Principal Types for Nominal Theories

Elliot Fairweather, Maribel Fernández and Murdoch J. Gabbay

Abstract. We define rank 1 polymorphic types for nominal rewrite rules and equations. Typing environments type atoms, variables, and function symbols, and since we follow a Curry-style approach there is no need to fully annotate terms with types. Our system has principal types, and we give rule and axiom formats to guarantee preservation of types under both rewriting and equality reasoning. This is non-trivial because substitution does not avoid capture, so a substituted symbol can—if we are not careful—appear in inconsistent typing contexts.
Keywords: binding, polymorphism, type inference, rewriting.

1 Introduction

Suppose we want to specify λ-calculus $\beta\eta$-equivalence. We might well write this:

$$\begin{array}{ll} (\lambda \overset{\substack{\textit{object-variable } x\\ \downarrow}}{x}.\,(\lambda y.s))t =_\beta \lambda y.((\lambda x.s)t) & \text{provided } y \overset{\substack{\textit{freshness side-condition}\\ \downarrow}}{\text{not}} \text{ free in } t \\ \underset{\substack{\uparrow\\ \textit{binding term-former } \lambda}}{\lambda} x.\,(rx) =_\eta \underset{\substack{\uparrow\\ \textit{meta-variable } r}}{r} & \text{provided } x \text{ not free in } r \end{array}$$

Equalities like this are typical, and appear in specifications of the λ-calculus (as above), substitution, logic with quantifiers, π-calculus, and more. They have common features as annotated above. To formalise and reason with this kind of systems, in this paper we follow the nominal approach:

– Nominal terms directly represent syntax with binding using term-formers, two levels of variable, and freshness side-conditions [20]. The terms and side-conditions above can be represented symbol-for-symbol in a formal syntax.
– Equality on nominal terms can be formalised in *nominal rewriting* [10] (for oriented equality) and *nominal algebra* [14] (for unoriented equality).
– Nominal terms can be given Hindley-Milner style types [9]. Given a type environment, principal types for nominal terms can be automatically deduced.

This paper combines types and equality reasoning—so we type terms *and* equations between them. This is not trivial because equality gives terms a dynamic behaviour, and type systems must ensure types are robust with respect to it.

For a large class of *closed* theories, corresponding to what can be defined in higher-order rewriting and equality reasoning, nominal rewriting is sound and complete for nominal algebra [11]. The technical contributions of this paper are notions of *typeable closed* rewrite rule and equality axiom such that types are preserved under rewriting and equality reasoning respectively. To ensure this, closedness and principal types play a key role, as our examples will demonstrate.

Related work. Nominal terms support a capturing substitution and the notation, though formal, is close to standard informal practice. For example β-reduction may be represented as $\mathsf{app}(\mathsf{lam}[a]X, Y) \to \mathsf{sub}([a]X, Y)$ where $\mathsf{sub}([a]X, Y)$ is a term which may be given the behaviour of 'non-capturing substitution' by rewrite rules [12,10].

Now consider *static* semantics, i.e. types like $\mathbb{N}$ for **numbers** and $\tau \to \tau$ for **functions**. Assigning types to terms partitions the language into 'numbers', or 'functions between numbers', and so on. This paper will not make the case for types but the interested reader can find compelling practical and theoretical motivations elsewhere, e.g. Java [17], ML [7], and System F [16].

Two approaches have been used to give static semantics for nominal terms: In e.g. [20,19,18] atoms have a special type of atoms $\mathbb{A}$. But, when we write $\mathsf{lam}[a]X$ or $\mathsf{lam}[a]a$ to represent $\lambda x.$- (a *term with a hole*, or *context*) or $\lambda x.x$, we might not expect a to be forbidden from having any type other than $\mathbb{A}$. We can use explicit casting function symbols to inject $\mathbb{A}$ into other types; but the a in $\mathsf{lam}[a]X$ still has type $\mathbb{A}$, so we cannot infer more about a until X is instantiated. This notion of typing is useful for terms without variables or in systems without polymorphism. So an alternative approach was proposed in [9] which allows atoms to inhabit any type. It follows a Curry-style approach, and has rank 1 polymorphism (*ML-style polymorphism* or *Hindley-Milner types* [7]); that is, types are assigned to terms without requiring type annotations for atoms or variables. Thus, we can write $\mathsf{lam}[a]X$, or $\mathsf{fix}[f]X$, or $\mathsf{forall}[a]X$, or $\mathsf{app}(\mathsf{lam}[a]X, \mathsf{lam}[b]Y)$, and so on, and expect the type system to make sense of this. The syntax-directed typing rules for nominal terms given in [9] guarantee that every typeable term has a *principal type* (one which subsumes all others in a suitable sense) in a given environment, and types are compatible with α-equivalence on nominal terms. Type inference is decidable and an algorithm to compute principal types has been implemented [8].

In [9], a notion of typed nominal rewriting rule is given, with a condition to ensure type preservation under rewriting. In this paper we consider also nominal equational axioms, and give sufficient conditions for type preservation under rewriting and equational deduction that are intuitive and easy to implement.

Overview of the paper. In Section 2 we recall nominal rewriting and equational reasoning, as well as the Curry-style type system for nominal terms. Section 3 gives examples to motivate the design of the type system for rules and equations. Section 4 contains the main results of the paper: it defines closed rules and the closed rewrite relation (which is sound and complete for nominal equational reasoning when the axioms are closed), and presents a notion of typeable rule and typeable axiom that guarantee that rewriting and equational reasoning preserve types. Finally, in Section 5 we draw conclusions and discuss future work.

2 Background

In this section we recall the main notions underlying the nominal approach to the specification of systems with binders. We focus on equational specifications.

Fix denumerable sets of **atoms**, **variables**, and **term-formers**. a, b, c will range over distinct atoms, $X, Y, Z, \ldots$ over distinct variables, and $\mathsf{f}, \mathsf{g}, \ldots$ over distinct term-formers. A **permutation** π is a bijection on atoms such that $\mathit{nontriv}(\pi) = \{a \mid \pi(a) \neq a\}$ is finite. **Nominal terms** r, s, t are defined by:

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

$\pi{\cdot}X$ is a **(moderated) variable**; $[a]t$ is an **atom-abstraction**. We write $V(t)$ for the variables in t and $A(t)$ for the atoms in t (so $A([a]a) = \{a\}$ and $A(\pi{\cdot}X) = \mathit{nontriv}(\pi)$). We write $(a\ b)$ for the **swapping** permutation mapping a to b, b to a, and all other c to themselves; id for the identity permutation (so $\mathit{id}(a) = a$); $\pi \circ \pi'$ for composition (so $(\pi \circ \pi')(a) = \pi(\pi'(a))$ and π^{-1} for inverse).

