Nominal Rewriting and Types

Maribel Fernández

King’s College London

Joint work with M.J. Gabbay and I. Mackie
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.
Specifying binding operations — informal presentations:

- Operational semantics:
  \[ \text{let } a = N \text{ in } M \rightarrow (\text{fun } a \rightarrow M)N \]

\(\alpha\)-conversion is implicit, but
\[ (\text{fun } a \rightarrow M) \not\equiv_{\alpha} (\text{fun } b \rightarrow M) \text{ since } a \text{ may occur in } M. \]
Motivations

Specifying binding operations — informal presentations:

- Operational semantics:

  \[
  \text{let } a = N \text{ in } M \rightarrow (\text{fun } a \rightarrow M)N
  \]

- $\beta$ and $\eta$-reductions in the $\lambda$-calculus:

  \[
  (\lambda x. M)N \rightarrow M[x/N]
  \]

  \[
  (\lambda x. Mx) \rightarrow M \quad (x \not\in \text{fv}(M))
  \]

$\alpha$-conversion is implicit, but

\[
(\text{fun } a \rightarrow M) \not\equiv_\alpha (\text{fun } b \rightarrow M) \quad \text{since } a \text{ may occur in } M.
\]
Motivations

Specifying binding operations — informal presentations:

- Operational semantics:

  \[
  \text{let } a = N \text{ in } M \rightarrow (\text{fun } a \rightarrow M)N
  \]

- \(\beta\) and \(\eta\)-reductions in the \(\lambda\)-calculus:

  \[
  (\lambda x. M)N \rightarrow M[x/N]
  \]

  \[
  (\lambda x. Mx) \rightarrow M \quad (x \not\in \text{fv}(M))
  \]

- \(\pi\)-calculus:

  \[
  P \mid \nu a. Q \rightarrow \nu a. (P \mid Q) \quad (a \not\in \text{fv}(P))
  \]

\(\alpha\)-conversion is implicit, but

\[
(\text{fun } a \rightarrow M) \not\equiv_{\alpha} (\text{fun } b \rightarrow M) \text{ since } a \text{ may occur in } M.
\]
Specifying binding operations — informal presentations:

- Operational semantics:
  
  $$\text{let } a = N \text{ in } M \rightarrow (\text{fun } a \rightarrow M)N$$

- $\beta$ and $\eta$-reductions in the $\lambda$-calculus:
  
  $$(\lambda x. M)N \rightarrow M[x/N]$$
  $$(\lambda x. Mx) \rightarrow M \quad (x \notin \text{fv}(M))$$

- $\pi$-calculus:
  
  $$P | \nu a. Q \rightarrow \nu a. (P | Q) \quad (a \notin \text{fv}(P))$$

- Logic equivalences:
  
  $$P \text{ and } (\forall x. Q) \iff \forall x (P \text{ and } Q) \quad (x \notin \text{fv}(P))$$

$\alpha$-conversion is implicit, but

$$(\text{fun } a \rightarrow M) \not\equiv_\alpha (\text{fun } b \rightarrow M) \text{ since } a \text{ may occur in } M.$$
There are several alternatives.

- **First-order rewrite systems.**

  \[
  \begin{align*}
  \text{append}(\text{nil}, x) & \rightarrow x \\
  \text{append}(\text{cons}(x, z), y) & \rightarrow \text{cons}(x, \text{append}(z, y))
  \end{align*}
  \]
There are several alternatives.

- First-order rewrite systems.

\[
\begin{align*}
append(nil, x) & \rightarrow x \\
append(cons(x, z), y) & \rightarrow cons(x, append(z, y))
\end{align*}
\]

\Rightarrow No binders. (-)
There are several alternatives.

- First-order rewrite systems.

\[
\begin{align*}
\text{append}(\text{nil}, x) & \rightarrow x \\
\text{append}(\text{cons}(x, z), y) & \rightarrow \text{cons}(x, \text{append}(z, y))
\end{align*}
\]

- No binders. (-)

$\Rightarrow$ First-order matching: we need to ’specify’ $\alpha$-conversion. (-)
Formally: Rewrite Systems

There are several alternatives.

