Nominal Matching Logic

Maribel Fernández
Joint work with James Cheney

Brasilia, 1 Nov 2022

Reasoning with binding operators

K is a successful first-order verification framework to specify and implement programming languages [Rosu 2017].
But binders are not a primitive notion.
Nominal Logic [Pitts 2003] is a first-order theory of binding.

Aim:
Combine the benefits of Matching Logic (K's foundation) with the advantages of Nominal Logic to reason about binding.

Key Notion: α-equivalence
terms defined modulo renaming of bound names

Examples: $\forall x.\phi$, $\lambda x.Mx$, $\nu x.P$
x: name, M, ϕ, P: variables, $\forall, \lambda, \nu$: binders

Key ideas: Names, abstraction, freshness, name swapping.

Signature $\mathbf{\Sigma} = (S, \mathcal{V}ar, \Sigma)$ where Σ includes
swapping $(-\ -) \cdot - : \alpha \times \alpha \times \tau \to \tau$
abstraction $[-]- : \alpha \times \tau \to [\alpha]\tau$
equality $=: \tau \times \tau$
freshness $\#: \alpha \times \tau$

Axioms below.

Semantics given by nominal sets

Nominal Logic Axioms

$$(a\ a)\cdot x = x \qquad (S1)$$
$$(a\ a')\cdot(a\ a')\cdot x = x \qquad (S2)$$
$$(a\ a')\cdot a = a' \qquad (S3)$$
$$(a\ a')\cdot(b\ b')\cdot x = ((a\ a')\cdot b\ (a\ a')\cdot b')\cdot(a\ a')\cdot x \qquad (E1)$$
$$b \# x \Rightarrow (a\ a')\cdot b \# (a\ a')\cdot x \qquad (E2)$$
$$(a\ a')\cdot f(\bar{x}) = f((a\ a')\cdot\bar{x}) \qquad (E3)$$
$$p(\bar{x}) \Rightarrow p((a\ a')\cdot\bar{x}) \qquad (E4)$$
$$(b\ b')\cdot[a]x = [(b\ b')\cdot a](b\ b')\cdot x \qquad (E5)$$
$$a \# x \wedge a' \# x \Rightarrow (a\ a')\cdot x = x \qquad (F1)$$
$$a \# a' \iff a \neq a' \qquad (F2)$$
$$\forall a:\alpha, a':\alpha'.a \# a' \qquad (\alpha \neq \alpha') \qquad (F3)$$
$$\forall\bar{x}.\exists a.a \# \bar{x} \qquad (F4)$$
$$\forall\bar{x}.(\text{И}a.\phi \iff \exists a.a \# \bar{x}\wedge\phi) \qquad (FV(\text{И}a.\phi)\subseteq\bar{x}) \qquad (Q)$$
$$[a]x = [a']x' \iff (a = a' \wedge x = x') \vee (a \# x' \wedge (a\ a')\cdot x = x') \qquad (A1)$$
$$\forall x:[\alpha]s.\exists a:\alpha, y:s.x = [a]y \qquad (A2)$$

Nominal Logic Axioms: Swapping

Swapping is defined for everything we can talk about in nominal logic.

$$\begin{array}{cr} (a\ a) \cdot x = x & (S1) \\ (a\ a') \cdot (a\ a') \cdot x = x & (S2) \\ (a\ a') \cdot a = a' & (S3) \end{array}$$

Swapping a name for itself has no effect.
Swappings are involutions.
Swapping acts on names as expected.

Nominal Logic Axioms: Equivariance

$$
\begin{aligned}
(a\ a')\cdot(b\ b')\cdot x &= ((a\ a')\cdot b\ (a\ a')\cdot b')\cdot(a\ a')\cdot x && (E1)\\
b \# x &\Rightarrow (a\ a')\cdot b \# (a\ a')\cdot x && (E2)\\
(a\ a')\cdot f(\bar{x}) &= f((a\ a')\cdot\bar{x}) && (E3)\\
p(\bar{x}) &\Rightarrow p((a\ a')\cdot\bar{x}) && (E4)\\
(b\ b')\cdot[a]x &= [(b\ b')\cdot a](b\ b')\cdot x && (E5)
\end{aligned}
$$

Equivariance means (roughly) "commutes with swappings"

Nominal Logic Axioms: Freshness and Abstraction

Given a name and any other thing the freshness relation tells us whether the name is fresh for or "appears free" in that thing.