A **substitution** σ is a map from variables to terms. Write $[X \mapsto r]$ for the substitution mapping X to r and all other Y to $\mathit{id}{\cdot}Y$.

Permutation and **substitutions** act on terms:

$$\begin{array}{rclrclrcl}
\pi{\cdot}a &=& \pi(a) & \pi{\cdot}(\pi'{\cdot}X) &=& (\pi \circ \pi'){\cdot}X & \pi{\cdot}(\mathsf{f}\,r) &=& \mathsf{f}\,\pi{\cdot}r \\
\pi{\cdot}(r_1, \ldots, r_n) &=& (\pi{\cdot}r_1, \ldots, \pi{\cdot}r_n) & \pi{\cdot}[a]r &=& [\pi(a)]\pi{\cdot}r & & &
\end{array}$$

$$\begin{array}{rclrclrcl}
a\sigma &=& a & (\pi{\cdot}X)\sigma &=& \pi{\cdot}(\sigma(X)) & (\mathsf{f}\,r)\sigma &=& \mathsf{f}(r\sigma) \\
(r_1, \ldots, r_n)\sigma &=& (r_1\sigma, \ldots, r_n\sigma) & ([a]r)\sigma &=& [a](r\sigma) & & &
\end{array}$$

Definition 1. α**-equality** and **freshness** are defined by the following rules, where $ds(\pi, \pi') = \{a \mid \pi(a) \neq \pi'(a)\}$ is the **difference set** of π, π':

$$\frac{}{a\#b} \quad \frac{a\#s}{a\#\mathsf{f}s} \quad \frac{a\#s_i \quad (1 \leq i \leq n)}{a\#(s_1, \ldots, s_n)} \quad \frac{}{a\#[a]s} \quad \frac{a\#s}{a\#[b]s} \quad \frac{\pi^{-1}(a)\#X}{a\#\pi{\cdot}X} \quad \frac{s \approx_\alpha t}{\mathsf{f}\,s \approx_\alpha \mathsf{f}\,t}$$

$$\frac{}{a \approx_\alpha a} \quad \frac{s \approx_\alpha t}{[a]s \approx_\alpha [a]t} \quad \frac{s \approx_\alpha (a\ b){\cdot}t \quad a\#t}{[a]s \approx_\alpha [b]t} \quad \frac{ds(\pi, \pi')\#X}{\pi{\cdot}X \approx_\alpha \pi'{\cdot}X} \quad \frac{s_i \approx_\alpha t_i \quad (1 \leq i \leq n)}{(s_1, \ldots, s_n) \approx_\alpha (t_1, \ldots, t_n)}$$

The intuition of $a\#t$ is 'a is not free in t', the intuition of $s \approx_\alpha t$ is 's and t are α-equivalent'. Both are conditional on freshness assumptions on the variables X. So **freshness contexts** Δ, ∇, ..., are finite sets of **freshness constraints** $a\#X$, and we write $\Delta \vdash a\#r$ when $a\#r$ follows from Δ; similarly for $\Delta \vdash r \approx_\alpha s$.

For example, $[a]a \approx_\alpha [b]b$ and $ds((a\ b), \mathit{id}) = \{a, b\}$ so $a\#X, b\#X \vdash (a\ b){\cdot}X \approx_\alpha X$; see [20,10] for more examples.

We now recall the Curry-style type system from [9], which can typecheck nominal terms even if they contain variables (representing unknown subterms); see [8] for an implementation in Haskell.

Fix denumerable sets of **base data types** (e.g. δ, $\mathbb{N}$), **type variables** (e.g. α), and **type-formers** (e.g. C, List). **Types** τ, **type-schemes** σ, and **type-declarations** (**arities**) ρ are generated by:

$$\tau ::= \delta \mid \alpha \mid \tau_1 \times \ldots \times \tau_n \mid C\ \tau \mid [\tau']\tau \qquad \sigma ::= \forall \overline{\alpha}.\tau \qquad \rho ::= (\tau')\tau$$

$\overline{\alpha}$ denotes any finite set of type variables (if empty we omit $\forall$); call them **bound** in σ and call **free** type variables mentioned in σ not in $\overline{\alpha}$. Write $TV(\tau)$ for the set of type variables in τ, and $\equiv$ for **equality** modulo α-equivalence.[1]

[1] Types could be expressed in nominal syntax; our focus here is the term language.

We associate a type declaration to each term-former. E.g. we can have $\mathsf{succ} : (\mathbb{N})\mathbb{N}$ and $0 : ()\mathbb{N}$ (we may write just $0 : \mathbb{N}$ in this case).

Type substitutions S, T, U are finite partial functions from type variables to types. Write $[\alpha{\mapsto}\tau']$ for the partial function just mapping α to τ', and id for the **identity** substitution. Substitutions act on types, type schemes and arities as usual; application is written τS and composition SS' (apply S then S'). Write $\sigma \succcurlyeq \tau$ when $\sigma \equiv \forall\overline{\alpha}.\tau'$ and $\tau' S \equiv \tau$ for S with domain in $\overline{\alpha}$. τ may contain other type variables; only *bound* type variables in σ may be instantiated; so $\forall\alpha.(\alpha{\times}\beta) \succcurlyeq (\beta{\times}\beta)$ but $(\alpha{\times}\beta) \not\succcurlyeq (\beta{\times}\beta)$. Write $\rho \succcurlyeq \rho'$ when $\rho S \equiv \rho'$ for some S. All variables in arities are bound, but since they are *all* bound we omit $\forall$.

Definition 2. A **typing context** Γ is a partial function from atoms and variables to type schemes, with finite domain. We write $\Gamma, a\colon \sigma$ for Γ **updated with** $a\colon \sigma$, that is, $(\Gamma \setminus \{a\colon \sigma'\}) \cup \{a\colon \sigma\}$. Similarly we write $\Gamma, X : \sigma$. We write ΓS for the typing context obtained by applying S to the types in Γ. Similarly, $\pi{\cdot}\Gamma$ denotes the context obtained by applying π to the atoms in Γ. $TV(\Gamma)$ denotes the set of type variables occurring free in Γ.

