Types for Nominal Terms and Rewrite Rules

Maribel Fernández   Murdoch J. Gabbay

DCS, King's College London

TYPES, April 2006

Motivations

Specifying binding operations — informal presentations:

- Operational semantics:

$$\text{let } a = N \text{ in } M \;\longrightarrow\; (\text{fun } a \to M)N$$

Motivations

Specifying binding operations — informal presentations:

- Operational semantics:

$$\text{let } a = N \text{ in } M \longrightarrow (\text{fun } a \rightarrow M)N$$

- β and η-reductions in the λ-calculus:

$$\begin{array}{lll} (\lambda x.M)N & \rightarrow & M[x/N] \\ (\lambda x.Mx) & \rightarrow & M \qquad (x \notin \mathrm{fv}(M)) \end{array}$$

Motivations

Specifying binding operations — informal presentations:

- Operational semantics:

$$\text{let } a = N \text{ in } M \longrightarrow (\text{fun } a \rightarrow M)N$$

- β and η-reductions in the λ-calculus:

$$\begin{array}{lcll} (\lambda x.M)N & \rightarrow & M[x/N] & \\ (\lambda x.Mx) & \rightarrow & M & (x \notin \text{fv}(M)) \end{array}$$

- π-calculus:

$$P \mid \nu a.Q \rightarrow \nu a.(P \mid Q) \qquad (a \notin \text{fv}(P))$$

Motivations

Specifying binding operations — informal presentations:

- Operational semantics:

$$\text{let } a = N \text{ in } M \longrightarrow (\text{fun } a \to M)N$$

- β and η-reductions in the λ-calculus:

$$\begin{array}{lll} (\lambda x.M)N & \to & M[x/N] \\ (\lambda x.Mx) & \to & M \qquad (x \notin \text{fv}(M)) \end{array}$$

- π-calculus:

$$P \mid \nu a.Q \to \nu a.(P \mid Q) \qquad (a \notin \text{fv}(P))$$

- α-conversion is implicit, but

Motivations

Specifying binding operations — informal presentations:

- Operational semantics:

$$\text{let } a = N \text{ in } M \longrightarrow (\text{fun } a \to M)N$$

- β and η-reductions in the λ-calculus:

$$\begin{array}{lcll} (\lambda x.M)N & \to & M[x/N] & \\ (\lambda x.Mx) & \to & M & (x \notin \text{fv}(M)) \end{array}$$

- π-calculus:

$$P \mid \nu a.Q \to \nu a.(P \mid Q) \qquad (a \notin \text{fv}(P))$$

- α-conversion is implicit, but
- $(\text{fun } a \to M) \neq_\alpha (\text{fun } b \to M)$ since a may occur in M.

Formally:

There are several alternatives.

- First-order rewrite systems.

$$\begin{array}{lcl} append(nil, x) & \rightarrow & x \\ append(cons(x, z), y) & \rightarrow & cons(x, append(z, y)) \end{array}$$

Formally:

There are several alternatives.

- First-order rewrite systems.

$$
\begin{array}{lcl}
append(nil, x) & \to & x \\
append(cons(x, z), y) & \to & cons(x, append(z, y))
\end{array}
$$

  - No binders. (-)

Formally:

There are several alternatives.

- First-order rewrite systems.

$$\begin{array}{lcl} append(nil, x) & \rightarrow & x \\ append(cons(x, z), y) & \rightarrow & cons(x, append(z, y)) \end{array}$$

  - No binders. (-)
  - First-order matching: we need to 'specify' α-conversion. (-)

Formally:

There are several alternatives.

- First-order rewrite systems.

$$\begin{array}{lcl} append(nil, x) & \rightarrow & x \\ append(cons(x, z), y) & \rightarrow & cons(x, append(z, y)) \end{array}$$

  - No binders. (-)
  - First-order matching: we need to 'specify' α-conversion. (-)
  - Simple notion of substitution. (+)

Higher-order frameworks

- Higher-order rewrite systems (CRS, HRS, etc.)
  β-rule:
  $$app(lam([a]Z(a)), Z') \to Z(Z')$$
  Then $app(lam([a]f(a, g(a)), b) \to f(b, g(b))$
  using higher-order matching.

