Semantic nominal terms

Murdoch J. Gabbay and Dominic P. Mulligan

1 Introduction

Nominal terms [UPG04] extend first-order syntax with binding symbols. They have been quite widely applied; examples include [FG07,CU04,LP08,Mat07]. Much of the flavour of first-order terms is preserved, and we can represent informal statements like

"If $y \notin fv(t)$ then $\lambda x.t$ is α-equivalent with $\lambda y.[y/x]t$", and
"How can we choose t and u to make $\lambda x.\lambda y.(y\ t)$ equal to $\lambda x.\lambda x.(x\ u)$?".

Nominal terms use a characteristic combination of features: Two levels of variable: *atoms* a and *unknowns* X. *Freshness conditions* $a\#X$ and *permutations* π. An abstraction $[a]r$ with a distinctive α-equivalence based on freshness and permutation.

The first statement above is rendered in nominal terms as the equality judgement $b\#X \vdash [a]X =_\alpha [b](b\ a) \cdot X$. a and b denote atoms, which represent the 'x' and 'y'; X denotes an unknown, it represents the 't'; $b\#X$ is a *freshness side-condition*, it represents the '$y \notin fv(t)$'; $(b\ a)$ is a *permutation* meaning 'map a to b and b to a', it represents the '$[y/x]$' (it can do this, since we assume $y \notin fv(t)$).

The second statement above is rendered as the nominal unification problem $\{\lambda[a]\lambda[b](b\ X) \stackrel{?}{=} \lambda[a]\lambda[a](a\ Y)\}$.

Nominal terms display some curious properties:

– The native notion of equivalence of first-order syntax is *syntactic identity*. For nominal terms, a notion of derivable α-equality is also needed. We cannot 'just quotient terms by α-equality' because α-equality is not a property of terms but of terms-in-freshness-contexts. Thus $[a]X = [b](b\ a) \cdot X$ has no meaning, only, for example, $\emptyset \vdash [a]X = [b](b\ a) \cdot X$ and $b\#X \vdash [a]X = [b](b\ a) \cdot X$ have meaning (the former is not derivable, the latter is).
– With first-order and higher-order terms, it is always possible to 'just pick a fresh name', and therefore we can 'always α-convert'.

In nominal terms it may not be possible to 'just pick a fresh atom' or 'just α-convert'. For example in the empty freshness context $\emptyset$, there is no a such that $a\#X$, and there is no atom b such that we can α-convert a to b in $\emptyset \vdash [a]X$. [1]

Nominal terms also raise a question which has puzzled us for some time. Atoms in nominal terms are denoted by themselves in the cumulative hierarchy (Definition 2.1, or [GP01]). Unknowns take denotation using a valuation (the standard first-order semantics for variables). There is nothing wrong with this but we find it inelegant because it is asymmetric with the denotation of atoms (and also, using functions to give semantics to unknowns is a circularity we want to avoid; one motivation of nominal terms is to axiomatise function binding). We want an alternative

[1] We encountered this before: In [GM07, Figure 2, axiom (**fr**)] and in [GM09, e.g. Lemma 25 and Theorem 33] we 'freshly extend' the freshness context. In [GL08,FG05] we can extend nominal terms syntax with an explicit 'fresh' binder.

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: `www.elsevier.nl/locate/entcs`

denotation of unknowns, which uses fixed elements in the cumulative hierarchy just as for atoms.

We will examine these issues through the prism of the following idea: nominal unknowns denote *infinite lists of distinct atoms*. This yields a structure in which unknowns have a single fixed 'nominal' denotation, very much like atoms (and unlike variables). We do not have to add an extra class of unknowns to the cumulative hierarchy; the denotation for unknowns is built out of what is already there. Freshness contexts are built in to terms. α-equivalence coincides with syntactic identity (for example in Definition 2.13, $[a]a$ and $[b]b$ are syntactically identical semantic nominal terms; in [UPG04] they are syntactically distinct nominal terms, which happen to be α-equivalent). We also generalise the nominal substitution action, such that capture-avoiding and capturing substitution fuse together.

