Nominal Completion for Rewrite Systems with Binders

Maribel Fernández

King's College London

July 2012

Joint work with Albert Rubio

Summary

Motivations

Nominal Rewriting

Closed nominal rules

Confluence and Termination

Completion

Future work

First-order languages vs. languages with binders

First-order data structures: numbers, lists, trees, etc.
available in most programming languages.

Few languages provide data structures with binding,
needed in type checkers, program analysers, compilers, etc.

Binding operators: Examples

Some concrete examples of binding operators (informally):

- Operational semantics:

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

Binding operators: Examples

Some concrete examples of binding operators (informally):

- Operational semantics:

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

- π-calculus:

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

Binding operators: Examples

Some concrete examples of binding operators (informally):

- Operational semantics:

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

- π-calculus:

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

- Logic equivalences:

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

Binders - α-equivalence

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

Example:
In $\forall x.P$ the variable x can be renamed.

- α-conversion is implicit, but
  $\forall x.P \neq_\alpha \forall y.P$ — x may occur in P.

Binders - α-equivalence

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

Example:
In $\forall x.P$ the variable x can be renamed.

- α-conversion is implicit, but
  $\forall x.P \neq_\alpha \forall y.P$ — x may occur in P.
- $\forall x.P =_\alpha \forall y.P\{x \mapsto y\}$

  Substitution of a bound name by a term has to avoid capture of other bound names: y fresh?

Binders - α-equivalence

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

Example:
In $\forall x.P$ the variable x can be renamed.

- α-conversion is implicit, but
  $\forall x.P \neq_\alpha \forall y.P$ — x may occur in P.
- $\forall x.P =_\alpha \forall y.P\{x \mapsto y\}$

  Substitution of a bound name by a term has to avoid capture of other bound names: y fresh?
- Formal definition.
  There are several alternatives.

Formally:

Alternatives:

- Encode in a first-order rewrite system
  e.g. use De Bruijn indices, encode alpha using operators such as "lift" and "shift"; explicit substitutions.
  (-) α-equivalence has to be "implemented"
  (+) simple notion of substitution, efficient first-order matching

Formally:

Alternatives:

- Encode in a first-order rewrite system
  e.g. use De Bruijn indices, encode alpha using operators such as "lift" and "shift"; explicit substitutions.
  (-) α-equivalence has to be "implemented"
  (+) simple notion of substitution, efficient first-order matching
- Higher-order rewrite systems (CRS, HRS, ERS, etc.), e.g.

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

  (+) Binders, functions, implicit α-equivalence
  (-) Substitution as a meta-operation, using β
  (-) Unification is undecidable in general

Alternative: Nominal Approach [Pitts, Gabbay, Urban, ...]

Inspired by nominal set theory (Fraenkel-Mostowski).
Key ideas: Freshness conditions $a\#t$, name swapping $(a\ b) \cdot t$.
Example:

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

- Terms with binders
- Built-in α-equivalence
- Efficient matching and unification algorithms
- Simple notion of substitution ("first-order"), non-capturing substitution has to be specified using rewrite rules.
- Dependencies of terms on names are implicit.

Nominal Syntax [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

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 Rewriting

Rewriting on nominal terms.

Applications:

- equational reasoning on data structures with binding;
- algebraic specifications of operators and data structures;
- operational semantics of programs;
- compilers, program transformations, etc.

Nominal Rewriting

Nominal Rewriting Rules:

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

Example: prenex normal forms in first-order logic

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

Reduction relation generated by (equivariant) nominal matching.

Nominal matching problem

$l \approx_\alpha t$ where $V(l) \cap V(t) = \emptyset$ has solution (Δ, θ) if

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

- Nominal matching is decidable [Urban, Pitts, Gabbay 2003]
- A solvable problem Pr has a unique most general solution: (Γ, θ) such that $\Gamma \vdash Pr\theta$.
- Efficient algorithms: linear in time and space [Calves-F.2008] BTW, nominal unification is quadratic [Calves-F.2010,Levy-Villaret2010].
- But for general NRSs nominal matching is not sufficient: we need Equivariant Nominal Matching — NP [Cheney2004].
- If rules are **closed**, then nominal matching is sufficient.

