Nominal Techniques: from Nominal Logic to Nominal Games

Nikos Tzevelekos
Oxford University Computing Laboratory

LINT workshop, Amsterdam, December 2008
Nominal Techniques := formal techniques for names,
Names := identifiers/atoms in constructions.

There are two parts in this talk; nominal techniques for:
- abstract syntax,
- semantics.

Different issues, same techniques.
From Nominal Logic to Nominal Sets
\[ \int_0^1 f(x) \, dx \]

In the above expression we say that \( x \) is bound in \( \int_0^1 f(x) \, dx \). Alternatively, the constructor \( \int_0^1 \, dx \) binds \( x \).

This is a very well understood notion: for example, we can easily spot the error below.

\[ \int_0^1 \int_0^1 xy \, dx \, dy = \int_0^1 \int_0^1 xx \, dx \, dy = \int_0^1 \frac{1}{3} \, dx = \frac{1}{3} \]
Consider the simply-typed $\lambda$-calculus.

**Types**

$A, B ::= B \mid A \rightarrow B$

**Terms**

$M, N ::= x \mid MN \mid \lambda x.M$

The constructor $\lambda x.$ is a binder. We consider terms *modulo choices of names in binding positions*. That is,

$$\text{Term} ::= \text{Var} + (\text{Term} \times \text{Term}) + (\text{Var} \times \text{Term})$$

$$\alpha\text{Term} ::= \text{Term}/=\alpha$$

where $M =_\alpha M'$ if $M$ and $M'$ differ solely in their choices of bound names.
The problem

Term := \( \text{Var} + (\text{Term} \times \text{Term}) + (\text{Var} \times \text{Term}) \)

\( \alpha \text{Term} := \text{Term}/_{=\alpha} \)

Most of the times:

- we say that we use \([M]_\alpha \in \alpha \text{Term},\)
- but in fact we use (specific!) \(M' \in [M]_\alpha.\)

This introduces (at best) an amount of informality in definitions and proofs regarding \(\alpha\)-terms.
Typing rules for $\alpha$-terms.

$$(x:A) \in \Gamma \quad \frac{\Gamma \vdash M : A \rightarrow B \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B} \quad \frac{\Gamma, x : A \vdash M : B \quad x \notin \text{dom}(\Gamma)}{\Gamma \vdash \lambda x.M : A \rightarrow B}$$

What does this formally mean?

- That $\Gamma \vdash [M]_\alpha : A$ has a derivation if $\Gamma \vdash M : A$ does?
Typing rules for $\alpha$-terms.

\[
\begin{align*}
(x:A) \in \Gamma & \quad \Gamma \vdash M : A \rightarrow B \quad \Gamma \vdash N : A \\
\Gamma & \vdash MN : B \\
\Gamma, x : A & \vdash M : B \quad x \notin \text{dom}(\Gamma) \\
\Gamma & \vdash \lambda x. M : A \rightarrow B
\end{align*}
\]

What does this formally mean?

- That $\Gamma \vdash [M]_\alpha : A$ has a derivation if $\Gamma \vdash M : A$ does?
- That $\Gamma \vdash [M]_\alpha : A$ has a derivation if $\Gamma \vdash M' : A$ does, some $M' \in [M]_\alpha$?
Typing rules for $\alpha$-terms.

\[
\frac{(x:A) \in \Gamma}{\Gamma \vdash x : A} \quad \frac{\Gamma \vdash M : A \to B \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B} \quad \frac{\Gamma, x : A \vdash M : B \quad x \not\in \text{dom}(\Gamma)}{\Gamma \vdash \lambda x. M : A \to B}
\]

What does this formally mean?

- That $\Gamma \vdash [M]_\alpha : A$ has a derivation if $\Gamma \vdash M : A$ does?
- That $\Gamma \vdash [M]_\alpha : A$ has a derivation if $\Gamma \vdash M' : A$ does, some $M' \in [M]_\alpha$?
- That derivations are considered modulo $\alpha$-equivalence and that $\Gamma \vdash [M]_\alpha : A$ has a derivation $[D]_\alpha$ if $\Gamma \vdash M' : A$ has a derivation $D$, some ("sufficiently fresh") $M' \in [M]_\alpha$?
Can’t we do things in a way that is both simple and formal?

