Formalisation of Nominal Equational Reasoning

Mauricio Ayala-Rincón
37th International Workshop on Unification - UNIF (FSCD 2023)
Sapienza Università di Roma, July 2nd, 2023

Mathematics and Computer Science Departments

Universidade de Brasília

† Research supported by the Brazilian agencies CAPES, CNPq, and FAPDF
1. Motivation
   Synthesis on Unification modulo

2. Bindings and Nominal Syntax

3. Nominal C-unification

4. Issues Adapting First-Order to Nominal AC-Unification
   An Algorithm for Nominal AC-Matching

5. Synthesis on Nominal Equational Modulo

6. Work in Progress and Future Work
Motivation
Equational Problems

- Equality check: $s = t$? 
- Matching: There exists $\sigma$ such that $s\sigma = t$? 
- Unification: There exists $\sigma$ such that $s\sigma = t\sigma$?

$s$ and $t$ are terms in some signature and $\sigma$ is a substitution.
Goal: to identify two expressions.
Method: replace variables by other expressions.

Example: for $x$ and $y$ variables, $a$ and $b$ constants, and $f$ a function symbol,

- Identify $f(x, a)$ and $f(b, y)$
Goal: to identify two expressions.
Method: replace variables by other expressions.

Example: for $x$ and $y$ variables, $a$ and $b$ constants, and $f$ a function symbol,

- Identify $f(x, a)$ and $f(b, y)$
- solution $\{x/b, y/a\}$. 
• $\mathcal{F}$ set of function symbols.
• $\mathcal{V}$ set of variables.
• $x, y, z$ variables.
• $a, b, c$ constant symbols.
• $f, g, h$ function symbols.
• $\mathcal{T}(\mathcal{F}, \mathcal{V})$ set of terms over $\mathcal{F}$ and $\mathcal{V}$.
• $s, t, u$ terms.
• $\sigma, \gamma, \delta : \mathcal{V} \rightarrow \mathcal{T}(\mathcal{F}, \mathcal{V})$ set of substitutions.

Substitutions have finite domain: $\{ v | v\sigma \neq v \}$ is finite.
Example:

- Solution $\sigma = \{x/b\}$ for $f(x, y) = f(b, y)$ is more general than solution $\gamma = \{x/b, y/b\}$.

$\sigma$ is more general than $\gamma$:

there exists $\delta$ such that $\sigma \delta = \gamma$;

$\delta = \{y/b\}$. 
Equational Problems - Syntactic Unification

Goal: *algorithm* that *unifies* terms.

Example:

- \( h(x, y, z) = h(f(w, w), f(x, x), f(y, y)) \)
Equational Problems - Syntactic Unification

Goal: algorithm that unifies terms.

Example:

- \( h(x, y, z) = h(f(w, w), f(x, x), f(y, y)) \)
- \( h(f(w, w), y, z) = h(f(w, w), f(f(w, w), f(w, w)), f(y, y)), \) partial solution:
  \( \{x / f(w, w)\} \)
Goal: algorithm that unifies terms.

Example:

\[ h(x, y, z) = h(f(w, w), f(x, x), f(y, y)) \]

\[ h(f(w, w), y, z) = h(f(w, w), f(f(w, w), f(w, w)), f(y, y)), \text{ partial solution:} \]
\[ \{x/f(w, w)\} \]

\[ h(f(w, w), f(f(w, w), f(w, w)), z) = h(f(w, w), f(f(w, w), f(w, w)), f(f(w, w), f(w, w))), \text{ partial solution:} \]
\[ \{x/f(w, w), y/f(f(w, w), f(w, w))\} \]
Equational Problems - Syntactic Unification

Goal: *algorithm* that *unifies* terms.

Example:

- $h(x, y, z) = h(f(w, w), f(x, x), f(y, y))$

- $h(f(w, w), y, z) =$
  $h(f(w, w), f(f(w, w), f(w, w)), f(y, y))$, partial solution:
  \{x/f(w, w)\}

- $h(f(w, w), f(f(w, w), f(w, w)), z) =$
  $h(f(w, w), f(f(w, w), f(w, w)), f(f(w, w), f(w, w)))$, partial solution:  \{x/f(w, w), y/f(f(w, w), f(w, w))\}

- $h(f(w, w), f(f(w, w), f(w, w)), f(f(w, w), f(w, w))) =$
  $h(f(w, w), f(f(w, w), f(w, w)), f(f(w, w), f(w, w)))$, solution:  \{x/f(w, w), y/f(f(w, w), f(w, w)), z/f(f(f(w, w), f(w, w)), f(f(w, w), f(w, w)))\}. 
Interesting questions:

- Correctness and Completeness.
- Complexity.
- With adequate data structures, there are linear solutions (Huet, Martelli-Montanari 1976, Petterson-Wegman 1978).

Syntactic unification is of type *unary* and linear.
When operators have algebraic equational properties, the problem is not as simple.

**Example:** for \( f \) commutative (C), \( f(x, y) \approx f(y, x) \):

- \( f(x, y) = f(a, b) \)?

The unification problem is of type *finitary*. 
When operators have algebraic equational properties, the problem is not as simple.