2 Terms and semantics

Definition 2.1 Fix an uncountably infinite set $\mathbb{A}$ of **atoms**. $a, b, c, \ldots$ will range over distinct atoms (we call this the **permutative** convention).

If X is a set write $\mathcal{P}(X)$ for the powerset of X. Let α and β range over ordinals. Define the **cumulative hierarchy** universe by transfinite induction:

$$\mathcal{U}_{\beta+1} = \mathcal{U}_\beta \cup \mathcal{P}(\mathcal{U}_\beta) \qquad \mathcal{U}_\beta = \mathbb{A} \cup \bigcup_{\alpha<\beta} \mathcal{U}_\alpha \quad (\beta \text{ a limit ordinal}) \qquad \mathcal{U} = \bigcup_{\alpha} \mathcal{U}_\alpha$$

$x, y, z, \ldots$ will range over elements of $\mathcal{U}$.[2] Note that 0 is a limit ordinal, so $\mathcal{U}_0 = \mathbb{A}$.

Remark 2.2 Standard set-wise constructions are definable within $\mathcal{U}$. In particular, we can form ordered tuples $(x_1, \ldots, x_n)$, the set of integers $\mathbb{Z}$, function-sets, and so on. For details see [Joh87]. i will range over elements of $\mathbb{Z}$.

Remark 2.3 [GP01] uses a universe built on a *countable* set of atoms. For our purposes here, we just want access to infinite lists of atoms. A generalisation of 'finite support' by Cheney [Che04, Section 3, *support ideals*] could be used, which admits infinite support without using uncountably many atoms. Definition 2.1 follows [Gab07], which includes further treatment of the theory of $\mathcal{U}$.

Definition 2.4 A **permutation** is a bijection π on atoms such that $\{a \mid \pi(a) \neq a\}$ is finite. π, π', π'' will range over permutations (it may be that $\pi = \pi'$).

Write id for the **identity** permutation, such that $id(a) = a$ always. Write $\circ$ for functional composition, so $(\pi \circ \pi')(a) = \pi(\pi'(a))$ always. Write π^{-1} for the **inverse** of π. As standard [GP01] write $(b\ a)$ for the **swapping**, such that $(b\ a)(a) = b$, $(b\ a)(b) = a$, and $(b\ a)(c) = c$ for all other c.

Definition 2.5 Define a **permutation action** by: $\pi \cdot a = \pi(a)$ and $\pi \cdot U = \{\pi \cdot x \mid x \in U\}$ for $U \in \mathcal{U} \setminus \mathbb{A}$.

Definition 2.6 Write $a\#x$ when $\{b \mid (b\ a) \cdot x \neq x\}$ is countable. Write $supp(x) = \{a \mid \neg a\#x\}$; $supp(x)$ is the set of $a \in \mathbb{A}$ such that $\{b \mid (b\ a) \cdot x = x\}$ is uncountable.

[2] This is not the finitely-supported hierarchy of [GP01], but we do not need to restrict to finitely- or countably-supported elements. We need only use those parts of $\mathcal{U}$ which we find convenient.

Definition 2.7 and Lemmas 2.8 and 2.9 mirror [GP01,Gab07].

Definition 2.7 Define $[a]x = \{(a, x)\} \cup \{(b, (b\ a) \cdot x) \mid b\#x\}$.

Lemma 2.8 $\mathrm{supp}([a]x) = \mathrm{supp}(x) \setminus \{a\}$.

Lemma 2.9 $[a]x = [b]y$ *if and only if* $(b\ a) \cdot x = y$ *and* $b\#x$.

Definition 2.10 Let $\mathbb{A}_\omega$ range over injective functions X from $\mathbb{Z}$ to $\mathbb{A}$. We call these **semantic unknowns**. $X, Y, Z, \ldots$ will range over semantic unknowns.

We do not introduce a distinct class of tokens as in [UPG04, Definition 2.3]. Semantic unknowns are constructible in $\mathcal{U}$ with integers and injective functions ('for free', in the cumulative hierarchy). Semantic unknowns can be considered as infinite lists; X is identified with $(\ldots, X(-2), X(-1), X(0), X(1), X(2), \ldots)$.