In particular, can’t we have a syntax which directly incorporates name-binding?

\[
\alpha \text{Term} := \text{Var} + (\alpha \text{Term} \times \alpha \text{Term}) + \langle \text{Var} \rangle \alpha \text{Term}
\]
[Pitts, 2001]: “A first order theory of names and binding”. A many-sorted logic with:

- sorts for data, names and name-abstractions:
  \[
  S ::= A \mid D \mid \langle A \rangle S
  \]

- constructors for functions; in particular:
  \[
  \begin{align*}
  &\text{if } t_1, t_2 : A, t : S \text{ then } (t_1 \ t_2) \cdot t : S, \\
  &\text{if } t_1 : A, t : S \text{ then } t_1.t : \langle A \rangle S,
  \end{align*}
  \]

- constructors for relations; in particular:
  \[
  \begin{align*}
  &\text{if } t_1 : A, t : S \text{ then } t_1 \# t \text{ is a formula,}
  \end{align*}
  \]

- quantifiers \( \forall, \exists, \ni \),

- axioms.
Example axioms:

\[ \forall a : A. \phi(\vec{x}) \iff \exists a : A. a\#\vec{x} \land \phi(\vec{x}) \]  

(Q)
Example axioms (note sorts should match):

\[ \forall a : A. \phi(\vec{x}) \iff \exists a : A. a \# \vec{x} \land \phi(\vec{x}) \]  
\text{(Q)}

\[ (a \ a') \cdot (b \ b') \cdot x = ((a \ a') \cdot b \cdot (a \ a') \cdot b') \cdot (a \ a') \cdot x \]  
\text{(E1)}

\[ b \# x \Rightarrow (a \ a') \cdot b \# (a \ a') \cdot x \]  
\text{(E2)}

\[ a \# x \land a' \# x \Rightarrow (a \ a') \cdot x = x \]  
\text{(F1)}

\[ a \cdot x = a' \cdot x' \iff (a = a' \lor a' \# x) \land x' = (a \ a') \cdot x \]  
\text{(A1)}

NL gives us a strong handle on names. For example:

- \[ \phi(\vec{x}) \iff \phi((a \ a') \cdot \vec{x}) \]
- \[ (\exists a : A. a \# \vec{x} \land \phi(\vec{x})) \iff (\forall a : A. a \# \vec{x} \Rightarrow \phi(\vec{x}')) \]
- \[ b \# a \cdot x \iff b = a \lor b \# x \]
- \[ a \cdot x = a' \cdot x' \iff \forall b : A. (a\ b) \cdot x = (a'\ b) \cdot x' \]
Consider a countably infinite set $A$ of *atoms* and its group of finite permutations $\text{PERM}(A)$.

A *nominal set* is a pair $(X, \cdot)$ such that $X$ is a set and

- $\cdot : \text{PERM}(A) \times X \to X$ is an action on $X$,
  - i.e. $\text{id} \cdot x = x$, $\pi \cdot (\pi' \cdot x) = (\pi \circ \pi') \cdot x$,
- each $x \in X$ has finite support,
  - i.e. there exists finite $S \subseteq A$,
    $\forall \pi. (\forall a \in S. \pi(a) = a) \implies \pi \cdot x = x$,

In particular, each $x \in X$ has a least support, $\text{supp}(x)$.
Consider a countably infinite set $A$ of atoms and its group of finite permutations $\text{PERM}(A)$.