**Example:** for \( f \) **commutative** (C), \( f(x, y) \approx f(y, x) \):

- \( f(x, y) = f(a, b) \)?
- Solutions: \( \{ x/a, y/b \} \) and \( \{ x/b, y/a \} \).

The unification problem is of type **finitary**.
Example: for $f$ associative $(A)$, $f(f(x, y), z) \approx f(x, f(y, z))$:

- $f(x, a) = f(a, x)$?

The unification problem is of type *infinitary*. 
Example: for $f$ associative (A), $f(f(x,y),z) \approx f(x,f(y,z))$:

- $f(x,a) = f(a,x)$?
- Solutions: $\{x/a\}, \{x/f(a,a)\}, \{x/f(a,f(a,a))\}, \ldots$

The unification problem is of type *infinitary*. 
Example: for $f$ AC with unity (U), $f(x, e) \approx x$:

- $f(x, y) = f(a, b)$?

The unification problem is of type *finitary*. 
Example: for $f$ AC with $unity$ (U), $f(x, e) \approx x$:

- $f(x, y) = f(a, b)$?
- Solutions: $\{x/e, y/f(a, b)\}$, $\{x/f(a, b), y/e\}$, $\{x/a, y/b\}$, and $\{x/b, y/a\}$.

The unification problem is of type $finitary$. 
Example: for \( f \) A, and idempotent (I), \( f(x, x) \approx x \):

- \( f(x, f(y, x)) = f(f(x, z), x) \)?
Example: for $f$ A, and idempotent (I), $f(x, x) \approx x$:

- $f(x, f(y, x)) = f(f(x, z), x)$?
- Solutions: $\{y/f(u, f(x, u)), z/u\}, \ldots$

The unification problem is of type zero (Schmidt-Schauß 1986, Baader 1986).
Example: for + AC, and $h$ homomorphism ($h$),
$h(x + y) \approx h(x) + h(y)$:

- $h(y) + a = y + z$?

Example: for + AC, and \( h \) homomorphism (\( h \)),
\[ h(x + y) \approx h(x) + h(y) : \]

- \( h(y) + a = y + z \)?
- Solutions: \( \{ y / a, z / h(a) \} , \{ y / h(a) + a, z / h^2(a) \} , \ldots , \{ y / h^k(a) + \ldots + h(a) + a, z / h^{k+1}(a) \} , \ldots \)

The unification problem is of type zero and undecidable (Narendran 1996). The same happens for ACU\( h \) (Nutt 1990, Baader 1993).
Motivation

Synthesis on Unification modulo
<table>
<thead>
<tr>
<th>Theory</th>
<th>Unif. type</th>
<th>Equality-checking</th>
<th>Matching</th>
<th>Unification</th>
<th>Related work</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntactic</td>
<td>1</td>
<td>O(n)</td>
<td>O(n)</td>
<td>O(n)</td>
<td>R65, MM76, PW78</td>
</tr>
<tr>
<td>C</td>
<td>ω</td>
<td>O(n^2)</td>
<td>NP-comp.</td>
<td>NP-comp.</td>
<td>BKN87, KN87</td>
</tr>
<tr>
<td>A</td>
<td>∞</td>
<td>O(n)</td>
<td>NP-comp.</td>
<td>NP-hard</td>
<td>M77, BKN87</td>
</tr>
<tr>
<td>AU</td>
<td>∞</td>
<td>O(n)</td>
<td>NP-comp.</td>
<td>decidable</td>
<td>M77, KN87</td>
</tr>
<tr>
<td>AI</td>
<td>0</td>
<td>O(n)</td>
<td>NP-comp.</td>
<td>NP-comp.</td>
<td>Klíma02, SS86, Baader86</td>
</tr>
<tr>
<td>Theory</td>
<td>Unif. type</td>
<td>Equality-checking</td>
<td>Matching</td>
<td>Unification</td>
<td>Related work</td>
</tr>
<tr>
<td>----------</td>
<td>------------</td>
<td>-------------------</td>
<td>------------</td>
<td>-------------</td>
<td>--------------</td>
</tr>
<tr>
<td>AC</td>
<td>$\omega$</td>
<td>$O(n^3)$</td>
<td>NP-comp.</td>
<td>NP-comp.</td>
<td>BKN87, KN87, KN92</td>
</tr>
<tr>
<td>ACU</td>
<td>$\omega$</td>
<td>$O(n^3)$</td>
<td>NP-comp.</td>
<td>NP-comp.</td>
<td>KN92</td>
</tr>
<tr>
<td>AC(U)I</td>
<td>$\omega$</td>
<td>-</td>
<td>-</td>
<td>NP-comp.</td>
<td>KN92, BMMO20</td>
</tr>
<tr>
<td>D</td>
<td>$\omega$</td>
<td>-</td>
<td>NP-hard</td>
<td>NP-hard</td>
<td>TA87</td>
</tr>
<tr>
<td>ACh</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>undecidable</td>
<td>B93, N96, EL18</td>
</tr>
<tr>
<td>ACUh</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>undecidable</td>
<td>B93, N96</td>
</tr>
</tbody>
</table>
Bindings and Nominal Syntax
Systems with bindings frequently appear in mathematics and computer science, but are not captured adequately in first-order syntax.