$$
\begin{array}{cr}
a \# x \wedge a' \# x \Rightarrow (a\ a') \cdot x = x & (F1) \\
a \# a' \iff a \neq a' & (F2) \\
\forall a : \alpha, a' : \alpha' . a \# a' \qquad (\alpha \neq \alpha') & (F3) \\
\forall \bar{x} . \exists a . a \# \bar{x} & (F4)
\end{array}
$$

The abstraction operation constructs alpha-equivalence classes.

$$
\begin{array}{cr}
[a]x = [a']x' \iff (a = a' \wedge x = x') \vee (a \# x' \wedge (a\ a') \cdot x = x) & (A1) \\
\forall x : [\alpha]s . \exists a : \alpha, y : s . x = [a]y & (A2)
\end{array}
$$

Nominal Logic Axioms: И-quantifier

$$\forall \bar{x}.(\text{И}a.\phi \iff \exists a.a \# \bar{x} \wedge \phi) \qquad (FV(\text{И}a.\phi) \subseteq \bar{x}) \quad (Q)$$

И binds a variable of name-sort

Some/any property: we can prove (using the other axioms) that

$$\text{И}a.\phi \iff \exists a.a \# \bar{x} \wedge \phi \iff \forall a.a \# \bar{x} \Rightarrow \phi$$

Example: λ-calculus abstract syntax

Sorts: Var (name sort) and Exp

$var : Var \rightarrow Exp \quad app : Exp \times Exp \rightarrow Exp \quad lam : [Var]Exp \rightarrow Exp$

$\lambda x.e$ represented as $lam([x]\mathbf{e})$ where **e** is the representation of e.

α-equivalence by construction (abstraction in nominal logic).

Substitution:

$$
\begin{array}{rcl}
subst(var(x), x, z) & = & z \\
x \neq y \Rightarrow subst(var(x), y, z) & = & var(x) \\
subst(app(x_1, x_2), y, z) & = & app(subst(x_1, y, z), subst(x_2, y, z)) \\
a \# y, z \Rightarrow subst(lam([a]x), y, z) & = & lam([a]subst(x, y, z))
\end{array}
$$

Matching Logic - Syntax

Signature $\mathbf{\Sigma} = (S, \mathcal{V}ar, \Sigma)$

Patterns:

$$\phi_\tau ::= x\colon \tau \mid \phi_\tau \wedge \psi_\tau \mid \neg\phi_\tau \mid \exists x\colon \tau'.\phi_\tau \mid \sigma(\phi_{\tau_1}, \ldots, \phi_{\tau_n})$$

where $x \in \mathcal{V}ar_\tau$ and $\sigma \in \Sigma_{\tau_1,\ldots,\tau_n;\tau}$.

Disjunction, implication, $\forall$, true and false defined as abbreviations:
e.g. $\top_\tau \equiv \exists x\colon \tau.x\colon \tau$ and $\bot_\tau \equiv \neg\top_\tau$.

Matching Logic - Model

$$M = (\{M_\tau\}_{\tau \in S}, \{\sigma_M\}_{\sigma \in \Sigma})$$

- non-empty carrier set M_τ for each $\tau \in S$
- $\sigma_M : M_{\tau_1} \times \ldots \times M_{\tau_n} \to \mathcal{P}(M_\tau)$ for each $\sigma \in \Sigma_{\tau_1,\ldots,\tau_n;\tau}$.

Matching Logic - Model

Valuation $\rho\colon \mathcal{V}ar \to M$ respecting sorts.
Extension to patterns:
$\overline{\rho}(x) = \{\rho(x)\}$ for all $x \in \mathcal{V}ar$, $\overline{\rho}(\phi_1 \wedge \phi_2) = \overline{\rho}(\phi_1) \cap \overline{\rho}(\phi_2)$,
$\overline{\rho}(\neg\phi_\tau) = M_\tau - \overline{\rho}(\phi_\tau)$, $\overline{\rho}(\exists x\colon \tau'.\phi_\tau) = \bigcup_{a \in M_{\tau'}} \overline{\rho[a/x]}(\phi_\tau)$,
$\overline{\rho}(\sigma(\phi_{\tau_1}, \ldots, \phi_{\tau_n}) = \overline{\sigma_M}(\overline{\rho}(\phi_{\tau_1}), \ldots, \overline{\rho}(\phi_{\tau_n}))$, for $\sigma \in \Sigma_{\tau_1,\ldots,\tau_n;\tau}$,
where
$\overline{\sigma_M}(V_1, \ldots, V_n) = \bigcup\{\sigma_M(v_1, \ldots, v_n) \mid v_1 \in V_1, \ldots, v_n \in V_n\}$.

