Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 174 (2007) 37–52
www.elsevier.com/locate/entcs

Hierarchical Nominal Terms and Their Theory of Rewriting

Murdoch J. Gabbay[1]

Computer Science Department
Heriot-Watt University, Riccarton
Edinburgh EH14 4AS
Great Britain
murdoch.gabbay@gmail.com

Abstract

Nominal rewriting introduced a novel method of specifying rewriting on syntax-with-binding. We extend this treatment of rewriting with hierarchy of variables representing increasingly 'meta-level' variables, e.g. in hierarchical nominal term rewriting the meta-level unknowns (representing unknown terms) in a rewrite rule can be 'folded into' the syntax itself (and rewritten). To the extent that rewriting is a mathematical meta-framework for logic and computation, and nominal rewriting is a framework with native support for binders, hierarchical nominal term rewriting is a meta-to-the-omega level framework for logic and computation with binders.

Keywords: Nominal rewriting, meta-theory of logic and programming, nominal techniques.

1 Introduction

Fix a set of **atoms** (or **object-level variable symbols**) $a, b, c, \ldots \in \mathbb{A}$ for the rest of this paper. The syntax of the λ-calculus is inductively generated by the grammar

$$s ::= a \mid ss \mid \lambda a.s.$$

Consider the λ-term '$\lambda a.s$'. Here s is a *meta-level* variable ranging over terms; s is not itself a λ-term.

Mathematical writing is full of this kind of language. Nominal terms model it closely. A relevant subset of nominal terms is inductively generated by the following grammar:

$$u ::= a \mid [a]u \mid \mathsf{f}(u, \ldots, u) \mid X$$

[1] Thanks to Aad Mathijssen, Alberto Momigliano, and anonymous referees for help and suggestions. We acknowledge the support of EPSRC grant number EP/C013573/1.

doi:10.1016/j.entcs.2007.01.017

Here X is one of a countably infinite collection of **unknowns symbols** $X, Y, Z, \ldots$; a represents object-level variable symbols; $[a]u$ represents abstraction; f is a term-former, for example λ.

X here corresponds to s above. $\lambda[a]X$ (the term-former λ syntactically acting on the abstraction of X with a) represents $\lambda a.s$.

Instantiation of X is direct textual replacement and does not avoid capture by abstractors, so $(\lambda[a]X)[a/X]$ is equal to $\lambda[a]a$ (here $[a/X]$ means 'instantiate X to a'). This is exactly what happens when we say 'take s to be a in $\lambda a.s$'; we expect to obtain $\lambda a.a$ and *not* $\lambda a'.a$, as a *capture-avoiding* notion of substitution delivers.

Nominal terms have a well-developed meta-theory [19,5,4]. A table presents the encoding of mathematical discourse into nominal terms:

$$\text{Meta-variable } \phi \text{ or } s \longmapsto \text{Unknown } X \qquad \text{Binding} \longmapsto \text{Abstraction}$$

But we used u as a meta-variable to range over nominal terms!

So we have not eliminated the meta-level, though we have internalised it. Does a language exist which is a fixed point of this process, in some sense? What if we iterate by allowing abstraction by unknowns $[X]u$, then internalise u as a 'stronger' unknown, and repeat this again, and again, and infinitely often? Taking the limit we obtain hierarchical nominal terms, in which infinitely many levels of meta-level discourse can be represented. What is the mathematics of this new language?

We give a theory of rewriting and a critical pairs result; there turn out to be unexpected differences with respect to nominal terms, which only have one level of atoms. We give example rewriting theories of substitution, scope and scope-extrusion, a λ-calculus, and a treatment of α-equivalence. This is arguably a comprehensive range of applications with which we lay groundwork for more advanced investigations.

2 Hierarchical nominal terms

Fix a set of **term-formers** f.

For each number $i \geq 1$ fix disjoint countably infinite **sets of atoms** $a_i, b_i, c_i, \ldots$. Say that a_i **has level** i.

The syntax of **hierarchical nominal terms** is inductively defined by

$$t, u, v ::= a_i \mid X \mid [a_i]t \mid \mathsf{f}(t, \ldots, t).$$

We may call a_i an 'atom of level i'. The intuition here is of a 'hole' which behaves like a variable towards weaker atoms, and like a constant symbol towards stronger atoms. Intuitively, weaker atoms have no access to stronger atoms; they must wait for those stronger atoms to 'become' terms; stronger atoms on the other hand have full access to weaker atoms, including to their names.

As for the rest of the syntax, $[a_i]t$ is an abstraction and $\mathsf{f}(t_1, \ldots, t_n)$ is a term-former applied to some terms. Subscripts on t, u, and v are for identification only and have nothing to do with levels. We shall see examples later; for now it suffices

to mention that λ and $\forall$ are example term-formers, but also $+$ and 2, and $\lambda[a_2]c_1$, $\lambda[a_2]b_3$, $2+2$, and a_1+2, are valid hierarchical nominal terms.

Unknowns X are variable symbols representing unknown terms. They behave like atoms of level ω. We still *need* unknowns because *something* has to represent unknown terms so that we can define rewrite rules and do rewriting!

For the rest of this paper we adhere to a convention that i, j, k vary over nonzero natural numbers and a_i, b_i, c_i range *permutatively* over atoms of level i. That is, a_i and b_i represent two *distinct* atoms of the same level so $a_i \neq b_j$ holds necessarily because we called on atom a_i and the other b_j. If we write a_i and c_k and $i = k$ then by our convention we assume that a_i and c_k are distinct. Typically it will be the case that $k \leq i < j$, though not always; we shall always be clear about what we assume, when we assume it.

Call a pair of an atom and a term $a_i\#t$ a **freshness assertion**. The intuition is 'a_i does not occur in t'. For example we expect $a_2\#a_1$ to hold, because a_1 is 'far too weak and puny' to ever have a hole as big as a_2. We do not expect $a_1\#a_2$ to hold, necessarily.