Higher-order frameworks

- Higher-order rewrite systems (CRS, HRS, etc.)
  β-rule:
  $$app(lam([a]Z(a)), Z') \to Z(Z')$$
  Then $app(lam([a]f(a, g(a)), b) \to f(b, g(b))$
  using higher-order matching.
- Higher-Order Abstract Syntax:

$$\text{let } a = N \text{ in } M(a) \quad \longrightarrow \quad (\text{fun } a \to M(a))N$$

Higher-order frameworks

- Higher-order rewrite systems (CRS, HRS, etc.)
  β-rule:
  $$app(lam([a]Z(a)), Z') \to Z(Z')$$
  Then $app(lam([a]f(a, g(a)), b) \to f(b, g(b))$
  using higher-order matching.
- Higher-Order Abstract Syntax:
  $$\text{let } a = N \text{ in } M(a) \longrightarrow (\text{fun } a \to M(a))N$$
  - Terms with binders. $(+)$

Higher-order frameworks

- Higher-order rewrite systems (CRS, HRS, etc.)
  β-rule:

  $$app(lam([a]Z(a)), Z') \to Z(Z')$$

  Then $app(lam([a]f(a, g(a)), b) \to f(b, g(b))$
  using higher-order matching.
- Higher-Order Abstract Syntax:

  $$\text{let } a = N \text{ in } M(a) \quad \longrightarrow \quad (\text{fun } a \to M(a))N$$

  - Terms with binders. $(+)$
  - Implicit α-conversion. $(+)$

Higher-order frameworks

- Higher-order rewrite systems (CRS, HRS, etc.)
  β-rule:

  $$app(lam([a]Z(a)), Z') \rightarrow Z(Z')$$

  Then $app(lam([a]f(a, g(a)), b) \rightarrow f(b, g(b))$
  using higher-order matching.
- Higher-Order Abstract Syntax:

  $$\text{let } a = N \text{ in } M(a) \;\longrightarrow\; (\text{fun } a \rightarrow M(a))N$$

  - Terms with binders. (+)
  - Implicit α-conversion. (+)
  - We targeted α but now we have to deal with β too. (-)

Higher-order frameworks

- Higher-order rewrite systems (CRS, HRS, etc.)
  β-rule:
  $$app(lam([a]Z(a)), Z') \to Z(Z')$$
  Then $app(lam([a]f(a, g(a)), b) \to f(b, g(b))$
  using higher-order matching.
- Higher-Order Abstract Syntax:
  $$\text{let } a = N \text{ in } M(a) \quad \longrightarrow \quad (\text{fun } a \to M(a))N$$
  - Terms with binders. (+)
  - Implicit α-conversion. (+)
  - We targeted α but now we have to deal with β too. (-)
  - Substitution is a meta-operation using β. (-)

Higher-order frameworks

- Higher-order rewrite systems (CRS, HRS, etc.)
  β-rule:
  $$app(lam([a]Z(a)), Z') \to Z(Z')$$
  Then $app(lam([a]f(a, g(a)), b) \to f(b, g(b))$
  using higher-order matching.
- Higher-Order Abstract Syntax:
  $$\text{let } a = N \text{ in } M(a) \longrightarrow (\text{fun } a \to M(a))N$$
  - Terms with binders. (+)
  - Implicit α-conversion. (+)
  - We targeted α but now we have to deal with β too. (-)
  - Substitution is a meta-operation using β. (-)
  - Unification is undecidable in general. (-)

Higher-order frameworks

- Higher-order rewrite systems (CRS, HRS, etc.)
  β-rule:
  $$app(lam([a]Z(a)), Z') \to Z(Z')$$
  Then $app(lam([a]f(a, g(a)), b) \to f(b, g(b))$
  using higher-order matching.
- Higher-Order Abstract Syntax:
  $$\text{let } a = N \text{ in } M(a) \longrightarrow (\text{fun } a \to M(a))N$$
  - Terms with binders. (+)
  - Implicit α-conversion. (+)
  - We targeted α but now we have to deal with β too. (-)
  - Substitution is a meta-operation using β. (-)
  - Unification is undecidable in general. (-)
  - Leaving name dependencies implicit is convenient (e.g. $\forall x.P$).