• First-order rewrite systems.

  \[
  \text{append}(\text{nil}, x) \rightarrow x
  \\
  \text{append}(\text{cons}(x, z), y) \rightarrow \text{cons}(x, \text{append}(z, y))
  \]

• No binders. (-)
• First-order matching: we need to ’specify’ \(\alpha\)-conversion. (-)

  \[\Rightarrow \text{Simple notion of substitution. (+)}\]
There are several alternatives.

- First-order rewrite systems.
  
  $$\text{append}(\text{nil}, x) \rightarrow x$$
  $$\text{append}(\text{cons}(x, z), y) \rightarrow \text{cons}(x, \text{append}(z, y))$$

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

- Algebraic $\lambda$-calculi: First-order rewriting $+$ $\beta$-rule.
Higher-order rewrite systems (CRS, HRS, etc.)
\(\beta\)-rule:
\[
\text{app}(\text{lam}([a]Z(a)), Z') \rightarrow Z(Z')
\]

Then \(\text{app}(\text{lam}([a]f(a, g(a)), b) \rightarrow f(b, g(b))\)
using higher-order matching.
• Higher-order rewrite systems (CRS, HRS, etc.)
  \(\beta\)-rule:
  \[\text{app}(\text{lam}([a]Z(a)), Z') \rightarrow Z(Z')\]

Then \(\text{app}(\text{lam}([a]f(a, g(a)), b) \rightarrow f(b, g(b))\)
using higher-order matching.

• Higher-Order Abstract Syntax:
  \[
  \text{let } a = N \text{ in } M(a) \rightarrow (\text{fun } a \rightarrow M(a))N
  \]
Higher-order frameworks

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

  \( \beta \)-rule:

  \[
  \text{app(lam([a]Z(a)), Z')} \rightarrow Z(Z')
  \]

  Then \( \text{app(lam([a]f(a, g(a)), b) \rightarrow f(b, g(b))} \)
  using higher-order matching.

- Higher-Order Abstract Syntax:

  \[
  \text{let } a = N \text{ in } M(a) \rightarrow (\text{fun } a \rightarrow M(a))N
  \]

  \( \Rightarrow \) Terms with binders. (+)
• Higher-order rewrite systems (CRS, HRS, etc.)

$\beta$-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:

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

• Terms with binders. (+)

$\Rightarrow$ Implicit $\alpha$-equivalence. (+)
Higher-order frameworks

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

\[ \text{\(\beta\)-rule:} \]

\[ \text{app}(\lambda a Z(a), Z') \rightarrow Z(Z') \]

Then \(\text{app}(\lambda a f(a, g(a)), b) \rightarrow f(b, g(b))\)
using higher-order matching.

- Higher-Order Abstract Syntax:

\[ \text{let } a = N \text{ in } M(a) \rightarrow (\text{fun } a \rightarrow M(a))N \]

- Terms with binders. (+)
- Implicit \(\alpha\)-equivalence. (+)

\[ \Rightarrow \text{ We targeted } \alpha \text{ but now we have to deal with } \beta \text{ too. (-)} \]
Higher-order frameworks

- Higher-order rewrite systems (CRS, HRS, etc.)
  \( \beta \)-rule:
  \[
  \text{app(lam}([a]Z(a)), Z') \rightarrow Z(Z')
  \]
  Then \( \text{app(lam}([a]f(a, g(a)), b) \rightarrow f(b, g(b)) \)
  using higher-order matching.

- Higher-Order Abstract Syntax:
  \[
  \text{let } a = N \text{ in } M(a) \rightarrow (\text{fun } a \rightarrow M(a))N
  \]
  - Terms with binders. (+)
  - Implicit \( \alpha \)-equivalence. (+)
  - We targeted \( \alpha \) but now we have to deal with \( \beta \) too. (-)
  \( \Rightarrow \) Substitution is a meta-operation using \( \beta \). (-)
Higher-order frameworks

- Higher-order rewrite systems (CRS, HRS, etc.)
  \[ \beta \text{-rule:} \]
  \[
  \text{app(lam([a]Z(a)), Z')} \rightarrow Z(Z')
  \]
  Then \( \text{app(lam([a]f(a, g(a)), b) \rightarrow f(b, g(b))} \)
  using higher-order matching.

