Nominal Techniques
for the Specification of Languages with Binders

Maribel Fernández
King’s College London

Online Logic Seminar
2nd February 2023
Overview:

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


Binding operators: some informal examples

- Operational semantics:

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

- \(\beta\) 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:

  \[
  P \land (\forall x. Q) \iff \forall x.(P \land Q) \quad (x \notin \text{fv}(P))
  \]
Terms are defined **modulo renaming of bound variables**, i.e., \(\alpha\)-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.
Encode $\alpha$-equivalence:

- Example: $\lambda$-calculus using De Bruijn’s indices with “lift” and “shift” operators to encode non-capturing substitution
- We need to ‘implement’ $\alpha$-equivalence from scratch (-)
- Simple (first-order) (+)
- Efficient matching and unification algorithms (+)
- No metavariables (-)
• Logical frameworks based on Higher-Order Abstract Syntax work modulo $\alpha$-equivalence ($\lambda$-calculus as metalanguage).

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

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

using (a restriction of) higher-order matching.
Higher-order frameworks

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

\[(a \ a)x = x\]  \hspace{1cm} (S1)
\[(a \ a')(a \ a')x = x\]  \hspace{1cm} (S2)
\[(a \ a')a = a'\]  \hspace{1cm} (S3)
Nominal Logic Axioms

\( (a \ a)x = x \) \hfill (S1)
\( (a \ a')(a \ a')x = x \) \hfill (S2)
\( (a \ a')a = a' \) \hfill (S3)
\( (a \ a')(b \ b')x = ((a \ a')b (a \ a')b')(a \ a')x \) \hfill (E1)
\( b \not\in x \Rightarrow (a \ a')b \not\in (a \ a')x \) \hfill (E2)
\( (a \ a')f(\overline{x}) = f((a \ a')\overline{x}) \) \hfill (E3)
\( p(\overline{x}) \Rightarrow p((a \ a')\overline{x}) \) \hfill (E4)
\( (b \ b')[a]x = [(b \ b')a](b \ b')x \) \hfill (E5)
Nominal Logic Axioms

\[(a a)x = x\]  \hspace{1cm} (S1)

\[(a a')(a a')x = x\]  \hspace{1cm} (S2)

\[(a a')a = a'\]  \hspace{1cm} (S3)

\[(a a')(b b')x = ((a a')b (a a')b')(a a')x\]  \hspace{1cm} (E1)

\[b \# x \Rightarrow (a a')b \# (a a')x\]  \hspace{1cm} (E2)

\[(a a')f(x) = f((a a')x)\]  \hspace{1cm} (E3)

\[p(x) \Rightarrow p((a a')x)\]  \hspace{1cm} (E4)

\[(b b')[a]x = [(b b')a](b b')x\]  \hspace{1cm} (E5)

\[a \# x \land a' \# x \Rightarrow (a a')x = x\]  \hspace{1cm} (F1)

\[a \# a' \iff a \neq a'\]  \hspace{1cm} (F2)

\[\forall a: ns, a': ns'. a \# a' \quad (ns \neq ns')\]  \hspace{1cm} (F3)

\[\forall x. \exists a. a \# x\]  \hspace{1cm} (F4)
Nominal Logic Axioms

\[(a \ a)x = x\]  \hspace{1cm} (S1)

\[(a \ a')(a \ a')x = x\]  \hspace{1cm} (S2)

\[(a \ a')a = a'\]  \hspace{1cm} (S3)

\[(a \ a')(b \ b')x = ((a \ a')b (a \ a')b')(a \ a')x\]  \hspace{1cm} (E1)

\[b \# x \Rightarrow (a \ a')b \# (a \ a')x\]  \hspace{1cm} (E2)

\[(a \ a')f(x) = f((a \ a')x)\]  \hspace{1cm} (E3)

\[p(x) \Rightarrow p((a \ a')x)\]  \hspace{1cm} (E4)

\[(b \ b')[a]x = [(b \ b')a](b \ b')x\]  \hspace{1cm} (E5)

\[a \# x \land a' \# x \Rightarrow (a \ a')x = x\]  \hspace{1cm} (F1)