For instance, the formulas

\[ \forall x_1, x_2 : x_1 + 1 + x_2 > 0 \quad \text{and} \quad \forall y_1, y_2 : 1 + y_2 + y_1 > 0 \]

are not syntactically equal, but should be considered equivalent in a system with binding and AC operators.
The nominal setting extends first-order syntax, replacing the concept of syntactical equality by $\alpha$-equivalence, which let us represent smoothly those systems.

Profiting from the nominal paradigm implies adapting basic notions (substitution, rewriting, equality) to it.
Consider a set of variables $\mathbf{X} = \{X, Y, Z, \ldots\}$ and a set of atoms $\mathbf{A} = \{a, b, c, \ldots\}$. 
Definition 1 (Nominal Terms)
Nominal terms are inductively generated according to the grammar:

\[ s, t ::= a \mid \pi \cdot X \mid \langle \rangle \mid [a]t \mid \langle s, t \rangle \mid f t \mid f^{AC} t \]

where \( \pi \) is a permutation that exchanges a finite number of atoms.

To guarantee that AC function applications have at least two arguments, we have the notion of well-formed terms.
$a\#t$ means that if $a$ occurs in $t$ then it does so under an abstractor $[a]$.

A context is a set of constraints of the form $a\#X$. Contexts are denoted as $\Delta$, $\Gamma$ or $\nabla$. 
An atom permutation $\pi$ represents an exchange of a finite amount of atoms in $\mathcal{A}$ and is presented by a list of swappings:

$$\pi = (a_1\ b_1) :: \ldots :: (a_n\ b_n) :: nil$$
Permutations act on atoms and terms:

- $(a \ b) \cdot a = b$;
- $(a \ b) \cdot b = a$;
- $(a \ b) \cdot f(a, c) = f(b, c)$;
- $(a \ b) :: (b \ c) \cdot [a]⟨a, c⟩ = (b \ c)[b]⟨b, c⟩ = [c]⟨c, b⟩$. 
Two important predicates are the *freshness* predicate $\#$, and the *$\alpha$-equality* predicate $\approx_{\alpha}$.

- $a \# t$ means that if $a$ occurs in $t$ then it must do so under an abstractor $[a]$.
- $s \approx_{\alpha} t$ means that $s$ and $t$ are $\alpha$-equivalent.
A context is a set of constraints of the form $a \# X$. Contexts are denoted by the letters $\Delta$, $\nabla$ or $\Gamma$. 
Advantages of the name binding nominal approach

*Freshness conditions* \( a \# s \), and *atom permutations* \( \pi \cdot s \).

**Example**

\( \beta \) and \( \eta \) rules as nominal rewriting rules:

\[
\text{app} \langle \text{lam}[a] M, N \rangle \rightarrow \text{subs} \langle [a] M, N \rangle \quad (\beta)
\]

\[
a \# M \vdash \text{lam}[a] \text{app} \langle M, a \rangle \rightarrow M \quad (\eta)
\]

Some substitution rules:

\[
b \# M \vdash \text{subs} \langle [b] M, N \rangle \rightarrow M
\]

\[
a \# N \vdash \text{subs} \langle [b] \text{lam}[a] M, N \rangle \rightarrow \text{lam}[a] \text{sub} \langle [b] M, N \rangle
\]

\[
c \# M, c \# N \vdash \text{subs} \langle [b] \text{lam}[a] M, N \rangle \rightarrow \text{lam}[c] \text{sub} \langle [b] (a \ c) \cdot M, N \rangle
\]
Advantages of the name binding nominal approach

- First-order terms with binders and *implicit* atom dependencies.
- Easy syntax to present *name binding* predicates as
  \[ a \in \text{FreeVar}(M), \; a \in \text{BoundVar}([a]s), \]  and operators as renaming: \( (a \; b) \cdot s \).
- Built-in \( \alpha \)-equivalence and first-order *implicit substitution*.
- Feasible syntactic equational reasoning: efficient equality-check, matching, and unification algorithms.
Derivation Rules for Freshness

\[
\begin{align*}
\Delta \vdash a \# \langle \rangle & \quad (\# \langle \rangle) \\
(\pi^{-1}(a) \# X) \in \Delta & \quad (\# X) \\
\Delta \vdash a \# \pi \cdot X & \quad (\# X) \\
\Delta \vdash a \# t & \quad (\# [a] t) \\
\Delta \vdash a \# [b] t & \quad (\# [a] b) \\
\Delta \vdash a \# t & \quad (\# s) \\
\Delta \vdash a \# [s, t] & \quad (\# pair) \\
\Delta \vdash a \# t & \quad (\# app)
\end{align*}
\]
Derivation Rules for alpha-Equivalence

\[ \Delta \vdash \emptyset \approx_{\alpha} \emptyset \] (\( \approx_{\alpha} \emptyset \))

\[ \Delta \vdash s \approx_{\alpha} t \] (\( \approx_{\alpha} \text{app} \))

\[ \Delta \vdash fs \approx_{\alpha} ft \] (\( \approx_{\alpha} \text{app} \))

\[ \Delta \vdash s \approx_{\alpha} (a \ b) \cdot t, a \# t \] (\( \approx_{\alpha} [a]b \))