Nominal Terms, Unification, Rewriting

Inspired by the work on Nominal Logic and Fresh ML.
Key ideas: Freshness conditions $a\#t$, name swapping $(ab)t$.
Example: β and η rules as Nominal Rewriting Systems:

$$\begin{array}{rcl} & app(lam([a]Z), Z') & \rightarrow & subst([a]Z, Z') \\ a\#M \vdash & (\lambda([a]app(M, a)) & \rightarrow & M \end{array}$$

$\Rightarrow$ Terms with binders.

Nominal Terms, Unification, Rewriting

Inspired by the work on Nominal Logic and Fresh ML.
Key ideas: Freshness conditions $a\#t$, name swapping $(ab)t$.
Example: β and η rules as Nominal Rewriting Systems:

$$\begin{array}{rcl} & app(lam([a]Z), Z') & \rightarrow & subst([a]Z, Z') \\ a\#M \vdash & (\lambda([a]app(M, a)) & \rightarrow & M \end{array}$$

- Terms with binders.

$\Rightarrow$ Matching modulo α (but terms are not defined as α-equivalence classes)

Nominal Terms, Unification, Rewriting

Inspired by the work on Nominal Logic and Fresh ML.
Key ideas: Freshness conditions $a\#t$, name swapping $(ab)t$.
Example: β and η rules as Nominal Rewriting Systems:

$$
\begin{array}{rlcl}
 & app(lam([a]Z), Z') & \rightarrow & subst([a]Z, Z') \\
a\#M \vdash & (\lambda([a]app(M, a)) & \rightarrow & M
\end{array}
$$

- Terms with binders.
- Matching modulo α (but terms are not defined as α-equivalence classes)

$\Rightarrow$ Simple notion of substitution (first order).

Nominal Terms, Unification, Rewriting

Inspired by the work on Nominal Logic and Fresh ML.
Key ideas: Freshness conditions $a\#t$, name swapping $(ab)t$.
Example: β and η rules as Nominal Rewriting Systems:

$$\begin{array}{rcl} & app(lam([a]Z), Z') & \rightarrow & subst([a]Z, Z') \\ a\#M \vdash & (\lambda([a]app(M, a)) & \rightarrow & M \end{array}$$

- Terms with binders.
- Matching modulo α (but terms are not defined as α-equivalence classes)
- Simple notion of substitution (first order).

$\Rightarrow$ Dependencies of terms on names are implicit.

Nominal Terms, Unification, Rewriting

Inspired by the work on Nominal Logic and Fresh ML.
Key ideas: Freshness conditions $a\#t$, name swapping $(ab)t$.
Example: β and η rules as Nominal Rewriting Systems:

$$\begin{array}{rlcl} & app(lam([a]Z), Z') & \rightarrow & subst([a]Z, Z') \\ a\#M \vdash & (\lambda([a]app(M, a)) & \rightarrow & M \end{array}$$

- Terms with binders.
- Matching modulo α (but terms are not defined as α-equivalence classes)
- Simple notion of substitution (first order).
- Dependencies of terms on names are implicit.

⇒ Easy to express constraints such as $a \notin \mathrm{fv}(M)$.

Nominal Terms, Unification, Rewriting

Inspired by the work on Nominal Logic and Fresh ML.
Key ideas: Freshness conditions $a\#t$, name swapping $(ab)t$.
Example: β and η rules as Nominal Rewriting Systems:

$$
\begin{array}{rlcl}
 & app(lam([a]Z), Z') & \to & subst([a]Z, Z') \\
a\#M \vdash & (\lambda([a]app(M, a)) & \to & M
\end{array}
$$

- Terms with binders.
- Matching modulo α (but terms are not defined as α-equivalence classes)
- Simple notion of substitution (first order).
- Dependencies of terms on names are implicit.
- Easy to express constraints such as $a \notin \mathrm{fv}(M)$.

$\Rightarrow$ Can be easily generalised to express more general constraints.