A **typing judgement** is a tuple $\Gamma; \Delta \vdash s : \tau$ where Γ is a typing context, Δ a freshness context, s a term and τ a type (when Δ is empty we omit the separating ';'). Define **derivable typing judgements** by:

$$\frac{\sigma \succcurlyeq \tau}{\Gamma, a : \sigma; \Delta \vdash a : \tau} \qquad \frac{\sigma \succcurlyeq \tau \quad \Gamma; \Delta \vdash \pi{\cdot}X : \diamond}{\Gamma, X : \sigma; \Delta \vdash \pi{\cdot}X : \tau} \qquad \frac{\Gamma, a : \tau; \Delta \vdash t : \tau'}{\Gamma; \Delta \vdash [a]t : [\tau]\tau'}$$

$$\frac{\Gamma; \Delta \vdash t_i : \tau_i \quad (1 \leq i \leq n)}{\Gamma; \Delta \vdash (t_1, \ldots, t_n) : \tau_1 \times \ldots \times \tau_n} \qquad \frac{\Gamma; \Delta \vdash t : \tau' \quad \mathsf{f} : \rho \succcurlyeq (\tau')\tau}{\Gamma; \Delta \vdash \mathsf{f}\, t : \tau}$$

$\Gamma; \Delta \vdash \pi{\cdot}X : \diamond$ holds if, for any a such that $\pi{\cdot}a \neq a$, $\Delta \vdash a\#X$ or $a\colon \sigma,\ \pi{\cdot}a\colon \sigma \in \Gamma$ for some σ. The condition $\mathsf{f} : \rho \succcurlyeq (\tau')\tau$ is shorthand for $\mathsf{f} : \rho$ and $\rho \succcurlyeq (\tau')\tau$. If $\Gamma; \Delta \vdash s{:}\tau$ for some τ then call $\Gamma; \Delta \vdash s$ **typeable**; otherwise call it **untypeable**.

$\pi{\cdot}X$ represents an unknown term in which π permutes atoms. The second rule ensures compatibility between this and α-equivalence (see Theorem 4 below) since substitution does not avoid capture: e.g. $([a]X)[X{\mapsto}s] = [a]s$.

We refer the reader to [9] for examples and give just the typing of one term here: $([a](a, b), [a][b](a, b), a, b) : [\alpha_1](\alpha_1 \times \beta) \times [\alpha_2][\beta_2](\alpha_2 \times \beta_2) \times \alpha \times \beta)$.

Lemma 3. *If $\Gamma; \Delta \vdash t : \tau$ then $\Gamma[\alpha{\mapsto}\tau']; \Delta \vdash t : \tau[\alpha{\mapsto}\tau']$.*
If $\Gamma;\Delta \vdash t{:}\tau$ and $a{,}b{:}\sigma \in \Gamma$ for some σ, then $\Gamma;\Delta \vdash (a\ b){\cdot}t{:}\tau$.

Theorem 4. *$\Delta \vdash s \approx_\alpha t$ and $\Gamma; \Delta \vdash s : \tau$ imply $\Gamma; \Delta \vdash t : \tau$.*

Definition 5. A **typing problem** is a triple $\Gamma; \Delta \vdash s$, and its **solution** is a pair (S, τ) of a type-substitution S and a type τ such that $\Gamma S; \Delta \vdash s{:}\tau$. Write $S|_\Gamma$ for the restriction of S to $TV(\Gamma)$. Solutions are naturally ordered by instantiation of substitutions; call a minimal element in a set of solutions **principal**.

Principal solutions for $\Gamma; \Delta \vdash s$ are unique modulo renaming type-variables. A function pt to **compute principal types**, generalising Hindley-Milner [7] to nominal terms, is given in [9, Definition 4]: $pt(\Gamma; \Delta \vdash s)$ is a principal solution (S, τ) such that $\Gamma S; \Delta \vdash s\colon \tau$, and is undefined if s is untypeable in $\Gamma; \Delta$.

Lemma 6. *1. **Type weakening**: If $\Gamma;\Delta \vdash s:\tau$ then $\Gamma,\Gamma';\Delta \vdash s:\tau$, provided that Γ' and Γ are disjoint.*

*2. **Type strengthening for atoms**: If $\Gamma, a:\tau';\Delta \vdash s:\tau$ then $\Gamma;\Delta \vdash s:\tau$ provided that $\Delta \vdash a\#s$.*

*3. **Type strengthening for variables**: If $\Gamma, X:\tau';\Delta \vdash s:\tau$ then $\Gamma;\Delta \vdash s:\tau$ provided X does not occur in s.*

We now introduce nominal algebra [14] and nominal rewriting [10]. See [11] for a detailed treatment of their connections.

An **equality judgement** is a tuple $\Delta \vdash s = t$ of a freshness context and two terms. Similarly a **rewrite judgement** is a tuple $\Delta \vdash s \to t$. An **equational theory** $\mathsf{T} = (\Sigma, Ax)$ is a pair of a signature Σ and a possibly infinite set of equality judgements Ax (**axioms**) in Σ. A **rewrite theory** $\mathsf{R} = (\Sigma, Rw)$ is a pair of Σ and a possibly infinite set of rewrite judgements Rw (**rewrite rules**) in Σ. Where Σ is clear, we identify T with Ax and R with Rw.

Definition 7. Write $\Delta \vdash (\phi_1, \dots, \phi_n)$ for $\Delta \vdash \phi_1, \dots, \Delta \vdash \phi_n$. A **position** C is a pair (s, X) of a term and a distinguished unknown X that occurs precisely once in s, as $id{\cdot}X$. If $C = (s, X)$ then we write $C[t]$ for $s[X{\mapsto}t]$.

Nominal rewriting: The *one-step rewrite relation* $\Delta \vdash s \xrightarrow{R} t$ is the least relation such that for every $(\nabla \vdash l \to r) \in \mathsf{R}$, freshness context Δ, position C, term s', permutation π, and substitution θ,

$$\frac{s \equiv C[s'] \qquad \Delta \vdash \big(\nabla\theta, \quad s' \approx_\alpha \pi{\cdot}(l\theta), \quad C[\pi{\cdot}(r\theta)] \approx_\alpha t\big)}{\Delta \vdash s \xrightarrow{R} t}\,(\mathbf{Rew}_{\nabla\vdash\mathbf{l}\to\mathbf{r}}). \qquad (1)$$

The *rewrite relation* $\Delta \vdash_{\mathsf{R}} s \to t$ is the reflexive transitive closure of the one-step rewrite relation. So, in particular, $\Delta \vdash s \approx_\alpha s'$ implies $\Delta \vdash_{\mathsf{R}} s \to s'$.

