Nominal Rewriting and Types

Maribel Fernández

King's College London

Joint work with M.J. Gabbay and I. Mackie

Motivations

Rewrite rules can be used to define

- equational theories, and theorem provers;
- algebraic specifications of operators and data structures;
- operational semantics of programs;
- a theory of functions;
- a theory of processes;
- etc.

Motivations

Specifying binding operations — informal presentations:

- Operational semantics:

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

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

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:

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

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

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.

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:

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

- Logic equivalences:

$$P \text{ and } (\forall x.Q) \Leftrightarrow \forall x(P \text{ and } Q) \qquad (x \notin \text{fv}(P))$$

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

Formally: Rewrite Systems

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: Rewrite Systems

There are several alternatives.

- First-order rewrite systems.

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

$\Rightarrow$ No binders. (-)

Formally: Rewrite Systems

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. (-)

$\Rightarrow$ First-order matching: we need to 'specify' α-conversion. (-)

Formally: Rewrite Systems

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. (-)

  $\Rightarrow$ Simple notion of substitution. (+)

Formally: Rewrite Systems

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. (+)

- Algebraic λ-calculi: First-order rewriting + β-rule.

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 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:

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

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

  $\Rightarrow$ 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:
  $$\text{let } a = N \text{ in } M(a) \longrightarrow (\text{fun } a \to M(a))N$$
  - Terms with binders. $(+)$

  $\Rightarrow$ Implicit α-equivalence. $(+)$

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-equivalence. $(+)$

  $\Rightarrow$ 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:
  $$\text{let } a = N \text{ in } M(a) \longrightarrow (\text{fun } a \to M(a))N$$
  - Terms with binders. $(+)$
  - Implicit α-equivalence. $(+)$
  - We targeted α but now we have to deal with β too. (-)

  $\Rightarrow$ 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-equivalence. $(+)$
  - We targeted α but now we have to deal with β too. (-)
  - Substitution is a meta-operation using β. (-)

  $\Rightarrow$ 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-equivalence. $(+)$
  - We targeted α but now we have to deal with β too. (-)
  - Substitution is a meta-operation using β. (-)
  - Unification is undecidable in general. (-)

  $\Rightarrow$ Leaving name dependencies implicit is convenient (e.g. $\forall x.P$).

Nominal Rewriting

Inspired by the work on Nominal Logic (Pitts et al.)
Key ideas: Freshness conditions $a\#t$, name swapping $(a\ b) \cdot t$.
Example: β and η rules as NRS:

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

$\Rightarrow$ Terms with binders.

Nominal Rewriting

Inspired by the work on Nominal Logic (Pitts et al.)
Key ideas: Freshness conditions $a\#t$, name swapping $(a\ b) \cdot t$.
Example: β and η rules as NRS:

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

- Terms with binders.

$\Rightarrow$ Built-in α-equivalence.

Nominal Rewriting

Inspired by the work on Nominal Logic (Pitts et al.)
Key ideas: Freshness conditions $a\#t$, name swapping $(a\ b) \cdot t$.
Example: β and η rules as NRS:

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

- Terms with binders.
- Built-in α-equivalence.

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

Nominal Rewriting

Inspired by the work on Nominal Logic (Pitts et al.)
Key ideas: Freshness conditions $a\#t$, name swapping $(a\ b) \cdot t$.
Example: β and η rules as NRS:

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

- Terms with binders.
- Built-in α-equivalence.
- Simple notion of substitution (first order).

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

Nominal Rewriting

Inspired by the work on Nominal Logic (Pitts et al.)
Key ideas: Freshness conditions $a\#t$, name swapping $(a\ b) \cdot t$.
Example: β and η rules as NRS:

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

- Terms with binders.
- Built-in α-equivalence.
- Simple notion of substitution (first order).
- Dependencies of terms on names are implicit.

$\Rightarrow$ Easy to express conditions such as $a \notin \mathrm{fv}(M)$

Nominal Syntax

- 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 Syntax

- 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.

Nominal Syntax

- 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', $t[a \mapsto t']$

α-equivalence

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

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

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

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

where

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

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

α-equivalence

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

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

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

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

where

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

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

Freshness

Also defined by induction:

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

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

Nominal Matching [Urban, Pitts, Gabbay 2003]

- $s = t$ has solution (Δ, θ) if $\Delta \vdash s\theta \approx_\alpha t$

Nominal Matching [Urban, Pitts, Gabbay 2003]

- $s = t$ has solution (Δ, θ) if $\Delta \vdash s\theta \approx_\alpha t$
- Examples:
  $\lambda([a]X) = \lambda([b]b)$??
  $\lambda([a]X) = \lambda([b]X)$??