A nominal set is a pair $(X, \cdot)$ such that $X$ is a set and

- $\cdot$ : $\text{PERM}(A) \times X \to X$ is an action on $X$,
  - i.e. $\text{id} \cdot x = x$, $\pi \cdot (\pi' \cdot x) = (\pi \circ \pi') \cdot x$,

- each $x \in X$ has finite support,
  - i.e. there exists finite $S \subseteq A$,
    $\forall \pi. (\forall a \in S. \pi(a) = a) \implies \pi \cdot x = x$,

In particular, each $x \in X$ has a least support, $\text{supp}(x)$.
For example, any set is trivially nominal, $A$ is a nominal set, products of nominal sets are nominal, etc.

Nominal sets derived from FM permutation models of ZFA.
Nominal Logic in Nominal Sets

Nominal sets provide a model for NL:

- map each $D$ to some $X_D$,
- map $A$ to $\mathbb{A}$,
- for each $a, b \in \mathbb{A}$ and $x \in X$ take:
  - $(a \ b) \cdot x$ as given,
  - $a \# x$ if $a \notin \text{supp}(x)$,
  - $a.x := \{(b, y) \mid (a = b \lor b \# y) \land y = (a \ b) \cdot x\}$. 

Thus, $\langle A \rangle X := \{a.x \mid a \in \mathbb{A} \land x \in X\}$.

- etc.

\[
\begin{align*}
  & t_1, t_2 : A \quad t : S \\
  & \quad \frac{}{(t_1 \ t_2) \cdot t : S} \\
  & t_1 : A \quad t : S \\
  & \quad \frac{}{t_1.t : \langle A \rangle S} \\
  & t_1 : A \quad t : S \\
  & \quad \frac{}{t_1 \# t : \text{wff}}
\end{align*}
\]
Nominal techniques introduced in [Gabbay & Pitts’99]. The original presentation was set-theoretic, in ZFA.

Nominal techniques have had a huge impact on abstract syntax:

- nominal algebras,
- nominal rewriting systems,
- nominal theorem provers,
- nominal metalanguages, etc.

See e.g. works of: Cheney, Gabbay, Mathijssen, Pitts, Shinwell, Urban, and collaborators.

but also in semantics via nominal sets:

- nominal domains [Shinwell & Pitts],
- nominal games.
From Nominal Sets to Nominal Games
Denotational Semantics assigns to terms denotations in some abstract mathematical domain (a category).

- Issues with α-equivalence disappear at the level of semantics.
- Different approach: \( \lambda x. f(x) \) represents
  - a name-abstraction (no comp. content) in syntax,
  - an exponential (a function) in semantics.
Denotational Semantics assigns to terms denotations in some abstract mathematical domain (a category).

- Issues with $\alpha$-equivalence disappear at the level of semantics.
- Different approach: $\lambda x.f(x)$ represents
  - a name-abstraction (no comp. content) in syntax,
  - an exponential (a function) in semantics.
- But there is still space for nominal techniques, in languages with names:
  - names for references,
  - names for objects, exceptions,
  - names for threads, channels, etc.
The nu-calculus

Types  \( A, B ::= \mathbf{B} \mid \mathbf{A} \mid A \to B \)

Terms  \( M, N ::= t \mid f \mid x \mid MN \mid \lambda x.M \mid \text{if } M \text{ then } N_1 \text{ else } N_2 \mid a \mid [M_1 = M_2] \mid \nu a.M \)

- base types: booleans, names
- \( \lambda \)-calculus over booleans
- name
- name-equality test
- name-abstraction
### The nu-calculus

#### Types

\[ A, B ::= B \mid A \mid A \rightarrow B \]  
base types: booleans, names

#### Terms

\[ M, N ::= t \mid f \mid x \mid MN \mid \lambda x.M \]  
\[ \lambda\text{-calculus over booleans} \]

\[ \mid \text{if } M \text{ then } N_1 \text{ else } N_2 \]  
\[ \text{name} \]

\[ \mid a \]  
\[ \text{name} \]

\[ \mid [M_1 = M_2] \]  
\[ \text{name-equality test} \]

\[ \mid \nu a. M \]  
\[ \text{name-abstraction} \]

- Terms form a nominal set \((a \in A)\).
- \(\nu a. M\) creates a *fresh* name \(a\) for \(M\) – it is a binder.
- Terms are taken modulo \(\alpha\)-equivalence (wrt both bindings).
The nu-calculus