\[ \Delta \vdash \pi \cdot X \approx_{\alpha} \pi' \cdot X \] (\( \approx_{\alpha} \text{var} \))

\[ ds(\pi, \pi') \# X \subseteq \Delta \] (\( \approx_{\alpha} \text{var} \))

\[ \Delta \vdash a \approx_{\alpha} a \] (\( \approx_{\alpha} \text{atom} \))

\[ \Delta \vdash a \approx_{\alpha} a \] (\( \approx_{\alpha} \text{atom} \))

\[ \Delta \vdash s \approx_{\alpha} t \] (\( \approx_{\alpha} \text{atom} \))

\[ \Delta \vdash [a]s \approx_{\alpha} [a]t \] (\( \approx_{\alpha} [a]a \))

\[ \Delta \vdash \langle s_0, s_1 \rangle \approx_{\alpha} \langle t_0, t_1 \rangle \] (\( \approx_{\alpha} \text{pair} \))
Let $f$ be a C function symbol.

We add rule $(\approx_{\alpha} \text{c-app})$ for dealing with C functions:

$$
\Delta \vdash s_2 \approx_{\alpha} t_1 \quad \Delta \vdash s_1 \approx_{\alpha} t_2
$$

$$
\Delta \vdash f^C\langle s_1, s_2 \rangle \approx_{\alpha} f^C\langle t_1, t_2 \rangle
$$
Let $f$ be an AC function symbol.

We add rule $\left( \approx_\alpha \text{ac-app} \right)$ for dealing with AC functions:

\[
\begin{align*}
\Delta \vdash S_i(f^{AC} s) & \approx_\alpha S_j(f^{AC} t) \quad \Delta \vdash D_i(f^{AC} s) & \approx_\alpha D_j(f^{AC} t) \\
\Delta \vdash f^{AC} s & \approx_\alpha f^{AC} t
\end{align*}
\]

$S_n(f\ *)$ selects the $n^{th}$ argument of the flattened subterm $f\ *$.

$D_n(f\ *)$ deletes the $n^{th}$ argument of the flattened subterm $f\ *$. 
Derivation Rules as a Sequent Calculus

Deriving $\vdash \forall[a] \oplus \langle a, fa \rangle \approx_\alpha \forall[b] \oplus \langle fb, b \rangle$, where $\oplus$ is C:

\[
\begin{align*}
a \approx_\alpha a & \quad (\approx_\alpha \text{atom}) \\
fa \approx_\alpha fa & \quad (\approx_\alpha \text{app}) \\
\oplus \langle a, fa \rangle & \approx_\alpha (a \ b) \cdot \oplus \langle fb, b \rangle & \quad (\approx_\alpha \text{c-app}) \\
\end{align*}
\]

\[
\begin{align*}
a \approx_\alpha a & \quad (\approx_\alpha \text{atom}) \\
fa \approx_\alpha fa & \quad (\approx_\alpha \text{app}) \\
a \# b & \quad (\# \text{atom}) \\
a \# fb & \quad (\# \text{app}) \\
a \# \langle fb, b \rangle & \quad (\# \text{pair}) \\
\end{align*}
\]

\[
\begin{align*}
[a] \oplus \langle a, fa \rangle & \approx_\alpha [b] \oplus \langle fb, b \rangle & \quad (\approx_\alpha \text{app}) \\
\forall[a] \oplus \langle a, fa \rangle & \approx_\alpha \forall[b] \oplus \langle fb, b \rangle & \quad (\approx_\alpha \text{app}) \\
\end{align*}
\]
Nominal C-unification
Nominal C-unification

Unification problem: \( \langle \Gamma, \{ s_1 \approx_\alpha t_1, \ldots s_n \approx_\alpha t_n \} \rangle \)

Unification solution: \( \langle \Delta, \sigma \rangle \), such that

- \( \Delta \vdash \Gamma \sigma \);
- \( \Delta \vdash s_i \sigma \approx_\alpha t_i \sigma, 1 \leq i \leq n \).

We introduced nominal (equality-check, matching) and unification algorithms that provide solutions given as triples of the form:

\[ \langle \Delta, \sigma, FP \rangle \]

where \( FP \) is a set of fixed-point equations of the form \( \pi \cdot X \approx_\alpha X \).

This provides a finite representation of the infinite set of solutions that may be generated from such fixed-point equations.
**Nominal C-unification**

*Fixed point equations* such as $\pi \cdot X \simeq_\alpha^? X$ may have infinite independent solutions.

For instance, in a signature in which $\oplus$ and $\star$ are C, the unification problem: $\langle \emptyset, \{(a \ b)X \simeq_\alpha^? X\} \rangle$

has solutions:

\[
\begin{align*}
\langle \{a\#X, b\#X\}, id\rangle, \\
\langle \emptyset, \{X/a \oplus b\}\rangle, \langle \emptyset, \{X/a \star b\}\rangle, \ldots \\
\langle \{a\#Z, b\#Z\}, \{X/(a \oplus b) \oplus Z\}\rangle, \ldots \\
\langle \emptyset, \{X/(a \oplus b) \star (b \oplus a)\}\rangle, \ldots
\end{align*}
\]
Issues Adapting First-Order to Nominal AC-Unification
We modified Stickel-Fages’s seminal AC-unification algorithm to avoid mutual recursion and verified it in the PVS proof assistant.