Nominal Matching [Urban, Pitts, Gabbay 2003]

- $s = t$ has solution (Δ, θ) if $\Delta \vdash s\theta \approx_\alpha t$
- Examples:
  $\lambda([a]X) = \lambda([b]b)$??
  $\lambda([a]X) = \lambda([b]X)$??
- Solutions: $(\emptyset, [X \mapsto a])$ and $(\{a\#X, b\#X\}, Id)$ resp.

Nominal Matching [Urban, Pitts, Gabbay 2003]

- $s = t$ has solution (Δ, θ) if $\Delta \vdash s\theta \approx_\alpha t$
- Examples:
  $\lambda([a]X) = \lambda([b]b)$??
  $\lambda([a]X) = \lambda([b]X)$??
- Solutions: $(\emptyset, [X \mapsto a])$ and $(\{a\#X, b\#X\}, Id)$ resp.
- Nominal matching is decidable, and linear in time [Calves, Fernandez 07].

Nominal Matching [Urban, Pitts, Gabbay 2003]

- $s = t$ has solution (Δ, θ) if $\Delta \vdash s\theta \approx_\alpha t$
- Examples:
  $\lambda([a]X) = \lambda([b]b)$??
  $\lambda([a]X) = \lambda([b]X)$??
- Solutions: $(\emptyset, [X \mapsto a])$ and $(\{a\#X, b\#X\}, Id)$ resp.
- Nominal matching is decidable, and
  linear in time [Calves, Fernandez 07].
- A solvable problem has a unique most general solution
  [Urban, Pitts, Gabbay 04].

Nominal Rewrite Rules

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

- Examples:

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

Nominal Rewrite Rules

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

- Examples:

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

- Equivariance: Rules are defined modulo permutative renamings of atoms.
  Equivariant nominal matching is exponential... BUT

Nominal Rewrite Rules

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

- Examples:

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

- Equivariance: Rules are defined modulo permutative renamings of atoms.
  Equivariant nominal matching is exponential... BUT
- if rules are CLOSED then it is linear. Intuitively, closed means no free atoms. The example above is closed.

Confluence Properties

Critical Pair Lemma:
If all critical pairs of a nominal rewrite system are joinable, then it is locally confluent. If the rules are closed then it is sufficient that non-trivial critical pairs be joinable.

Orthogonality:
If all the rules are closed, left-linear, and without superpositions nominal rewriting is confluent.

A polymorphic type system for nominal rewriting

Types built from

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

Types and type schemes:

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

Type declarations (arity):

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

E.g. *succ*: (Nat)Nat

Instantiation: $\sigma \leq \tau$ E.g. $\forall\alpha.\alpha \leq Nat$, $(\alpha)\alpha \leq (Nat)Nat$

Typing Rules

Typing judgement: $\Gamma; \Delta \vdash s : \tau$ where Γ is a typing context, Δ a freshness context, s a term and τ a type.

$$\frac{\sigma \leq \tau}{\Gamma, a : \sigma; \Delta \vdash a : \tau} \qquad \frac{\sigma \leq \tau \quad \Gamma; \Delta \vdash \pi \cdot X : \diamond}{\Gamma, X : \sigma; \Delta \vdash \pi \cdot X : \tau}$$

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

$$\frac{\Gamma; \Delta \vdash t : \tau' \quad f : \rho \leq (\tau')\tau}{\Gamma; \Delta \vdash f\ t : \tau}$$

$\Gamma; \Delta \vdash \pi \cdot X : \diamond$ holds if for any a such that $\pi \cdot a \neq a$, $\Delta \vdash a \# X$ or for some σ, $a : \sigma, \pi \cdot a : \sigma \in \Gamma$.

- Every term has a principal type.

Typing Rules

Typing judgement: $\Gamma; \Delta \vdash s : \tau$ where Γ is a typing context, Δ a freshness context, s a term and τ a type.

$$\frac{\sigma \leq \tau}{\Gamma, a : \sigma; \Delta \vdash a : \tau} \qquad \frac{\sigma \leq \tau \quad \Gamma; \Delta \vdash \pi \cdot X : \diamond}{\Gamma, X : \sigma; \Delta \vdash \pi \cdot X : \tau}$$

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

$$\frac{\Gamma; \Delta \vdash t : \tau' \quad f : \rho \leq (\tau')\tau}{\Gamma; \Delta \vdash f\ t : \tau}$$

$\Gamma; \Delta \vdash \pi \cdot X : \diamond$ holds if for any a such that $\pi \cdot a \neq a$, $\Delta \vdash a \# X$ or for some σ, $a : \sigma, \pi \cdot a : \sigma \in \Gamma$.