Lemma 2.11 $\pi \cdot X$ *maps* i *to* $\pi(X(i))$. *Also,* $\mathrm{supp}(X) = \{X(i) \mid i \in \mathbb{Z}\}$.

In [UPG04, Definition 2.4], '$\pi \cdot X$' is written, but this is just a pair of π and X (π 'acts' when X is substituted for a term in in Definition 2.13, but still suspends on unknowns). In contrast, in Lemma 2.11 $\pi \cdot X$ is π acting on the element $X \in \mathcal{U}$.

Remark 2.12 $\mathrm{supp}(X) = \mathrm{supp}(Y)$ does not imply $X = Y$. $\mathrm{supp}(X) = \mathrm{supp}(Y)$ does not imply that there exists a permutation π such that $Y = \pi \cdot X$. We provide counterexamples: Fix a semantic unknown X. Consider Y mapping i to $X(i+1)$. Then $\mathrm{supp}(X) = \mathrm{supp}(Y)$ but $X \neq Y$; there is no permutation π such that $Y = \pi \cdot X$.

Thus X generates infinitely many 'incomparable' semantic unknowns with the same support. We can use this to inject nominal terms-in-context into semantic nominal terms. Details of the construction will be in a full paper; the reader can view [DGM09, Section 6], where the construction is carried out in a similar situation.

Definition 2.13 Define **semantic nominal terms** by:

$$r, s, t, \ldots ::= a \mid X \mid [a]r \mid (r_1, \ldots, r_n) \mid i \in \mathbb{Z}$$

r, s, t will range over semantic nominal terms. [3]

Here, $[a]r$ denotes abstraction in the sense of [GP01] and Definition 2.7.

Lemma 2.14 $\pi \cdot r$ *is inductively characterised by:*

$$\pi{\cdot}a{=}\pi(a) \quad (\pi{\cdot}X)(i){=}\pi(X(a)) \quad \pi{\cdot}[a]r{=}[\pi(a)]\pi{\cdot}r \quad \pi{\cdot}(r_1, \ldots){=}(\pi{\cdot}r_1, \ldots) \quad \pi{\cdot}i{=}i$$

Corollary 2.15 $\mathrm{supp}(r)$ *is inductively characterised by:*

$$\mathrm{supp}(a) = \{a\} \qquad \mathrm{supp}(X) = \{X(i) \mid i \in \mathbb{Z}\} \qquad \mathrm{supp}([a]r) = \mathrm{supp}(r) \setminus \{a\}$$
$$\mathrm{supp}(r_1, \ldots, r_n) = \bigcup_{1 \leq i \leq n} \mathrm{supp}(r_i) \qquad \mathrm{supp}(i) = \emptyset$$

In 'ordinary' nominal terms, to generate a fresh atom we must extend the freshness context (for instance, see [GM09, Definition 24]). Here, X has support $\mathrm{supp}(X)$, but $\mathbb{A} \setminus \mathrm{supp}(X)$ is infinite so an infinite supply of fresh atoms is guaranteed, and it is always possible to α-convert:

[3] We could take term-formers as primitive, but tuples and integers are more convenient here.

Theorem 2.16 *For every r, there exist an infinite number of a such that $a\#r$.*

Also, for every a and r there exist uncountably many b such that there exists some s such that $[a]r = [b]s$.

Proof The first part is by an easy induction on r. We consider only the case of X. $\mathrm{supp}(X)$ is countable and $\mathbb{A}$ is uncountable. The result follows.

The second part follows from Lemma 2.9, taking $s = (b\ a) \cdot r$. □

Some results from [UPG04], which have relatively long proofs by induction on derivable α-equality, become trivial in the 'semantic' world, because α-equality coincides now with identity. For example: "If $\pi(a) = \pi'(a)$ for every $a \in \mathit{supp}(r)$, then $\pi \cdot r = \pi' \cdot r$" [UPG04, Lemma 2.8] is now an easy property of supporting sets [GP01, Proposition 3.4]. "If $r = s$ then $\pi \cdot r = \pi \cdot s$" [UPG04, Theorem 2.11] is a fact of equality. This saving of labour scales well; the authors wish it had been available in [GM09] and [GM08], and look forward to using it in future.

