Names and Symmetry in Computer Science

Andrew Pitts

UNIVERSITY OF CAMBRIDGE
Computer Laboratory

LICS 2015 Tutorial

An introduction to
nominal techniques
motivated by

Programming language semantics

Automata theory

Constructive type theory

Compositionality in semantics

> The meaning of a compound phrase should be a well-defined function of the meanings of its constituent subphrases.

Although one can define behaviour of whole

C/Java/OCaml/Haskell... <u>programs</u>

(e.g. via an abstract machine) it's often more illuminating, but harder, to give a semantics for individual

C/Java/OCaml/Haskell... <u>statement-forming constructs</u>

as operations for composing suitable mathematical structures.

Compositionality in semantics

The meaning of a compound phrase should be a well-defined function of the meanings of its constituent subphrases.

Denotational semantics are compositional:

- Each program phrase P is given a denotation $[\![P]\!]$—a mathematical object representing the contribution of P to the meaning of *any* complete program in which it occurs.
- The denotation of a phrase is a function of the denotations of its subphrases.

Question: are these OCaml expressions behaviourally equivalent?

$$\begin{array}{l}\texttt{let}\,a = \texttt{ref}\,42\ \texttt{in}\\ \texttt{fun}\ x \to a := (!a + x)\,;\\ \qquad\qquad !a\end{array}$$

$$\begin{array}{l}\texttt{let}\,b = \texttt{ref}(-42)\ \texttt{in}\\ \texttt{fun}\ y \to b := (!b - y)\,;\\ \qquad\qquad -(!b)\end{array}$$

Question: are these OCaml expressions behaviourally equivalent?

$$\begin{array}{l}\mathtt{let}\ a = \mathtt{ref}\ 42\ \mathtt{in} \\ \mathtt{fun}\ x \to a := (!a + x)\,; \\ \qquad\qquad !a\end{array}$$

$$\begin{array}{l}\mathtt{let}\ b = \mathtt{ref}(-42)\ \mathtt{in} \\ \mathtt{fun}\ y \to b := (!b - y)\,; \\ \qquad\qquad -(!b)\end{array}$$

Answer: yes, but devising denotational semantics that exactly match the execution behaviour of

> higher-order functions + locally scoped names (of storage locations, of exceptions, of ...)

has proved hard.

Current state-of-the-art: nominal game semantics (see forthcoming tutorial article by Murawski & Tzevelekos in *Foundations & Trends in Programming Languages*).

Dependence & Symmetry

What does it mean for mathematical structures [needed for denotational semantics] to

$$\begin{cases} \text{depend upon some names?} \\ \text{be independent of some names?} \end{cases}$$

- Conventional answer: parameterization (explicit dependence via functions).
  Can lead to 'weakening hell'.

Dependence & Symmetry

What does it mean for mathematical structures [needed for denotational semantics] to

$$\begin{cases} \text{depend upon some names?} \\ \text{be independent of some names?} \end{cases}$$

- Conventional answer: parameterization (explicit dependence via functions).
  Can lead to 'weakening hell'.
- *Nominal techniques* answer: independence via invariance properties of symmetries.

Name permutations

- $\mathbb{A}$ = fixed countably infinite set of atomic names (a,b,...)
- $\mathbf{S}_{\mathbb{A}}$ = group of all (finite) permutations of $\mathbb{A}$

Typical elements:

$$\begin{tikzcd} a \arrow[r, bend left] & b \arrow[l, bend left] & c \arrow[dr, bend right] & & e \arrow[ll, bend right] & \cdots \\ & & & d \arrow[ur, bend right] & & \end{tikzcd}$$

(but NOT $\begin{tikzcd} f \arrow[dr, bend right] & & g \arrow[dl, bend left] \\ & h & \end{tikzcd}$)

Name permutations

- $\mathbb{A}$ = fixed countably infinite set of atomic names (a,b,...)
- $\mathbf{S}_{\mathbb{A}}$ = group of all (finite) permutations of $\mathbb{A}$
  - each π is a bijection $\mathbb{A} \cong \mathbb{A}$ (injective and surjective function)
  - π finite means: $\{a \in \mathbb{A} \mid \pi(a) \neq a\}$ is finite.
  - group: multiplication is composition of functions $\pi' \circ \pi$; identity is identity function $\mathbf{id}$; inverses are inverse functions π^{-1}.