- Higher-Order Abstract Syntax:
  \[
  \text{let } a = N \text{ in } M(a) \rightarrow (\text{fun } a \rightarrow M(a))N
  \]

- Terms with binders. (+)
- Implicit \( \alpha \)-equivalence. (+)
- We targeted \( \alpha \) but now we have to deal with \( \beta \) too. (-)
- Substitution is a meta-operation using \( \beta \). (-)
  \[ \Rightarrow \text{Unification is undecidable in general.} \]
Higher-order frameworks

• Higher-order rewrite systems (CRS, HRS, etc.)
  \( \beta \)-rule:
  \[
  \text{app}(\text{lam}([a]Z(a)), Z') \rightarrow Z(Z')
  \]
  Then \( \text{app}(\text{lam}([a]f(a, g(a)), b) \rightarrow f(b, g(b)) \)
  using higher-order matching.

• Higher-Order Abstract Syntax:
  \[
  \text{let } a = N \text{ in } M(a) \rightarrow (\text{fun } a \rightarrow M(a))N
  \]

• Terms with binders. (+)
• Implicit \( \alpha \)-equivalence. (+)
• We targeted \( \alpha \) but now we have to deal with \( \beta \) too. (-)
• Substitution is a meta-operation using \( \beta \). (-)
• Unification is undecidable in general. (-)
\[\Rightarrow\] Leaving name dependencies implicit is convenient (e.g. \( \forall x.P \)).
Inspired by the work on Nominal Logic (Pitts et al.)
Key ideas: Freshness conditions $a \not\in t$, name swapping $(a \ b) \cdot t$.
Example: $\beta$ and $\eta$ rules as NRS:

\[
\begin{align*}
    \text{app}(\text{lam}([a]Z), Z') & \rightarrow \text{subst}([a]Z, Z') \\
    a \not\in M \vdash (\lambda([a]\text{app}(M, a)) & \rightarrow M
\end{align*}
\]

$\Rightarrow$ Terms with binders.
Inspired by the work on Nominal Logic (Pitts et al.)
Key ideas: Freshness conditions $a \not\in t$, name swapping $(a \ b) \cdot t$.
Example: $\beta$ and $\eta$ rules as NRS:

$$\text{app}(\lambda([a]Z), Z') \rightarrow \text{subst}([a]Z, Z')$$

$$a \not\in M \vdash (\lambda([a]\text{app}(M, a)) \rightarrow M$$

- Terms with binders.

$\Rightarrow$ Built-in $\alpha$-equivalence.
Inspired by the work on Nominal Logic (Pitts et al.)
Key ideas: Freshness conditions $a \not\in t$, name swapping $(a \ b) \cdot t$
Example: $\beta$ and $\eta$ rules as NRS:

$$app(lam([a]Z), Z') \rightarrow subst([a]Z, Z')$$
$$a \not\in M \vdash (\lambda([a]app(M, a)) \rightarrow M$$

- Terms with binders.
- Built-in $\alpha$-equivalence.

$\Rightarrow$ Simple notion of substitution (first order).
Inspired by the work on Nominal Logic (Pitts et al.)
Key ideas: Freshness conditions $a \not\in t$, name swapping $(a \ b) \cdot t$.
Example: $\beta$ and $\eta$ rules as NRS:

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

\[
a \not\in M \vdash (\lambda[a]\text{app}(M, a)) \rightarrow M
\]

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

$\Rightarrow$ Dependencies of terms on names are implicit.
Inspired by the work on Nominal Logic (Pitts et al.)

Key ideas: Freshness conditions $a \not\# t$, name swapping $(a \ b) \cdot t$.

Example: $\beta$ and $\eta$ rules as NRS:

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

$$a \not\# M \vdash (\lambda([a]app(M, a)) \rightarrow M$$

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

$\Rightarrow$ Easy to express conditions such as $a \notin \text{fv}(M)$
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 $\pi$ ($\text{Id}$ 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 $\pi$ ($Id$ 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 $X$. 
• 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 $\pi$ ($Id$ empty).