ϕ_τ valid in M, $M \vDash \phi_\tau$, if $\overline{\rho}(\phi_\tau) = M_\tau$ for all $\rho\colon Var \to M$.

Adding NL Features to ML

1. Nominal Logic can be embbeded as a Matching Logic Theory: NLML (see [PPDP 2022])
   $\Rightarrow$ it can be directly implemented in K
   But...
   - ground names, which are useful in rewriting, logic programming and program verification, are not available in NLML
   - not clear how to incorporate the И-quantifier in a first-class way, which is needed to simplify reasoning with freshness constraints.
2. NML: Matching Logic with Built-in Names and И

Matching Logic with Built-in Names and И

NML signature $\mathbf{\Sigma} = (S, \mathcal{V}ar, Name, \Sigma)$ consists of

- a non-empty set S of sorts $\tau, \tau_1, \tau_2 \ldots$, split into a set NS of name sorts $\alpha, \alpha_1, \alpha_2, \ldots$, a set DS of data sorts $\delta, \delta_1, \delta_2, \ldots$ including a sort $Pred$, and a set AS of abstraction sorts $[\alpha]\tau$
- an S-indexed family $\mathcal{V}ar = \{\mathcal{V}ar_\tau \mid \tau \in S\}$ of countable sets of variables $x\colon \tau, y\colon \tau, \ldots$,
- an NS-indexed family $Name = \{Name_\alpha \mid \alpha \in NS\}$ of countable sets of names $\mathsf{a}\colon \alpha, \mathsf{b}\colon \alpha, \ldots$ and
- an $(S^* \times S)$-indexed family Σ of sets of many-sorted symbols σ, written $\Sigma_{\tau_1,\ldots,\tau_n;\tau}$.

NML Syntax

Patterns:

$$\phi_\tau \quad ::= \quad x : \tau \mid \mathsf{a} : \alpha \mid \phi_\tau \wedge \psi_\tau \mid \neg\phi_\tau \mid \exists x : \tau'.\phi_\tau$$
$$\mid \sigma(\phi_{\tau_1}, \ldots, \phi_{\tau_n}) \mid \text{И}\mathsf{a} : \alpha.\phi_\tau$$

where $x \in \mathcal{V}ar_\tau$, a $\in Name_\alpha$, and both $\exists$ and И are binders (i.e., we work modulo α-equivalence).

Σ includes the following families of sort-indexed symbols (subscripts omitted):

$(- \ -) \cdot -$	$:$	$\alpha \times \alpha \times \tau \to \tau$	swapping (function)
$[-]-$	$:$	$\alpha \times \tau \to [\alpha]\tau$	abstraction (function)
$-@-$	$:$	$[\alpha]\tau \times \alpha \rightharpoonup \tau$	concretion (partial function)
$fresh_{\tau,\alpha}$	$\in$	$\Sigma_{\tau;\alpha}$	freshness (multivalued operation)
$- \#_{\alpha,\tau} -$	$:$	$\alpha \times \tau \rightharpoonup Pred$	freshness relation
$-^\dagger$	$:$	$\Sigma_{Pred;\tau}$	coercion operator, often left implicit.

Given $\mathbf{\Sigma} = (S, \mathcal{V}ar, Name, \Sigma)$
let $\mathbb{A}$ be $\bigcup_{\alpha \in NS} \mathbb{A}_\alpha$ where each $\mathbb{A}_\alpha$ is an infinite countable set of atoms and the $\mathbb{A}_\alpha$ are pairwise disjoint,
let G be a product of permutation groups $\prod_i Sym(\mathbb{A}_i)$

An NML model $M = (\{M_\tau\}_{\tau \in S}, \{\sigma_M\}_{\sigma \in \Sigma})$ consists of

- a non-empty nominal G-set M_τ for each $\tau \in S - NS$;
- an equivariant interpretation $\sigma_M : M_{\tau_1} \times \cdots \times M_{\tau_n} \to \mathcal{P}_{fin}(M_\tau)$ for each $\sigma \in \Sigma_{\tau_1, \ldots, \tau_n; \tau}$.

