Nominal Techniques
for the Specification of Languages with Binders

Maribel Fernández

King's College London

Online Logic Seminar
2nd February 2023

Overview:

- Specifying binders: α-equivalence and meta-variables
- Nominal Logic
- Nominal terms: unification and matching modulo α
- Nominal rewriting
- Examples

References

1. A. Pitts. Nominal Logic. Information and Computation 183, 165–193, 2003.
2. C. Urban, A. Pitts, M.J. Gabbay. Nominal Unification. Theoretical Computer Science 323, pages 473-497, 2004.
3. M. Fernández, M.J. Gabbay. Nominal Rewriting. Information and Computation 205, pages 917-965, 2007.
4. C. Calvès, M. Fernández. Matching and Alpha-Equivalence Check for Nominal Terms. J. Computer and System Sciences, 2010.
5. E. Fairweather, M. Fernández. Typed Nominal Rewriting. ACM Transactions on Computational Logic, 2018.
6. M. Ayala-Rincón, W. de Carvalho-Segundo, M. Fernández, D. Nantes-Sobrinho, A. Rocha Oliveira. A Formalisation of Nominal Alpha-Equivalence with A, C and AC Function Symbols. Theoretical Computer Science, 2019.

Binding operators: some informal examples

- Operational semantics:

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

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

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

- π-calculus:

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

- Logic equivalences:

$$P \wedge (\forall x.Q) \Leftrightarrow \forall x.(P \wedge Q) \qquad (x \notin \mathrm{fv}(P))$$

Binding operators - α-equivalence and Metavariables

Terms are defined modulo renaming of bound variables, i.e., α-equivalence.

Example:

$$\forall x.P =_\alpha \forall y.P\{x \mapsto y\}$$

for any fresh variable y

How can we formally specify and reason with binding operators? There are several alternatives.

First-order frameworks

Encode α-equivalence:

- Example: λ-calculus using De Bruijn's indices with "lift" and "shift" operators to encode non-capturing substitution
- We need to 'implement' α-equivalence from scratch (-)
- Simple (first-order) (+)
- Efficient matching and unification algorithms (+)
- No metavariables (-)

- Logical frameworks based on Higher-Order Abstract Syntax work modulo α-equivalence (λ-calculus as metalanguage).

$$\forall(\lambda x.P(x))$$

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

using (a restriction of) *higher-order matching*.

Higher-order frameworks

- The syntax includes binders (+)
- Implicit α-equivalence (+)
- We targeted α but now we have to deal with β too (-)
- Unification is undecidable in general (-)

Nominal Approach

Nominal Logic [Pitts 2003]: a sorted first-order logic theory

Key ideas: Names (which can be swapped), abstraction, freshness.

Semantics given by nominal sets.

Nominal Logic Axioms

$$(a\ a)x = x \qquad (S1)$$

$$(a\ a')(a\ a')x = x \qquad (S2)$$

$$(a\ a')a = a' \qquad (S3)$$

Nominal Logic Axioms

$$(a\ a)x = x \qquad (S1)$$
$$(a\ a')(a\ a')x = x \qquad (S2)$$
$$(a\ a')a = a' \qquad (S3)$$
$$(a\ a')(b\ b')x = ((a\ a')b\ (a\ a')b')(a\ a')x \qquad (E1)$$
$$b \# x \Rightarrow (a\ a')b \# (a\ a')x \qquad (E2)$$
$$(a\ a')f(\vec{x}) = f((a\ a')\vec{x}) \qquad (E3)$$
$$p(\vec{x}) \Rightarrow p((a\ a')\vec{x}) \qquad (E4)$$
$$(b\ b')[a]x = [(b\ b')a](b\ b')x \qquad (E5)$$