We formalised the algorithm’s termination, soundness, and completeness [AFSS22].
Let $f$ be an AC function symbol. The solutions that come to mind when unifying:

$$f(X, Y) \approx? f(a, W)$$

are:

$$\{X \rightarrow a, Y \rightarrow W\} \text{ and } \{X \rightarrow W, Y \rightarrow a\}$$

Are there other solutions?
Yes!

For instance, \( \{X \rightarrow f(a, Z_1), \ Y \rightarrow Z_2, \ W \rightarrow f(Z_1, Z_2)\} \) and \( \{X \rightarrow Z_1, \ Y \rightarrow f(a, Z_2), \ W \rightarrow f(Z_1, Z_2)\} \).
Example

the AC Step for AC-unification.

How do we generate a complete set of unifiers for:

\[ f(X, X, Y, a, b, c) \approx? f(b, b, b, c, Z) \]
Eliminate common arguments in the terms we are trying to unify.

Now, we must unify

\[ f(X, X, Y, a) \approx f(b, b, Z) \]
According to the number of times each argument appears, transform the unification problem into a linear equation on $\mathbb{N}$:

$$2X_1 + X_2 + X_3 = 2Y_1 + Y_2,$$

Above, variable $X_1$ corresponds to argument $X$, variable $X_2$ corresponds to argument $Y$, and so on.
Generate a basis of solutions to the linear equation.

**Table 1:** Solutions for the Equation \(2X_1 + X_2 + X_3 = 2Y_1 + Y_2\)

<table>
<thead>
<tr>
<th>X_1</th>
<th>X_2</th>
<th>X_3</th>
<th>Y_1</th>
<th>Y_2</th>
<th>2X_1 + X_2 + X_3</th>
<th>2Y_1 + Y_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>
Associate new variables with each solution.

**Table 2:** Solutions for the Equation $2X_1 + X_2 + X_3 = 2Y_1 + Y_2$

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$X_1$</td>
<td>$X_2$</td>
<td>$X_3$</td>
<td>$Y_1$</td>
<td>$Y_2$</td>
<td>$2X_1 + X_2 + X_3$</td>
<td>$2Y_1 + Y_2$</td>
<td>New Variables</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>$Z_1$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>$Z_2$</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>$Z_3$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>$Z_4$</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>$Z_5$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>$Z_6$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>$Z_7$</td>
</tr>
</tbody>
</table>
Observing the previous Table, relate the “old” variables and the “new” ones:

\[ X_1 \approx Z_6 + Z_7 \]
\[ X_2 \approx Z_2 + Z_4 + 2Z_5 \]
\[ X_3 \approx Z_1 + 2Z_3 + Z_4 \]
\[ Y_1 \approx Z_3 + Z_4 + Z_5 + Z_7 \]
\[ Y_2 \approx Z_1 + Z_2 + 2Z_6 \]
Decide whether we will include (set to 1) or not (set to 0) every “new” variable. Every “old” variable must be different than zero.

In our example, we have $2^7$ possibilities of including/excluding the variables $Z_1, \ldots, Z_7$, but after observing that $X_1, X_2, X_3, Y_1, Y_2$ cannot be set to zero, only 69 cases remain.
Drop the cases where the variables representing constants or subterms headed by a different AC function symbol are assigned to more than one of the “new” variables.

For instance, the potential new unification problem

\[
\{ X_1 \approx? Z_6, X_2 \approx? Z_4, X_3 \approx? f(Z_1, Z_4), \\
Y_1 \approx? Z_4, Y_2 \approx? f(Z_1, Z_6, Z_6) \}
\]

should be discarded as the variable \( X_3 \), which represents the constant \( a \), cannot unify with \( f(Z_1, Z_4) \).
Replace “old” variables by the original terms they substituted and proceed with the unification.

Some new unification problems may be unsolvable and will be discarded later. For instance:

\[
\{ X \approx ? Z_6, \ Y \approx ? Z_4, \ a \approx ? Z_4, \ b \approx ? Z_4, \ Z \approx ? f(Z_6, Z_6) \}\]
In our example,

\[ f(X, X, Y, a, b, c) \approx? f(b, b, b, c, Z) \]

the solutions are:

\[
\begin{align*}
\sigma_1 &= \{ Y \to f(b, b), Z \to f(a, X, X) \} \\
\sigma_2 &= \{ Y \to f(Z_2, b, b), Z \to f(a, Z_2, X, X) \} \\
\sigma_3 &= \{ X \to b, Z \to f(a, Y) \} \\
\sigma_4 &= \{ X \to f(Z_6, b), Z \to f(a, Y, Z_6, Z_6) \}
\end{align*}
\]
We found a loop while solving nominal AC-unification problems using Stickel-Fages’ Diophantine-based algorithm.