NML Model

A model is *standard* if the interpretation of:

1. each name sort α is $\mathbb{A}_\alpha$
2. the sort $Pred$ is a singleton set $\{*\}$, where $*$ is equivariant: $\{*\}$ is a nominal set whose powerset is isomorphic to Bool
3. each abstraction sort $[\alpha]\tau$ is $[M_\alpha]M_\tau$
4. the swapping symbol $(- \ -) \cdot - : \alpha \times \alpha \times \tau \to \tau$ is the swapping function on M_τ
5. the abstraction symbol is the quotienting function mapping $\langle a, x \rangle$ to its alpha-equivalence class, i.e. $(a, x) \mapsto (a, x)/_{\equiv_\alpha}$
6. the concretion symbol is the (partial) concretion function $(X, \mathsf{a}) \mapsto \{y \mid (\mathsf{a}, y) \in X\}$
7. the freshness operation $fresh_{\tau,\alpha}$ is the function $x \mapsto \{a \mid a \notin supp(x)\}$
8. the freshness relation $\#_{\alpha,s}$ is the freshness predicate on $\mathbb{A}_\alpha \times M_\tau$, i.e., it holds for the tuples $\{(a, x) \mid a \notin supp(x)\}$.

NML Valuation

A function $\rho : \mathcal{V}ar \cup Name \rightharpoonup M$ with finite domain that is compatible with sorting (i.e., $\rho(x\colon \tau) \in M_\tau$, $\rho(\mathsf{a}\colon \alpha) \in M_\alpha$), injective on names and finitely supported is called a *valuation*.

Injectivity ensures that two different names in the syntax are interpreted by different elements in the valuation.

A valuation is equivariant if $(\pi \cdot \rho)(e) = \pi \cdot \rho(e)$ for every element in its domain.

NML Pattern Semantics

Given valuation ρ whose domain includes the free variables and free names of ϕ:

$$\begin{array}{rcl}
\overline{\rho}(x : \tau) & = & \{\rho(x)\} \\
\overline{\rho}(\mathsf{a} : \alpha) & = & \{\rho(\mathsf{a})\} \\
\overline{\rho}(\sigma(\phi_1, \dots, \phi_n)) & = & \overline{\sigma_M}(\overline{\rho}(\phi_1), \dots, \overline{\rho}(\phi_n)) \\
\overline{\rho}(\phi_1 \wedge \phi_2) & = & \overline{\rho}(\phi_1) \cap \overline{\rho}(\phi_2) \\
\overline{\rho}(\neg\phi) & = & M_\tau - \overline{\rho}(\phi) \\
\overline{\rho}(\exists x : \tau.\phi) & = & \bigcup_{a \in M_\tau} \overline{\rho[a/x]}(\phi) \\
\overline{\rho}(\text{И}\mathsf{a} : \alpha.\phi) & = & \bigcup_{a \in \mathbb{A}_\alpha - supp(\rho)} \{v \in \overline{\rho[a/\mathsf{a}]}(\phi) \mid a \notin supp(v)\}
\end{array}$$

In the interpretation of the И pattern, ρ is extended by assigning to a any fresh element a of $\mathbb{A}_\alpha$

$\reflectbox{N}$ Pattern - Examples

- Suppose ϕ does not contain a as a free name; then $\reflectbox{N}\mathsf{a}.\phi$ is equivalent to ϕ.
- $\phi_1 = \reflectbox{N}\mathsf{a}.\mathsf{a}$ is a pattern that matches nothing. Likewise $\phi_2 = \reflectbox{N}\mathsf{a}.\langle \mathsf{a}, \mathsf{a}\rangle$ and $\phi_3 = \reflectbox{N}\mathsf{a}.\reflectbox{N}\mathsf{b}.\langle \mathsf{a}, \mathsf{b}\rangle$.
- $\phi_4 = \reflectbox{N}\mathsf{a}.[\mathsf{a}]\mathsf{a}$ matches any abstraction whose body is the abstracted name.
- $\phi_5 = \reflectbox{N}\mathsf{a}.\mathsf{a} = \mathsf{a}$ is a valid predicate (equivalent to $\top$)
- $\phi_6 = \exists x.\reflectbox{N}\mathsf{a}.\mathsf{a} = x$ is false/empty since whatever x is, a must be chosen fresh for (and in particular distinct from) it. However, $\reflectbox{N}\mathsf{a}.\exists x.\mathsf{a} = x$ is true since we may choose $x = \mathsf{a}$.

Ꮇ Pattern - Example

Consider three possible rules representing eta-equivalence for the lambda-calculus

$$
\begin{array}{rcl}
x : Exp & = & lam([\mathsf{a}]app(x, var(\mathsf{a}))) \\
x : Exp & = & lam(\exists a.[a]app(x, var(a))) \\
x : Exp & = & lam(\mathsf{И}\mathsf{a}.[\mathsf{a}]app(x, var(\mathsf{a})))
\end{array}
$$

Only the third one is correct.