Actions

A $\mathbf{S}_{\mathbb{A}}$-action on a set X is a function

$$\pi \in \mathbf{S}_{\mathbb{A}},\ x \in X \ \mapsto\ \pi \cdot x \in X$$

satisfying

- $\pi' \cdot (\pi \cdot x) = (\pi' \circ \pi) \cdot x$
- $\mathbf{id} \cdot x = x$

Actions

A $\mathbf{S}_{\mathbb{A}}$-action on a set X is a function

$$\pi \in \mathbf{S}_{\mathbb{A}},\ x \in X \;\mapsto\; \pi \cdot x \in X$$

satisfying

- $\pi' \cdot (\pi \cdot x) = (\pi' \circ \pi) \cdot x$
- $\mathbf{id} \cdot x = x$

Simple example: $\mathbf{S}_{\mathbb{A}}$ acts on sets of names $A \subseteq \mathbb{A}$ via

$$\pi \cdot A = \{\pi(a) \mid a \in A\}$$

E.g.

$$\big(\, a \quad b \,\big) \cdot \{c \mid c \neq a\} = \{c \mid c \neq b\}$$

Support – the key definition

Suppose $\mathbf{S}_{\mathbb{A}}$ acts on a set X and $x \in X$.

A set of names $A \subseteq \mathbb{A}$ supports x if for all $\pi \in \mathbf{S}_{\mathbb{A}}$

$$(\forall a \in A.\, \pi(a) = a) \;\Rightarrow\; \pi \cdot x = x$$

X is a nominal set if every $x \in X$ has a finite support.
[AMP-Gabbay, LICS 1999]

Support – the key definition

Suppose $\mathbf{S}_{\mathbb{A}}$ acts on a set X and $x \in X$.

A set of names $A \subseteq \mathbb{A}$ supports x if for all $\pi \in \mathbf{S}_{\mathbb{A}}$

$$(\forall a \in A.\, \pi(a) = a) \;\Rightarrow\; \pi \cdot x = x$$

X is a nominal set if every $x \in X$ has a <u>finite</u> support.
[AMP-Gabbay, LICS 1999]

E.g. for $X = \{A \mid A \subseteq \mathbb{A}\}$,

- $\{b \mid b \neq a\}$ is infinite, but is supported by $\{a\}$
  because if $\pi(a) = a$, then π is a permutation of $\mathbb{A} - \{a\}$

Support – the key definition

Suppose $\mathbf{S}_{\mathbb{A}}$ acts on a set X and $x \in X$.

A set of names $A \subseteq \mathbb{A}$ supports x if for all $\pi \in \mathbf{S}_{\mathbb{A}}$

$$(\forall a \in A.\, \pi(a) = a) \;\Rightarrow\; \pi \cdot x = x$$

X is a nominal set if every $x \in X$ has a finite support. [AMP-Gabbay, LICS 1999]

E.g. for $X = \{A \mid A \subseteq \mathbb{A}\}$,

- $\{b \mid b \neq a\}$ is infinite, but is supported by $\{a\}$
- no finite set of names supports $\{a_0, a_2, a_4, \ldots\}$ (supposing $a_0, a_1, a_2, a_3, \ldots$ enumerates $\mathbb{A}$), so $\{A \mid A \subseteq \mathbb{A}\}$ is not a nominal set
- $\{A \subseteq \mathbb{A} \mid A \text{ finite, or } \mathbb{A} - A \text{ finite}\}$ is a nominal set

Support – the key definition

Suppose $\mathbf{S}_{\mathbb{A}}$ acts on a set X and $x \in X$.

A set of names $A \subseteq \mathbb{A}$ supports x if for all $\pi \in \mathbf{S}_{\mathbb{A}}$

$$(\forall a \in A.\, \pi(a) = a) \;\Rightarrow\; \pi \cdot x = x$$

X is a nominal set if every $x \in X$ has a finite support. [AMP-Gabbay, LICS 1999]

E.g. for λ-terms with (free & bound) variables from $\mathbb{A}$

$$t ::= a \mid t\,t \mid \lambda a.t \quad \text{modulo } \alpha\text{-equivalence}$$

with $\mathbf{S}_{\mathbb{A}}$-action: $\begin{cases} \pi \cdot a &= \pi(a) \\ \pi \cdot (t\,t') &= (\pi \cdot t)(\pi \cdot t') \\ \pi \cdot (\lambda a.t) &= \lambda \pi(a).(\pi \cdot t) \end{cases}$

FACT: each t is supported by its finite set of *free variables*.