Inductively define a notion of entailment on freshness assertions as follows:

$$\frac{}{a_i\#c_k}\,(\#\mathbf{diff})\quad (k \leq i) \qquad \frac{a_i\#t_1 \ldots a_i\#t_n}{a_i\#\mathsf{f}(t_1,\ldots,t_n)}\,(\#\mathsf{f})$$

$$\frac{}{a_i\#[a_i]t}\,(\#\mathbf{abs}{=}) \qquad \frac{\begin{array}{c}[a_i\#b_j]\\ \vdots \\ a_i\#t\end{array}}{a_i\#[b_j]t}\,(\#\mathbf{abs}{<})\quad (i<j) \qquad \frac{a_i\#t}{a_i\#[c_k]t}\,(\#\mathbf{abs}{\geq})\ (i \geq k)$$

- (#**diff**): This implements our intuition that a strong atom 'looks like' an unknown to weaker atoms but not conversely (since it may contain them, but not conversely). Between atoms of the same level, # encodes distinctness.
- (#**abs**=), (#**abs**<), and (#**abs**≥): a_i is abstracted in $[a_i]t$. Intuitively, we can prove that a_i is abstracted in $[b_j]t$ when we can prove it is abstracted in t, where we are allowed to assume that a_i is abstracted in the hole in t called b_j. In (#**abs**<) $[a_i\#b_j]$ denotes **discharge** in the natural deduction sense [2]; in sequent style (#**abs**<) would be

$$\frac{\Phi, a_i\#b_j \vdash a_i\#t}{\Phi \vdash a_i\#[b_j]t}$$

This is a surprising twist not present in normal nominal terms and their freshness [19,5]. So we are able to derive $a_i\#[b_j]b_j$ always even if $j > i$ because of the extra freshness assumption. This issue does not arise when proving $a_i\#[c_k]u$ for $k \leq i$, because then we can deduce $a_i\#c_k$ with (#**diff**). In particular this issue *cannot* arise if there is only one level, as is the case for nominal terms. [2]

[2] This insight derives partly from work with Giulio Manzonetto during his visit to Université Paris VII in

- (#f): An atom is fresh for $\mathsf{f}(t_1,\ldots,t_n)$ when it is fresh for all the t_1 up to t_n.
- There is no rule for deriving $a_i\#X$; the only way to know this, is to assume it beforehand. In this sense X is like an atom of level ω.

Call $a_i\#b_j$ for $j > i$ or $a_i\#X$ **primitive freshness assertions**. Call a possibly infinite set Δ of freshness assertions a **freshness context**. Call Δ **primitive** when all the assertions it contains are primitive.

We say $a_i\#t$ is **entailed** by Δ and write $\Delta \vdash a_i\#t$, when $a_i\#t$ can be derived from Δ using these rules. If Δ is empty write $\Delta \vdash a_i\#t$ just as $\vdash a_i\#t$. If Δ' is another freshness context write $\Delta \vdash \Delta'$ when $\Delta \vdash a_i\#t$ for every $a_i\#t \in \Delta'$.

We now develop the primitive notion of substitution for unknowns, and in the next section we treat unification and finally rewriting.

A **substitution** σ is a map from unknowns to hierarchical nominal terms. Write $[X{:=}t]$ for the substitution mapping X to t and Y to Y. We extend the action of substitutions to all hierarchical nominal terms by

$$a_i\sigma = a_i \quad X\sigma = \sigma(X) \quad ([a_i]t)\sigma = [a_i](t\sigma) \quad \mathsf{f}(t_1,\ldots,t_n)\sigma = \mathsf{f}(t_1\sigma,\ldots,t_n\sigma).$$

Extend the substitution action point-wise to things mentioning terms, such as sets of terms and freshness assertions, and sets thereof. For example, if Δ is a freshness context then $\Delta\sigma = \{a_i\#(t\sigma) \mid a_i\#t \in \Delta\}$.

The rules above are highly syntax-directed and have a computational content by which we can calculate for each $a_i\#t$ a minimal (in a suitable sense) set of assumptions necessary to entail it:

$$b_j\#a_i, \Delta \Longrightarrow \Delta \quad (j > i) \qquad\qquad a_i\#b_i, \Delta \Longrightarrow \Delta$$

$$a_i\#\mathsf{f}(t_1,\ldots,t_n), \Delta \Longrightarrow a_i\#t_1,\ldots,a_i\#t_n,\Delta \qquad a_i\#[a_i]t, \Delta \Longrightarrow \Delta$$

$$\frac{a_i\#t, \Delta \Longrightarrow \Delta' \cup S}{a_i\#[b_j]t, \Delta \Longrightarrow \Delta'} \quad (i{<}j,\ S{\subseteq}\{a_i\#b_j\}) \qquad a_i\#[c_k]t, \Delta \Longrightarrow a_i\#t, \Delta \quad (k{\leq}i)$$

Here we omit singleton set brackets, e.g. writing $a_i\#b_j$ for $\{a_i\#b_j\}$. On the left of the arrow $\Longrightarrow$ comma indicates disjoint set union. On the right of the arrow comma indicates possibly non-disjoint set union.

The following results are easy to prove:

Lemma 2.1 • *If Δ is finite and $\Delta \Longrightarrow \Delta'$ then Δ' is finite.*

- *$\Longrightarrow$ is terminating as a rewrite relation on finite freshness contexts.*
- *$\Longrightarrow$ is confluent on possibly infinite freshness contexts.*

Proof. The first part is easy.

2004, while I was supported by LIX and the École Polytechnique.

For the second part assign a numerical measure $|t|$ to terms by: $|a_i| = 1$, $\mathsf{f}(t_1,\ldots,t_n) = \Sigma_{1\leq i\leq n}|t_i| + 1$, $|[a_i]t| = |t| + 1$. Extend the measure to Δ by taking $|\Delta|$ to be a function on numbers $n > 0$ given by: $|\Delta|(n)$ is the number of freshness assertions $a_i\#t$ in Δ such that $|t| = n$. For a suitable ordering on such functions (essentially a lexicographic ordering) it is very easy to show that $\Longrightarrow$ makes the measure strictly decrease.

For the third part, we must show that if $\Delta \Longrightarrow \Delta_1$ and $\Delta \Longrightarrow \Delta_2$, then there is some Δ' such that $\Delta_1 \Longrightarrow \Delta'$ and $\Delta_2 \Longrightarrow \Delta'$. We can prove this by considering all possible cases for both reductions; this is long but routine. □

Write $\langle\Delta\rangle_{nf}$ for the unique $\Longrightarrow$ normal form of Δ. It is not hard to check that $\langle\Delta\rangle_{nf}$ is of the form $\Delta' \cup \Delta''$ where Δ' is a primitive freshness context and Δ'' contains only problems of the form $a_i\#a_i$. If Δ'' is empty call Δ **consistent**, otherwise call Δ **inconsistent**. Intuitively, Δ is 'satisfiable' if and only if it is consistent. Obviously, Δ is consistent if and only if $\langle\Delta\rangle_{nf}$ is consistent.