- Every term has a principal type.
- Type inference is decidable.

Typing Rules

Typing judgement: $\Gamma; \Delta \vdash s : \tau$ where Γ is a typing context, Δ a freshness context, s a term and τ a type.

$$\frac{\sigma \leq \tau}{\Gamma, a : \sigma; \Delta \vdash a : \tau} \qquad \frac{\sigma \leq \tau \quad \Gamma; \Delta \vdash \pi \cdot X : \diamond}{\Gamma, X : \sigma; \Delta \vdash \pi \cdot X : \tau}$$

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

$$\frac{\Gamma; \Delta \vdash t : \tau' \quad f : \rho \leq (\tau')\tau}{\Gamma; \Delta \vdash f\ t : \tau}$$

$\Gamma; \Delta \vdash \pi \cdot X : \diamond$ holds if for any a such that $\pi \cdot a \neq a$, $\Delta \vdash a \# X$ or for some σ, $a : \sigma, \pi \cdot a : \sigma \in \Gamma$.

- Every term has a principal type.
- Type inference is decidable.
- Typable rules preserve types.

Examples

$$
\begin{array}{rcl}
a : \forall\alpha.\alpha, X : \beta & \vdash & (a, X) : \beta \times \beta \\
 & \vdash & [a]a : [\alpha]\alpha \\
a : \beta & \vdash & [a]a : [\alpha]\alpha \\
a : \alpha, b : \alpha, X : \tau & \vdash & (a\ b) \cdot X : \tau \\
X : \tau; a\#X, b\#X & \vdash & (a\ b) \cdot X : \tau \\
X : \tau, a : \alpha, b : \alpha & \vdash & [a]((a\ b) \cdot X, b) : [\alpha](\tau \times \alpha)
\end{array}
$$

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.

Conclusion

- Nominal Terms: first-order syntax, with a notion of matching modulo α.

Conclusion

- Nominal Terms: first-order syntax, with a notion of matching modulo α.
- Higher-order substitutions are easy to define using freshness.

Conclusion

- Nominal Terms: first-order syntax, with a notion of matching modulo α.
- Higher-order substitutions are easy to define using freshness.
- Nominal matching is decidable; equivariant matching is linear with closed rules.

Conclusion

- Nominal Terms: first-order syntax, with a notion of matching modulo α.
- Higher-order substitutions are easy to define using freshness.
- Nominal matching is decidable; equivariant matching is linear with closed rules.
- Closed NRSs have the expressive power of higher-order rewriting.

Conclusion

- Nominal Terms: first-order syntax, with a notion of matching modulo α.
- Higher-order substitutions are easy to define using freshness.
- Nominal matching is decidable; equivariant matching is linear with closed rules.
- Closed NRSs have the expressive power of higher-order rewriting.
- Closed NRSs have the properties of first-order rewriting (critical pair, lemma, orthogonality).

Conclusion

- Nominal Terms: first-order syntax, with a notion of matching modulo α.
- Higher-order substitutions are easy to define using freshness.
- Nominal matching is decidable; equivariant matching is linear with closed rules.
- Closed NRSs have the expressive power of higher-order rewriting.
- Closed NRSs have the properties of first-order rewriting (critical pair, lemma, orthogonality).
- To model name generation, locality or scope constraints: add a new construct $И$ and a **scope** relation $a@t$.

Conclusion

- Nominal Terms: first-order syntax, with a notion of matching modulo α.
- Higher-order substitutions are easy to define using freshness.
- Nominal matching is decidable; equivariant matching is linear with closed rules.
- Closed NRSs have the expressive power of higher-order rewriting.
- Closed NRSs have the properties of first-order rewriting (critical pair, lemma, orthogonality).
- To model name generation, locality or scope constraints: add a new construct $И$ and a **scope** relation $a@t$.
- To model general constraints on the applicability of rules: generalise 'contexts' in rules with other predicates: e.g.'is-in', 'is-closed', etc.

Conclusion

- Nominal Terms: first-order syntax, with a notion of matching modulo α.
- Higher-order substitutions are easy to define using freshness.
- Nominal matching is decidable; equivariant matching is linear with closed rules.
- Closed NRSs have the expressive power of higher-order rewriting.
- Closed NRSs have the properties of first-order rewriting (critical pair, lemma, orthogonality).
- To model name generation, locality or scope constraints: add a new construct $И$ and a **scope** relation $a@t$.
- To model general constraints on the applicability of rules: generalise 'contexts' in rules with other predicates: e.g.'is-in', 'is-closed', etc.
- Types are not sufficient for termination: adapt results from algebraic λ calculi.

That's all!

Questions ?