Category of nominal sets, **Nom**

objects are nominal sets

= sets equipped with an action of all (finite) permutations of $\mathbb{A}$, all of whose elements have finite support

morphisms are equivariant functions

= functions preserving the permutation action.

identities and **composition**

= as usual for functions

Why use category theory?

- **equivalence of categories** from different mathematical realms (or even just functors between them) can tell us a lot.
  For example, the following are all equivalent:
  - **Nom**
  - the Schanuel topos
    (from Grothendieck's generalized Galois theory)
  - the category of named sets
    (from the work of Montanari *et al* on model-checking π-calculus)
- **universal properties** (adjoint functors) can characterize a mathematical construction uniquely up to isomorphism and help predict its properties.
  For example…

Nominal exponentials

$$\boxed{X \to_{\mathbf{fs}} Y} \triangleq$$
$$\left\{ f \in Y^X \,\middle|\, \begin{array}{l} f \text{ is finitely supported w.r.t. the action} \\ \pi \cdot f = \lambda x \to \pi \cdot (f(\pi^{-1} \cdot x)) \end{array} \right\}$$

This is characterised uniquely up to isomorphism by the fact that it give the right adoint to $(\_) \times X : \mathbf{Nom} \to \mathbf{Nom}$:

$$\frac{Z \times X \to Y}{Z \to (X \to_{\mathbf{fs}} Y)}$$

(Products in $\mathbf{Nom}$ are created by the forgetful functor to the category of sets.)

Nominal exponentials

$\boxed{X \to_{\mathbf{fs}} Y} \triangleq$

N.B. permutations have inverses

$$\left\{ f \in Y^X \;\middle|\; \begin{array}{l} f \text{ is finitely supported w.r.t. the action} \\ \pi \cdot f = \lambda x \to \pi \cdot (f(\pi^{-1} \cdot x)) \end{array} \right\}$$

This is characterised uniquely up to isomorphism by the fact that it give the right adoint to $(\_) \times X : \mathbf{Nom} \to \mathbf{Nom}$:

$$\frac{Z \times X \to Y}{Z \to (X \to_{\mathbf{fs}} Y)}$$

(Products in $\mathbf{Nom}$ are created by the forgetful functor to the category of sets.)

Name abstraction

Each $X \in \mathbf{Nom}$ yields a nominal set $\boxed{[\mathbb{A}]X}$ of

name-abstractions $\langle a \rangle x$ are $\sim$-equivalence classes of pairs $(a, x) \in \mathbb{A} \times X$, where

$$
\begin{aligned}
(a, x) \sim (a', x') \;\Leftrightarrow\; & \exists\, b \,\#\, (a, x, a', x') \\
& (b\ a) \cdot x = (b\ a') \cdot x'
\end{aligned}
$$

Name abstraction

Each $X \in \mathbf{Nom}$ yields a nominal set $\boxed{[\mathbb{A}]X}$ of

name-abstractions $\langle a \rangle x$ are $\sim$-equivalence classes of pairs $(a, x) \in \mathbb{A} \times X$, where

$$(a, x) \sim (a', x') \ \Leftrightarrow\ \exists\, b \,\#\, (a, x, a', x')\ \ (b\ a) \cdot x = (b\ a') \cdot x'$$

Freshness relation: $a \# x$ means $a \notin A$ for some finite support A for x

The freshness relation gives a
well-behaved, syntax-independent
notion of freeness, or non-occurrence.

Name-abstraction

Name abstraction $[\mathbb{A}](\_) : \mathbf{Nom} \to \mathbf{Nom}$ is right adjoint to adjoint 'separated tensor' $\_ * \mathbb{A}$

$$X * \mathbb{A} \triangleq \{(x, a) \in X \times \mathbb{A} \mid a \# x\}$$

$$\frac{X * \mathbb{A} \to Y}{X \to [\mathbb{A}]Y}$$

So $[\mathbb{A}]X$ is a kind of (affine) linear function space from $\mathbb{A}$ to X, but with great properties – e.g. it too has a right adjoint

$$\frac{([\mathbb{A}]X) \to Y}{X \to \{f \in Y^{\mathbb{A}} \mid (\forall a \in \mathbb{A})\ a \# f(a)\}}$$