Lemma 2.2 *Suppose that Δ is a primitive freshness context and suppose that $\{a_i\#t\}$ is consistent. Then $\Delta \vdash a_i\#t$ if and only if $\Delta \vdash \langle a_i\#t\rangle_{nf}$.*

Proof. By an easy induction on the derivation of $\Delta \vdash a_i\#t$. □

Lemma 2.3 *If Δ and $\Delta\sigma$ are consistent and $\Delta \vdash a_i\#t$ then $\langle\Delta\sigma\rangle_{nf} \vdash a_i\#(t\sigma)$.*

Proof. By induction on the derivation of $\Delta \vdash a_i\#t$. We consider two cases:

- Suppose our derivation of $\Delta \vdash a_i\#[b_j]u$ concludes with (#**abs**<). Then we have a derivation of $\Delta, a_i\#b_j \vdash a_i\#u$. Note that $a_i\#(b_j\sigma)$ equals $a_i\#b_j$. By inductive hypothesis $\langle\Delta\sigma\rangle_{nf}, a_i\#b_j \vdash a_i\#u\sigma$ is derivable. The result follows.
- Suppose $\Delta \vdash a_i\#X$ holds because $a_i\#X \in \Delta$. By Lemma 2.2 we can deduce that $\langle a_i\#\sigma(X)\rangle_{nf} \vdash a_i\#\sigma(X)$ and by some easy calculations the result follows.

□

3 Unification

An **equality assertion** is a pair of terms $t = u$. We say that '$t = u$ **holds**' when t and u are syntactically identical, we may abbreviate this just to '$t = u$', and we may shorten '$t = u$ does not hold' to $t \neq u$.

A **unification problem** is a set of freshness or equality assertions Φ. We define a **noninstantiating** reduction relation on these unification problems as follows:

$$a_i\#t \Longrightarrow \langle a_i\#t\rangle_{nf} \qquad a_i = a_i, \Phi \Longrightarrow \Phi \qquad X = X, \Phi \Longrightarrow \Phi$$
$$[a_i]t = [a_i]u, \Phi \Longrightarrow t = u, \Phi$$
$$\mathsf{f}(t_1,\ldots,t_n) = \mathsf{f}(u_1,\ldots,u_n), \Phi \Longrightarrow t_1 = u_1,\ldots,t_n = u_n, \Phi$$

Here we omit singleton set brackets, e.g. writing $t = u$ for $\{t = u\}$. On the left of the arrow $\Longrightarrow$, comma indicates disjoint set union. On the right of the arrow comma indicates possibly non-disjoint set union.

Lemma 3.1 *The noninstantiating reductions on unification problems are terminating and confluent.*

Recall that $[X{:=}t]$ is the substitution mapping X to t. We may extend the reduction relation with **instantiating rules** as follows:

$$X = u, \Phi \stackrel{X:=u}{\Longrightarrow} \Phi[X{:=}u] \qquad\qquad t = X, \Phi \stackrel{X:=t}{\Longrightarrow} \Phi[X{:=}t]$$

Here we extend the substitution action point-wise to the terms in the freshness or equality assertions in Φ.

Call the following equality assertions **reduced**:

- $a_i = b_j$.
- $X = t$ and X occurs in t.
- $\mathsf{f}(t_1, \ldots, t_m) = \mathsf{g}(u_1, \ldots, u_n)$ (where f and g are different term-formers).
- $a_i = \mathsf{g}(u_1, \ldots, u_n)$, or $a_i = [b_j]u$, or $a_i = [a_i]u$, or $\mathsf{f}(t_1, \ldots, t_n) = [b_j]u$, or symmetric versions such as $[a_i]t = a_i$.

We may call reduced equality assertions **inconsistent**.

A **solution** to a unification problem Φ is a pair (Γ, σ) of a consistent hierarchical nominal freshness context Γ and a substitution σ such that

- For every $t = u \in \Phi$ it is the case that $t\sigma = u\sigma$.
- For every $a_i \# t \in \Phi$ it is the case that $\Gamma \vdash a_i \# t\sigma$.
- For every X it is the case that X does not occur in $X\sigma$ (or equivalently, $X\sigma\sigma = X\sigma$).

Lemma 3.2 *If* $\Phi \Longrightarrow \Phi'$ *then* (Γ, σ) *solves* Φ *if and only if* (Γ, σ) *solves* Φ'.

Lemma 3.3 *If* $\Phi \stackrel{X:=t}{\Longrightarrow} \Phi'$ *then* (Γ, σ) *solves* Φ *if and only if* (Γ, σ) *solves* Φ'.

Define a partial ordering on solutions to a hierarchical nominal unification problem by: $(\Gamma', \sigma') < (\Gamma, \sigma)$ when for some σ'' it is the case that $\Gamma' \vdash \Gamma\sigma''$ and $X\sigma' = X\sigma\sigma''$ for all X.

Say a solution to a problem is **principal** when it is a least element in the instantiation ordering amongst solutions to the problem.

Theorem 3.4 *The non-instantiating and instantiating rules are terminating. Write* $\langle\Phi\rangle_{nf}$ *for some arbitrary but fixed choice of normal form. Then* $\langle\Phi\rangle_{nf}$ *is a least element in the set of solutions to* Φ*, and* $\langle\Phi\rangle_{nf}$ *is principal.*

Proof. By a standard proof-method similar to that used to prove Lemma 36 in [5] and using the previous two lemmas; the hierarchy causes no difficulties since we are only acting on unknowns. □

So unification of hierarchical nominal terms is decidable and has principal solutions.

4 Hierarchical nominal rewrite rules

To 'do' rewriting we need to be able to address some position within a term (at which to do the rewrite!).

Say a term has a **position** when it mentions a distinguished unknown, we usually write it -, precisely once (which identifies the position in the term at which that unknown occurs). Let L, C, P vary over terms with a position. Write $C[s]$ for $C[\text{-}\mapsto s]$ and write [-] when the term *is* its (unique) unknown. Since C is only of interest inasmuch as - may be substituted for a term, we tend to silently assume - is fresh, and we may say 'C is a position' when we mean 'C is a term *with* a distinguished position'.

For example, $[a_1](a_1, \text{-})$ is position and $(\text{-}, \text{-})$ is not.