Nominal algebra: $\Delta \vdash_{\mathsf{T}} s = t$ is the least transitive reflexive symmetric relation such that for every $(\nabla \vdash l = r) \in \mathsf{T}$, freshness context Δ, position C, permutation π, substitution θ, and fresh Γ (so if $a\#X \in \Gamma$ then $a \notin A(\Delta, s, t)$),

$$\frac{\Delta, \Gamma \vdash \big(\nabla\theta, \quad s \approx_\alpha C[\pi{\cdot}(l\theta)], \quad C[\pi{\cdot}(r\theta)] \approx_\alpha t\big)}{\Delta \vdash_{\mathsf{T}} s = t}\,(\mathbf{Axi}_{\nabla\vdash\mathbf{l}=\mathbf{r}}). \qquad (2)$$

3 Examples

Consider a type Λ and term-formers $\mathsf{lam} : ([\Lambda]\Lambda)\Lambda$, $\mathsf{app} : (\Lambda \times \Lambda)\Lambda$ and $\mathsf{sub} : ([\Lambda]\Lambda \times \Lambda)\Lambda$, sugared to $\lambda[a]s$, $s\ t$, and $s[a{\mapsto}t]$. Then $X, Y{:}\Lambda \vdash (\lambda[a]X)Y : \Lambda$ by Definition 2. The function pt can automatically infer this type and a most general type for a, namely Λ, since $\mathsf{lam} : ([\Lambda]\Lambda)\Lambda$. Anticipating Section 4, here are rewrite rules for β-reduction (note typing *and* freshness conditions in rules); types of abstracted atoms can be inferred and are Λ:

$$X, Y{:}\Lambda \vdash (\lambda[a]X)Y \to X[a{\mapsto}Y]{:}\Lambda \qquad X, Y{:}\Lambda; a\#X \vdash X[a{\mapsto}Y] \to X{:}\Lambda$$
$$Y{:}\Lambda \vdash a[a{\mapsto}Y] \to Y{:}\Lambda \qquad X, Y{:}\Lambda; b\#Y \vdash (\lambda[b]X)[a{\mapsto}Y] \to \lambda[b](X[a{\mapsto}Y]){:}\Lambda$$
$$X, Y, Z{:}\Lambda \vdash (XY)[a{\mapsto}Z] \to (X[a{\mapsto}Z])\,Y[a{\mapsto}Z]{:}\Lambda$$

And here is an equality axiom for η: $X{:}\Lambda; a\#X \vdash \lambda[a](Xa) = X{:}\Lambda$.

Rewriting and axioms are used to compute and reason with terms—not only ground terms, but also terms with variables. Types should be compatible with the resulting dynamics on terms. Since a variable might occur several times in a term, under different abstractions, the interaction between types and rewriting or equational reasoning is subtle. *Closedness* [10] ensures the interaction between abstractions and variables is safe. Later we leverage this to conditions on closed rewrite rules and axioms to ensure also type preservation. E.g. the rules for β and η (as a rewrite or an equality) preserve types. Thus, the type Λ of $b : \Lambda \vdash \lambda[a](ba)$ is preserved in the reduced term $b : \Lambda \vdash b$.

We get more interesting types for λ-terms with a type-former $\Rightarrow$ of arity 2 and term-formers $\lambda : ([\alpha]\beta)(\alpha{\Rightarrow}\beta)$, $\circ : ((\alpha{\Rightarrow}\beta){\times}\alpha)\beta$, and $\sigma : (([\alpha]\beta){\times}\alpha)\beta$. As before, write $\sigma([a]s, t)$ as $s[a{\mapsto}t]$. Then the following rules preserve types:

$$X{:}\alpha, Y{:}\beta;\ a\#X \vdash X[a{\mapsto}Y] \to X : \alpha \qquad Y{:}\gamma \vdash a[a{\mapsto}Y] \to Y : \gamma$$
$$X{:}\alpha \Rightarrow \beta,\ Y{:}\alpha,\ Z{:}\gamma \vdash (XY)[a{\mapsto}Z] \to (X[a{\mapsto}Z])(Y[a{\mapsto}Z]) : \beta$$
$$X{:}\beta,\ Y{:}\gamma;\ b\#Y \vdash (\lambda[b]X)[a{\mapsto}Y] \to \lambda[b](X[a{\mapsto}Y]) : \alpha{\Rightarrow}\beta$$
$$X{:}\beta, Y{:}\alpha \vdash (\lambda[a]X)Y \to \sigma([a]X, Y) : \beta$$

Assume types $\mathbb{B}$ and $\mathbb{N}$. Then $B{:}\mathbb{B}, N{:}\mathbb{N} \vdash ((\lambda[a]a)B, (\lambda[a]a)N) : \mathbb{B} \times \mathbb{N}$ and $B{:}\mathbb{B},\ N{:}\mathbb{N} \vdash ((\lambda[a]a)B, (\lambda[a]a)N) \to (B, N) : \mathbb{B} \times \mathbb{N}$.

$\lambda[a]a$ takes different types just like $\lambda x.x$ in the Hindley-Milner type system; $pt(\vdash \lambda[a]a) = (id, \alpha \Rightarrow \alpha)$. Our system will not type $B{:}\mathbb{B}, N{:}\mathbb{N} \vdash BN$ or $\lambda[a]aa$—the previous system with a unique type Λ types the second term.

Below we show that η is compatible with this notion of type: $X{:}\alpha{\Rightarrow}\beta; a\#X \vdash \lambda[a](Xa) = X{:}\ \alpha \Rightarrow \beta$. A term obtained from a typeable term by typed equational reasoning is also typeable.

4 Types for (closed) nominal rewriting and algebra

We recall closed rewrite rules [10] and closed equational axioms [11], then give conditions to ensure nominal rewriting and equational reasoning preserve types.

4.1 Closed rules and closed rewriting

Intuitively, a closed term has no unabstracted atoms and all occurrences of X (representing an unknown subterm) appear under *the same* abstracted atoms. So $\mathsf{f}([a]X, X)$ is not closed since a is abstracted in the first X but not the second, and $a\#X \vdash \mathsf{f}([a]X, X)$ is closed, since the freshness constraint ensures that a cannot occur in X. Closedness was introduced in [12]. Definition 8 tests to check for closedness using *freshened variants* [10] (inductive definitions are also possible [6,13]).