Name-abstraction

Name abstraction $[\mathbb{A}](\_) : \mathbf{Nom} \to \mathbf{Nom}$ is right adjoint to adjoint 'separated tensor' $\_ * \mathbb{A}$

$$X * \mathbb{A} \triangleq \{(x, a) \in X \times \mathbb{A} \mid a \# x\}$$

$$\frac{X * \mathbb{A} \to Y}{\overline{X \to [\mathbb{A}]Y}}$$

So $[\mathbb{A}]X$ is a kind of (affine) linear function space from $\mathbb{A}$ to X, but with great properties – e.g. it too has a right adjoint

$\rightsquigarrow$ an initial algebra semantics for syntax with binders using $[\mathbb{A}](\_)$ to model binding operations with user-friendly inductive/recursive properties. . .

Initial algebras

- $[\mathbb{A}](\_)$ can be combined with $\_ \times \_$ and $\_ + \_$ to give functors $\mathbf{T} : \mathbf{Nom} \to \mathbf{Nom}$ that have initial algebras $I : \mathbf{T}\,D \to D$

$$
\begin{tikzcd}
\mathbf{T}\,D \arrow[d, "I"'] & & \mathbf{T}\,X \arrow[d, "F", "\text{for all}"'] \\
D & & X
\end{tikzcd}
$$

Initial algebras

- $[\mathbb{A}](\_)$ can be combined with $\_ \times \_$ and $\_ + \_$ to give functors $\mathbf{T} : \mathbf{Nom} \to \mathbf{Nom}$ that have initial algebras $I : \mathbf{T}\,D \to D$

$$
\begin{tikzcd}
\mathbf{T}\,D \arrow[r, dashed, "\mathbf{T}\,\hat{F}"] \arrow[d, "I"'] & \mathbf{T}\,X \arrow[d, "F"] \\
D \arrow[r, dashed, "\text{exists unique}", "\hat{F}"'] & X
\end{tikzcd}
$$

Initial algebras

- $[\mathbb{A}](\_)$ can be combined with $\_ \times \_$ and $\_ + \_$ to give functors $\mathbf{T} : \mathbf{Nom} \to \mathbf{Nom}$ that have initial algebras $I : \mathbf{T}\, D \to D$
- For a wide class of such functors ('nominal algebraic functors') the initial algebra D coincides with sets of abstract syntax trees modulo α-equivalence.

  E.g. the initial algebra for

  $$\mathbf{T}(\_) \triangleq \mathbb{A} + (\_ \times \_) + [\mathbb{A}](\_)$$

  is the usual set of untyped λ-terms and the initial-algebra universal property yields. . . .

α-Structural recursion

Theorem.
Given any $X \in \mathbf{Nom}$ and $\begin{cases} f_1 & \in & \mathbb{A} \to_{\mathrm{fs}} X \\ f_2 & \in & X \times X \to_{\mathrm{fs}} X \\ f_3 & \in & [\mathbb{A}]X \to_{\mathrm{fs}} X \end{cases}$

$\exists!\ \hat{f} \in \Lambda \to_{\mathrm{fs}} X$

$$\begin{cases} \hat{f}\,a = f_1\,a \\ \hat{f}\,(t_1\,t_2) = f_2(\hat{f}\,t_1, \hat{f}\,t_2) \\ \hat{f}(\lambda a.t) = f_3(\langle a\rangle(\hat{f}\,t)) & \text{if } a \,\#\, (f_1, f_2, f_3) \end{cases}$$

untyped λ-terms:
$t ::= a \mid t\,t \mid \lambda a.t$

α-Structural recursion

Theorem.
Given any $X \in \mathbf{Nom}$ and $\begin{cases} f_1 & \in & \mathbb{A} \to_{\mathrm{fs}} X \\ f_2 & \in & X \times X \to_{\mathrm{fs}} X \\ f_3 & \in & \mathbb{A} \times X \to_{\mathrm{fs}} X \end{cases}$ s.t.

$$(\forall a)\ a \# (f_1, f_2, f_3) \Rightarrow (\forall x)\ a \# f_3(a, x) \qquad \text{(FCB)}$$