Closed Rules

Intuitively: no free names.
$R \equiv \nabla \vdash l \to r$ is **closed** when

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

has a solution σ (where R' is a freshened copy of R).

Given $R \equiv \nabla \vdash l \to r$ and $\Delta \vdash s$ we write

$$\Delta \vdash s \xrightarrow[c]{R} t \quad \text{when} \quad \Delta, A(R')\#V(\Delta, s) \vdash s \xrightarrow{R'} t$$

and call this **closed rewriting**.

Examples

The following rules are not closed:

$$g(a) \to a$$

$$[a]X \to X$$

Why?

Examples

The following rule is closed:

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

Why?

Examples

The following rules are closed, they define capture-avoiding substitution in the lambda calculus.
We introduce a term-former *subst* which we sugar to $t[a \mapsto t']$.

$$
\begin{array}{lllll}
(\text{Beta}) & & (\lambda[a]X)X' & \to & X[a \mapsto X'] \\
(\sigma_{App}) & & (XX')[a \mapsto Y] & \to & X[a \mapsto Y]X'[a \mapsto Y] \\
(\sigma_{var}) & & a[a \mapsto X] & \to & X \\
(\sigma_{\epsilon}) & a\#Y \vdash & Y[a \mapsto X] & \to & Y \\
(\sigma_{fn}) & b\#Y \vdash & (\lambda[b]X)[a \mapsto Y] & \to & \lambda[b](X[a \mapsto Y])
\end{array}
$$

Properties of Closed Rewriting

Closed Nominal Rewriting:

- works uniformly in α equivalence classes of terms.

Properties of Closed Rewriting

Closed Nominal Rewriting:

- works uniformly in α equivalence classes of terms.
- is expressive: can encode Combinatory Reduction Systems.

Properties of Closed Rewriting

Closed Nominal Rewriting:

- works uniformly in α equivalence classes of terms.
- is expressive: can encode Combinatory Reduction Systems.
- efficient matching.

Properties of Closed Rewriting

Closed Nominal Rewriting:

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

Confluence

Suppose

1. $R_i = \nabla_i \vdash l_i \to r_i$ for $i = 1, 2$ are copies of two rules in $\mathcal{R}$ such that $V(R_1) \cap V(R_2) = \emptyset$ (R_1 and R_2 could be copies of the same rule).
2. $l_1 \equiv L[l_1']$ such that $\nabla_1, \nabla_2, l_1' \stackrel{?}{\approx}_? l_2$ has a principal solution (Γ, θ), so that $\Gamma \vdash l_1'\theta \approx_\alpha l_2\theta$ and $\Gamma \vdash \nabla_i\theta$ for $i = 1, 2$.

Then $\Gamma \vdash (r_1\theta, L\theta[r_2\theta])$ is a **critical pair**.
If $L = [\text{-}]$ and R_1, R_2 are copies of the same rule, or if l_1' is a variable, then we say the critical pair is **trivial**.

Critical Pair Lemma:
A closed nominal rewrite system where all non-trivial critical pairs are joinable, is locally confluent.

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

Termination

Many techniques for first-order systems: well developed area.

Example: recursive path ordering [Dershowitz]

rpo: A well-founded precedence on function symbols generates a well-founded ordering on first-order terms.

Nominal rpo - Version 1

Idea:

- Equate all atoms (by collapsing them into a unique atom a).
- Precedence: extend a precedence on the signature $\Sigma \cup \{a, [a]\cdot\}$

Formally:

1. $\hat{t}$ coincides with t except that all the atoms have been replaced by a distinguished atom a.
2. $\Delta \vdash s > t$ if $\hat{s} >_{rpo} \hat{t}$ using a precedence $>_\Sigma$ on $\Sigma \cup \{a, [a]\cdot\}$, such that $f >_\Sigma [a]\cdot >_\Sigma a$ for all $f \in \Sigma$.

Nominal rpo version 1

Let Δ be a freshness context, s, t nominal terms over Σ, and $>_\Sigma$ a precedence on Σ. We write $\Delta \vdash s > t$, if $\hat{s} > \hat{t}$ as defined below, where $\geq$ is the reflexive closure of $>$, and $>_{mul}$ is the multiset extension of $>$.