- Function symbols: $f, g \ldots$
  Variables: $M, N, X, Y, \ldots$
  Atoms: $a, b, \ldots$
  Swappings: $(a\ b)$
  Def. $(a\ b)a = b$, $(a\ b)b = a$, $(a\ b)c = c$
  Permutations: lists of swappings, denoted π (Id empty).
- Nominal Terms:

$$s, t ::= a \mid \pi \cdot X \mid [a]t \mid f\ t \mid (t_1, \ldots, t_n)$$

  $Id \cdot X$ written as X.
- Example (ML): $var(a)$, $app(t, t')$, $lam([a]t)$, $let(t, [a]t')$, $letrec[f]([a]t, t')$, $subst([a]t, t')$
  Syntactic sugar:
  a, (tt'), $\lambda a.t$, let $a = t$ in t', letrec $fa = t$ in t', $t[a \mapsto t']$

Types for Nominal Terms

Types built from

- a set of base data sorts δ (e.g. Nat, Bool, Exp, ...)
- type variables α, and
- type constructors *tf* (e.g. $\times$, $\rightarrow$, List, ...)

$$\tau ::= \delta \mid \alpha \mid (\tau_1 \times \ldots \times \tau_n \mid \mathit{tf}\ \tau \mid [\tau]\tau' \qquad \sigma ::= \forall \overline{\alpha}\tau$$

Type declarations (arity):

$$\rho ::= (\tau')\tau$$

Instantiation relation: $\sigma \leq \tau$

Typing Rules

$$\frac{\sigma \leq \tau}{\Gamma, a : \sigma \vdash a : \tau} \qquad \frac{\sigma \leq \tau}{\Gamma, X : \sigma \vdash \pi \cdot X : \tau} \qquad \frac{\Gamma \vdash t : \tau' \quad f : \rho \leq (\tau')\tau}{\Gamma \vdash f\ t : \tau}$$

$$\frac{\Gamma, a : \tau \vdash t : \tau'}{\Gamma \vdash [a]t : [\tau]\tau'} \qquad \frac{\Gamma \vdash t_i : \tau_i}{\Gamma \vdash (t_1, \ldots, t_n) : (\tau_1 \times \ldots \times \tau_n)}$$

Example:

$X : \tau,\ b : \beta \vdash [a]((a\ b) \cdot X, b) : [\alpha](\tau \times \beta)$

Remark:

- Permutations are ignored in the typing rules (but will be taken into account when instantiating terms).
- Generalisation of Hindley-Milner's type system: atoms (can be abstracted or unabstracted), variables (cannot be abstracted but can be instantiated, with non-capture-avoiding substitutions), suspended permutations.

Principal Types

- Every term has a principal type, obtained using the function $pt(\Gamma \vdash s)$.
  pt is sound and complete.

Principal Types

- Every term has a principal type, obtained using the function $pt(\Gamma \vdash s)$.
  pt is sound and complete.
- Type inference is decidable.

Principal Types

- Every term has a principal type, obtained using the function $pt(\Gamma \vdash s)$.
  pt is sound and complete.
- Type inference is decidable.
- Types are preserved by α-equivalence.

α-equivalence (Freshness)

We use freshness to avoid name capture.
$a\#X$ means $a \notin \text{fv}(X)$ when X is instantiated.

$$\frac{}{a\#b} \qquad \frac{}{a\#[a]s} \qquad \frac{\pi^{-1}(a)\#X}{a\#\pi \cdot X}$$

$$\frac{a\#s_1 \quad \cdots \quad a\#s_n}{a\#(s_1, \ldots, s_n)} \qquad \frac{a\#s}{a\#fs} \qquad \frac{a\#s}{a\#[b]s}$$

α-equivalence

$$\frac{}{a \approx_\alpha a} \qquad \frac{ds(\pi, \pi')\#X}{\pi \cdot X \approx_\alpha \pi' \cdot X}$$

$$\frac{s_1 \approx_\alpha t_1 \quad \cdots \quad s_n \approx_\alpha t_n}{(s_1, \ldots, s_n) \approx_\alpha (t_1, \ldots, t_n)} \qquad \frac{s \approx_\alpha t}{fs \approx_\alpha ft}$$

$$\frac{s \approx_\alpha t}{[a]s \approx_\alpha [a]t} \qquad \frac{a\#t \qquad s \approx_\alpha (a\ b) \cdot t}{[a]s \approx_\alpha [b]t}$$

where

$$ds(\pi, \pi') = \{n | \pi(n) \neq \pi'(n)\}$$

- $a\#X, b\#X \vdash (a\ b) \cdot X \approx_\alpha X$

α-equivalence

$$\frac{}{a \approx_\alpha a} \qquad \frac{ds(\pi, \pi')\#X}{\pi \cdot X \approx_\alpha \pi' \cdot X}$$