A **freshened variant** $t^{\text{\tiny N}}$ of t is t in which atoms and unknowns have been replaced by 'fresh' atoms and unknowns (not in $A(t)$ and $V(t)$, and perhaps also fresh for some atoms and unknowns from other syntax, which we will always specify). We omit an inductive definition. Similarly, if ∇ is a freshness context then $\nabla^{\text{и}}$ will denote a freshened variant of ∇ (so if $a\#X \in \nabla$ then $a^{\text{и}}\#X^{\text{и}} \in \nabla^{\text{и}}$, where $a^{\text{и}}$ and $X^{\text{и}}$ are fresh for the atoms and unknowns of ∇). We may extend this to other syntax, like equality and rewrite judgements. If $\nabla^{\text{и}} \vdash l^{\text{и}} \to r^{\text{и}}$ is a freshened variant of $\nabla \vdash l \to r$ then $V(\nabla^{\text{и}} \vdash l^{\text{и}} \to r^{\text{и}}) \cap V(\nabla \vdash l \to r) = \varnothing$.

For example: $[a^{\text{и}}][b^{\text{и}}]X^{\text{и}}$ is a freshened variant of $[a][b]X$, $a^{\text{и}}\#X^{\text{и}}$ is a freshened variant of $a\#X$, and $\varnothing \vdash a^{\text{и}} \to b^{\text{и}}$ is a freshened variant of $\varnothing \vdash a \to b$.

Definition 8 (Closedness). Call $\nabla \vdash l$ **closed** when there exists a substitution θ with $dom(\theta) \subseteq V(\nabla^{\text{и}} \vdash l^{\text{и}})$ such that $\nabla, A(\nabla^{\text{и}}, l^{\text{и}})\#V(\nabla, l) \vdash (\nabla^{\text{и}}\theta,\ l \approx_\alpha l^{\text{и}}\theta)$.

Call $R = (\nabla \vdash l \to r)$ and $A = (\nabla \vdash l = r)$ **closed** when $\nabla \vdash (l, r)$ is closed.

Given a rewrite rule $R = (\nabla \vdash l \to r)$ and a term-in-context $\Delta \vdash s$, the (one-step) **closed rewriting** relation $\Delta \vdash s \overset{R}{\to}_c t$ holds if there are C and θ, such that, for a freshened variant $R^{\text{и}}$ of R (fresh for R, Δ, s, t):

$$s \equiv C[s'] \quad \text{and} \quad \Delta, A(R^{\text{и}}) \# V(\Delta, s, t) \vdash (\nabla^{\text{и}}\theta,\ s' \approx_\alpha l^{\text{и}}\theta,\ C[r^{\text{и}}\theta] \approx_\alpha t). \qquad (3)$$

Closed rewriting $\Delta \vdash_{\mathsf{R}} s \to_c t$ is the reflexive transitive closure, as in Def. 7.

The choice of freshened variant in Definition 8 does not matter. Closed rewriting is sound and complete for nominal algebra, if all axioms are closed [11].

4.2 Typed closed rewriting

We define a notion typeable rule such that rewriting preserves types:

Definition 9. A **typeable closed rewrite rule** $R \equiv \Phi; \nabla \vdash l \to r : \tau$ is a tuple of a type context Φ which only types the variables in l and has no type-schemes (in particular, Φ mentions no atoms), a freshness context ∇, and terms l and r such that

1. $V(r, \nabla, \Phi) \subseteq V(l)$,
2. $pt(\Phi; \nabla \vdash l) = (id, \tau)$ and $\Phi; \nabla \vdash r : \tau$,
3. R^T is closed, where R^T is a variant of R in which each abstracted atom is decorated by appending the type of that atom to its name; a renamed atom occurring in a freshness constraint in ∇ is replaced by new freshness constraints, one for each differently decorated variant of the atom.

For example, under the signature $\mathsf{f}\colon ([N]N)N$, $\mathsf{g}\colon ([Bool]N)N$, the variant of the rule $X : N; a\#X \vdash \mathsf{f}([a]X) \to \mathsf{g}([a]X) : N$ used in the third condition in Definition 9 is $X\colon N; aN\#X, aBool\#X \vdash \mathsf{f}([aN]X) \to \mathsf{g}([aBool]X)$. The condition is satisfied: although one occurrence of X is under an abstraction for aN and the other is under an abstraction for $aBool$, we have freshness assumptions for both atoms: $aN\#X, aBool\#X$.

The type decorations for R^T can be determined mechanically by running pt to compute principal types on the left-hand side and right-hand side of R.

The first condition in the definition of typeable closed rule is standard. The second condition says that l is typeable using Φ and ∇, and r is typeable with a type *at least* as general. The third condition ensures that atom abstractions are typed in a consistent manner throughout the rule. To check this, we compute types for the abstracted atoms using the type inference algorithm, and annotate atom abstractions with types. In a Church-style system, we would use the type annotations provided by the user; thanks to the type inference algorithm, in our system users need not provide explicit type annotations for every abstraction.

The typed closed rewriting relation is defined using typed matching.

Definition 10. A **(typed) matching problem** $(\Phi;\nabla\vdash l)\ {}_?{\approx}\ (\Gamma;\Delta\vdash s)$ is a pair of tuples (Φ and Γ are typing contexts, ∇ and Δ are freshness contexts, l and s are terms) such that variables and type-variables on the left are disjoint from those in Γ,Δ,s, and Φ mentions no atoms or type-schemes.

A **solution** to this matching problem, if it exists, is the least pair (S,θ) of a type- and term-substitution (the ordering on substitutions extends to pairs component-wise) such that:

1. $X\theta\equiv X$ for $X\notin V(\Phi,\nabla,l)$, $\alpha S\equiv\alpha$ for $\alpha\notin TV(\Phi)$[2], $\Delta\vdash l\theta{\approx_\alpha}s$ and $\Delta\vdash\nabla\theta$.
2. $pt(\Phi;\nabla\vdash l)=(id,\tau)$, $pt(\Gamma;\Delta\vdash s)=(id,\tau S)$, and for each $X\colon\phi'\in\Phi$, we have $\Gamma,\Gamma';\Delta\vdash X\theta\colon\phi' S$, where $\Gamma'=\Phi' S$ and Φ' contains the typings for the atoms abstracted above an occurrence of X in l and which may occur in instances of X.