We can now get down to defining rewriting and proving some of its properties. A **hierarchical nominal rewrite rule** is a triple

$$\nabla \vdash l \longrightarrow r$$

where ∇ is a primitive freshness context (primitive freshness contexts are necessarily consistent) and l and r are terms, such that r and ∇ mention only unknowns in l.

If $(\mathbf{R}) = \nabla \vdash l \longrightarrow r$ and $\Delta \vdash t$ is a hierarchical nominal term-in-context, write $\Delta \vdash t \xrightarrow{(\mathbf{R})} u$ and say '$\Delta \vdash t$ rewrites with $(\mathbf{R})$ to u' when

- There is a position C and substitution σ such that
- $\Delta \vdash \nabla\sigma$ and
- $C[l\sigma] = t$, and $C[r\sigma] = u$.

Write $\longrightarrow^*$ for the reflexive transitive closure of $\longrightarrow$. So $\Delta \vdash t \longrightarrow^* u$ holds when $t = u$ or when there is some sequence of $\longrightarrow$-reductions from t to u. If Δ is irrelevant or known we may write $\Delta \vdash t \longrightarrow^* u$ as just $t \longrightarrow^* u$.

Call a possibly infinite set of hierarchical nominal rewrite rules a **hierarchical nominal (term) rewrite system**.

Call a hierarchical nominal rewrite system **confluent** when if $\Delta \vdash t \longrightarrow^* u$ and $\Delta \vdash t \longrightarrow^* u'$, then v exists such that $\Delta \vdash u \longrightarrow^* v$ and $\Delta \vdash u' \longrightarrow^* v$.

Confluence is an important property because it ensures uniqueness of normal forms, a form of determinism. Local confluence is a weaker property, it is defined as 'joinability of peaks'. More precisely:

Call a pair of rewrites of the form $\Delta \vdash t \longrightarrow u_1$ and $\Delta \vdash t \longrightarrow u_2$ a **peak**. Call a hierarchical nominal rewrite system **locally confluent** when if $\Delta \vdash t \longrightarrow u_1$ and $\Delta \vdash t \longrightarrow u_2$, then a v exists such that $\Delta \vdash u_1 \longrightarrow^* v$ and $\Delta \vdash u_2 \longrightarrow^* v$. We may call such a peak **joinable**.

Suppose:

(i) $R_i = \nabla_i \vdash l_i \longrightarrow r_i$ for $i = 1, 2$ are two rules mentioning disjoint unknowns,

(ii) $l_1 = L[l_1']$ such that $\nabla_1, \nabla_2, l_1' = l_2$ has a principal solution (Γ, θ), so that $l_1'\theta = l_2\theta$ and $\Gamma \vdash \nabla_1\theta$, $\nabla_2\theta$.

Then call the pair of terms-in-context

$$\Gamma \vdash (r_1\theta, L\theta[r_2\theta])$$

a **critical pair**. If $L = [\text{-}]$ and R_1, R_2 are copies of the same rule, or if l'_1 is an unknown, then we call the critical pair **trivial**.[3]

Call a rewrite rule $R = \nabla \vdash l \longrightarrow r$ **uniform** when if $\Delta \vdash t \xrightarrow{R} u$ then $\Delta, \langle a_i\#t\rangle_{nf} \vdash a_i\#u$ for any a_i such that $\langle a_i\#t\rangle_{nf}$ is consistent.

Checking uniformity looks hard. In fact it is not:

Lemma 4.1 *$R = \nabla \vdash l \longrightarrow r$ is uniform if and only if $\nabla, \langle a_i\#l\rangle_{nf} \vdash a_i\#r$ for all a_i occurring in the syntax of R, and for one atom a which does not.*

Proof. See [5]. □

Uniformity ensures freshness properties are not destroyed by rewriting:

Lemma 4.2 *If R is uniform and $\Delta \vdash t \xrightarrow{R} u$ and $\Delta \vdash a_i\#t$, then $\Delta \vdash a_i\#u$.*

Proof. Suppose $\Delta \vdash a_i\#t$. By uniformity $\Delta, \langle a_i\#t\rangle_{nf} \vdash a_i\#u$. By elementary properties of natural deduction style proofs, $\Delta \vdash a_i\#u$. □

Theorem 4.3 *In a* uniform *rewrite system, peaks which are instances of trivial critical pairs are joinable.*

Proof. Suppose two rules $R_i = \nabla_i \vdash l_i \to r_i$ for $i = 1, 2$ have a critical pair

$$\Gamma \vdash (r_1\theta, L\theta[r_2\theta])$$

Then $l_1 = L[l'_1]$, and (Γ, θ) is such that $l'_1\theta = l_2\theta$, and $\Gamma \vdash \nabla_1\theta, \nabla_2\theta$. Recall also that we call the critical pair trivial when $L = [\text{-}]$ and R_1, R_2 are copies of the same rule, or l'_1 is a unknown.

If R_1 and R_2 are identical, then their rewrites are identical. If R_1 and R_2 differ and l'_1 is a unknown, then the only way we might not be able to apply R_1 in $L\theta[r_2\theta]$ or its instances, is if some freshness condition on l'_1 in ∇_1 is unsatisfiable after R_2, which was satisfiable before R_2. For uniform rules Lemma 4.2 guarantees that this cannot happen. □

5 Rewrites for substitution

Our idea when designing hierarchical nominal rewriting is that it should be able to represent meta-levels and instantiation. We used atoms to represent variable symbols. Our first task is therefore to use the framework of rewriting to give some framework by which atoms may be instantiated to terms.

[3] We assume that unknowns in rules may be renamed. This is standard both in first-order and nominal rewriting [5].

Introduce a binary term-former sub and sugar $\mathsf{sub}([a]u,t)$ to $u[a{\mapsto}t]$. Rewrites for sub are:

$$\begin{array}{ll}
\textbf{(suba)} & a_i[a_i{\mapsto}X] \longrightarrow X \\
\textbf{(sub\#)} & a_i\#Z \vdash Z[a_i{\mapsto}X] \longrightarrow Z \\
\textbf{(subaa)} & Z[a_i{\mapsto}a_i] \longrightarrow Z \\
\textbf{(subf)} & \mathsf{f}(Z_1,\ldots,Z_n)[a_i{\mapsto}X] \longrightarrow \mathsf{f}(Z_1[a_i{\mapsto}X],\ldots,Z_n[a_i{\mapsto}X]) \\
\textbf{(subabs}{>}\textbf{)} & ([c_k]Z)[a_i{\mapsto}X] \longrightarrow [c_k](Z[a_i{\mapsto}X]) \qquad (i>k) \\
\textbf{(subabs}{\leq}\textbf{)} & b_j\#X \vdash ([b_j]Z)[a_i{\mapsto}X] \longrightarrow [b_j](Z[a_i{\mapsto}X]) \qquad (i\leq j)
\end{array}$$

These are axiom-schemes for all i and j and every n, and for every term-former f (if we like). We could avoid having such schemes by enriching the syntax of rewrite rules, but it does not seem worth the trouble here. We *always* assume at least an axiom **(subsub)**.