$$\frac{s_1 \approx_\alpha t_1 \quad \cdots \quad s_n \approx_\alpha t_n}{(s_1, \ldots, s_n) \approx_\alpha (t_1, \ldots, t_n)} \qquad \frac{s \approx_\alpha t}{fs \approx_\alpha ft}$$

$$\frac{s \approx_\alpha t}{[a]s \approx_\alpha [a]t} \qquad \frac{a\#t \qquad s \approx_\alpha (a\ b) \cdot t}{[a]s \approx_\alpha [b]t}$$

where

$$ds(\pi, \pi') = \{n | \pi(n) \neq \pi'(n)\}$$

- $a\#X, b\#X \vdash (a\ b) \cdot X \approx_\alpha X$
- $b\#X \vdash \lambda[a]X \approx_\alpha \lambda[b](a\ b) \cdot X$

α-equivalence

$$\frac{}{a \approx_\alpha a} \qquad \frac{ds(\pi, \pi')\#X}{\pi \cdot X \approx_\alpha \pi' \cdot X}$$

$$\frac{s_1 \approx_\alpha t_1 \quad \cdots \quad s_n \approx_\alpha t_n}{(s_1, \ldots, s_n) \approx_\alpha (t_1, \ldots, t_n)} \qquad \frac{s \approx_\alpha t}{fs \approx_\alpha ft}$$

$$\frac{s \approx_\alpha t}{[a]s \approx_\alpha [a]t} \qquad \frac{a\#t \qquad s \approx_\alpha (a\ b) \cdot t}{[a]s \approx_\alpha [b]t}$$

where

$$ds(\pi, \pi') = \{n | \pi(n) \neq \pi'(n)\}$$

- $a\#X, b\#X \vdash (a\ b) \cdot X \approx_\alpha X$
- $b\#X \vdash \lambda[a]X \approx_\alpha \lambda[b](a\ b) \cdot X$
- α-equivalence respects types:
  $\Delta \ \vdash \ s \approx_\alpha t$ and $\Gamma \ \vdash \ s : \tau \Rightarrow \Gamma \ \vdash \ t : \tau$.

Nominal Unification and Matching

- $l\ {}_?{\approx}_?\ t$ has solution (Δ, θ) if

$$\Delta \vdash l\theta \approx_\alpha t\theta$$

A solvable problem Pr has a unique most general solution: (Γ, θ) such that $\Gamma \vdash Pr\theta$

Nominal Unification and Matching

- $l\ {}_?\!\approx_?\ t$ has solution (Δ, θ) if

$$\Delta \vdash l\theta \approx_\alpha t\theta$$

  A solvable problem Pr has a unique most general solution: (Γ, θ) such that $\Gamma \vdash Pr\theta$

- Nominal unification (and matching) is decidable [Urban, Pitts, Gabbay 2003, TCS 04]

Nominal Unification and Matching

- $l \; {}_?{\approx}_? \; t$ has solution (Δ, θ) if

  $$\Delta \vdash l\theta \approx_\alpha t\theta$$

  A solvable problem Pr has a unique most general solution: (Γ, θ) such that $\Gamma \vdash Pr\theta$
- Nominal unification (and matching) is decidable [Urban, Pitts, Gabbay 2003, TCS 04]
- and polynomial [TERMGRAPH 06].

Rules:

$$\Delta \vdash l \to r \qquad V(r) \cup V(\Delta) \subseteq V(l)$$

Examples:

$$\begin{array}{rlcl}
 & (\lambda[a]X)Y & \to & X[a\mapsto Y] \\
 & (XX')[a\mapsto Y] & \to & X[a\mapsto Y]X'[a\mapsto Y] \\
a\#Y \vdash & Y[a\mapsto X] & \to & Y \\
b\#Y \vdash & (\lambda[b]X)[a\mapsto Y] & \to & \lambda[b](X[a\mapsto Y])
\end{array}$$

Typed Nominal Rewriting

Typed Rules:

$$\Phi; \nabla \ \vdash \ l \to r : \tau$$

where:

- Φ types only variables and has no type-schemes,

Typed Nominal Rewriting

Typed Rules:

$$\Phi; \nabla \ \vdash \ l \to r : \tau$$

where:

- Φ types only variables and has no type-schemes,
- $pt(\Phi \ \vdash \ l) = (Id, \tau)$ and $\Phi \ \vdash \ r : \tau$.