For instance

\[ f(X, W) \approx ? f(\pi \cdot X, \pi \cdot Y) \]

Variables are associated as below:

- \( U_1 \) is associated with argument \( X \),
- \( U_2 \) is associated with argument \( W \),
- \( V_1 \) is associated with argument \( \pi \cdot X \), and
- \( V_2 \) is associated with argument \( \pi \cdot Y \).
The Diophantine equation associated is $U_1 + U_2 = V_1 + V_2$.

The table with the solutions of the Diophantine equations is shown below. The name of the new variables was chosen to make clearer the loop we will fall into.

**Table 3:** Solutions for the Equation $U_1 + U_2 = V_1 + V_2$

<table>
<thead>
<tr>
<th>$U_1$</th>
<th>$U_2$</th>
<th>$V_1$</th>
<th>$V_2$</th>
<th>$U_1 + U_2$</th>
<th>$V_1 + V_2$</th>
<th>New variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>$Z_1$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>$W_1$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>$Y_1$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>$X_1$</td>
</tr>
</tbody>
</table>
\{X \approx? \, X_1, \, W \approx? \, Z_1, \, \pi \cdot X \approx? \, X_1, \, \pi \cdot Y \approx? \, Z_1\}
\{X \approx? \, Y_1, \, W \approx? \, W_1, \, \pi \cdot X \approx? \, W_1, \, \pi \cdot Y \approx? \, Y_1\}
\{X \approx? \, Y_1 + X_1, \, W \approx? \, W_1, \, \pi \cdot X \approx? \, W_1 + X_1, \, \pi \cdot Y \approx? \, Y_1\}
\{X \approx? \, Y_1 + X_1, \, W \approx? \, Z_1 + W_1, \, \pi \cdot X \approx? \, W_1 + X_1, \, \pi \cdot Y \approx? \, Z_1 + Y_1\}
\{X \approx? \, X_1, \, W \approx? \, Z_1 + W_1, \, \pi \cdot X \approx? \, W_1 + X_1, \, \pi \cdot Y \approx? \, Z_1\}
\{X \approx? \, Y_1, \, W \approx? \, Z_1 + W_1, \, \pi \cdot X \approx? \, W_1, \, \pi \cdot Y \approx? \, Z_1 + Y_1\}
\{X \approx? \, Y_1 + X_1, \, W \approx? \, Z_1 + W_1, \, \pi \cdot X \approx? \, W_1 + X_1, \, \pi \cdot Y \approx? \, Z_1 + Y_1\}
After instantiateStep

Seven branches are generated:

\[ B_1 \rightarrow \{ \pi \cdot X \approx? X \}, \sigma = \{ W \mapsto \pi \cdot Y \} \]

\[ B_2 \rightarrow \sigma = \{ W \mapsto \pi^2 \cdot Y, X \mapsto \pi \cdot Y \} \]

\[ B_3 \rightarrow \{ f(\pi^2 \cdot Y, \pi \cdot X_1) \approx? f(W, X_1) \}, \sigma = \{ X \mapsto f(\pi \cdot Y, X_1) \} \]

\[ B_4 \rightarrow \text{No solution} \]

\[ B_5 \rightarrow \text{No solution} \]

\[ B_6 \rightarrow \sigma = \{ W \mapsto f(Z_1, \pi \cdot X), Y \mapsto f(\pi^{-1} \cdot Z_1, \pi^{-1} \cdot X) \} \]

\[ B_7 \rightarrow \{ f(\pi \cdot Y_1, \pi \cdot X_1) \approx? f(W_1, X_1) \}, \]

\[ \sigma = \{ X \mapsto f(Y_1, X_1), W \mapsto f(Z_1, W_1), Y \mapsto f(\pi^{-1} \cdot Z_1, \pi^{-1} \cdot Y_1) \} \]
Focusing on Branch 7, notice that the problem before the AC Step and the problem after the AC Step and instantiating the variables are, respectively:

\[ P = \{ f(X, W) \approx^? f(\pi \cdot X, \pi \cdot Y) \} \]

\[ P_1 = \{ f(X_1, W_1) \approx^? f(\pi \cdot X_1, \pi \cdot Y_1) \} \]
Issues Adapting First-Order to Nominal AC-Unification

An Algorithm for Nominal AC-Matching
Nominal AC-matching is matching in the nominal setting in the presence of associative-commutative function symbols.

We proposed (to the best of our knowledge) the first nominal AC-matching algorithm, and formalised it in the PVS proof assistant ([AFFKS23]).
Given an algorithm of unification, one can adapt it by adding as a parameter a set of *protected variables* \( \mathcal{X} \), which cannot be instantiated.

The adapted algorithm can then be used for:

- **Unification** - By putting \( \mathcal{X} = \emptyset \).
- **Matching** - By putting \( \mathcal{X} \) as the set of variables in the right-hand side.
- **\( \alpha \)-Equivalence** - By putting \( \mathcal{X} \) as the set of variables that appear in the problem.
We modify our first-order AC-unification formalisation to obtain a formalised algorithm for nominal AC-matching.
The algorithm is recursive and needs to keep track of:

- the current context $\Gamma$,  
- the equational constraints we must unify $P$,  
- the substitution $\sigma$ computed so far,  
- the set of variables $V$ that are/were in the problem, and  
- the set of protected variables $X$.