Even without term-formers aside from sub, these rules have very interesting structure. The following rewrites are derivable, where here $j>i$ and $k\leq i$:

$$\begin{array}{r}
Z[a_i{\mapsto}X][b_j{\mapsto}Y] \longrightarrow^* Z[b_j{\mapsto}Y][a_i{\mapsto}X[b_j{\mapsto}Y]] \\
a_i\#Y \vdash Z[a_i{\mapsto}X][c_k{\mapsto}Y] \longrightarrow^* Z[c_k{\mapsto}Y][a_i{\mapsto}X[c_k{\mapsto}Y]]
\end{array} \quad (1)$$

Rewrites for the first case are:

$$\begin{array}{l}
Z[a_i{\mapsto}X][b_j{\mapsto}Y] \overset{\textbf{(subf)}}{\longrightarrow^*} \mathsf{sub}(([a_i]Z)[b_j{\mapsto}Y], X[b_j{\mapsto}Y]) \\
\qquad\qquad \overset{\textbf{(subabs}{>}\textbf{)}}{\longrightarrow^*} \mathsf{sub}([a_i](Z[b_j{\mapsto}Y]), X[b_j{\mapsto}Y]) = Z[b_j{\mapsto}Y][a_i{\mapsto}X[b_j{\mapsto}Y]].
\end{array}$$

The second case is similar, but we have to use **(subabs$\leq$)** and to do that we must prove $a_i\#Y$.

Thus *strong* substitution distributes over *weak* substitution without avoiding capture whereas *weak* substitution distributes over *strong* substitution but only subject to a capture-avoidance condition $b_j\#X$. Thus $a_2[a_1{\mapsto}2][a_2{\mapsto}a_1] \longrightarrow^* 2$ arguably models *exactly* what we mean when we say 'let t be a in t with a replaced by 2' (where 2 is some term-former; any term would do as well).

Since rewriting is more general than a particular calculus or logic, this example is meant to exhibit capture-avoiding substitution, and non-capture-avoiding instantiation, as two sides of a single unified theory of sub.

Recall that *uniform* rewrite rules satisfy Theorem 4.3.

Theorem 5.1 *The rewrite rules for* sub *are all uniform. Nontrivial critical pairs may be joined. The rules above are locally confluent.*

Proof. By Lemma 4.1 we need only check a finite number of properties such as $\langle a_i\#a_i[a_i{\mapsto}X]\rangle_{nf} \vdash a_i\#X$. They are all routine. It is detailed but routine to check the nontrivial critical pairs. The third part follows by standard reasoning using Theorem 4.3. □

We believe that our rewrite system is confluent but proving this is nontrivial even in the two-level case. The problem is (1) above, which is non-directed and makes terms syntactically larger. These problems *have* been investigated and overcome (see [7] and see the brief discussion in Section 7) but investigating them here is outside the scope of this paper.

6 Scope extrusion of И

Introduce a term-former И. Sugar $И[a_i]t$ to $Иa_i.t$. Read this as 'generate a fresh name a_i in t'.

Our framework can express scope-extrusion rules consistent with this intuition, similar to the behaviour of the π-calculus restriction operator ν [13]. Assume the term-formers and rewrites of substitution above. Introduce rewrites:

$$(И\#) \qquad b_j\#Z \vdash Иb_j.Z \longrightarrow Z$$
$$(Иsub) \qquad b_j\#Y \vdash И(([b_j]Z)[a_i{\mapsto}Y]) \longrightarrow Иb_j.(Z[a_i{\mapsto}Y]) \qquad (j > i)$$

The effect of (Иsub) is handled by (**subabs**>) when $j \leq i$, so the following rewrite is *always* valid:

$$b_j\#Y \vdash И(([b_j]Z)[a_i{\mapsto}Y]) \longrightarrow Иb_j.(Z[a_i{\mapsto}Y])$$

This beautifully implements that the abstracted atom *really is* private in the scope of И. There is no rewrite

$$(И\bot) \qquad b_j\#Z \vdash Z[a_i{\mapsto}Иb_j.Y] \longrightarrow Иb_j.(Z[a_i{\mapsto}Y])$$

because substitution might copy $Иb_j.Y$ and each copy should have a private copy of the fresh atom. For example assume a term-former f and consider scope-extrusion rewrite rules

$$b_j\#Y \vdash \mathsf{f}(Иb_j.X, Y) \longrightarrow Иb_j.\mathsf{f}(X,Y) \quad b_j\#X \vdash \mathsf{f}(X, Иb_j.Y) \longrightarrow Иb_j.\mathsf{f}(X,Y).$$

Then we may reduce

$$\vdash (\mathsf{f}(a_i,a_i))[a_i{\mapsto}Иb_j.b_j] \overset{(\mathbf{subf}),(\mathbf{suba})}{\longrightarrow^*} \mathsf{f}(Иb_j.b_j, Иb_j.b_j).$$

In the presence of (И⊥) there is a second reduction path:

$$\vdash (\mathsf{f}(a_i,a_i))[a_i{\mapsto}Иb_j.b_j] \overset{(И\bot)}{\longrightarrow} Иb_j.(\mathsf{f}(a_i,a_i)[a_i{\mapsto}b_j]) \overset{(\mathbf{subf}),(\mathbf{suba})}{\longrightarrow^*} Иb_j.(b_j,b_j)$$

This is not desired behaviour so we rule out (И⊥).