1. $[a]s > t$ if $s \geq t$
2. $s > [a]t$ if $root(s) \in \Sigma$ and $s > t$
3. $[a]s > [a]t$ if $s > t$
4. $[a]s > a$
5. $f(s_1, \ldots, s_n) > t$ if $s_i \geq t$ for some i, $f \in \Sigma$
6. $f(s_1, \ldots, s_n) > g(t_1, \ldots, t_m)$ if $f >_\Sigma g$ and $f(s_1, \ldots, s_n) > t_i$ for all i
7. $f(s_1, \ldots, s_n) > a$
8. $f(s_1, \ldots, s_n) > g(t_1, \ldots, t_m)$ if $f =_\Sigma g$ and $\{s_1, \ldots, s_n\} >_{mul} \{t_1, \ldots, t_m\}$.

Examples

We can order the nominal rewriting rules that compute prenex normal form of first-order logic formulas:

Use a precedence
$\wedge >_\Sigma \forall$, $\wedge >_\Sigma \exists$,
$\vee >_\Sigma \forall$, $\vee >_\Sigma \exists$,
$\neg >_\Sigma \forall$, $\neg >_\Sigma \exists$

Then
$a\#P \ \vdash \ P \wedge \forall[a]Q > \forall[a](P \wedge Q)$

Example[Jouannaud-RubioCSL2008]

HORPO to compare terms in λ-calculus extended with constants
Example: $\vdash\ f(g([x]f(x,x)), g([x]f(x,x))) > [x]f(x,x)$ in the HORPO and also in the nominal rpo (because $g([a]f(a,a)) > a$).
More interesting:

$$\vdash\ f(g(X,Y))) > g(X,[a]f(h(Y,a)))$$

using the nominal rpo but not comparable with HORPO

Proof: Assume $f >_\Sigma g >_\Sigma h$, then $\vdash\ f(g(X,Y))) > X$ and $\vdash\ f(g(X,Y))) > [a]f(h(Y,a)))$. The first one is trivial (subterm relation). For the second, we show $\vdash\ f(g(X,Y))) > f(h(Y,a)))$, which requires $g(X,Y) > h(Y,a)$. Since $g(X,Y) > Y$, and $g(X,Y) > a$, we are done.

Nominal rpo - Version 1

Properties
It is a reduction ordering:

1. transitive, irreflexive, well-founded, and preserved by context and substitution, because we have $\Delta \vdash s > t$ if and only if $\hat{s} >_{rpo} \hat{t}$
2. preserved by $\approx_\alpha$, that is, it works uniformly in α-equivalence classes

Not preserved under atom substitution.
To obtain an ordering preserved by atom substitution we need to distinguish unabstracted atoms.

Nominal rpo

Let $>_\Sigma$ be a precedence on Σ.
Let $\mathbb{C} = \{c_1, c_2, \ldots\}$ such that $\mathbb{C} \cap \mathbb{A}(\Delta, s, t) = \emptyset$
$\Delta \vdash s > t$, defined by cases below, where $\geq$ is $> \cup \bowtie$.

1. $\Delta \vdash [a]s > t$ if $\Delta \cup \Delta_{c\#s,t} \vdash (a\ c) \cdot s \geq t$, for an arbitrary $c \in \mathbb{C} - \mathbb{A}(\Delta, [a]s, t)$.
2. $\Delta \vdash s > [a]t$ if $root(s) \in \Sigma$ and $\Delta \cup \Delta_{c\#s,t} \vdash s > (a\ c) \cdot t$, for an arbitrary $c \in \mathbb{C} - \mathbb{A}(\Delta, s, [a]t)$.
3. $\Delta \vdash [a]s > [b]t$ if $\Delta \cup \Delta_{c\#s,t} \vdash (a\ c) \cdot s > (b\ c) \cdot t$, for an arbitrary $c \in \mathbb{C} - \mathbb{A}(\Delta, [a]s, [b]t)$.
4. $\Delta \vdash [a]s > [a]t$ if $\Delta \vdash s > t$.
5. $\Delta \vdash [a]s > c$ if $c \in \mathbb{C}$.
6. $\Delta \vdash f(s_1, \ldots, s_n) > t$ if $\Delta \vdash s_i \geq t$ for some i.
7. $\Delta \vdash f(s_1, \ldots, s_n) > g(t_1, \ldots, t_m)$ if $f >_\Sigma g$ and $\Delta \vdash f(s_1, \ldots, s_n) > t_i$ for all i.
8. $\Delta \vdash f(s_1, \ldots, s_n) > c$ if $c \in \mathbb{C}$.
9. $\Delta \vdash f(s_1, \ldots, s_n) > g(t_1, \ldots, t_m)$ if $f =_\Sigma g$ and $\Delta \vdash \{s_1, \ldots, s_n\} >_{mul} \{t_1, \ldots, t_m\}$.