Properties

Theorem (Equivariant Semantics)

If $v \in \overline{\rho}(\phi)$ *then* $(a\ a') \cdot v \in \overline{(a\ a') \cdot \rho}(\phi)$. *I.e., for all* ϕ, $(a\ a') \cdot \overline{\rho}(\phi) = \overline{(a\ a') \cdot \rho}(\phi)$.

The Ͷ pattern satisfies the following equivalences:

1. $\text{И}\mathsf{a}\colon \alpha.\phi_\tau(\mathsf{a}, \bar{\mathsf{b}}, \bar{x}) \Leftrightarrow$
   $\exists z_\mathsf{a}\colon \alpha.((\exists y\colon \tau.y \wedge z_\mathsf{a} \#_s^\dagger(\bar{\mathsf{b}}, \bar{x}, y)) \wedge \phi_\tau\{\mathsf{a} \mapsto z_\mathsf{a}\}(\mathsf{a}, \bar{\mathsf{b}}, \bar{x}))$
2. $\text{И}\mathsf{a}\colon \alpha.\phi_\tau(\mathsf{a}, \bar{\mathsf{b}}, \bar{x}) \Leftrightarrow \forall z_\mathsf{a}\colon \alpha.((\exists y\colon \tau.y \wedge z_\mathsf{a} \#_s^\dagger(\bar{\mathsf{b}}, \bar{x}, y)) \Rightarrow \phi_\tau\{\mathsf{a} \mapsto z_\mathsf{a}\}(\mathsf{a}, \bar{\mathsf{b}}, \bar{x}))$

Applications: The λ-calculus in NML

To reason about the typed lambda-calculus we use sorts Exp (expressions), Ty (types), and Var (variables, a name-sort) interpreted as nominal sets M_{Var}, M_{Exp}, and M_{Ty} satisfying the following equations:

$$M_{Exp} = M_{Var} + (M_{Exp} \times M_{Exp}) + [M_{Var}]M_{Exp}$$

$$M_{Ty} = 1 + M_{Ty} \times M_{Ty} + \cdots$$

We assume at least one constant type (e.g. int or $unit$) and a binary constructor $fn : Ty \times Ty \to Ty$ for function types

M_{Exp} is the set of lambda-terms quotiented by alpha-equivalence. We fix M_Λ as the standard model obtained taking M_{Exp} and M_{Ty} as defined above.

Induction principle schematic over $P \in \Sigma_{Exp,\tau_1,\ldots,\tau_n;Pred}$:

$$\begin{array}{rl}(\forall a\colon Var.P(var(a),\bar{y})) & \Rightarrow \\ (\forall t_1\colon Exp, t_2\colon Exp.P(t_1,\bar{y}) \wedge P(t_2,\bar{y}) \Rightarrow P(app(t_1,t_2),\bar{y})) & \Rightarrow \\ (\forall a, t.a \# \bar{y} \Rightarrow P(t,\bar{y}) \Rightarrow P(lam([a]t)),\bar{y}) & \Rightarrow \\ \forall t\colon Exp.P(t,\bar{y}) & \end{array}$$

Example: Substitution Lemma proved by induction on x.

$$\begin{array}{rcl}P(x,y,z,y',z') & \triangleq & y \# y', z' \Rightarrow \\ & & subst(subst(x,y,z),y',z') = \\ & & subst(subst(x,y',z'),y,subst(z,y',z')).\end{array}$$

The λ-calculus in NML

In NML we can axiomatize substitution equationally (no side condition)

$$\begin{array}{rcl} subst(var(a), a, z) & = & z \\ subst(var(a), \neg a, z) & = & var(a) \\ subst(app(x_1, x_2), y, z) & = & app(subst(x_1, y, z), subst(x_2, y, z)) \\ subst(lam(x), y, z) & = & lam(\text{И}\mathsf{a}.[\mathsf{a}]subst(x@\mathsf{a}, y, z)) \end{array}$$

Induction principle using И avoiding freshness constraints

$$
\begin{array}{rl}
(\forall x : Var.P(var(x))) & \Rightarrow \\
(\forall t_1 : Exp, t_2 : Exp.P(t_1) \wedge P(t_2) \Rightarrow P(app(t_1, t_2))) & \Rightarrow \\
(\forall t : [Var]Exp.\text{И}\mathsf{a} : Var.P(t@\mathsf{a}) \Rightarrow P(lam(t)) & \Rightarrow \\
\forall t : Exp.P(t) &
\end{array}
$$