Theorem 6.1 (И#) *and* (Иsub) *are uniform (and so we can apply tools such as Theorem 4.3 to rewrite systems making use of* И*).*

Proof. We must show that:

$$b_j\#Z,\ \langle a_i\#\mathit{И}b_j.Z\rangle_{nf} \vdash a_i\#Z \qquad b_j\#Z,\ \langle b_j\#\mathit{И}b_j.Z\rangle_{nf} \vdash b_j\#Z$$
$$b_j\#Z,\ \langle c_k\#[a_i]\mathit{И}b_j.Z\rangle_{nf} \vdash c_k\#\mathit{И}b_j.[a_i]Z$$
$$b_j\#Z,\ \langle a_i\#[a_i]\mathit{И}b_j.Z\rangle_{nf} \vdash a_i\#\mathit{И}b_j.[a_i]Z$$
$$b_j\#Z,\ \langle b_j\#[a_i]\mathit{И}b_j.Z\rangle_{nf} \vdash b_j\#\mathit{И}b_j.[a_i]Z.$$

We consider a few cases, they are very easy:

- $\langle a_i\#\mathit{И}b_j.Z\rangle_{nf} = \{a_i\#Z\}$. The result follows.
- $b_j\#Z \in \{b_j\#Z\}$. The result follows.
- $\langle c_j\#[a_i]\mathit{И}b_j.Z\rangle_{nf} = c_j\#Z$. The result follows using the derivation rules for freshness assertions.
- $\vdash a_i\#\mathit{И}b_j.[a_i]Z$ and $\vdash b_j\#\mathit{И}b_j.[a_i]Z$ are easy to derive using the derivation rules for freshness assertions. The result follows.

□

7 A hierarchical λ-calculus

Assume term-formers sub, $\mathit{И}$, λ and app. Sugar $\mathsf{app}(t,u)$ to tu. Sugar $\lambda([a_i]t)$ to $\lambda a_i.t$. Rewrites of a hierarchical λ-calculus are given by rewrites:

$$(\beta)\quad (\lambda a_i.Z)X \longrightarrow Z[a_i{\mapsto}X] \qquad (\sigma\#)\quad a_i\#Z \vdash Z[a_i{\mapsto}X] \longrightarrow Z$$
$$(\sigma\mathbf{a})\quad a_i[a_i{\mapsto}X] \longrightarrow X$$
$$(\sigma\mathbf{p})\quad (a_iZ_1\dots Z_n)[b_j{\mapsto}Y] \longrightarrow (a_i[b_j{\mapsto}Y])\dots(Z_n[b_j{\mapsto}Y])$$
$$(\sigma\mathbf{p}')\quad (a_iZ_1\dots Z_n)[a_i{\mapsto}X] \longrightarrow (a_i[a_i{\mapsto}X])\dots(Z_n[a_i{\mapsto}X])$$
$$(\sigma\sigma)\quad Z[a_i{\mapsto}X][b_j{\mapsto}Y] \longrightarrow Z[b_j{\mapsto}Y][a_i{\mapsto}X[b_j{\mapsto}Y]] \quad (j{>}i)$$
$$(\sigma\lambda)\quad a_i\#X \vdash (\lambda a_i.Z)[c_k{\mapsto}X] \longrightarrow \lambda a_i.(Z[c_k{\mapsto}X]) \quad (k{\leq}i)$$
$$(\sigma\lambda')\quad (\lambda a_i.Z)[b_j{\mapsto}Y] \longrightarrow \lambda a_i.(Z[b_j{\mapsto}Y]) \quad (j{>}i)$$
$$(\sigma\mathbf{tr})\quad Z[a_i{\mapsto}a_i] \longrightarrow Z \qquad (\mathit{И}\mathbf{p})\quad b_j\#Y \vdash (\mathit{И}b_j.X)Y \longrightarrow \mathit{И}b_j.(XY)$$
$$(\mathit{И}\lambda)\quad \lambda a_i.\mathit{И}b_j.Z \longrightarrow \mathit{И}b_j.\lambda a_i.Z \qquad (\mathit{И}\#)\quad b_j\#Z \vdash \mathit{И}b_j.Z \longrightarrow Z$$
$$(\mathit{И}\sigma)\quad b_j\#X \vdash (\mathit{И}b_j.Z)[a_i{\mapsto}X] \longrightarrow \mathit{И}b_j.(Z[a_i{\mapsto}X])$$

This system is discussed in detail elsewhere [6], though we since simplified the presentation. Note the weaker treatment of substitution compared to Section 5, e.g. $(\sigma\mathbf{p})$ and $(\sigma\mathbf{p}')$ and a closely connected *lack* of a rule $(\sigma\sigma')$ corresponding to (1). This is what is needed to avoid (1) while retaining confluence.

Theorem 7.1 *Rewrites in the system above are confluent.*

Proof. For local confluence it suffices by Theorem 4.3 along with some standard further calculations, to check that *nontrivial* critical pairs may be joined. This is detailed work but essentially routine. For confluence we use Theorem 7.2 below taking $\mathcal{R}_1$ to be the system with just (β), and $\mathcal{R}_2$ to be the system with all the other rules. □

Call two hierarchical nominal term rewrite systems $\mathcal{R}_1$ and $\mathcal{R}_2$ when there is no nontrivial critical pair between a pair of rules one in $\mathcal{R}_1$ and the other in $\mathcal{R}_2$.

Theorem 7.2 *If $\mathcal{R}_1$ and $\mathcal{R}_2$ are left-linear, confluent, and orthogonal, then their union is confluent.*

The proof is identical to one the literature [18, Thm 5.10.5].

Call a term which does not mention unknowns and which mentions only atoms of level 1 a **value**. We briefly indicate how to translate the untyped λ-calculus to values: a translates to a_1 (assume some arbitrary injection of untyped λ-calculus variable symbols to atoms of level 1), tu translates to $t'u'$ if t and u translate to t' and u' respectively, and $\lambda a.t$ translates to $И a_i.([a_i]t')$ if t translates to t'. This translation *is* correct in a natural sense and preserves strong normalisation.

8 α-equivalence

We can use substitution to recover α-equivalence:

$$(\alpha) \qquad b_i\#X \vdash ([a_i]X) \longrightarrow [b_i](X[a_i{\mapsto}b_i])$$
$$(\alpha') \qquad b_i\#Z \vdash Z[a_i{\mapsto}b_i][b_i{\mapsto}Y] \longrightarrow Z[a_i{\mapsto}Y]$$

Lemma 8.1 *In the presence of (α) and (α') and the rules for* sub *except for* **(subaa)**, *the rewrite $Z[a_i{\mapsto}a_i] \longrightarrow Z$ is valid.*

Proof. We use (α) (recall that $Z[a_i{\mapsto}a_i] = \mathsf{sub}([a_i]Z, a_i)$), (α'), and **(sub#)**. □

The rewrite $[a_i]a_i \overset{(\alpha),(\mathbf{suba})}{\longrightarrow}{}^* [b_i]b_i$ is valid.