Examples

Precedence to order the rules for prenex normal form:
$\wedge >_\Sigma \forall$, $\wedge >_\Sigma \exists$, $\vee >_\Sigma \forall$, $\vee >_\Sigma \exists$, $\neg >_\Sigma \forall$, $\neg >_\Sigma \exists$
Example:
$a\#P \ \vdash \ P \wedge \forall[a]Q >_{nrpo} \forall[a](P \wedge Q)$
Proof: since $\wedge >_\Sigma \forall$, we show $a\#P \ \vdash \ P \wedge \forall[a]Q > [a](P \wedge Q)$
We have an abstraction in the rhs, so we show

$$a\#P, c\#P, c\#Q \ \vdash \ P \wedge \forall[a]Q > (a\ c) \cdot P \wedge (a\ c) \cdot Q$$

We now compare their subterms. Since
$a\#P, c\#P, c\#Q \ \vdash \ P \approx_\alpha (a\ c) \cdot P$, it boils down to proving
$a\#P, c\#P, c\#Q \ \vdash \ \forall[a]Q > (a\ c) \cdot Q$,
We show $a\#P, c\#P, c\#Q \ \vdash \ [a]Q \geq (a\ c) \cdot Q$
This is a consequence of
$a\#P, c\#P, c\#Q, c'\#Q \ \vdash \ (a\ c') \cdot Q \geq (a\ c) \cdot Q$, which holds since $ds((a\ c'), (a\ c)) = \{a, c, c'\}$ and we can derive

$$a\#P, c\#P, c\#Q, c'\#Q \ \vdash \ (a\ c') \cdot Q \bowtie (a\ c) \cdot Q$$

Properties

The nominal rpo has the following properties:

1. If $\Delta \vdash s >_{nrpo} t$ then:
   1. For all $C[\text{-}]$, $\Delta \vdash C[s] >_{nrpo} C[t]$.
   2. For all π, $\Delta \vdash \pi \cdot s >_{nrpo} \pi \cdot t$.
   3. For all Γ such that $\Gamma \vdash \Delta\sigma$, $\Gamma \vdash s\sigma >_{nrpo} t\sigma$.
2. It is a decidable, irreflexive and transitive relation.
3. It is well founded when the precedence is well-founded: there are no infinite descending chains $\Delta \vdash s_1 >_{nrpo} s_2 >_{nrpo} \dots$
4. Compatible with $\approx_\alpha$ and preserved by capture-avoiding atom-substitution.

Confluence and Termination of Nominal Rewriting

If an equational theory can be represented by a confluent and terminating rewrite system, then equational reasoning can be mechanised.

Closed rewriting is sufficient to decide equality modulo a nominal equational theory if the axioms can be oriented to form a confluent and terminating closed NRS [LFMTP2010].

Confluence and Termination of Nominal Rewriting

Theorem

If for all $\nabla \vdash l \to r$ *in an NRS* $\mathcal{R}$ *we have* $\nabla \vdash l > r$, *where* $>$ *is one of the orderings defined above, then* $\mathcal{R}$ *is terminating.*

Newman's Lemma: Local confluence + termination implies confluence.

Kunth-Bendix Completion

If a nominal rewriting system is not confluent, but it is terminating: compute critical pairs and try to add rules to join them.