• Nominal Terms:

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

$Id \cdot X$ 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:

$a, (tt'), \lambda a.t, let \ a = t \ in \ t', letrec \ fa = t \ in \ t', t[a \mapsto t']$
\(\alpha\)-equivalence

We use freshness to avoid name capture.

\(a \# X\) means \(a \notin \text{fv}(X)\) when \(X\) is instantiated.

\[
\begin{align*}
\frac{a \approx_{\alpha} a}{\pi \cdot X \approx_{\alpha} \pi' \cdot X} \\
\frac{ds(\pi, \pi') \# X}{ds(\pi, \pi') \# X}
\end{align*}
\]

\[
\begin{align*}
(s_1, \ldots, s_n) &\approx_{\alpha} (t_1, \ldots, t_n) \\
\frac{s_1 \approx_{\alpha} t_1 \ldots s_n \approx_{\alpha} t_n}{(s_1, \ldots, s_n) \approx_{\alpha} (t_1, \ldots, t_n)} \\
s &\approx_{\alpha} t \\
fs &\approx_{\alpha} ft \\
\frac{s \approx_{\alpha} t}{fs \approx_{\alpha} ft}
\end{align*}
\]

\[
\begin{align*}
[a]s &\approx_{\alpha} [a]t \\
[a]s &\approx_{\alpha} [b]t
\end{align*}
\]

where

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

- \(a \# X, b \# X \vdash (a \ b) \cdot X \approx_{\alpha} X\)
\(\alpha\)-equivalence

We use freshness to avoid name capture.

\(a \# X\) means \(a \not\in \text{fv}(X)\) when \(X\) is instantiated.

\[
\begin{array}{c}
\frac{a \approx_{\alpha} a}{a \approx_{\alpha} a} \\
\frac{ds(\pi, \pi') \# X}{\pi \cdot X \approx_{\alpha} \pi' \cdot X}
\end{array}
\]

\[
\begin{array}{c}
s_1 \approx_{\alpha} t_1 \cdots s_n \approx_{\alpha} t_n \\
(s_1, \ldots, s_n) \approx_{\alpha} (t_1, \ldots, t_n)
\end{array}
\]

\[
\begin{array}{c}
s \approx_{\alpha} t \\
fs \approx_{\alpha} ft
\end{array}
\]

\[
\begin{array}{c}
s \approx_{\alpha} t \\
[a]s \approx_{\alpha} [a]t
\end{array}
\]

\[
\begin{array}{c}
fs \approx_{\alpha} ft \\
fs \approx_{\alpha} ft
\end{array}
\]

\[
\begin{array}{c}
[s]t \approx_{\alpha} [bt]
\end{array}
\]

where

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

\[
\begin{array}{c}
a \# X, b \# X \vdash (a \ b) \cdot X \approx_{\alpha} X
\end{array}
\]

\[
\begin{array}{c}
b \# X \vdash \lambda[a]X \approx_{\alpha} \lambda[b](a \ b) \cdot X
\end{array}
\]
Also defined by induction:

\[
\begin{align*}
& a \# b \\
& a \# [a]s \\
& \pi^{-1}(a) \# X \\
& a \# \pi \cdot X \\
& a \# s_1 \cdots a \# s_n \\
& a \# (s_1, \ldots, s_n) \\
& a \# s \\
& a \# fs \\
& a \# [b]s
\end{align*}
\]
• $s = t$ has solution $(\Delta, \theta)$ if $\Delta \vdash s\theta \approx_\alpha t$
• $s = t$ has solution $(\Delta, \theta)$ if $\Delta \vdash s\theta \approx_\alpha t$

• Examples:
  $\lambda([a]X) = \lambda([b]b)$
  $\lambda([a]X) = \lambda([b]X)$
• $s = t$ has solution $(\Delta, \theta)$ 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 $(\{a\#X, b\#X\}, \text{Id})$ resp.
Nominal Matching [Urban, Pitts, Gabbay 2003]

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

- Examples:
  \[
  \lambda([a]X) = \lambda([b]b) \quad \lambda([a]X) = \lambda([b]X) 
  \]

- Solutions: $(\emptyset, [X \mapsto a])$ and $(\{a\#X, b\#X\}, Id)$ resp.