Nominal Logic Axioms

$$
\begin{aligned}
&(a\ a)x = x && (S1)\\
&(a\ a')(a\ a')x = x && (S2)\\
&(a\ a')a = a' && (S3)\\
&(a\ a')(b\ b')x = ((a\ a')b\ (a\ a')b')(a\ a')x && (E1)\\
&b \# x \Rightarrow (a\ a')b \# (a\ a')x && (E2)\\
&(a\ a')f(\vec{x}) = f((a\ a')\vec{x}) && (E3)\\
&p(\vec{x}) \Rightarrow p((a\ a')\vec{x}) && (E4)\\
&(b\ b')[a]x = [(b\ b')a](b\ b')x && (E5)\\
&a \# x \wedge a' \# x \Rightarrow (a\ a')x = x && (F1)\\
&a \# a' \iff a \neq a' && (F2)\\
&\forall a\colon ns, a'\colon ns'.\ a \# a' \qquad (ns \neq ns') && (F3)\\
&\forall \vec{x}.\exists a.\ a \# \vec{x} && (F4)
\end{aligned}
$$

Nominal Logic Axioms

$$(a\ a)x = x \qquad (S1)$$
$$(a\ a')(a\ a')x = x \qquad (S2)$$
$$(a\ a')a = a' \qquad (S3)$$
$$(a\ a')(b\ b')x = ((a\ a')b\ (a\ a')b')(a\ a')x \qquad (E1)$$
$$b \# x \Rightarrow (a\ a')b \# (a\ a')x \qquad (E2)$$
$$(a\ a')f(\vec{x}) = f((a\ a')\vec{x}) \qquad (E3)$$
$$p(\vec{x}) \Rightarrow p((a\ a')\vec{x}) \qquad (E4)$$
$$(b\ b')[a]x = [(b\ b')a](b\ b')x \qquad (E5)$$
$$a \# x \wedge a' \# x \Rightarrow (a\ a')x = x \qquad (F1)$$
$$a \# a' \iff a \neq a' \qquad (F2)$$
$$\forall a : ns, a' : ns'.\ a \# a' \qquad (ns \neq ns') \qquad (F3)$$
$$\forall \vec{x}. \exists a.\ a \# \vec{x} \qquad (F4)$$
$$[a]x = [a']x' \iff (a = a' \wedge x = x') \vee (a \# x' \wedge (a\ a')x = x') \qquad (A1)$$
$$\forall x : [ns]s. \exists a : ns, y : s.\ x = [a]y \qquad (A2)$$

Nominal Logic Axioms

$$(a\ a)x = x \qquad (S1)$$
$$(a\ a')(a\ a')x = x \qquad (S2)$$
$$(a\ a')a = a' \qquad (S3)$$
$$(a\ a')(b\ b')x = ((a\ a')b\ (a\ a')b')(a\ a')x \qquad (E1)$$
$$b \# x \Rightarrow (a\ a')b \# (a\ a')x \qquad (E2)$$
$$(a\ a')f(\vec{x}) = f((a\ a')\vec{x}) \qquad (E3)$$
$$p(\vec{x}) \Rightarrow p((a\ a')\vec{x}) \qquad (E4)$$
$$(b\ b')[a]x = [(b\ b')a](b\ b')x \qquad (E5)$$
$$a \# x \wedge a' \# x \Rightarrow (a\ a')x = x \qquad (F1)$$
$$a \# a' \iff a \neq a' \qquad (F2)$$
$$\forall a\colon ns, a'\colon ns'.\ a \# a' \qquad (ns \neq ns') \qquad (F3)$$
$$\forall \vec{x}.\exists a.\ a \# \vec{x} \qquad (F4)$$
$$[a]x = [a']x' \iff (a = a' \wedge x = x') \vee (a \# x' \wedge (a\ a')x = x') \qquad (A1)$$
$$\forall x\colon [ns]s.\exists a\colon ns, y\colon s.\ x = [a]y \qquad (A2)$$
$$\forall \vec{x}.(\text{И}a.\phi \iff \exists a.a \# \vec{x} \wedge \phi) \qquad (FV(\text{И}a.\phi) \subseteq \vec{x}) \qquad (Q)$$