Types  \( A, B ::= B \mid A \mid A \rightarrow B \)  

Terms \( M, N ::= t \mid f \mid x \mid MN \mid \lambda x.M \mid \text{if } M \text{ then } N_1 \text{ else } N_2 \mid a \mid [M_1 = M_2] \mid \nu a.M \)

- Terms form a nominal set \( (a \in A) \).
- \( \nu a.M \) creates a fresh name \( a \) for \( M \) – it is a binder.
- Terms are taken modulo \( \alpha \)-equivalence (wrt both bindings).

“Names are created with local scope, can be tested for equality and can be passed around via function application, but that is all.”
Reduction happens in *state-environments*. Reduction rules include:

- $\lambda$-calculus rules (call by value),
- nominal rules:

$$
S, \nu a. M \rightarrow S \oplus a, M \\
S, [a = b] \rightarrow S, f (a \neq b) \\
S, [a = a] \rightarrow S, t
$$

So reduction is non-deterministic, in a “nominal way”.

Two terms are *(observationally) equivalent* ($\equiv$) if no context of type $C[\_] : B$ can distinguish them.
This simple calculus is quite expressive. For example:

\[ \nu a. \lambda x. a \not\equiv \lambda x. \nu a. a \]
\[ \nu a. \lambda x. [a = x] \cong \lambda x. f \]

...\n
As \( n \) ranges in \( \omega \) we get infinitely many (observationally) different terms of type \( A \rightarrow A \) by:

\[ \nu a_1. ... \nu a_n. \lambda x. \text{if} [x = a_1] \text{then} a_2 \text{else if} [x = a_2] \text{then} a_3 \text{else} ... \text{if} [x = a_{n-1}] \text{then} a_n \text{else} a_1 \]

Although introduced in [Pitts & Stark, 1993], its first fully abstract semantics was given in [AGMOS, 2004].
[AGMOS’04] and [Laird’04] introduced Nominal Games.

Names excluded, the $\nu$-calculus is game-semantically easy. The extra feature needed was *plays-with-names*:

- names in plays as first-class moves (like integers),
- strategies unable to distinguish between fresh names (unlike integers),
- some notion of *local state* (or *name-availability*).
[AGMOS’04] and [Laird’04] introduced Nominal Games.

Names excluded, the $\nu$-calculus is game-semantically easy. The extra feature needed was \emph{plays-with-names}:

- Names in plays as first-class moves (like integers),
- Strategies unable to distinguish between fresh names (unlike integers),
- Some notion of \emph{local state} (or \emph{name-availability}).

All of the above achieved elegantly by use of nominal sets at the basis of moves, plays, strategies, etc.

This is no coincidence: the first two specifications go back to the notions of atomic, bindable names at the very basis of nominal techniques!
Conclusions in Nominal Games

[What has been accomplished] A series of FA models:

- for the $\nu$-calculus [AGMOS’04, Tz’07],
- $\nu$-calc.+HO-references, exceptions [Tz’07, Tz’08],
- $\nu$-calc.+pointers [Laird’04, Laird’08],
- $\nu$-calc.+HO-concurrency [Laird’06],
- $\nu$-calc.+int-references [Tz & Murawski’08].

[What to do next] Examine (at least):

- more nominal languages (...),
- decidability of nominal languages,
- other structures under the “nominal lense” (e.g. AJM-games)!
[What has been accomplished] A series of FA models:

- for the $\nu$-calculus [AGMOS’04, Tz’07],
- $\nu$-calc.+HO-references, exceptions [Tz’07, Tz’08],
- $\nu$-calc.+pointers [Laird’04, Laird’08],
- $\nu$-calc.+HO-concurrency [Laird’06],
- $\nu$-calc.+int-references [Tz & Murawski’08].

[What to do next] Examine (at least):

- more nominal languages (...),
- decidability of nominal languages,
- other structures under the “nominal lense” (e.g. AJM-games)!

THANKS!