Hence, it’s input is a quintuple $\langle \Gamma, P, \sigma, V, X \rangle$. 
We assume the input satisfies $\text{Vars}(\text{rhs}(P)) \subseteq \mathcal{X}$ (notice that to obtain a nominal AC-unification algorithm, we would have to eliminate this hypothesis from the proofs).
The output is a list of solutions, each of the form $\langle \Gamma_1, \sigma_1 \rangle$. 
The AC part of the algorithm (ACMatch) is handled by function applyACStep, which relies on two functions: solveAC and instantiateStep.

- **solveAC** builds the linear Diophantine equational system associated with the AC-matching equational constraint, generates the basis of solutions, and uses these solutions to generate the new AC-matching equational constraints.
- **instantiateStep** instantiates the moderated variables that it can.
Idea: for the particular case of matching (unlike unification) all the new moderated variables introduced by \texttt{solveAC} are instantiated by \texttt{instantiateStep}.
Hence, termination is much easier in nominal AC-matching than in first-order AC-unification.
\( \nabla' \vdash \nabla \sigma \) denotes that \( \nabla' \vdash a\#X\sigma \) holds for each \((a\#X) \in \nabla \).

\( \nabla \vdash \sigma \equiv_{\mathcal{V}} \sigma' \) denotes that \( \nabla \vdash X\sigma \equiv_{\alpha} X\sigma' \) for all \( X \) in \( \mathcal{V} \). When \( \mathcal{V} \) is the set of all variables \( X \), we write \( \nabla \vdash \sigma \equiv \sigma' \).
Our algorithm receives as input quintuples. Hence, to state the theorems of soundness and completeness, we need the definition of a solution $\langle \Delta, \delta \rangle$ to a quintuple $\langle \Gamma, P, \sigma, V, \mathcal{X} \rangle$. 
Definition 2 (Solution for a Quintuple)

A solution to a quintuple $\langle \Gamma, P, \sigma, V, \mathcal{X} \rangle$ is a pair $\langle \Delta, \delta \rangle$, where the following conditions are satisfied:

1. $\Delta \vdash \Gamma \delta$.
2. If $a \not\approx t \in P$ then $\Delta \vdash a \not\approx t \delta$.
3. If $t \approx s \in P$ then $\Delta \vdash t \delta \approx_\alpha s \delta$.
4. There exists $\lambda$ such that $\Delta \vdash \lambda \circ \sigma \approx V \delta$.
5. $\text{dom}(\delta) \cap \mathcal{X} = \emptyset$. 
Note that if $\langle \Delta, \delta \rangle$ is a solution of $\langle \Gamma, \emptyset, \sigma, X, \mathcal{X} \rangle$ this corresponds to the notion of $\langle \Delta, \delta \rangle$ being an instance of $\langle \Gamma, \sigma \rangle$ that does not instantiate variables in $\mathcal{X}$. 
Theorem 3 (Soundness for AC-Matching)

Let the pair $\langle \Gamma_1, \sigma_1 \rangle$ be an output of $\text{ACMatch}(\langle \emptyset, \{ t \approx s \}, \text{id}, \text{Vars}(t, s), \text{Vars}(s) \rangle)$. 

If $\langle \Delta, \delta \rangle$ is an instance of $\langle \Gamma_1, \sigma_1 \rangle$ that does not instantiate the variables in $s$, then 

$\langle \Delta, \delta \rangle$ is a solution to $\langle \emptyset, \{ t \approx s \}, \text{id}, \overline{X}, \text{Vars}(s) \rangle$. 

An interpretation of the previous Theorem is that if $\langle \Delta, \delta \rangle$ is an AC-matching instance to one of the outputs of $\text{ACMatch}$, then $\langle \Delta, \delta \rangle$ is an AC-matching solution to the original problem.
Theorem 4 (Completeness for AC-Matching)

Suppose that \( \langle \Delta, \delta \rangle \) is a solution to \( \langle \emptyset, \{ t \approx ? s \}, \text{id}, \mathbb{X}, \text{Vars}(s) \rangle \), that \( \delta \subseteq V \) and that \( \text{Vars}(\Delta) \subseteq V \).

Then, there exists

\[
(\langle \Gamma, \sigma \rangle \in \text{ACMatch}(\langle \emptyset, \{ t \approx ? s \}, \text{id}, V, \text{Vars}(s) \rangle))
\]

such that \( \langle \Delta, \delta \rangle \) is an instance (restricted to the variables of \( V \)) of \( \langle \Gamma, \sigma \rangle \) that does not instantiate the variables of \( s \).
An interpretation of the previous Theorem is that if $\langle \Delta, \delta \rangle$ is an AC-matching solution to the initial problem, then $\langle \Delta, \delta \rangle$ is an AC-matching instance of one of the outputs of ACMatch.
The hypotheses $\delta \subseteq V$ and $\text{Vars}(\Delta) \subseteq V$ are just a technicality that was put to guarantee that the new variables introduced by the algorithm in the AC-part do not clash with the variables in $\text{dom}(\delta)$ or in the terms in $\text{im}(\delta)$ or in $\text{Vars}(\Delta)$. 
Synthesis on Nominal Equational Modulo
## Synthesis of results on Nominal Unification Modulo