\[a \# a' \iff a = a'\]  \hspace{1cm} (F2)

\[\forall a: ns, a': ns'. a \# a' \quad (ns \neq ns')\]  \hspace{1cm} (F3)

\[\forall \bar{x}. \exists a. a \# \bar{x}\]  \hspace{1cm} (F4)

\[[a]x = [a']x' \iff (a = a' \land x = x') \lor (a \# x' \land (a \ a')x = x')\]  \hspace{1cm} (A1)

\[\forall x: [ns]s. \exists a: ns, y: s. x = [a]y\]  \hspace{1cm} (A2)
Nominal Logic Axioms

\[(a \ a)x = x\]  \(\text{(S1)}\)
\[(a \ a')(a \ a')x = x\]  \(\text{(S2)}\)
\[(a \ a')a = a'\]  \(\text{(S3)}\)
\[(a \ a')(b \ b')x = ((a \ a')b \ (a \ a')b')(a \ a')x\]  \(\text{(E1)}\)
\[b \# x \Rightarrow (a \ a')b \# (a \ a')x\]  \(\text{(E2)}\)
\[(a \ a')f(\vec{x}) = f((a \ a')\vec{x})\]  \(\text{(E3)}\)
\[p(\vec{x}) \Rightarrow p((a \ a')\vec{x})\]  \(\text{(E4)}\)
\[(b \ b')[a]x = [(b \ b')a](b \ b')x\]  \(\text{(E5)}\)
\[a \# x \land a' \# x \Rightarrow (a \ a')x = x\]  \(\text{(F1)}\)
\[a \# a' \iff a \neq a'\]  \(\text{(F2)}\)
\[\forall a : ns, a' : ns'. a \# a' \quad (ns \neq ns')\]  \(\text{(F3)}\)
\[\forall \vec{x}. \exists a. a \# \vec{x}\]  \(\text{(F4)}\)
\[[a]x = [a']x' \iff (a = a' \land x = x') \lor (a \# x' \land (a \ a')x = x')\]  \(\text{(A1)}\)
\[\forall x : [ns]s. \exists a : ns, y : s. x = [a]y\]  \(\text{(A2)}\)
\[\forall \vec{x}. (\forall a. \phi \iff \exists a.a \# \vec{x} \land \phi) \quad (\text{FV}(\forall a. \phi) \subseteq \vec{x})\]  \(\text{(Q)}\)
Nominal Languages

Freshness conditions $a\not\equiv t$, name swapping $(a\ b)\cdot t$, abstraction $[a]\ t$

- Terms with binders
- Built-in $\alpha$-equivalence
- Simple notion of substitution (first order)
- Efficient matching and unification algorithms
- Dependencies of terms on names are implicit
• Variables: $M, N, X, Y, \ldots$
  
  Names: $a, b, \ldots$

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

• Nominal Terms:

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

\( \pi \) is a permutation: finite bijection on names, represented as a list of swappings, e.g., \( (a \ b)(c \ d) \), \( \text{Id} \) (empty list).

\( \pi \cdot t \): \( \pi \) acts on \( t \), permutes names, suspends on variables.

\( (a \ b) \cdot a = b \), \( (a \ b) \cdot b = a \), \( (a \ b) \cdot c = c \)

\( \text{Id} \cdot X \) written as \( X \).
• **Variables:** $M, N, X, Y, \ldots$

**Names:** $a, b, \ldots$

**Function symbols** (term formers): $f, g \ldots$

• **Nominal Terms:**

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

$\pi$ is a **permutation**: finite bijection on names, represented as a list of **swappings**, e.g., $ (a \ b)(c \ d)$, $Id$ (empty list).