- Nominal matching is decidable, and linear in time [Calves, Fernandez 07].
• $s = t$ has solution $(\Delta, \theta)$ if $\Delta \vdash s\theta \approx_\alpha t$

• Examples:
  $\lambda([a]X) = \lambda([b]b)$
  $\lambda([a]X) = \lambda([b]X)$

• Solutions: $(\emptyset, [X \rightarrow a])$ and $(\{a\#X, b\#X\}, I_d)$ 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

\[ \Delta \vdash l \rightarrow r \quad V(r) \cup V(\Delta) \subseteq V(l) \]

- **Examples:**
  
  \[
  (\lambda[a]X)Y \quad \rightarrow \quad X[a\mapsto Y] \\
  (XX')[a\mapsto Y] \quad \rightarrow \quad X[a\mapsto Y]X'[a\mapsto Y] \\
  a[a\mapsto X] \quad \rightarrow \quad X \\
  a\# Y \vdash Y[a\mapsto X] \quad \rightarrow \quad Y \\
  b\# Y \vdash (\lambda[b]X)[a\mapsto Y] \quad \rightarrow \quad \lambda[b](X[a\mapsto Y])
  \]
Nominal Rewrite Rules

\[ \Delta \vdash l \rightarrow r \quad V(r) \cup V(\Delta) \subseteq V(l) \]

- Examples:

\[
\begin{align*}
(\lambda[a]X)Y & \rightarrow X[a \mapsto Y] \\
(XX')[a \mapsto Y] & \rightarrow X[a \mapsto Y]X'[a \mapsto Y] \\
a[a \mapsto X] & \rightarrow X \\
a \# Y \vdash Y[a \mapsto X] & \rightarrow Y \\
b \# Y \vdash (\lambda[b]X)[a \mapsto Y] & \rightarrow \lambda[b](X[a \mapsto Y])
\end{align*}
\]

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

\[ \Delta \vdash l \rightarrow r \quad V(r) \cup V(\Delta) \subseteq V(l) \]

- Examples:

  \[
  (\lambda[a]X)Y \quad \rightarrow \quad X[a \mapsto Y] \\
  (XX')[a \mapsto Y] \quad \rightarrow \quad X[a \mapsto Y]X'[a \mapsto Y] \\
  a[a \mapsto X] \quad \rightarrow \quad X \\
  a \# Y \vdash \ Y[a \mapsto X] \quad \rightarrow \quad Y \\
  b \# Y \vdash \ (\lambda[b]X)[a \mapsto Y] \quad \rightarrow \quad \lambda[b](X[a \mapsto Y])
  \]

- 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.
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.
Types built from

- a set of base data sorts $\delta$ (e.g. Nat, Bool, Exp, ...)
- type variables $\alpha$, and
- type constructors $C$ (e.g. List, $\times$, $\to$, ...)

Types and type schemes:

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

Type declarations (arity):

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

E.g. $\text{succ}: (\text{Nat})\text{Nat}$

Instantiation: $\sigma \leq \tau$  
E.g. $\forall\alpha.\alpha \leq \text{Nat}$, $(\alpha)\alpha \leq (\text{Nat})\text{Nat}$
Typing Rules

Typing judgement: \( \Gamma; \Delta \vdash s : \tau \) where \( \Gamma \) is a typing context, \( \Delta \) a freshness context, \( s \) a term and \( \tau \) a type.

\[
\frac{\sigma \leq \tau}{\Gamma, a : \sigma; \Delta \vdash a : \tau}
\]

\[
\frac{\sigma \leq \tau}{\Gamma; \Delta \vdash \pi \cdot X : \diamond}
\]

\[
\frac{\Gamma, X : \sigma; \Delta \vdash \pi \cdot X : \tau}{\Gamma, X : \sigma; \Delta \vdash \pi \cdot X : \tau}
\]

\[
\frac{\Gamma, a : \tau; \Delta \vdash t : \tau'}{\Gamma; \Delta \vdash [a]t : [\tau]\tau'}
\]

\[
\frac{\Gamma; \Delta \vdash t_i : \tau_i \quad (1 \leq i \leq n)}{\Gamma; \Delta \vdash (t_1, \ldots, t_n) : \tau_1 \times \ldots \times \tau_n}
\]

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

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

- Every term has a principal type.