$\exists!$ $\hat{f} \in \Lambda \to_{\mathrm{fs}} X$

$$\begin{cases} \hat{f}\, a = f_1\, a \\ \hat{f}\,(t_1\, t_2) = f_2(\hat{f}\, t_1, \hat{f}\, t_2) \\ \hat{f}(\lambda a.t) = f_3(a, \hat{f}\, t) \quad \text{if } a \# (f_1, f_2, f_3) \end{cases}$$

α-Structural recursion

Theorem.
Given any $X \in \mathbf{Nom}$ and $\begin{cases} f_1 \in \mathbb{A} \to_{\mathrm{fs}} X \\ f_2 \in X \times X \to_{\mathrm{fs}} X \\ f_3 \in \mathbb{A} \times X \to_{\mathrm{fs}} X \end{cases}$ s.t.

$$(\forall a)\ a \mathbin{\#} (f_1, f_2, f_3) \ \Rightarrow\ (\forall x)\ a \mathbin{\#} f_3(a, x) \qquad \text{(FCB)}$$

$\exists!\ \hat{f} \in \Lambda \to_{\mathrm{fs}} X$

$$\begin{cases} \hat{f}\, a = f_1\, a \\ \hat{f}\,(t_1\, t_2) = f_2(\hat{f}\, t_1, \hat{f}\, t_2) \\ \hat{f}(\lambda a.t) = f_3(a, \hat{f}\, t) \quad \text{if } a \mathbin{\#} (f_1, f_2, f_3) \end{cases}$$

E.g. capture-avoiding substitution $(\_)[t'/a']$ is the $\hat{f}$ for

$$\begin{aligned} f_1\, a &\triangleq \textbf{if } a = a' \textbf{ then } t' \textbf{ else } a \\ f_2(t_1, t_2) &\triangleq t_1\, t_2 \\ f_3(a, t) &\triangleq \lambda a.t \end{aligned}$$

for which (FCB) holds, since $a \mathbin{\#} \lambda a.t$

Nominal System T

The notion of 'α-structural recursion' generalizes smoothly from λ-terms to any nominal algebraic signature, giving a version of Gödel's System T for nominal data types [J ACM 53(2006)459–506].

Urban & Berghofer's Nominal package for Isabelle/HOL (interactive theorem prover for classical higher-order logic) implements this, and more.

Seems to capture informal practice quite well.

Applications: PL semantics

- operational semantics of ‘nominal’ calculi
- game semantics; domain theory
- equational logic & rewriting modulo α-equivalence
- logic programming (Cheney-Urban: αProlog)
- functional metaprogramming (AMP-Shinwell: Fresh Ocaml)

Take-home message: permutation comes before substitution when dealing with the meta-theory of names and binding operations in syntactical structures.

An introduction to nominal techniques motivated by

Programming language semantics

Automata theory

Constructive type theory

Support – the key definition

Suppose $\mathbf{S}_{\mathbb{A}}$ acts on a set X and $x \in X$.

A set of names $A \subseteq \mathbb{A}$ supports x if for all $\pi \in \mathbf{S}_{\mathbb{A}}$

$$(\forall a \in A.\, \pi(a) = a) \;\Rightarrow\; \pi \cdot x = x$$

X is a nominal set if every $x \in X$ has a finite support. [AMP-Gabbay, LICS 1999]

Consider replacing $\mathbf{S}_{\mathbb{A}}$ by a subgroup G
i.e. use a more restrictive notion of symmetry

Different symmetries

Three interesting examples:

1. **Equality**: $G =$ all (finite) permutations.
2. **Linear order**: $\mathbb{A} = \mathbb{Q}$ and $G =$ order-preserving perms.
3. **Graphs**: $\mathbb{A} =$ vertices of the Rado graph and $G =$ graph automorphisms.

Different symmetries

Three interesting examples:

1. **Equality**: $G =$ all (finite) permutations.
2. **Linear order**: $\mathbb{A} = \mathbb{Q}$ and $G =$ order-preserving perms.
3. **Graphs**: $\mathbb{A} =$ vertices of the Rado graph and $G =$ graph automorphisms.

In general [Bojańczyk, Klin, Lasota]:

$\mathbb{A} =$ Carrier of the universal homogeneous structure (Fraïssé limit) for a finite relational signature

$G =$ automorphisms w.r.t. the signature

yields a universe $\mathcal{U} = \mathbf{P}_{\mathbf{fs}}\,(\mathbb{A} + \mathcal{U})$ with interesting applications for nominal computation.

