh'.

Timeline (or: suggested reading)

2001 Gabbay thesis introduces FM sets. FreshML project implements operational semantics for it based on ML.

2001 Pitts Nominal Logic introduces Hilbert-style axiomatisation of $\#$ and $И$. Uses Atoms-as-constants (AAC).

2003 Gabbay Fresh Logic introduces Natural-Deduction style system for $\#$ and $И$, with proof-normalisation and sound and complete semantics (in FM sets). Uses AAC.

2004 A Sequent Calculus for Nominal Logic introduces Sequent-style system with cut-elimination, and also demonstrates it as a logic programming language. Uses Atoms-as-variables (AAV).

Also of relevance: Nominal Unification by Urban, Pitts,and Gabbay, and Nominal Rewriting by Fernández, Gabbay, and Mackie.

Deduction rules

$$\frac{\Gamma, n\#ts \vdash C}{\Gamma \vdash C}\,(new\mathbb{A}) \quad (n \notin V(\Gamma, C))$$

$$\frac{\Gamma, a\#ts \vdash P\{a/n\}}{\Gamma, a\#ts \vdash \mathsf{И}n.\, P}\,(\mathsf{И}R) \quad (P \equiv P'[n, ts])$$

$$\frac{\Gamma, a\#ts, P\{a/n\} \vdash C}{\Gamma, a\#ts, \mathsf{И}n.\, P \vdash C}\,(\mathsf{И}L) \quad (P \equiv P'[n, ts])$$

Deduction rules

$(new\mathbb{A})$says "we can always introduce a fresh atom; call it n and say 'it's fresh"'.

$(И L)$and $(И R)$are symmetric so we consider just $(И R)$. The best way of understanding this rule is to consider the case when every t in ts is an x. Then:

$$\frac{\Gamma, a\#xs \vdash P[n\mapsto a]}{\Gamma, a\#xs \vdash И n.\, P} \qquad (V(P) = \{n, xs\})$$

The more complex form is obtained just by observing that, for good meta-theoretic properties, we want the deduction rules to each be closed under substituting ts for xs. We do not want to assume $n \equiv a$ since the ts might contain a. It is a bit like the existential right rule.

Deduction rules

$$\frac{\Gamma, a\#b \vdash C \quad \Gamma, a = b \vdash C}{\Gamma \vdash C}\,(case\mathbb{A}) \qquad \frac{}{\Gamma, a\#a \vdash C}\,(\#\mathbb{A})$$

$$\frac{\Gamma, (a\,b)\cdot t = t \vdash P}{\Gamma,\ a\#t, b\#t \vdash P}\,(\pi\#) \qquad \frac{\Gamma, P \vdash C}{\Gamma, (a\,b)\cdot P \vdash C}\,(\pi L)$$

In Hilbert-style these say:

$$a\#b \vee a = b \qquad \neg(a\#a)$$

$$a, b\#t \Rightarrow (a\,b)\cdot t = t \qquad (a\,b)\cdot P \implies P.$$

Deduction rules

Finally, there are some equational rules:

$$\frac{\Gamma, A \vdash C}{\Gamma \vdash C}\,(\mathcal{A}L) \quad (A \in \mathcal{A})$$

$$\mathcal{A} = \left\{ \begin{array}{l} (a\ a)\cdot t = t, \quad (a\ b)\cdot(a\ b)\cdot t = t \\ (a\ b)\cdot a = b, \quad (a\ b)\cdot \mathsf{c} = \mathsf{c}, \\ (a\ b)\cdot[t\ u] = [(a\ b)\cdot t]\ [(a\ b)\cdot u], \\ (a\ b)\cdot \lambda x.t = \lambda x.(a\ b)\cdot[t\{(a\ b)\cdot x/x\}] \end{array} \right\}$$

Logic programming (Horn)

First-order nominal Horn clause goals and program clauses:

$$\begin{array}{rcl} G & ::= & \top \mid p(ts) \mid G \wedge G \mid \exists x.G \mid И n.G \\ D & ::= & \top \mid p(ts) \mid D \wedge D \mid G \supset D \\ & \mid & \forall x.D \mid И n.D \end{array}$$

The И-free fragment is essentially Horn clauses.

Uniform proof search applies just right-rules or $(new\mathbb{A})$ until the goal is atomic, then just left-rules or $(new\mathbb{A})$. The above is a logic programming language: if $D \vdash G$ is derivable then a uniform derivation exists.

Note that $(new\mathbb{A})$ may be applied at any point. This is technically necessary because it cannot be permuted up past $(\exists R)$: the witness t may mention the fresh variable generated by $(new\mathbb{A})$.

Logic programming (Harrop)

First-order nominal hereditary Harrop goals G and program clauses D:

$$\begin{array}{rcl} G & ::= & \top \mid p(ts) \mid G \wedge G \mid \exists x.G \mid \mathit{И} n.G \\ & \mid & \forall x.G \mid G \vee G \mid D \supset G \\ D & ::= & \top \mid p(ts) \mid D \wedge D \mid G \supset D \\ & \mid & \forall x.D \mid \mathit{И} n.D \end{array}$$

This is also a logic programming language.

Adding atomic freshness and equality goals is no problem for uniform proof search since it does not comment on how atomic goals are to be derived.

Conclusions and future work (apologies to Cheney, Pitts, and Others)

This paper explores some of the consequences of taking names as a datatype. There are many more:

- (Cheney) Efficient logic programming and implementation.
- (Pitts) Semantic/denotational methods to prove program properties, using nominal domains.
- (Gabbay) Complex holes: What languages, logics, calculi, and semantics, arise from contexts and unknowns $C[X]$, for which substitution is not capture-avoiding? What are unification and rewriting in the presence of the above, or closure conditions ('t is closed' instead of just freshness $a\#t$), e.g. for closed rewriting.
- (Others: Urban, Fernández, Shinwell, following paper . . .) I'm in the happy position of having to confess that I'm losing track.

So someone asked 'why not use name-for-name substitution'

We use swappings $(a\ b)$ instead of substitutions $[a \mapsto b]$ because it is easier to logic program a theory for α-equivalence using it. Thus:

$$\frac{a \neq b}{a\#b} \qquad \frac{a\#s, t}{a\#st} \qquad a\#\lambda a.t \qquad \frac{a\#t \quad a \neq b}{a\#\lambda b.t}$$

$$a \equiv_\alpha a \qquad \frac{s \equiv_\alpha s' \quad t \equiv_\alpha t'}{st \equiv_\alpha st'}$$

$$\frac{s \equiv_\alpha t}{\lambda a.s \equiv_\alpha \lambda a.t} \qquad \frac{s \equiv_\alpha (a\ b) \cdot t \quad b\#s}{\lambda a.s \equiv_\alpha \lambda b.t}$$

This said, a theory based on substitutions is possible [Gabbay, unpublished].