Typed Nominal Rewriting

Typed Rules:

$$\Phi; \nabla \ \vdash \ l \rightarrow r : \tau$$

where:

- Φ types only variables and has no type-schemes,
- $pt(\Phi \ \vdash \ l) = (Id, \tau)$ and $\Phi \ \vdash \ r : \tau$.
- The *essential typings* of $\Phi \ \vdash \ r : \tau$ are a subset of the essential typings of $\Phi \ \vdash \ l : \tau$, up to weakening and strengthening of atoms not affected by permutations.

Typed Nominal Rewriting

Typed Rules:

$$\Phi; \nabla \ \vdash \ l \to r : \tau$$

where:

- Φ types only variables and has no type-schemes,
- $pt(\Phi \ \vdash \ l) = (Id, \tau)$ and $\Phi \ \vdash \ r : \tau$.
- The *essential typings* of $\Phi \ \vdash \ r : \tau$ are a subset of the essential typings of $\Phi \ \vdash \ l : \tau$, up to weakening and strengthening of atoms not affected by permutations.
- Essential typings of $\Phi \ \vdash \ r : \tau$ are the typings associated to $\pi \cdot X$ during $pt(\Phi \ \vdash \ r)$, where we apply π in the typing context.

Typed Rules:

$$\Phi; \nabla \vdash l \to r : \tau$$

where:

- Φ types only variables and has no type-schemes,
- $pt(\Phi \vdash l) = (Id, \tau)$ and $\Phi \vdash r : \tau$.
- The *essential typings* of $\Phi \vdash r : \tau$ are a subset of the essential typings of $\Phi \vdash l : \tau$, up to weakening and strengthening of atoms not affected by permutations.
- Essential typings of $\Phi \vdash r : \tau$ are the typings associated to $\pi \cdot X$ during $pt(\Phi \vdash r)$, where we apply π in the typing context.
- Example: The essential typings of $a : \alpha, X : \tau \vdash ((a\ b) \cdot X, [a]X) : \tau \times [\alpha']\tau$ are $b : \alpha, X : \tau \vdash X : \tau$ and $a : \alpha', X : \tau \vdash X : \tau$.

A **(typed) matching problem** $(\Phi; \nabla \vdash l) \;{}_?{\approx}\; (\Gamma; \Delta \vdash s)$ is a pair of tuples (Φ, Γ are typing contexts, ∇, Δ are freshness contexts, l, s are terms) such that the atoms, variables and type-variables mentioned on the left-hand side are disjoint from those mentioned in Γ, s.

A **solution** is the least pair (S, θ) of a type- and term-substitution such that:

1. $X\theta \equiv X$ for $X \notin V(\Phi, \nabla, l)$ and $\alpha S \equiv \alpha$ for $\alpha \notin TV(\Phi)$.
2. $\Delta \vdash l\theta \approx_\alpha s$ and $\Delta \vdash \nabla\theta$ are derivable.
3. $pt(\Phi \vdash l) = (Id, \tau)$ and $pt(\Gamma \vdash s) = (Id, \tau S)$;
4. For each $\Phi, \Phi' \vdash X : \phi'$ an essential typing of $\Phi \vdash l : \tau$, it is the case that $\Gamma, (\Phi' S) \vdash X\theta : \phi' S$.

We rewrite **terms-in-context** $\Delta \vdash s$.

- Take $\Delta \vdash s$, $\Delta \vdash t$ such that $pt(\Gamma \vdash s) = (Id, \mu)$; and $R \equiv \Phi; \nabla \vdash l \to r : \tau$, such that $V(R) \cap V(\Gamma, \Delta, s, t) = \emptyset$, $A(R) \cap A(\Gamma, \Delta, s, t) = \emptyset$ and $TV(R) \cap TV(\Gamma) = \emptyset$ (renaming variables and atoms in R if necessary).

Nominal Rewriting — Closed Rewriting

We rewrite **terms-in-context** $\Delta \vdash s$.