Key ideas:

- Critical pair lemma
- Nominal rpo to check termination
- Critical pairs CAN be added as rules (no problems with types or format as in other rewriting formalisms that manipulate binders)

Nominal Completion

Transformation rules on pairs $(E, \mathcal{R})$
(closed nominal equations and nominal rewriting rules)
Input: $(E, \emptyset)$ and a (well-founded) reduction ordering $>$
Transformation rules:

$$
\begin{array}{lcll}
(E, \mathcal{R}) & \Rightarrow & (E \cup \Delta \vdash s = t, \mathcal{R}) & \text{if } \Delta \vdash (s, t) \\
 & & & \text{critical pair of } \mathcal{R} \\
(E \cup \Delta \vdash s = t, \mathcal{R}) & \Rightarrow & (E, \mathcal{R} \cup \Delta \vdash s \to t) & \text{if } \Delta \vdash s > t \\
(E \cup \Delta \vdash s = t, \mathcal{R}) & \Rightarrow & (E, \mathcal{R} \cup \Delta \vdash t \to s) & \text{if } \Delta \vdash t > s \\
(E \cup \Delta \vdash s = t, \mathcal{R}) & \Rightarrow & (E, \mathcal{R}) & \text{if } \Delta \vdash s \approx_\alpha t \\
(E \cup \Delta \vdash s = t, \mathcal{R}) & \Rightarrow & (E \cup \Delta \vdash s' = t, \mathcal{R}) & \text{if } \Delta \vdash s \to^c_{\mathcal{R}} s' \\
(E \cup \Delta \vdash s = t, \mathcal{R}) & \Rightarrow & (E \cup \Delta \vdash s = t', \mathcal{R}) & \text{if } \Delta \vdash t \to^c_{\mathcal{R}} t' \\
(E, \mathcal{R} \cup \Delta \vdash s \to t) & \Rightarrow & (E, R \cup \Delta \vdash s \to t') & \text{if } \Delta \vdash t \to^c_{\mathcal{R}} t' \\
(E, \mathcal{R} \cup \Delta \vdash s \to t) & \Rightarrow & (E \cup \Delta \vdash s' = t, \mathcal{R}) & \text{if } \Delta \vdash s \to^c_{\mathcal{R}} s'
\end{array}
$$

As in the case of first-order rewriting, the rule that computes critical pairs should be used in a controlled way.

Example

The following rules are inspired by [Nipkow-Prehofer1998].

As ordering take nrpo with any precedence.

$$\begin{array}{lll} (\eta) & a\#X & \vdash \quad \lambda([a]app(X,a)) \to X \\ (\bot) & & \quad app(\bot, Y) \to \bot \end{array}$$

There is a critical pair since the unification problem $a\#X, app(X,a) \;{}_?{\approx}_?\; app(\bot, Y)$ has solution $\{X \mapsto \bot, Y \mapsto a\}$: $\lambda([a]\bot) = \bot$ can be oriented into $\lambda([a]\bot) \to \bot$.

Example

The following rules are inspired by Van de Pol's PhD thesis (1996).
As ordering take nrpo with precedence $\Sigma > +$ and $\Sigma > app > id$.

$$
\begin{array}{lcl}
a\#F & \vdash & \Sigma(0,[a]app(F,a)) \to 0 \\
a\#F & \vdash & \Sigma(s(N),[a]app(F,a)) \to app(F,s(N)) + \Sigma(N,[a]app(F,a)) \\
 & & id(X) \to X \\
 & & app(id0,X) \to id(X)
\end{array}
$$

With the fourth rule applied on the first and second we obtain two critical pairs: $\Sigma(0,[a]id(a)) = 0$ and $\Sigma(s(N),[a]id(a)) = app(id0,s(N)) + \Sigma(N,[a]app(id0,a))$ which can be simplified and oriented as follows:
$\Sigma(0,[a]a) \to 0$ and $\Sigma(s(N),[a]a) \to s(N) + \Sigma(N,[a]a)$.

Conclusion

- Nominal Terms: extension of first-order syntax, efficient matching modulo α.
- Clean semantics: Nominal Sets
- **Equational reasoning and rewriting**
  Completion as a tool to mechanise nominal equational logic
- Future work: functional abstraction and capture-avoiding substitution, more powerful type systems, ...