Curry-Style Types for Nominal Terms

Maribel Fernández Murdoch J. Gabbay

King's College London, Heriot-Watt University

Rewriting Calculus Workshop, October 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:

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

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

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

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

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.

Formally:

Alternatives:

- First-order rewrite systems.
  No binders, first-order matching: we need to 'specify' α-conversion. (-)
  But a simple notion of substitution. (+)
- Higher-order rewrite systems (CRS, HRS, ERS, etc.) using (restrictions of) higher-order matching, Higher-Order Abstract Syntax: let $a = N$ in $M(a) \longrightarrow (\text{fun } a \rightarrow M(a))N$
  Terms with binders and implicit α-conversion. (+)
  But we targeted α and end up having 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 Fresh ML.
Key ideas: Freshness conditions $a\#t$, name swapping $(a\ b) \cdot t$.
Example: β and η rules using nominal terms:

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

Terms with binders and 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.

Nominal Syntax (Untyped) [Urban,Pitts,Gabbay: TCS'04]

- 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 (Freshness)

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

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

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

α-equivalence

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

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

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

where

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

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

α-equivalence

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

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

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

where

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

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

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, ...)

Types and type schemes:

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

Type declarations (arity): $\rho ::= (\tau')\tau$
Instantiation relation: $\sigma \leq \tau$
Typing judgement: $\Gamma; \Delta \vdash s : \tau$ where Γ is a typing context, Δ a freshness context, s a term and τ a type.

Typing Rules

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

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

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

$\Gamma; \Delta \vdash \pi \cdot X : \diamond$ holds if for any a such that $\pi \cdot a \neq a$,

- either $\Delta \vdash a\#X$ and $\Delta \vdash \pi \cdot a\#X$,
- or for some σ it is the case that $a\colon \sigma \in \Gamma$ and $\pi \cdot a\colon \sigma \in \Gamma$.

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.

Principal Types

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

Principal Types

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

Principal Types

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

Nominal Rewriting

Nominal unification (and matching) is decidable [Urban, Pitts, Gabbay, TCS 04] and polynomial [TERMGRAPH 06].
Nominal Rewriting Rules:

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

Examples:

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

Typed Nominal Rewriting

Typed matching problem: $(\Phi; \nabla \vdash l) \stackrel{?}{\approx} (\Gamma; \Delta \vdash s)$
Solution: (S, θ) such that:

1. $X\theta \equiv X$ for $X \notin V(\Phi, \nabla, l)$, $\alpha S \equiv \alpha$ for $\alpha \notin TV(\Phi)$[1], $\Delta \vdash l\theta \approx_\alpha s$ and $\Delta \vdash \nabla\theta$.
2. $pt(\Phi; \nabla \vdash l) = (Id, \tau)$ and $pt(\Gamma; \Delta \vdash s) = (Id, \tau S)$; and for each $\Phi, \Phi'; \nabla \vdash \pi \cdot X : \phi'$ an essential typing in $\Phi; \nabla \vdash l : \tau$, we have $\Gamma, (\Phi' S); \Delta \vdash (\pi \cdot X)\theta : \phi' S$.

[1] 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 $\alpha \in TV(\Gamma)$.

Typed Rules

$\Phi; \nabla \vdash l \to r : \tau$ 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. The essential typings of $\Phi; \nabla \vdash r : \tau$ are a subset of the essential typings of $\Phi; \nabla \vdash l : \tau$.

Typed Rewriting

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

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

Subject Reduction: Let $R \equiv \Phi;\nabla \vdash l \to r : \tau$. If $\Gamma;\Delta \vdash s : \mu$ and $\Gamma;\Delta \vdash s \xrightarrow{R} t$ then $\Gamma;\Delta \vdash t : \mu$.

Applications

A (typed!) specification 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:

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

Applications

Surjective pairing:
Consider $\mathit{fst} : (\alpha \times \beta)\alpha$ and $\mathit{snd} : (\alpha \times \beta)\beta$.
Typable rewrite rules for projections and surjective pairing:

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

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

Conclusion

- Nominal Terms: first-order syntax, with a notion of matching modulo α (decidable, polynomial).
- Type system: Typing is decidable and there are principal types.
- α-equivalence preserves types.
- Rewriting with typed rewrite rules preserves types.
- Future work: denotational semantics for nominal terms; normalisation properties of nominal terms (intersection types).