Maribel Fernández  Nominal Rewriting and Types
Typing Rules

Typing judgement: \( \Gamma; \Delta \vdash s : \tau \) where \( \Gamma \) is a typing context, \( \Delta \) a freshness context, \( s \) a term and \( \tau \) a type.

\[
\begin{align*}
\sigma &\leq \tau & \Gamma, a : \sigma; \Delta \vdash a : \tau
\end{align*}
\]

\[
\begin{align*}
\sigma &\leq \tau & \Gamma, X : \sigma; \Delta \vdash \pi \cdot X : \diamond
\end{align*}
\]

\[
\begin{align*}
\Gamma, a : \tau; \Delta \vdash t : \tau' & \Gamma; \Delta \vdash t_i : \tau_i \quad (1 \leq i \leq n)
\end{align*}
\]

\[
\begin{align*}
\Gamma; \Delta \vdash [a] t : [\tau] \tau' & \Gamma; \Delta \vdash (t_1, \ldots, t_n) : \tau_1 \times \ldots \times \tau_n
\end{align*}
\]

\[
\begin{align*}
\Gamma; \Delta \vdash t : \tau' & f : \rho \leq (\tau') \tau
\end{align*}
\]

\[
\begin{align*}
\Gamma; \Delta \vdash f t : \tau
\end{align*}
\]

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

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

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

- $\sigma \leq \tau$  
  \[
  \frac{}{\Gamma, a : \sigma; \Delta \vdash a : \tau}
  \]

- $\Gamma; \Delta \vdash \pi \cdot X : \Diamond$  
  \[
  \frac{}{\Gamma, X : \sigma; \Delta \vdash \pi \cdot X : \tau}
  \]

- $\Gamma, a : \tau; \Delta \vdash t : \tau'$  
  \[
  \frac{}{\Gamma; \Delta \vdash [a]t : [\tau]\tau'}
  \]

- $\Gamma; \Delta \vdash t_i : \tau_i$ (1 \leq i \leq n)  
  \[
  \frac{}{\Gamma; \Delta \vdash (t_1, \ldots, t_n) : \tau_1 \times \ldots \times \tau_n}
  \]

- $\Gamma, \Delta \vdash t : \tau'$  
  \[
  \frac{}{f : \rho \leq (\tau')\tau}
  \]

- $\Gamma; \Delta \vdash f \ t : \tau$

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

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

\[ a : \forall \alpha. \alpha, X : \beta \quad \vdash (a, X) : \beta \times \beta \]
\[ \vdash [a]a : [\alpha]\alpha \]
\[ a : \beta \quad \vdash [a]a : [\alpha]\alpha \]
\[ a : \alpha, b : \alpha, X : \tau \quad \vdash (a \ b) \cdot X : \tau \]
\[ X : \tau; a\#X, b\#X \quad \vdash (a \ b) \cdot X : \tau \]
\[ X : \tau, a : \alpha, b : \alpha \quad \vdash [a]((a \ b) \cdot X, b) : [\alpha](\tau \times \alpha) \]

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.
• Nominal Terms: first-order syntax, with a notion of matching modulo $\alpha$.
• Nominal Terms: first-order syntax, with a notion of matching modulo $\alpha$.
• Higher-order substitutions are easy to define using freshness.
• Nominal Terms: first-order syntax, with a notion of matching modulo $\alpha$.
• Higher-order substitutions are easy to define using freshness.
• Nominal matching is decidable; equivariant matching is linear with closed rules.
• Nominal Terms: first-order syntax, with a notion of matching modulo $\alpha$.
• 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.
• Nominal Terms: first-order syntax, with a notion of matching modulo $\alpha$.
• 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).
• Nominal Terms: first-order syntax, with a notion of matching modulo $\alpha$.
• 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 $\mathcal{N}$ and a scope relation $a@t$. 

Conclusion

• Nominal Terms: first-order syntax, with a notion of matching modulo $\alpha$.
• 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 $\forall$ 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 $\alpha$.
- 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 $\bar{N}$ 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 $\lambda$ calculi.
Questions ?