Nominal Languages

Freshness conditions $a\#t$, name swapping $(a\ b) \cdot t$, abstraction [a]t

- Terms with binders
- Built-in α-equivalence
- Simple notion of substitution (first order)
- Efficient matching and unification algorithms
- Dependencies of terms on names are implicit

Nominal Syntax [Urban, Pitts, Gabbay 2004]

- Variables: $M, N, X, Y, \ldots$
  Names: $a, b, \ldots$
  Function symbols (term formers): $f, g \ldots$

Nominal Syntax [Urban, Pitts, Gabbay 2004]

- Variables: $M, N, X, Y, \ldots$
  Names: $a, b, \ldots$
  Function symbols (term formers): $f, g \ldots$
- Nominal Terms:

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

π is a permutation: finite bijection on names, represented as a list of swappings, e.g., $(a\ b)(c\ d)$, Id (empty list).
$\pi \cdot t$: π acts on t, permutes names, suspends on variables.
$(a\ b) \cdot a = b$, $(a\ b) \cdot b = a$, $(a\ b) \cdot c = c$
$Id \cdot X$ written as X.

Nominal Syntax [Urban, Pitts, Gabbay 2004]

- Variables: $M, N, X, Y, \ldots$
  Names: $a, b, \ldots$
  Function symbols (term formers): $f, g \ldots$
- Nominal Terms:

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

  π is a permutation: finite bijection on names, represented as a list of swappings, e.g., $(a\ b)(c\ d)$, Id (empty list).
  $\pi \cdot t$: π acts on t, permutes names, suspends on variables.
  $(a\ b) \cdot a = b$, $(a\ b) \cdot b = a$, $(a\ b) \cdot c = c$
  $Id \cdot X$ 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 $f\ a = t$ 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.

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

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

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

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.

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

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

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

where

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

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

Freshness

Also defined by induction:

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

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

Nominal Rewriting Rules:

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

Example

Beta-reduction in the Lambda-calculus:

$$\begin{array}{llllll}
Beta & & (\lambda[a]X)Y & \to & X[a \mapsto Y] \\
\sigma_a & & a[a \mapsto Y] & \to & Y \\
\sigma_{app} & & (XX')[a \mapsto Y] & \to & X[a \mapsto Y]X'[a \mapsto Y] \\
\sigma_\epsilon & a\#Y \vdash & Y[a \mapsto X] & \to & Y \\
\sigma_\lambda & b\#Y \vdash & (\lambda[b]X)[a \mapsto Y] & \to & \lambda[b](X[a \mapsto Y])
\end{array}$$

Rewriting steps: $(\lambda[c]c)Z \to c[c \mapsto Z] \to Z$

Rewriting relation generated by $R = \nabla \vdash l \to r$: $\Delta \vdash s \stackrel{R}{\to} t$

s **rewrites with** R **to** t **in the context** Δ when:

1. $s \equiv C[s']$ such that θ solves $(\nabla \vdash l) \; {}_{?}{\approx} \; (\Delta \vdash s')$
2. $\Delta \vdash C[r\theta] \approx_\alpha t$.

Example: Prenex Normal Forms

$$
\begin{array}{lcl}
a\#P & \vdash & P \wedge \forall[a]Q \rightarrow \forall[a](P \wedge Q) \\
a\#P & \vdash & (\forall[a]Q) \wedge P \rightarrow \forall[a](Q \wedge P) \\
a\#P & \vdash & P \vee \forall[a]Q \rightarrow \forall[a](P \vee Q) \\
a\#P & \vdash & (\forall[a]Q) \vee P \rightarrow \forall[a](Q \vee P) \\
a\#P & \vdash & P \wedge \exists[a]Q \rightarrow \exists[a](P \wedge Q) \\
a\#P & \vdash & (\exists[a]Q) \wedge P \rightarrow \exists[a](Q \wedge P) \\
a\#P & \vdash & P \vee \exists[a]Q \rightarrow \exists[a](P \vee Q) \\
a\#P & \vdash & (\exists[a]Q) \vee P \rightarrow \exists[a](Q \vee P) \\
 & \vdash & \neg(\exists[a]Q) \rightarrow \forall[a]\neg Q \\
 & \vdash & \neg(\forall[a]Q) \rightarrow \exists[a]\neg Q
\end{array}
$$

To implement rewriting (functional/logic programming) we need a matching/unification algorithm.