Applications: automata theory

Automata over infinite alphabets:

- HD-automata: π-calculus verification [Montanari el al]
- fresh-register automata [Tzevelekos] (extending finite-memory automata of Kaminski & Francez): verifying programs with local names
- automata & Turing machines in sets with atoms [Bojańczyk et al]; CSP with atoms [Klin et al, LICS 2015]; ...

Applications: automata theory

Automata over infinite alphabets:

- HD-automata: π-calculus verification [Montanari el al]
- fresh-register automata [Tzevelekos] (extending finite-memory automata of Kaminski & Francez): verifying programs with local names
- automata & Turing machines in sets with atoms [Bojańczyk et al]; CSP with atoms [Klin et al, LICS 2015]; ...

General approach: try to do 'ordinary' computation theory inside the universe/category of nominal sets (for a Fraïssé symmetry), but replace finite-state notions by orbit-finite ones...

Orbit finiteness

$=$ having finitely many equivalence classes for

$$x \sim y \triangleq \exists \pi .\ \pi \cdot x = y$$

E.g. for the equality symmetry, $\mathbb{A}^n$ is orbit-finite, $\mathbb{A}^*$ is not.

Orbit finiteness

$=$ having finitely many equivalence classes for

$$x \sim y \triangleq \exists \pi.\ \pi \cdot x = y$$

E.g. for the equality symmetry, $\mathbb{A}^n$ is orbit-finite, $\mathbb{A}^*$ is not.

Orbit-finite elements of $\mathcal{U}$ have good closure properties, except for powerset. Nominal automata (NA) satisfy:

- deterministic (D) $\neq$ non-deterministic (ND)
- emptiness for ND-NA is decidable
- equivalence for D-NA is decidbale
- D-NAs can be minimized

See [Bojańczyk-Klin-Lasota, LMCS 10(3) 2014].

An introduction to nominal techniques motivated by

Programming language semantics

Automata theory

Constructive type theory

Nominal System T

Questions:

- Martin-Löf Type Theory generalizes Gödel's System T to inductively defined families of dependent types.
  What is the right version of the notions of *nominal set*, *freshness* and *name abstraction* within constructive type theory?

  Stark-Schöpp CSL 2004 [extensional]
  Cheney LMCS 2012 [missing locally scoped names]
  AMP-Matthiesen-Derikx LSFA 2014 [locally scoped, judgementally fresh names]

Applications: Homotopy Type Theory

Cubical sets [Bezem-Coquand-Huber] model of Voevodsky's axiom of univalence can be described using nominal sets equipped with an operation of substitution $x \mapsto x(i/a)$ where $i \in \{0, 1\}$.

- names are names of directions (cartesian axes)

  (so e.g., if an object has support $\{a, b, c\}$ it is 3-dimensional)
- freshness ($a \# x$) = degeneracy ($x(i/a) = x$)
- identity types are modelled by name-abstraction: $\langle a \rangle x$ is a proof that $x(0/a)$ is equal to $x(1/a)$.

HoTT is about (proof-relevant) *mathematical foundations* (a topic no longer very popular with mathematicians). That's where the mathematics of nominal sets came from...

Impact can take a long time

The mathematics behind nominal sets goes back a long way...

Abraham Fraenkel, *Der Begriff "definit" und die Unabhängigkeit des Auswahlsaxioms*, Sitzungsberichte der Preussischen Akademie der Wissenschaften, Physikalisch-mathematische Klasse (1922), 253–257.

Andrzej Mostowski, *Uber die Unabhängigkeit des Wohlordnungssatzes vom Ordnungsprinzip*, Fundamenta Mathematicae 32 (1939), 201–252.

Impact can take a long time

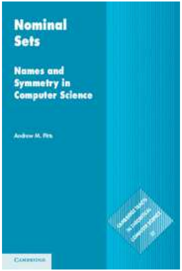
The mathematics behind nominal sets goes back a long way...

...and it's still too early to tell what will be the impact of the applications of it to CS developed over the last 15 years.

Take-home messages:

- Computation modulo symmetry deserves further exploration.
- Permutation comes before substitution and (hence) name-abstraction before lambda-abstraction...but it seems that constructive type theory and nominal techniques can coexist (wts).

Homework



Nominal Sets

Names and Symmetry in Computer Science

Cambridge Tracts in Theoretical Computer Science, Vol. 57 (CUP, 2013)