$\pi \cdot t$: $\pi$ 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):** 
  - $\text{var}(a)$, $\text{app}(t, t')$, $\text{lam}([a]t)$, $\text{let}(t, [a]t')$, $\text{letrec}[f]([a]t, t')$, $\text{subst}([a]t, t')$

  **Syntactic sugar:**

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

We use freshness to avoid name capture: 
\(a \not\approx X\) means \(a \not\in \text{fv}(X)\) when \(X\) is instantiated.

\[
\begin{align*}
\frac{a \approx_\alpha a}{\pi \cdot X \approx_\alpha \pi' \cdot X} \\
\frac{ds(\pi, \pi') \not\approx X}{ds(\pi, \pi') \not\approx X}
\end{align*}
\]

\[
\begin{align*}
\frac{s_1 \approx_\alpha t_1 \cdots s_n \approx_\alpha t_n}{(s_1, \ldots, s_n) \approx_\alpha (t_1, \ldots, t_n)} \\
\frac{s \approx_\alpha t}{[a]s \approx_\alpha [a]t} \\
\frac{a \not\approx t \quad s \approx_\alpha (a \ b) \cdot t}{[a]s \approx_\alpha [b]t}
\end{align*}
\]

where

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

- \(a \not\approx X, \ b \not\approx X \vdash (a \ b) \cdot X \approx_\alpha X\)
We use freshness to avoid name capture: \(a \# X\) means \(a \not\in \text{fv}(X)\) when \(X\) is instantiated.

\[
\frac{a \approx_{\alpha} a}{\pi \cdot X \approx_{\alpha} \pi' \cdot X}
\]

\[
\frac{ds(\pi, \pi') \# X}{(s_1, \ldots, s_n) \approx_{\alpha} (t_1, \ldots, t_n)}
\]

\[
\frac{s_1 \approx_{\alpha} t_1 \cdots s_n \approx_{\alpha} t_n}{(s_1, \ldots, s_n) \approx_{\alpha} (t_1, \ldots, t_n)}
\]

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

\[
\frac{a \# t}{[a] s \approx_{\alpha} [a] t}
\]

\[
\frac{b \# t}{[a] s \approx_{\alpha} [b] t}
\]

where

\[
ds(\pi, \pi') = \{ n | \pi(n) \neq \pi'(n) \}\]
Also defined by induction:

\[
\begin{aligned}
& a \# b & a \# [a] s & \frac{\pi^{-1}(a) \# X}{a \# \pi \cdot X} \\
& a \# s_1 \cdots a \# s_n & a \# f s & a \# [b] s
\end{aligned}
\]
Nominal Rewriting Rules:

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

Example

Beta-reduction in the Lambda-calculus:

- **Beta**
  $$\beta (\lambda [a]X)Y \rightarrow X[a\mapsto Y]$$

- **σ\_a**
  $$a[a\mapsto Y] \rightarrow Y$$

- **σ\_app**
  $$(XX')[a\mapsto Y] \rightarrow X[a\mapsto Y]X'[a\mapsto Y]$$

- **σ\_ε**
  $$a\# Y \vdash Y[a\mapsto X] \rightarrow Y$$

- **σ\_λ**
  $$b\# Y \vdash (\lambda [b]X)[a\mapsto Y] \rightarrow \lambda [b](X[a\mapsto Y])$$

Rewriting steps: $$(\lambda [c]c)Z \rightarrow c[c\mapsto Z] \rightarrow Z$$
Rewriting relation generated by $R = \nabla \vdash l \rightarrow r$: $\Delta \vdash s \overset{R}{\rightarrow} t$

$s$ rewrites with $R$ to $t$ in the context $\Delta$ when:

1. $s \equiv C[s']$ such that $\theta$ solves $(\nabla \vdash l) \approx (\Delta \vdash s')$
2. $\Delta \vdash C[r\theta] \approx_\alpha t$. 
Example: Prenex Normal Forms

\[ a \# P \vdash P \land \forall[a]Q \rightarrow \forall[a](P \land Q) \]
\[ a \# P \vdash (\forall[a]Q) \land P \rightarrow \forall[a](Q \land P) \]
\[ a \# P \vdash P \lor \forall[a]Q \rightarrow \forall[a](P \lor Q) \]
\[ a \# P \vdash (\forall[a]Q) \lor P \rightarrow \forall[a](Q \lor P) \]
\[ a \# P \vdash P \land \exists[a]Q \rightarrow \exists[a](P \land Q) \]
\[ a \# P \vdash (\exists[a]Q) \land P \rightarrow \exists[a](Q \land P) \]
\[ a \# P \vdash P \lor \exists[a]Q \rightarrow \exists[a](P \lor Q) \]
\[ a \# P \vdash (\exists[a]Q) \lor P \rightarrow \exists[a](Q \lor P) \]
\[ \vdash \neg(\exists[a]Q) \rightarrow \forall[a]\neg Q \]
\[ \vdash \neg(\forall[a]Q) \rightarrow \exists[a]\neg Q \]
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: $\alpha$-equivalence
- Higher-order unification is undecidable