Recall:

- efficient algorithms (linear time) for first-order terms
- We need more powerful algorithms: α-equivalence
- Higher-order unification is undecidable

Nominal terms have good computational properties:

- Unification is decidable and unitary
- Efficient algorithms: α-equivalence, matching, unification

$\Longrightarrow$ Programming languages (Alpha-Prolog, FreshML)
$\Longrightarrow$ Nominal Rewriting

Checking α-equivalence

Idea: The α-equivalence derivation rules become simplification rules

$$
\begin{array}{rcll}
a\#b, Pr & \Longrightarrow & Pr & \\
a\#fs, Pr & \Longrightarrow & a\#s, Pr & \\
a\#(s_1, \ldots, s_n), Pr & \Longrightarrow & a\#s_1, \ldots, a\#s_n, Pr & \\
a\#[b]s, Pr & \Longrightarrow & a\#s, Pr & \\
a\#[a]s, Pr & \Longrightarrow & Pr & \\
a\#\pi \cdot X, Pr & \Longrightarrow & \pi^{-1} \cdot a\#X, Pr & \quad \pi \not\equiv Id
\end{array}
$$

$$
\begin{array}{rcl}
a \approx_\alpha a, Pr & \Longrightarrow & Pr \\
(l_1, \ldots, l_n) \approx_\alpha (s_1, \ldots, s_n), Pr & \Longrightarrow & l_1 \approx_\alpha s_1, \ldots, l_n \approx_\alpha s_n, Pr \\
fl \approx_\alpha fs, Pr & \Longrightarrow & l \approx_\alpha s, Pr \\
[a]l \approx_\alpha [a]s, Pr & \Longrightarrow & l \approx_\alpha s, Pr \\
[b]l \approx_\alpha [a]s, Pr & \Longrightarrow & (a\ b) \cdot l \approx_\alpha s, a\#l, Pr \\
\pi \cdot X \approx_\alpha \pi' \cdot X, Pr & \Longrightarrow & ds(\pi, \pi')\#X, Pr
\end{array}
$$

Solving Equations [Urban, Pitts, Gabbay 2003]

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

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

  Nominal Matching: $s = t$ has solution (Δ, θ) if

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

  (t ground or variables disjoint from s)

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

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

  Nominal Matching: $s = t$ has solution (Δ, θ) if

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

  (t ground or variables disjoint from s)

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

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

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

  Nominal Matching: $s = t$ has solution (Δ, θ) if

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

  (t ground or variables disjoint from s)

- 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

- Nominal matching is decidable [Urban, Pitts, Gabbay 2003]
  A solvable problem Pr has a unique most general solution: (Γ, θ) such that $\Gamma \vdash Pr\theta$.
- Nominal matching algorithm: add an *instantiation rule*:

$$\pi \cdot X \approx_\alpha u, Pr \quad \Longrightarrow^{X \mapsto \pi^{-1} \cdot u} \quad Pr[X \mapsto \pi^{-1} \cdot u]$$

No occur-checks needed (left-hand side variables distinct from right-hand side variables).

Complexity

Alpha-equivalence check: linear if right-hand sides of constraints are ground. Otherwise, log-linear.

Matching: quadratic in the non-ground case

Complexity - Summary

Case	Alpha-equivalence	Matching
Ground	linear	linear
Non-ground and linear	log-linear	log-linear
Non-ground and non-linear	log-linear	quadratic

Remark:
The representation using higher-order abstract syntax does saturate the variables (they have to be applied to the set of atoms they can capture).
Conjecture: the algorithms are linear wrt HOAS also in the non-ground case.