This is all very well for terms not mentioning unknowns or atoms that are too strong, but if we have $[a_i]X$ or $[a_i]b_j$ for $j > i$, and want to rename a_i to some b_i such that $b_i\#X$ or $b_i\#b_j$. Where do we find this guaranteed-fresh b_i?

Say Δ has **sufficient freshnesses** when for every finite set S of atoms and/or unknowns, and for every level i, there is an atom $a_i \notin S$ such that $a_i\#b_j \in \Delta$ for every $b_j \in S$, and $a_i\#X \in \Delta$ for every X in Δ. It is not hard to prove the existence of contexts with sufficient freshnesses by an inductive construction. If Δ has sufficient freshnesses then it is infinite.

This achieves the effect of dynamic creation of names, since any syntax we rewrite is finite and we obtain the desired effect of 'always having a fresh atom', which we can always rename within its scope using α-equivalence.

In short, if the freshness context is sufficiently rich then (some fragment) of α-equivalence becomes accessible. The downside is of course that extra rules may mean extra critical pairs if we want to use Theorem 4.3.

So assuming sufficient freshnesses, how do these axioms behave?

Write $(a_i\ b_i)_{c_i} \cdot t$ for the term $И c_i.t[a_i{\mapsto}c_i][b_i{\mapsto}a_i][c_i{\mapsto}b_i]$.

Lemma 8.2 *Suppose we are rewriting in a context with sufficient freshnesses and*

suppose $c_i\#X$ and $d_i\#X$. Then the following rewrites are valid:

$$(a_i\ b_i)_{c_i} \cdot X \longrightarrow^* (a_i\ b_i)_{d_i} \cdot X \qquad (a_i\ b_i)_{c_i} \cdot X \longrightarrow^* (b_i\ a_i)_{c_i} \cdot X$$
$$(a_i\ b_i)_{c_i} \cdot (a_i\ b_i)_{c_i} \cdot X \longrightarrow^* X.$$

Proof. We just sketch the first reduction, the others are no harder:

$$(a_i\ b_i)_{c_i} \cdot X \xrightarrow{(\alpha)} \mathsf{N}d_i.(X[a_i{\mapsto}c_i][b_i{\mapsto}a_i][c_i{\mapsto}a_i][c_i{\mapsto}d_i])$$
$$\xrightarrow{(\alpha)} \mathsf{N}d_i.(X[a_i{\mapsto}c_i][b_i{\mapsto}a_i][c_i{\mapsto}d_i][d_i{\mapsto}a_i][c_i{\mapsto}d_i])$$
$$\longrightarrow^* \mathsf{N}d_i.(X[a_i{\mapsto}d_i][b_i{\mapsto}a_i][d_i{\mapsto}a_i])$$

□

In view of the lemma above we may write just $(a_i\ b_i) \cdot t$ for $(a_i\ b_i)_{c_i} \cdot t$. It is now not hard to prove that:

$$(a_i\ b_i) \cdot (c_i\ d_i) \cdot X \longrightarrow^* (c_i\ d_i) \cdot (a_i\ b_i) \cdot X$$
$$(a_i\ b_i) \cdot (a_i\ d_i) \cdot X \longrightarrow^* (a_i\ d_i) \cdot (d_i\ b_i) \cdot X \qquad (a_i\ a_i) \cdot X \longrightarrow^* X.$$

This suffices to verify that we have implemented a **permutation action** in terms of atom-for-atom substitution. Furthermore:

$$(a_i\ b_i) \cdot \mathsf{f}(t_1, \ldots, t_n) \longrightarrow^* \mathsf{f}((a_i\ b_i) \cdot t_1, \ldots, (a_i\ b_i) \cdot t_n)$$
$$(a_i\ b_i) \cdot a_i \longrightarrow^* b_i \qquad a_i\#X, b_i\#X \vdash (a_i\ b_i) \cdot X \longrightarrow^* X$$
$$(a_i\ b_i) \cdot [a_i]X \longrightarrow^* [b_i](a_i\ b_i) \cdot X \qquad (a_i\ b_i) \cdot [b_i]X \longrightarrow^* [a_i](a_i\ b_i) \cdot X$$
$$(a_i\ b_i) \cdot [c_i]X \longrightarrow^* [c_i](a_i\ b_i) \cdot X.$$

These are characteristic properties of the permutation action on nominal terms. More research on this is needed; in particular a detailed examination of how these axioms make atoms of different levels interact, might give useful information about an appropriate built-in permutation action on hierarchical nominal terms.

9 Concluding comments and future work

Many systems formalise aspects of meta-level logic and programming. Examples are first- and higher-order logic [2,20], rewriting [18,12], logical frameworks [1,10,15], and many more including of course nominal-based systems [19,5,4,17,8]. This work differs from others in several ways:

We investigated a *hierarchy* of levels modelling increasingly 'meta' treatments of an object-level theory (in the framework of rewriting). Our hierarchy of atoms can accurately capture our intuitions about how meta-level variables should be instantiated. For example the rewrite $a_2[a_1{\mapsto}2][a_2{\mapsto}a_1] \longrightarrow^* 2$ discussed in Section 5 is supposed to model what we mean when we say 'let t be a in t with a replaced by 2'.

As with all nominal-based systems we implement abstraction and freshness as an explicit and *logical* property of terms (e.g. recall the logical derivation rules for freshness from Section 2). But there are twists; the hierarchy demanded changes in derivation rules for freshness assertions, notably (#**abs**<). We also *omit* α-equivalence as primitive, which simplified the framework at the cost of having a weaker theory of equality of terms.

Recall that we have given rewrites for substitution, π-calculus style restriction [13], α-equivalence, and a computationally powerful λ-calculus inspired from previous work [6] for which we exploited the meta-level results about hierarchical nominal term rewrite systems presented here to indicate a particularly concise method of proof for confluence. We have sketched some indication of how the infinite hierarchy can give 'meta-levels within the rewrite system'.