<table>
<thead>
<tr>
<th>Theory</th>
<th>Unif. type</th>
<th>Equality-checking</th>
<th>Matching</th>
<th>Unification</th>
<th>Related work</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\approx_\alpha$</td>
<td>1</td>
<td>$O(n \log n)$</td>
<td>$O(n \log n)$</td>
<td>$O(n^2)$</td>
<td>UPG04 LV10 CF08 CF10 LSFA2015</td>
</tr>
<tr>
<td>C</td>
<td>$\infty$</td>
<td>$O(n^2 \log n)$</td>
<td>NP-comp.</td>
<td>NP-comp.</td>
<td>LOPSTR2017 FroCoS2017 TCS2019 LOPSTR2019 MSCS2021</td>
</tr>
<tr>
<td>A</td>
<td>$\infty$</td>
<td>$O(n \log n)$</td>
<td>NP-comp.</td>
<td>NP-hard</td>
<td>LSFA2016 TCS2019</td>
</tr>
<tr>
<td>AC</td>
<td>$\omega$</td>
<td>$O(n^3 \log n)$</td>
<td>NP-comp.</td>
<td>NP-comp.</td>
<td>LSFA2016 TCS2019 CICM2023</td>
</tr>
</tbody>
</table>
More on Nominal Reasoning

Also:

- Overlaps in Nominal Rewriting [LSFA 2015]
- Nominal Narrowing [FSCD 2016]
- Nominal Intersection Types [TCS 2018]
- Nominal Disequations [LSFA 2019]
- Nominal Syntax with Permutation Fixed Points [LMCS2020]
Work in Progress and Future Work
Removing the hypotheses $\delta \subseteq V$ and $\text{Vars}(\Delta) \subseteq V$ in the statement of completeness.

**Table 4: Quantitative Data.**

<table>
<thead>
<tr>
<th>Theory</th>
<th>Theorems</th>
<th>TCCs</th>
<th>Size (.pvs)</th>
<th>Size (.prf)</th>
<th>Size (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>[AFFKS23]</td>
<td>6</td>
<td>4</td>
<td>2.8 kB</td>
<td>0.02 MB</td>
<td>&lt; 1%</td>
</tr>
<tr>
<td>unification_alg</td>
<td>11</td>
<td>19</td>
<td>6.9 kB</td>
<td>2.1 MB</td>
<td>9%</td>
</tr>
<tr>
<td>ac_step</td>
<td>45</td>
<td>11</td>
<td>15.8 kB</td>
<td>1.6 MB</td>
<td>7%</td>
</tr>
<tr>
<td>inst_step</td>
<td>75</td>
<td>17</td>
<td>20.3 kB</td>
<td>2 MB</td>
<td>9%</td>
</tr>
<tr>
<td>aux.unification</td>
<td>140</td>
<td>52</td>
<td>44.9 kB</td>
<td>6.9 MB</td>
<td>30%</td>
</tr>
<tr>
<td>Diophantine</td>
<td>77</td>
<td>44</td>
<td>23.5 kB</td>
<td>1 MB</td>
<td>4%</td>
</tr>
<tr>
<td>unification</td>
<td>119</td>
<td>13</td>
<td>28.0 kB</td>
<td>1.7 MB</td>
<td>8%</td>
</tr>
<tr>
<td>fresh_subs</td>
<td>37</td>
<td>5</td>
<td>10.9 kB</td>
<td>0.6 MB</td>
<td>3%</td>
</tr>
<tr>
<td>substitution</td>
<td>166</td>
<td>34</td>
<td>30.1 kB</td>
<td>2.5 MB</td>
<td>11%</td>
</tr>
<tr>
<td>equality</td>
<td>83</td>
<td>20</td>
<td>15.1 kB</td>
<td>1.6 MB</td>
<td>7%</td>
</tr>
<tr>
<td>freshness</td>
<td>15</td>
<td>10</td>
<td>4.5 kB</td>
<td>0.1 MB</td>
<td>&lt; 1%</td>
</tr>
<tr>
<td>terms</td>
<td>147</td>
<td>53</td>
<td>29.1 kB</td>
<td>1.1 MB</td>
<td>5 %</td>
</tr>
<tr>
<td>atoms</td>
<td>14</td>
<td>3</td>
<td>3.7 kB</td>
<td>0.03 MB</td>
<td>&lt; 1 %</td>
</tr>
<tr>
<td>list</td>
<td>265</td>
<td>113</td>
<td>54.9 kB</td>
<td>1.4 MB</td>
<td>6 %</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>1200</td>
<td>398</td>
<td>290.5 kB</td>
<td>22.6 MB</td>
<td>100%</td>
</tr>
</tbody>
</table>

The approach in progress is similar to the one applied for removing variables to the first-order AC-unification algorithm formalization in [FSCD2022].
Future Work

Study how to avoid the circularity in nominal AC-unification.

- How circularity enriches the set of computed solutions?
- Under which conditions can circularity be avoided?

Consider the alternative approach to AC-unification proposed by Boudet, Contejean and Devie [BCD90, Bou93], which was used to define AC higher-order pattern unification.

Explore the connection between nominal and higher-order patterns to obtain a nominal AC-unification algorithm.
Thank you!