For more details on the implementation see [4],
see [6] for formalisations in Coq and PVS

Back to Nominal Rewriting

Let $R = \nabla \vdash l \to r$ where $V(l) \cap V(s) = \emptyset$

s **rewrites with** R **to** t **in the context** Δ, written $\Delta \vdash s \xrightarrow{R} t$, when:

1. $s \equiv C[s']$ such that θ solves $(\nabla \vdash l) \;{}_?{\approx}\; (\Delta \vdash s')$
2. $\Delta \vdash C[r\theta] \approx_\alpha t$.

- To define the reduction relation generated by nominal rewriting rules we use nominal matching.

Let $R = \nabla \vdash l \to r$ where $V(l) \cap V(s) = \emptyset$

s **rewrites with** R **to** t **in the context** Δ, written $\Delta \vdash s \xrightarrow{R} t$, when:

1. $s \equiv C[s']$ such that θ solves $(\nabla \vdash l) \;{}_?{\approx}\; (\Delta \vdash s')$
2. $\Delta \vdash C[r\theta] \approx_\alpha t$.

- To define the reduction relation generated by nominal rewriting rules we use nominal matching.
- $(\nabla \vdash l) \;{}_?{\approx}\; (\Delta \vdash s')$ if $\nabla, l \approx_\alpha s'$ has solution (Δ', θ), that is, $\Delta' \vdash \nabla\theta, l\theta \approx_\alpha s'$ and $\Delta \vdash \Delta'$

Equivariance: Rules defined modulo permutative renamings of atoms.

Beta-reduction in the Lambda-calculus:

$$
\begin{array}{lrlcl}
Beta & & (\lambda[a]X)Y & \to & X[a \mapsto Y] \\
\sigma_a & & a[a \mapsto Y] & \to & Y \\
\sigma_{app} & & (XX')[a \mapsto Y] & \to & X[a \mapsto Y]X'[a \mapsto Y] \\
\sigma_\epsilon & a\#Y \vdash & Y[a \mapsto X] & \to & Y \\
\sigma_\lambda & b\#Y \vdash & (\lambda[b]X)[a \mapsto Y] & \to & \lambda[b](X[a \mapsto Y])
\end{array}
$$

Nominal Matching vs. Equivariant Matching

- Nominal matching is efficient.

Nominal Matching vs. Equivariant Matching

- Nominal matching is efficient.
- Equivariant nominal matching is exponential... BUT

Nominal Matching vs. Equivariant Matching

- Nominal matching is efficient.
- Equivariant nominal matching is exponential... BUT
- if rules are CLOSED then nominal matching is sufficient.
  Intuitively, closed means no free atoms.
  The rules in the examples above are closed.

$R \equiv \nabla \ \vdash \ l \to r$ is **closed** when

$$(\nabla' \ \vdash \ (l', r'))\ {}_{?}{\approx} \ (\nabla, A(R')\#V(R) \ \vdash \ (l, r))$$

has a solution σ (where R' is freshened with respect to R).

Given $R \equiv \nabla \ \vdash \ l \to r$ and $\Delta \ \vdash \ s$ a term-in-context we write

$$\Delta \ \vdash \ s \stackrel{R}{\to}_c t \quad \text{when} \quad \Delta, A(R')\#V(\Delta, s) \ \vdash \ s \stackrel{R'}{\to} t$$

and call this **closed rewriting**.

Examples

The following rules are not closed:

$$g(a) \to a$$

$$[a]X \to X$$

Why?

The following rule is closed:

$$a\#X \ \vdash \ [a]X \quad \to \quad X$$

Why?

Examples

Closed rules that define **capture-avoiding substitution** in the lambda calculus:
(explicit) substitutions, $subst([x]M, N)$ abbreviated $M[x \mapsto N]$.

$$
\begin{array}{lcrcl}
(\text{Beta}) & & (\lambda[a]X)X' & \to & X[a \mapsto X'] \\
(\sigma_{app}) & & (XX')[a \mapsto Y] & \to & X[a \mapsto Y]X'[a \mapsto Y] \\
(\sigma_a) & & a[a \mapsto X] & \to & X \\
(\sigma_\epsilon) & a\#Y \ \vdash & Y[a \mapsto X] & \to & Y \\
(\sigma_\lambda) & b\#Y \ \vdash & (\lambda[b]X)[a \mapsto Y] & \to & \lambda[b](X[a \mapsto Y])
\end{array}
$$

Properties of Closed Rewriting

Closed Nominal Rewriting:

- works uniformly in α equivalence classes of terms.
- is expressive: can encode Combinatory Reduction Systems.
- is efficient: linear matching.
- inherits confluence conditions from first order rewriting.