3 A new, more general substitution action

Definition 3.1 Suppose $S \subseteq \mathbb{A}$. Specify π/S by:
– If $a \in S$ then $(\pi/S)(a) = \pi(a)$.
– If $a \notin S$ and $\pi^{-1}(a) \notin S$ then $(\pi/S)(a) = a$.
– If $a \notin S$ and $\pi^{-1}(a) \in S$ then $(\pi/S)(a) = a_n$, where $a_n \in S$ and there exist a_1, $\ldots$, a_{n-1} such that $\pi(a) = a_1$ and $a_i \notin S$ and $a_{i+1} = \pi(a_i)$ for $i < n$.

If $\pi = (a\ b\ c\ d\ e)(f\ g)$ (π maps a to b to c to d to e to a, and f to g to f), then $\pi/\{a\} = (a\ b)$ $\pi/\{a,b\} = (a\ b\ c)$ $\pi/\{a,c\} = (a\ b\ c\ d)$ $\pi/\{a,f\} = (a\ b)(f\ g)$.

Definition 3.2 Write $\mathit{Orb}(X)$ for $\{\pi \cdot X \mid \text{all } \pi\}$; call this the **(permutation) orbit** of X. A **substitution** is a function from semantic unknowns to terms such that for each orbit $\mathit{Orb}(X)$ there exists some **representative** $X \in \mathit{Orb}(X)$ such that $\sigma(\pi \cdot X) = (\pi/\mathit{supp}(X)) \cdot \sigma(X)$ for all π.

$\sigma, \sigma', \sigma'', \ldots$ will range over substitutions. Write $[X \mapsto t]$ for the function mapping $\pi \cdot X$ to $(\pi/\mathit{supp}(X)) \cdot t$, and all other Y to Y.

In [UPG04] substitutions must observe a freshness context (e.g. the '$\nabla' \vdash \theta(\nabla)$' in Lemma 2.14, and '$\nabla \vdash a\#\theta(t)$' in Definition 3.1 of [UPG04]). In the semantic world this corresponds with insisting that $\mathit{supp}(\sigma(X)) \subseteq \mathit{supp}(X)$ always; then $(\pi/\mathit{supp}(X)) \cdot \sigma(X) = \pi \cdot \sigma(X)$ (by [GP01, Proposition 3.4]) and semantic nominal terms substitution specialises to (a semantic casting of) nominal terms substitution.

Definition 3.3 Define a **substitution action** by:

$$a\sigma = a \qquad X\sigma = \sigma(X) \qquad (r_1, \ldots, r_n)\sigma = (r_1\sigma, \ldots, r_n\sigma) \qquad i\sigma = i$$
$$([a]r)\sigma = [b](b\ a) \cdot (r\sigma) \quad (b\#r,\ b\#r\sigma)$$

In the final clause, we make an arbitrary fixed choice of fresh b for each r and $r\sigma$.

Suppose that $a, b \in \mathrm{supp}(X)$ and $c, d \notin \mathrm{supp}(X)$. Then

$$X[X \mapsto a] = a \quad ([a]X)[X \mapsto a] = [b]b = [a]a \quad ([c]X)[X \mapsto c] = [d]c$$
$$([a][c]X)[X \mapsto (a,c)] = [a][d](a,c).$$

Our substitution is capturing and capture-avoiding; in the examples above a is captured, and c is not, because $a \in supp(X)$ and $c \notin supp(X)$.

Theorem 3.4 *If* $a \in \mathrm{supp}(X)$ *then* $([a]X)\sigma = [a]\sigma(X)$.
If $a \notin \mathrm{supp}(X)$ *then* $([a]X)\sigma = [b]\sigma(X)$, *for* $b\#X$ *and* $b\#\sigma(X)$.

Proof Suppose $a \in \mathrm{supp}(X)$. Recall our choice of fresh b for X and $\sigma(X)$. Note $((b\ a)/\mathrm{supp}(X)) = (b\ a)$. Using Lemma 2.9, $([a]X)\sigma = [b](b\ a) \cdot (X\sigma) = [a]\sigma(X)$.