That is: we want to make l on the left α-equivalent to s on the right. The first condition defines a matching solution for untyped nominal terms [20,10]. The last condition enforces type consistency: terms should have compatible types, and the solution should instantiate the variables consistent with the typing assumptions. For typeable closed rules, the typing context Φ' is uniquely defined.

The assumption that Φ mentions no atoms or type-schemes may seem strong, but is all we need: we give applications in Section 4.4.

For example, $(X{:}\alpha\vdash X)\ {}_?{\approx}\ (a{:}\mathbb{B}\vdash a)$ has solution $([\alpha\mapsto\mathbb{B}],[X\mapsto a])$, whereas $(X{:}\mathbb{B}\vdash X)\ {}_?{\approx}\ (a{:}\alpha\vdash a)$ has no solution—α on the right is too general.

To see why we must check θ in the second condition, consider $\mathsf{g}(\mathsf{f}\ \mathsf{True})$ where $\mathsf{g}\colon(\alpha)\mathbb{N}$ and $\mathsf{f}\colon(\beta)\mathbb{N}$ (i.e. both functions are polymorphic and produce a result of type $\mathbb{N}$) and a rule $X\colon\mathbb{N}\vdash\mathsf{g}(\mathsf{f}\ X)\to\mathsf{suc}\,X$, where $\mathsf{suc}\colon(\mathbb{N})\mathbb{N}$. Then the untyped matching problem $\mathsf{g}(\mathsf{f}\,X))\ {}_?{\approx}\ \mathsf{g}(\mathsf{f}\,\mathsf{True})$ has solution $[X\mapsto\mathsf{True}]$, but the typed matching problem $(X\colon\mathbb{N}\vdash\mathsf{g}(\mathsf{f}\,X))\ {}_?{\approx}\ (\vdash\mathsf{g}(\mathsf{f}\,\mathsf{True}))$ has none: $[X\mapsto\mathsf{True}]$ fails the last condition since X should have type $\mathbb{N}$ but is instantiated with a Boolean. If matching did not fail, this rewrite would not preserve types.

Definition 11 (Typed closed rewriting). Take a typeable term $\Gamma;\Delta\vdash s{:}\mu$ and a typeable closed rule $R\equiv\Phi;\nabla\vdash l\to r:\tau$. Assume $s\equiv C[s']$ and $\Gamma';\Delta\vdash s':\mu'$ is the typing of s' at the corresponding position in s.[3] We say s **closed rewrites with R to t in the context** $\Gamma;\Delta$ and write $\Gamma;\Delta\vdash s\overset{R}{\to}_c t$ when there is some $R^{\text{\tiny И}}$ a freshened variant of R (fresh for R, Γ, Δ, s, and t), where Δ' is the freshness context $A(R^{\text{\tiny И}})\ \#\ V(\Gamma,\Delta,s,t)$, such that:

[2] So in particular, by the side-conditions on variables being disjoint between left and right of the problem, $X\theta\equiv X$ for $X{\in}V(\Gamma,\Delta,s)$ and $\alpha S\equiv\alpha$ for $\alpha\in TV(\Gamma)$.
[3] $\Gamma\subseteq\Gamma'$, and $\Gamma\subset\Gamma'$ if C contains abstractions over s'.

1. The typed matching problem $(\Phi^{\text{и}};\nabla^{\text{и}}\vdash l^{\text{и}})\ {}_?{\approx}\ (\Gamma';\Delta,\Delta'\vdash s')$ has solution (S,θ).
2. $\Delta,\Delta'\vdash C[r^{\text{и}}\theta]\approx_\alpha t$.

The conditions of typed closed rewriting extend those of closed nominal rewriting, with types. The following lemma, proved by induction, is used to prove that rewriting preserves types:

Lemma 12. *Consider $\Phi';\nabla\vdash r'$, where r' is either the right-hand side of a typeable closed rule $R\equiv\Phi;\nabla\vdash l\to r:\tau$ or a subterm of the right hand side; if r' is under atom abstractions in r, we assume $\Phi'=\Phi\cup\Phi''$ where Φ'' contains the atom typings of the removed atom abstractions. Assume $\Phi';\nabla\vdash r':\tau'$ (so $\tau'=\tau$ if $r'=r$) and let (S,θ) be the substitutions that solve the typed matching problem $(\Phi;\nabla\vdash l)\ {}_?{\approx}\ (\Gamma;\Delta\vdash s)$. Then $\Gamma,\Phi''S;\Delta\vdash r'\theta:\tau'S$.*

Theorem 13 (Subject Reduction). *Let $R\equiv\Phi;\nabla\vdash l\to r:\tau$ be a typeable closed rule. If $\Gamma;\Delta\vdash s:\mu$ and $\Gamma;\Delta\vdash s\overset{R}{\to}_c t$ then $\Gamma;\Delta\vdash t:\mu$.*

Proof. It suffices to prove that if $pt(\Gamma;\Delta\vdash s)=(id,\nu)$ and $\Gamma;\Delta\vdash s\overset{R}{\to}_c t$ then $\Gamma;\Delta\vdash t:\nu$. Suppose $\Gamma;\Delta\vdash s\overset{R}{\to}_c t$. Then (using the notation in the definition of matching and rewriting above) we know that:
1. $R^{\text{и}}$ is a freshened variant of R (fresh for R, Γ, Δ, s, and t) and Δ' is the freshness context $A(R^{\text{и}})\ \#\ V(\Gamma,\Delta,s,t)$.
2. $s\equiv C[s']$, $\Delta,\Delta'\vdash l^{\text{и}}\theta\approx_\alpha s'$, and $\Delta,\Delta'\vdash a\#X\theta$ for each $a\#X$ in $\nabla^{\text{и}}$.
3. $\Gamma';\Delta\vdash s':\nu'$ is the typing of s', and by the Weakening Lemma 6, also $\Gamma';\Delta,\Delta'\vdash s':\nu'$.
4. $pt(\Phi^{\text{и}};\nabla^{\text{и}}\vdash l^{\text{и}})=(id,\tau)$ and $pt(\Gamma',\Delta\vdash s')=(id,\tau S)$ so there is some S' such that $\Gamma'S'=\Gamma'$ and $\tau SS'=\nu'$.
5. $\Delta,\Delta'\vdash C[r^{\text{и}}\theta]\approx_\alpha t$.