We have heard the following reasoning: *If the intuition of* $u[a_1 \mapsto t]$ *is to replace any* a_1*-shaped holes in* u *with* t *in a capture-avoiding manner, then surely it should be forbidden that* t *mention any atom stronger than* 1*, since then we are trying to replace a 'big hole' with a 'smaller hole'?* This is not so. Informally we may write 'replace x by t in $\lambda y.x$ where y is not free in t; we obtain $\lambda y.t$'. Here t is a meta-variable. We may later say 'now actually t is 2; we obtain $\lambda y.2$'. This corresponds to a series of rewrites on $b_1\#X_2 \vdash (\lambda[b_1]a_1)[a_1 \mapsto X_2][X_2 \mapsto 2]$. Now $\vdash (\lambda[b_1]a_1)[a_1 \mapsto X_2][X_2 \mapsto 2]$ is a legal term. However the side-conditions on (**subabs**$\leq$) prevent us from distributing $[a_1 \mapsto X_2]$ under the abstraction because we cannot prove $b_1\#X_2$. If we wish we may add fresh atoms to the context and use α-equivalence to rename b_1.

We have also heard the following reasoning: *Hierarchical nominal terms are supposed to internalise meta-level variables, so why do they include explicit unknowns* X*,* Y*,* Z *as well as atoms* a_i*?* This is a matter of taste and presentation only. For our notion of rewriting to work, we do need access to atoms stronger than any other atoms in a rule to play the rôle of meta-variables. The easiest way to guarantee this is to add atoms of level ω, that is, our unknowns. We could alternatively leave atoms only with finite levels, let nominal rewriting automatically identify which are the strongest atoms in a rewrite rule, and phrase our rewrite rules accordingly. For example (**sub**#) becomes $a_i\#b_{i+1} \vdash b_{i+1}[a_i \mapsto b_{i+1}] \rightarrow b_{i+1}$. To us it seemed a cleaner presentation to add unknowns.

But note: the meta-level will always be with us no matter how strong or how weak our object-level. The issue is how much of our meta-level intuitions we internalise in a formal system, and how we do that. Hierarchical nominal rewriting *does not* and *cannot* 'make the meta-level go away'. We can only model it and use that model to learn.

An existing formalisation of the meta-level with an infinite hierarchy is the Russelian type hierarchy [16]. Since its inception a century ago this has sired higher-order logic [20], the polymorphic λ-calculus [9], dependent type systems [14], and higher-order rewriting [12] to name a few. We can see these systems as having a 'hierarchy of meta-levels' in the sense that objects of functional type 'talk about' the types they are functions on. Yet this forces a *particular* notion of meta-level because substitution is capture-avoiding, a syntactic identity and freshness is not directly

expressed. Some work discusses a λ-calculus with explicit substitution [3,11] but this is designed to calculate computational cost; the substitution is is still capture-avoiding, meta-variables and freshness are not explicit, and the underlying semantics remains unchanged. Our hierarchy of atoms with freshness contexts gives a us a different slant; it remains to relate nominal techniques to higher-order techniques in general, and in the specific case that we have a hierarchy of atoms.

Future work includes a more profound analysis of the NEW calculus of contexts, a λ-calculus based on the same ideas [6], restoring permutations as built-in, and further analyses of substitution and α-equivalence. We see logics based on hierarchical nominal terms which can internalise aspects of the meta-level using the hierarchy to avoid inconsistencies. We also see logics exploring the properties of freshness, of which we have caught interesting glimpses in our derivation system for #. Semantics are a very important tool and should be thoroughly investigated. We anticipate many interesting insights into the mathematical content of naming unknowns and the meta-level.

References

[1] A. Avron, F. A. Honsell, I. A. Mason, R. Pollack. Using typed lambda calculus to implement formal systems on a machine. *Journal of Automated Reasoning*, 9(3), 1992.

[2] J. Barwise. An introduction to first-order logic. In J. Barwise, ed., *Handbook of Mathematical Logic*. North Holland, 1977.

[3] R. Bloo. *Preservation of Termination for Explicit Substitution*. PhD thesis, Eindhoven University of Technology, Eindhoven, 1997.

[4] J. Cheney, C. Urban. Alpha-prolog: A logic programming language with names, binding and alpha-equivalence. In B. Demoen, V. Lifschitz, eds., *Proceedings of the 20th International Conference on Logic Programming (ICLP 2004)*, number 3132 in LNCS. Springer-Verlag, 2004.

[5] M. Fernández, M. J. Gabbay. Nominal rewriting. *Information and Computation*, 2005. Accepted for publication.

[6] M. J. Gabbay. A new calculus of contexts. In *Proc. 7th Int. ACM SIGPLAN Conf. on Principles and Practice of Declarative Programming (PPDP'2005)*. ACM, 2005.

[7] M. J. Gabbay, A. Mathijssen. Capture-avoiding substitution as a nominal algebra. In *ICTAC'2006*, 2006.

[8] M. J. Gabbay, A. Mathijssen. Nominal algebra. Submitted STACS'07, 2006.

[9] J. R. Hindley. *Basic Simple Type Theory*. Number 42 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1997.

[10] Huet, Kahn, Paulin-Mohring. The COQ tutorial. http://pauillac.inria.fr/coq/doc/tutorial.html. LogiCal Project.

[11] P. Lescanne. From lambda-sigma to lambda-upsilon a journey through calculi of explicit substitutions. In *POPL '94: Proc. 21st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. ACM Press, 1994.

[12] R. Mayr, T. Nipkow. Higher-order rewrite systems and their confluence. *Theoretical Computer Science*, 192, 1998.

[13] R. Milner, J. Parrow, D. Walker. A calculus of mobile processes, II. *Information and Computation*, 100(1), 1992.

[14] B. Nordstrom, K. Petersson, J. M. Smith. *Programming in Martin-Lof's Type Theory*, vol. 7 of *International Series of Monographs on Computer Science*. Clarendon Press, Oxford, 1990. Also online at `http://www.cs.chalmers.se/Cs/Research/Logic/book/`.

[15] L. Paulson. *The Isabelle reference manual*. Cambridge University Computer Laboratory, 2001.

[16] B. Russell, A. Whitehead. *Principia Mathematica*. Cambidge University Press, 1910, 1912, 1913. 3 vols.

[17] M. Shinwell, A. Pitts. Fresh objective caml user manual. Technical Report UCAM-CL-TR-621, University of Cambridge, 2005.

[18] Terese. *Term Rewriting Systems*. Number 55 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2003.

[19] C. Urban, A. M. Pitts, M. J. Gabbay. Nominal unification. *Theoretical Computer Science*, 323(1–3), 2004.

[20] J. van Benthem. Higher-order logic. In D. Gabbay, F. Guenthner, eds., *Handbook of Philosophical Logic, 2nd Edition*, vol. 1. Kluwer, 2001.