- Take $\Delta \vdash s$, $\Delta \vdash t$ such that $pt(\Gamma \vdash s) = (Id, \mu)$; and $R \equiv \Phi; \nabla \vdash l \rightarrow r : \tau$, such that $V(R) \cap V(\Gamma, \Delta, s, t) = \emptyset$, $A(R) \cap A(\Gamma, \Delta, s, t) = \emptyset$ and $TV(R) \cap TV(\Gamma) = \emptyset$ (renaming variables and atoms in R if necessary).
- s **rewrites with** R **to** t **in the context** $\Gamma; \Delta$, written $\Gamma; \Delta \vdash s \xrightarrow{R} t$, when:

We rewrite **terms-in-context** $\Delta \vdash s$.

- Take $\Delta \vdash s$, $\Delta \vdash t$ such that $pt(\Gamma \vdash s) = (Id, \mu)$; and $R \equiv \Phi; \nabla \vdash l \rightarrow r : \tau$, such that $V(R) \cap V(\Gamma, \Delta, s, t) = \emptyset$, $A(R) \cap A(\Gamma, \Delta, s, t) = \emptyset$ and $TV(R) \cap TV(\Gamma) = \emptyset$ (renaming variables and atoms in R if necessary).
- s **rewrites with** R **to** t **in the context** $\Gamma; \Delta$, written $\Gamma; \Delta \vdash s \xrightarrow{R} t$, when:
  1. $s = s''[s']$

We rewrite **terms-in-context** $\Delta \vdash s$.

- Take $\Delta \vdash s$, $\Delta \vdash t$ such that $pt(\Gamma \vdash s) = (Id, \mu)$; and $R \equiv \Phi; \nabla \vdash l \to r : \tau$, such that $V(R) \cap V(\Gamma, \Delta, s, t) = \emptyset$, $A(R) \cap A(\Gamma, \Delta, s, t) = \emptyset$ and $TV(R) \cap TV(\Gamma) = \emptyset$ (renaming variables and atoms in R if necessary).
- s **rewrites with** R **to** t **in the context** $\Gamma; \Delta$, written $\Gamma; \Delta \vdash s \xrightarrow{R} t$, when:
    1. $s = s''[s']$
    2. $\Gamma' \vdash s' : \mu'$ is the typing of s' at the corresponding position in a derivation for $\Gamma \vdash s''[s'] : \mu$;

We rewrite **terms-in-context** $\Delta \vdash s$.

- Take $\Delta \vdash s$, $\Delta \vdash t$ such that $pt(\Gamma \vdash s) = (Id, \mu)$; and $R \equiv \Phi; \nabla \vdash l \to r : \tau$, such that $V(R) \cap V(\Gamma, \Delta, s, t) = \emptyset$, $A(R) \cap A(\Gamma, \Delta, s, t) = \emptyset$ and $TV(R) \cap TV(\Gamma) = \emptyset$ (renaming variables and atoms in R if necessary).
- s **rewrites with** R **to** t **in the context** $\Gamma; \Delta$, written $\Gamma; \Delta \vdash s \xrightarrow{R} t$, when:
  1. $s = s''[s']$
  2. $\Gamma' \vdash s' : \mu'$ is the typing of s' at the corresponding position in a derivation for $\Gamma \vdash s''[s'] : \mu$;
  3. $(\Phi; \nabla \vdash l) \;{}_?{\approx}\; (\Gamma'; \Delta, A(\nabla, l) \# V(\Delta, s') \vdash s')$ has solution (S, θ).

We rewrite **terms-in-context** $\Delta \vdash s$.

- Take $\Delta \vdash s$, $\Delta \vdash t$ such that $pt(\Gamma \vdash s) = (Id, \mu)$; and $R \equiv \Phi; \nabla \vdash l \to r : \tau$, such that $V(R) \cap V(\Gamma, \Delta, s, t) = \emptyset$, $A(R) \cap A(\Gamma, \Delta, s, t) = \emptyset$ and $TV(R) \cap TV(\Gamma) = \emptyset$ (renaming variables and atoms in R if necessary).
- s **rewrites with** R **to** t **in the context** $\Gamma; \Delta$, written $\Gamma; \Delta \vdash s \xrightarrow{R} t$, when:
  1. $s = s''[s']$
  2. $\Gamma' \vdash s' : \mu'$ is the typing of s' at the corresponding position in a derivation for $\Gamma \vdash s''[s'] : \mu$;
  3. $(\Phi; \nabla \vdash l) \ {}_?{\approx} \ (\Gamma'; \Delta, A(\nabla, l) \# V(\Delta, s') \vdash s')$ has solution (S, θ).
  4. $\Delta \vdash s''[r\theta] \approx_\alpha t$.