By Theorem 4 (α-equivalence respects types), and 1, 2 and 3, we deduce $\Gamma';\Delta,\Delta'\vdash l^{\text{и}}\theta:\tau SS'$. Since $pt(\Phi^{\text{и}};\nabla^{\text{и}}\vdash l^{\text{и}})=(id,\tau)$, by our assumptions on rewrite rules also $\Phi^{\text{и}};\nabla^{\text{и}}\vdash r^{\text{и}}:\tau$, and by Lemma 3 also $\Phi^{\text{и}}SS';\nabla\vdash r^{\text{и}}:\tau SS'$. Since θ respects the context $\Phi^{\text{и}}SS';\nabla^{\text{и}}$ by definition of typed matching, then, by the auxiliary lemma 12, $\Gamma';\Delta,\Delta'\vdash r^{\text{и}}\theta:\tau SS'=\nu'$. Hence $\Gamma;\Delta,\Delta'\vdash C[r^{\text{и}}\theta]:\nu$. By Theorem 4, also $\Gamma;\Delta,\Delta'\vdash t:\nu$, and since $A(\Delta')\not\in t$ (by definition of closed rewriting), then $\Gamma;\Delta\vdash t:\nu$ as required. $\square$

Closure ensures that for each occurrence of X in l and r, instantiation behaves uniformly with respect to abstraction. This alone does not ensure preservation of types under reduction: consider $\mathsf{f}:([N]N)N$, $\mathsf{g}:([Bool]N)N$, $\mathsf{suc}:(N)N$. The rule $X:N\vdash \mathsf{f}[a]X\to\mathsf{g}[a]X:N$ is closed, and the principal type of the left-hand side is also the type of the right-hand side, but the term $\mathsf{f}[a]\mathsf{suc}(a):N$ rewrites to $\mathsf{g}[a]\mathsf{suc}(a)$, which is untypeable. Each X in l and r must be under the same abstractions, *typed in the same way.* So condition 3 of Definition 9 annotates abstracted atoms in $l\to r$ with types.

4.3 Typed equational reasoning

Definition 14. A **typeable closed axiom** $A \equiv \Phi; \nabla \vdash l = r : \tau$ is a tuple of a type context Φ which only types the variables in l and r and has no type-schemes (so Φ mentions no atoms), a freshness context ∇, and terms l and r such that $pt(\Phi; \nabla \vdash l) = pt(\Phi; \nabla \vdash r) = (id, \tau)$ and A^T is closed, where A^T is a decorated version of A from Definition 11.

Typed nominal equational reasoning is defined as in Definition 7, but with the additional requirement that the substitution θ applied to l and r solves *typed* matching problems, that is, it must respect the types given in the typing context. Thus, using Lemma 12, equational deduction preserves types.

Theorem 15 (Preservation of types). *Let* T *be a nominal theory with only typeable closed axioms. If* $\Gamma; \Delta \vdash s{:}\mu$ *and* $\Gamma; \Delta \vdash_{\mathsf{T}} s{=}t$ *is in the typeable nominal equality relation generated from* T *by typed nominal reasoning, then* $\Gamma; \Delta \vdash t : \mu$.

Proof. By induction on nominal algebra equality. The interesting case is the use of the rule $(\mathbf{Axi}_{\mathbf{\Phi;\nabla\vdash l=r}})$, for a typeable closed axiom $A \equiv \Phi; \nabla \vdash l = r : \tau$ in T. The conditions in Definition 14 ensure that $pt(\Phi; \nabla \vdash l) = pt(\Phi; \nabla \vdash r)$. Since the substitution used to instantiate l and r to derive $\Gamma; \Delta \vdash_{\mathsf{T}} s = t$ respects $\Phi; \nabla$ (it solves typed matching problems), we can proceed as for Theorem 13. □

4.4 Examples

Rewrites for **surjective pairing** cannot be implemented by compositional translation to λ-calculus [3]. We can define it; assume $\mathsf{fst} : (\alpha \times \beta)\alpha$ and $\mathsf{snd} : (\alpha \times \beta)\beta$:

$$X{:}\alpha,\ Y{:}\beta \vdash \mathsf{fst}(X, Y) \to X{:}\alpha \qquad X{:}\alpha,\ Y{:}\beta \vdash \mathsf{snd}(X, Y) \to Y{:}\beta$$
$$Z{:}\alpha{\times}\beta \vdash (\mathsf{fst}\, Z, \mathsf{snd}\, Z) \to Z{:}\alpha{\times}\beta$$

Consider a type o and term-formers $\top, \bot\colon o$, $\wedge, \Rightarrow, \Leftrightarrow\colon (o \times o)o$, $\approx\colon (\alpha \times \alpha)o$, $\forall\colon ([\alpha]o)o$, and substitution $\sigma\colon ([\alpha]o, \alpha)o$ sugared as before. Rewrite rules for equality and simplification for (a fragment of) **first-order logic** are typeable closed; we use infix notation for the binary connectives:

$$X{:}\alpha \vdash X \approx X \to \top{:}o \qquad X{:}o; a\#X \vdash \forall[a]X \to X{:}o$$
$$X, Y{:}o \vdash \forall[a](X \wedge Y) \to \forall[a]X \wedge \forall[a]Y{:}o$$

Using $P[a{\mapsto}T]$ as syntactic sugar for the term $\sigma([a]P, T)$ representing the substitution of a by T in P, we also have the following typeable closed axioms:

$$\begin{array}{l} P\colon o, T\colon \alpha \vdash\ \forall[a]P {\Rightarrow} P[a{\mapsto}T] = \top\colon o \\ P, Q\colon o;\ a\#P\ \vdash\ \forall[a](P {\Rightarrow} Q) \Leftrightarrow P {\Rightarrow} \forall[a]Q = \top\colon o \\ U, T\colon \alpha, P\colon o \vdash\ U \approx T \wedge P[a{\mapsto}T] {\Rightarrow} P[a{\mapsto}U] = \top\colon o \end{array}$$

We can extend this with arithmetic. Consider a type $\mathbb{N}$, term-formers $0 : \mathbb{N}$, $\mathsf{succ} : (\mathbb{N})\mathbb{N}$, $+ : (\mathbb{N} \times \mathbb{N})\mathbb{N}$. Then $[a](a \approx a)$ has principal type $[\alpha]o$, and $\forall[a](a \approx 0)$ is typeable (with type o) whereas $\forall[a](\top \approx 0)$ is not typeable.