Nominal terms have good computational properties:

- Unification is decidable and unitary
- Efficient algorithms: $\alpha$-equivalence, matching, unification

$\Rightarrow$ Programming languages (Alpha-Prolog, FreshML)

$\Rightarrow$ Nominal Rewriting
Idea: The $\alpha$-equivalence derivation rules become *simplification* rules

\[
\begin{align*}
    a \# b, \Pr & \implies \Pr \\
    a \# fs, \Pr & \implies a \# s, \Pr \\
    a \# (s_1, \ldots, s_n), \Pr & \implies a \# s_1, \ldots, a \# s_n, \Pr \\
    a \# [b]s, \Pr & \implies a \# s, \Pr \\
    a \# [a]s, \Pr & \implies \Pr \\
    a \# \pi \cdot X, \Pr & \implies \pi^{-1} \cdot a \# X, \Pr \quad \pi \neq \text{Id} \\
    a \approx_\alpha a, \Pr & \implies \Pr \\
    (l_1, \ldots, l_n) \approx_\alpha (s_1, \ldots, s_n), \Pr & \implies l_1 \approx_\alpha s_1, \ldots, l_n \approx_\alpha s_n, \Pr \\
    fl \approx_\alpha fs, \Pr & \implies l \approx_\alpha s, \Pr \\
    [a]l \approx_\alpha [a]s, \Pr & \implies l \approx_\alpha s, \Pr \\
    [b]l \approx_\alpha [a]s, \Pr & \implies (a \ b) \cdot l \approx_\alpha s, a \# l, \Pr \\
    \pi \cdot X \approx_\alpha \pi' \cdot X, \Pr & \implies ds(\pi, \pi') \# X, \Pr
\end{align*}
\]
• Nominal Unification: \( l \ ?\approx\? t \) has solution \((\Delta, \theta)\) if

\[
\Delta \vdash l\theta \approx_\alpha t\theta
\]

Nominal Matching: \( s = t \) has solution \((\Delta, \theta)\) if

\[
\Delta \vdash s\theta \approx_\alpha t
\]

\((t \text{ ground or variables disjoint from } s)\)
• Nominal Unification: \( l \approx t \) has solution \((\Delta, \theta)\) if

\[
\Delta \vdash l\theta \approx_\alpha t\theta
\]

Nominal Matching: \( s = t \) has solution \((\Delta, \theta)\) if

\[
\Delta \vdash s\theta \approx_\alpha t
\]

\((t \text{ ground or variables disjoint from } s)\)

• Examples:

\[
\lambda([a]X) = \lambda([b]b) \quad ??
\lambda([a]X) = \lambda([b]X) \quad ??
\]
• Nominal Unification: $l \approx t$ has solution $(\Delta, \theta)$ if

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

Nominal Matching: $s = t$ has solution $(\Delta, \theta)$ 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 $(\{a\#X, b\#X\}, ld)$ resp.
Nominal matching is decidable [Urban, Pitts, Gabbay 2003] A solvable problem $Pr$ has a unique most general solution: $(\Gamma, \theta)$ such that $\Gamma \vdash Pr\theta$.

Nominal matching algorithm: add an instantiation rule:

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

No occur-checks needed (left-hand side variables distinct from right-hand side variables).
**Alpha-equivalence check**: linear if right-hand sides of constraints are ground. Otherwise, log-linear.