Nominal Rewriting — Closed Rewriting

We rewrite **terms-in-context** $\Delta \vdash s$.

- Take $\Delta \vdash s$, $\Delta \vdash t$ such that $pt(\Gamma \vdash s) = (Id, \mu)$; and $R \equiv \Phi; \nabla \vdash l \to r : \tau$, such that $V(R) \cap V(\Gamma, \Delta, s, t) = \emptyset$, $A(R) \cap A(\Gamma, \Delta, s, t) = \emptyset$ and $TV(R) \cap TV(\Gamma) = \emptyset$ (renaming variables and atoms in R if necessary).
- s **rewrites with** R **to** t **in the context** $\Gamma; \Delta$, written $\Gamma; \Delta \vdash s \xrightarrow{R} t$, when:
  1. $s = s''[s']$
  2. $\Gamma' \vdash s' : \mu'$ is the typing of s' at the corresponding position in a derivation for $\Gamma \vdash s''[s'] : \mu$;
  3. $(\Phi; \nabla \vdash l) \,{}_?{\approx}\, (\Gamma'; \Delta, A(\nabla, l) \# V(\Delta, s') \vdash s')$ has solution (S, θ).
  4. $\Delta \vdash s''[r\theta] \approx_\alpha t$.
- Subject Reduction:
  Let $R \equiv \Phi; \nabla \vdash l \to r : \tau$. If $\Gamma \vdash s : \mu$ and $\Gamma; \Delta \vdash s \xrightarrow{R} t$ then $\Gamma \vdash t : \mu$.

Applications

A (typed!) implementation of the untyped λ-calculus:
Consider a type Λ and term-constructors $lam : ([\Lambda]\Lambda)\Lambda$, $app : (\Lambda \times \Lambda)\Lambda$, and $sub : ([\Lambda]\Lambda \times \Lambda)\Lambda$. We sugar these to $\lambda[a]s$, st, and $s[a \mapsto t]$ respectively.
Rewrite rules:

$$\begin{array}{rcl}
X, Y{:}\Lambda & \vdash & (\lambda[a]X)Y \rightarrow X[a \mapsto Y] : \Lambda \\
X, Y{:}\Lambda;\, a\#X & \vdash & X[a \mapsto Y] \rightarrow X : \Lambda \\
Y{:}\Lambda & \vdash & a[a \mapsto Y] \rightarrow Y : \Lambda \\
X, Y{:}\Lambda;\, b\#Y & \vdash & (\lambda[b]X)[a \mapsto Y] \rightarrow \lambda[b](X[a \mapsto Y]) : \Lambda \\
X, Y, Z{:}\Lambda & \vdash & (XY)[a \mapsto Z] \rightarrow X[a \mapsto Z]\; Y[a \mapsto Z] : \Lambda
\end{array}$$

Applications

Surjective pairing:
Consider *fst* : $(\alpha \times \beta)\alpha$ and *snd* : $(\alpha \times \beta)\beta$.
We can define typable rewrite rules for projections and surjective pairing as follows:

$$
\begin{array}{rl}
X:\alpha,\ Y:\beta \vdash & \mathit{fst}(X,Y) \to X:\alpha \\
X:\alpha,\ Y:\beta \vdash & \mathit{snd}(X,Y) \to Y:\beta \\
X:\alpha\times\beta \vdash & (\mathit{fst}(X),\mathit{snd}(X)) \to X:\alpha\times\beta
\end{array}
$$

Note that this rewrite system cannot be analysed as sugar in the λ-calculus [Barendregt 74].

Conclusion

- Nominal Rewriting Systems: first-order systems with matching modulo α (decidable, polynomial).
  Higher-order rewriting systems can be encoded.
- α-equivalence preserves types.
- Typing is decidable and there are principal types.
- Typing rules ignore permutations but typed-matching and typed-rewriting take them into account.
  Rewriting with typed rewrite rules preserves types.
- Future work: denotational semantics for nominal terms; normalisation properties of nominal terms (intersection types); type systems for nominal programming languages.

That's all!

Questions ?