Types

So far, we have discussed untyped nominal terms.

There are also typed versions

- many-sorted
- Simply typed — Church-style and Curry-style
- Polymorphic Curry-style systems
- Intersection type assignment systems
- Dependently typed systems

Given two nominal terms s and t and an equational theory E.
Question: is there a substitution σ and a freshness context ∇ such that $\nabla \vdash s\sigma \approx_{\alpha,E} t\sigma$?

Nominal E-Unification

Given two nominal terms s and t and an equational theory E.
Question: is there a substitution σ and a freshness context ∇ such that $\nabla \vdash s\sigma \approx_{\alpha,E} t\sigma$?

Interference: Commutative Symbols, e.g., OR, $+$

$(c\ d) \cdot X \approx^?_{\alpha,C} X$ has infinite principal solutions:
$X \mapsto c + d, X \mapsto f(c + d), X \mapsto [e]c + [e]d, \ldots$

Nominal C-Unification Procedure [Ayala-Rincón et al.]:

1. Simplification phase:
   Build a derivation tree (branching for C symbols)
2. Solve fixed point constraints $X \approx_{\alpha,C} \pi \cdot X$

Nominal C-Unification Procedure [Ayala-Rincón et al.]:

1. Simplification phase:
   Build a derivation tree (branching for C symbols)
2. Solve fixed point constraints $X \approx_{\alpha,C} \pi \cdot X$

First-order C-unification and nominal unification are finitary.

Nominal C-Unification Procedure [Ayala-Rincón et al.]:

1. Simplification phase:
   Build a derivation tree (branching for C symbols)
2. Solve fixed point constraints $X \approx_{\alpha,C} \pi \cdot X$

First-order C-unification and nominal unification are finitary.
Nominal C-unification is NOT, if we represent solutions using substitutions and freshness contexts.

Nominal C-Unification Procedure [Ayala-Rincón et al.]:

1. Simplification phase:
   Build a derivation tree (branching for C symbols)
2. Solve fixed point constraints $X \approx_{\alpha,C} \pi \cdot X$

First-order C-unification and nominal unification are finitary.
Nominal C-unification is NOT, if we represent solutions using substitutions and freshness contexts.

Alternative representation: fixed-point constraints instead of freshness constraints:
$\pi \curlywedge x \Leftrightarrow \pi \cdot x = x$

Using fixed-point constraints instead of freshness constraints, nominal C-unification is finitary.

Conclusion

- NRSs are first-order systems with built-in α-equivalence.
- Closed NRSs $\Leftrightarrow$ higher-order rewriting systems
  Capture-avoiding atom substitution easy to define. If included as primitive unification becomes undecidable
  See Jesus Dominguez's PhD thesis.
- Hindley-Milner style types [4]: principal types, α-equivalence preserves types. Sufficient conditions for Subject Reduction.
- Nominal unification is quadratic (unknown lower bound) [Levy&Villaret, Calvès & F.]
- Nominal matching is linear, equivariant matching is linear with closed rules.

Conclusions

- Applications: functional and logic programming languages, theorem provers, model checkers
- Implementations/formalisations:
  by Elliot Fairweather
  Nominal Datatypes Package for Haskell (Jamie Gabbay):
  `https://github.com/bellissimogiorno/nominal`
  Nominal Project, University of Brasilia:
  `http://nominal.cic.unb.br`
  alpha-Prolog (James Cheney, Christian Urban):
  `https://homepages.inf.ed.ac.uk/jcheney/programs/aprolog/`
  Nominal Isabelle (Christian Urban)