**Matching**: quadratic in the non-ground case
### Complexity - Summary

<table>
<thead>
<tr>
<th>Case</th>
<th>Alpha-equivalence</th>
<th>Matching</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ground</td>
<td>linear</td>
<td>linear</td>
</tr>
<tr>
<td>Non-ground and linear</td>
<td>log-linear</td>
<td>log-linear</td>
</tr>
<tr>
<td>Non-ground and non-linear</td>
<td>log-linear</td>
<td>quadratic</td>
</tr>
</tbody>
</table>

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.
Let \( R = \nabla \vdash l \rightarrow r \) where \( V(l) \cap V(s) = \emptyset \).

\( s \) rewrites with \( R \) to \( t \) in the context \( \Delta \), written \( \Delta \vdash s \xrightarrow{R} t \), when:

1. \( s \equiv C[s'] \) such that \( \theta \) 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 $R = \nabla \vdash l \rightarrow r$ where $V(l) \cap V(s) = \emptyset$

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

1. $s \equiv C[s']$ such that $\theta$ 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 $(\Delta', \theta)$, 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:

\[
\begin{align*}
\text{Beta} & \quad (\lambda[a]X)Y \quad \rightarrow \quad X[a \mapsto Y] \\
\sigma_a & \quad a[a \mapsto Y] \quad \rightarrow \quad Y \\
\sigma_{app} & \quad (XX')[a \mapsto Y] \quad \rightarrow \quad X[a \mapsto Y]X'[a \mapsto Y] \\
\sigma_\epsilon & \quad a \# Y \vdash Y[a \mapsto X] \quad \rightarrow \quad Y \\
\sigma_\lambda & \quad b \# Y \vdash (\lambda[b]X)[a \mapsto Y] \quad \rightarrow \quad \lambda[b](X[a \mapsto Y])
\end{align*}
\]
• Nominal matching is efficient.
• Nominal matching is efficient.

• **Equivariant nominal matching** is exponential... **BUT**
- 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.
$R \equiv \nabla \vdash l \rightarrow r$ is closed when

$$(\nabla' \vdash (l', r')) \approx (\nabla, A(R')#V(R) \vdash (l, r))$$

has a solution $\sigma$ (where $R'$ is freshened with respect to $R$).

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

$$\Delta \vdash s \xrightarrow{R} t$$

when

$$\Delta, A(R')#V(\Delta, s) \vdash s \xrightarrow{R'} t$$

and call this closed rewriting.
The following rules are not closed:

\[ g(a) \rightarrow a \]

\[ [a]X \rightarrow X \]

Why?
The following rule is closed:

\[ a\#X \vdash [a]X \rightarrow X \]

Why?
Closed rules that define **capture-avoiding substitution** in the lambda calculus:

(explicit) substitutions, \( \text{subst}(\,[x] M, N) \) abbreviated \( M[x \mapsto N] \).

\[
\begin{align*}
\text{(Beta)} & \quad (\lambda[a]X)X' & \rightarrow & \quad X[a \mapsto X'] \\
\text{(σ_{app})} & \quad (XX')[a \mapsto Y] & \rightarrow & \quad X[a \mapsto Y]X'[a \mapsto Y] \\
\text{(σ_{a})} & \quad a[a \mapsto X] & \rightarrow & \quad X \\
\text{(σ_{ε})} & \quad a \# Y \vdash Y[a \mapsto X] & \rightarrow & \quad Y \\
\text{(σ_{λ})} & \quad b \# Y \vdash (\lambda[b]X)[a \mapsto Y] & \rightarrow & \quad \lambda[b](X[a \mapsto Y])
\end{align*}
\]
Closed Nominal Rewriting:

- works uniformly in $\alpha$ equivalence classes of terms.
- is expressive: can encode Combinatory Reduction Systems.
- is efficient: linear matching.
- inherits confluence conditions from first order rewriting.
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 $\sigma$ and a freshness context $\nabla$ 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 $\sigma$ and a freshness context $\nabla$ 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:

$$\pi \triangleright x \iff \pi \cdot x = x$$

Using fixed-point constraints instead of freshness constraints, nominal C-unification is finitary.
• NRSs are first-order systems with built-in $\alpha$-equivalence.
• Closed NRSs $\iff$ 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, $\alpha$-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)