Suppose $a \notin \mathrm{supp}(X)$. Recall our choice of fresh b for X and $\sigma(X)$. Note $((b\ a)/\mathrm{supp}(X)) = id$. Then $([a]X)\sigma = [b]id \cdot (X\sigma) = [b]\sigma(X)$. □

Further properties will be described in a full paper.

Nominal terms can be interpreted in semantic nominal terms using a construction like that in [DGM09] (Remark 2.12). The practical consequences of semantic nominal terms are unclear. Computationally, semantic nominal terms are a step back relative to nominal terms, but they are not computationally prohibitive. We can reason and program on them using well-founded induction in the sense of [GP01]; we should think of them as finite trees-with-binding. Infinite structure enters the trees through the the labels, which may include infinite lists of atoms, but that does not place implementation out of the question — consider infinite precision floating point arithmetic.

Semantic nominal terms provide a new semantics for nominal terms. As often happens, we trade abstraction for computational properties. However, it is good to have an abstract model to go with our concrete syntax, and the model has inherent interest, explaining nominal unknowns in terms of the existing universe instead of as variables ranging over elements of that universe.

References

[Che04] James Cheney. *Nominal Logic Programming.* PhD thesis, Cornell University, August 2004.

[CU04] James Cheney and Christian Urban. Alpha-prolog: A logic programming language with names, binding and alpha-equivalence. In *Proc. of the 20th Int'l Conf. on Logic Programming (ICLP 2004)*, number 3132 in Lecture Notes in Computer Science, pages 269–283. Springer, 2004.

[DGM09] Gilles Dowek, Murdoch J. Gabbay, and Dominic P. Mulligan. Permissive Nominal Terms and their Unification. Submitted to Rewriting Techniques and Applications RTA 2009. Available online at `http://www.gabbay.org.uk/papers/perntu.pdf`, 2009.

[FG05] Maribel Fernández and Murdoch J. Gabbay. Nominal rewriting with name generation: abstraction vs. locality. In *PPDP 2005*, pages 47–58. ACM Press, 2005.

[FG07] Maribel Fernández and Murdoch J. Gabbay. Nominal rewriting (journal version). *Information and Computation*, 205(6):917–965, 2007.

[Gab07] Murdoch J. Gabbay. A General Mathematics of Names. *Information and Computation*, 205(7):982–1011, July 2007.

[GL08] Murdoch J. Gabbay and Stéphane Lengrand. The lambda-context calculus. *Electronic Notes in Theoretical Computer Science*, 196:19–35, 2008.

[GM07] Murdoch J. Gabbay and Aad Mathijssen. A formal calculus for informal equality with binding. In *Proceedings of* 14^{th} *Workshop on Logic, Language and Information in Computation (WoLLIC 2007)*, volume 4576 of *Lecture Notes in Computer Science*, pages 162–176, 2007.

[GM08] Murdoch J. Gabbay and Dominic P. Mulligan. One-and-a-halfth Order Terms: Curry-Howard for Incomplete Derivations. In *Proceedings of* 15^{th} *Workshop on Logic, Language and Information in Computation (WoLLIC 2008)*, volume 5110 of *Lecture Notes in Artificial Intelligence*, pages 180–194, 2008.

[GM09] Murdoch J. Gabbay and Dominic P. Mulligan. Two-and-a-halfth Order Lambda-calculus. *Electronic Notes in Theoretical Computer Science*, 2009. To appear.

[GP01] Murdoch J. Gabbay and A. M. Pitts. A New Approach to Abstract Syntax with Variable Binding (journal version). *Formal Aspects of Computing*, 13(3–5):341–363, 2001.

[Joh87] P. T. Johnstone. *Notes on logic and set theory.* Cambridge University Press, 1987.

[LP08] M. R. Lakin and A. M. Pitts. A metalanguage for structural operational semantics. In *Trends in Functional Programming*, volume 8, pages 19–35. Intellect, 2008.

[Mat07] Aad Mathijssen. *Logical Calculi for Reasoning with Binding.* PhD thesis, Technische Universiteit Eindhoven, 2007.

[UPG04] Christian Urban, Andrew M. Pitts, and Murdoch J. Gabbay. Nominal Unification. *Theoretical Computer Science*, 323(1–3):473–497, 2004.