We can also axiomatise **the λ-calculus** by these equalities [15], using the notation introduced in Section 3:

$$a[a\mapsto X] = X \quad a\#Z \vdash Z[a\mapsto X] = Z \qquad c\#X \vdash (\lambda[c]Z)[a\mapsto X] = \lambda[c](Z[a\mapsto X])$$
$$(Z'Z)[a\mapsto X] = (Z'[a\mapsto X])(Z[a\mapsto X]) \qquad Z[a\mapsto a] = Z$$

All are typeable closed axioms except for the last, which is not closed: it contains a free atom a on the left hand side and the atoms in an instance of Z on the left are under the scope of an abstraction for a, but they escape the abstraction on the right. The results presented here do not apply to this rule. Non-closed rules are arguably rare (and the standard higher-order rewriting formalisms do not accept them), but interesting examples exist—in this case, the rule is needed for completeness with respect to the models [15]. We conjecture that an extension with *essential typings* [9] (type assignments need not be unique but every type assignment for an occurrence of a variable X on the right must have already been made for an occurrence of X on the left) would let us deal with non-closed rules or axioms like this.

5 Conclusions and future work

Type inference is well-studied for the λ-calculus and Curry-style systems also exist for first-order rewriting systems [1] and algebraic λ-calculi (which combine term rewriting and λ-calculus) [2]. We know of no type assignment system for the standard higher-order rewriting formats (HRSs use a typed metalanguage, and restrict rewrite rules to base types). Our system is in Curry style; type annotations on terms are not required. We do rely on type declarations for term-formers (arities) and in future we may investigate conditions under which this assumption and the closedness assumptions could be relaxed.

The types produced resemble the Hindley-Milner polymorphic type system for the λ-calculus, but are acting on nominal terms which include variables X representing context holes as well as atoms a representing program variables.

The function *pt* is implemented [8]. The conditions on typeable closed rules and typeable closed axioms are decidable and easily mechanisable using *pt* and nominal matching, which is also implemented [5].

Theorem 4 proves our types compatible with the powerful notion of α-equivalence inherited from nominal terms [20]. Theorem 13 shows that a notion of typeable closed nominal rewrite rule exists which guarantees the preservation of types under closed nominal rewriting. Theorem 15 shows that types are also preserved under equational reasoning using a notion of typeable closed axioms.

Our type system has rank 1 polymorphism. More powerful systems, e.g. rank 2 polymorphic types or intersection types [4], should be possible. The latter have been used to characterise normalisation properties of λ-terms. Normalisation of nominal rewriting using type systems is a subject for future work, and one of our long-term goals is to come closer to applying logical semantics such as intersection types, to nominal rewriting.

References

1. Steffen van Bakel and Maribel Fernández, *Normalization results for typeable rewrite systems*, Information and Computation **133** (1997), no. 2, 73–116.
2. Franco Barbanera and Maribel Fernández, *Intersection type assignment systems with higher-order algebraic rewriting*, Theoretical Computer Science **170** (1996), 173–207.
3. Henk P. Barendregt, *Pairing without conventional constraints*, Zeitschrift für mathematischen Logik und Grundlagen der Mathematik **20** (1974), 289–306.
4. Henk P. Barendregt, Mario Coppo, and Mariangiola Dezani Ciancaglini, *A filter lambda model and the completeness of type assignment*, Journal of Symbolic Logic **48** (1983), no. 4, 931–940.
5. Christophe Calvès, *Complexity and implementation of nominal algorithms*, Ph.D. thesis, King's College London, 2010.
6. Ranald Clouston, *Closed terms (unpublished notes)*, `http://users.cecs.anu.edu.au/~rclouston/closedterms.pdf`, 2007.
7. Luis Damas and Robin Milner, *Principal type-schemes for functional programs*, Proceedings of the 9th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 82), ACM, 1982, pp. 207–212.
8. Elliot Fairweather, *Principal types for nominal terms: tool description*, Available from `http://www.inf.kcl.ac.uk/pg/elliotf/research`.
9. Maribel Fernández and Murdoch J. Gabbay, *Curry-style types for nominal terms*, Types for Proofs and Programs (TYPES 06), Lecture Notes in Computer Science, vol. 4502, Springer, September 2007, pp. 125–139.
10. ———, *Nominal rewriting (journal version)*, Information and Computation **205** (2007), no. 6, 917–965.
11. ———, *Closed nominal rewriting and efficiently computable nominal algebra equality*, Proceedings of the 5th International Workshop on Logical Frameworks and Meta-Languages (LFMTP 2010), 2010.
12. Maribel Fernández, Murdoch J. Gabbay, and Ian Mackie, *Nominal Rewriting Systems*, Proceedings of the 6th ACM SIGPLAN symposium on Principles and Practice of Declarative Programming (PPDP 2004), ACM Press, August 2004, pp. 108–119.
13. Murdoch J. Gabbay, *Nominal terms and nominal logics: from foundations to metamathematics*, Handbook of Philosphical Logic, vol. 17, Kluwer, 2011.
14. Murdoch J. Gabbay and Aad Mathijssen, *Nominal universal algebra: equational logic with names and binding*, Journal of Logic and Computation **19** (2009), no. 6, 1455–1508.
15. ———, *A nominal axiomatisation of the lambda-calculus*, Journal of Logic and Computation **20** (2010), no. 2, 501–531.
16. Jean-Yves Girard, *The system F of variable types, fifteen years later*, Theoretical Computer Science **45** (1986).
17. James Gosling, Bill Joy, and Guy Steele, *The Java language specification*, Addison-Wesley, 1996.
18. Andrew M. Pitts, *Nominal system T*, Proceedings of the 37th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL 2010), ACM Press, January 2010, pp. 159–170.
19. Mark R. Shinwell, Andrew M. Pitts, and Murdoch J. Gabbay, *FreshML: Programming with Binders Made Simple*, Proceedings of the 8th ACM SIGPLAN International Conference on Functional Programming (ICFP 2003), vol. 38, ACM Press, August 2003, pp. 263–274.
20. Christian Urban, Andrew M. Pitts, and Murdoch J. Gabbay, *Nominal Unification*, Theoretical Computer Science **323** (2004), no. 1–3, 473–497.

Acknowledgements. Thanks to the anonymous referees. Supported by grant RYC-2006-002131 at the Polytechnic University of Madrid.