Substitution Lemma (with just one freshness condition, formalizing the usual side-condition in textbooks)

$$
\begin{array}{rcl}
\mathsf{a} \# z' & \Rightarrow & subst(subst(x, \mathsf{a}, z), \mathsf{b}, z') = \\
& & subst(subst(x, \mathsf{b}, z'), \mathsf{a}, subst(z, \mathsf{b}, z'))
\end{array}
$$

The λ-calculus in NML: Reduction

$$
\begin{array}{rcl}
red(app(x,y)) & = & app(red(x),y) \vee app(x, red(y)) \\
 & \vee & (\exists z.x = lam(z) \wedge \text{И}\mathsf{a}.subst(z@\mathsf{a}, \mathsf{a}, y)) \\
red(lam([\mathsf{a}]y)) & = & lam([\mathsf{a}]red(y))
\end{array}
$$

Weak reduction: only first axiom (no reduction under λ)

The λ-calculus in NML: Type Checking

$wf \in \Sigma_{Ctx,Ty;Exp}$, given sorts for finite maps from variables to types $c : Map[Var, Ty]$ (abbreviated as Ctx) and types $t : Ty$.

$$
\begin{array}{rcl}
wf(c,t) & = & \exists a : Var.var(a) \wedge t = c[a] \\
 & \vee & \exists u : Ty.app(wf(c, fn(u,t)), wf(c,u)) \\
 & \vee & \exists t_1, t_2 : Ty.t = fn(t_1,t_2) \wedge lam(\text{И}\mathsf{a}.[\mathsf{a}]wf(c[\mathsf{a} := t_1], t_2))
\end{array}
$$

Here $c[a]$ denotes the (partial) operation that looks up a's binding in c and $c[a := t]$ the (partial) operation that extends c with a binding for a variable not already present in its domain.
e.g. $lam([\mathsf{a}]\mathsf{a}) \in wf([], fn(t,t))$ holds for any type t.
Finite maps satisfy standard axioms, such as

$$
c[a := t][a] = t \quad a \neq b \Rightarrow c[a := t][b] = c[b]
$$
$$
x \in c[a] \wedge y \in c[a] \Rightarrow x = y
$$

The standard property of subject reduction can be stated as:

$$red(wf(c,t)) \subseteq wf(c,t)$$

It can be proved using the axioms for red and wf and the following induction principle:

$$\begin{aligned}(\forall t\colon Ty, a\colon Var, c\colon Ctx.(var(a) \wedge t = c[a]) \subseteq P(c,t)) \Rightarrow \\ (\forall t_1\colon Ty, t_2\colon Ty, c\colon Ctx.app(P(c, fn(t_1,t_2)), P(c,t_1)) \subseteq P(c,t_2)) \Rightarrow \\ (\forall t_1\colon Ty, t_2\colon Ty, c\colon Ctx.lam(\text{Иa} : Var.[\mathsf{a}]P(c[\mathsf{a} := t_1], t_2)) \subseteq P(c, fn(t_1,t_2))) \\ \Rightarrow \forall c\colon Ctx, t\colon Ty.wf(c,t) \subseteq P(c,t)\end{aligned}$$

together with a lemma (well-typed substitutions preserve types):

$$\forall a,t,t',c,c'.c' \subseteq c \Rightarrow subst(wf(c[a := t'],t), a, wf(c',t')) \subseteq wf(c,t)$$

The λ-calculus in NML: Progress

$value = lam(\top)$ since we consider only a pure lambda-calculus.

Progress:

A well-formed closed term that is not weakly-reducible is a value:

$$wf([], t) \subseteq value \vee reducible$$

where $reducible$ is defined as $\exists x.x \wedge \exists y.y \in red(x)$

Auxiliary properties of $reducible$:

$$app(reducible, \top) \subseteq reducible \qquad app(lam(\top), \top) \subseteq reducible$$

Conclusions

- Being first-order, nominal logic is a natural candidate for supporting binding in Matching Logic (avoiding additional higher-order features and semantic complications)
- A straightforward approach (NL as a theory in ML) is workable, but has drawbacks
- We propose a deeper integration, supporting name constants (as in nominal unification) and using ML's support for nondeterminism and partial functions for fresh names and abstraction respectively
- We illustrate NML on small examples but much remains to consider, e.g. explicit induction/fixed point reasoning a la μML.