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

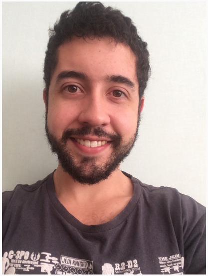
Universidade de Brasília

† Research supported by the Brazilian agencies CAPES, CNPq, and FAPDF

Joint Work With


Ana C. Rocha Oliveira

Washington L. de Carvalho



Gabriel Ferreira Silva


Maribel Fernández


Daniele Nantes-Sobrinho


Temur Kutsia

Outline

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 σ such that $s\sigma = t$?
- **Unification**: There exists σ such that $s\sigma = t\sigma$?

s and t are *terms* in some *signature* and σ is a *substitution*.

Equational Problems - Syntactic Unification

- 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 $f(b, y)$

Equational Problems - Syntactic Unification

- 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 $\{x/b, y/a\}$.

Equational Problems - Syntactic unification

- $\mathcal{F}$ 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.
- $\sigma, \gamma, \delta : \mathcal{V} \to \mathcal{T}(\mathcal{F}, \mathcal{V})$ set of substitutions.

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

Equational Problems - Syntactic Unification

Example:

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

σ is *more general* than γ:

$$\text{there exists } \delta \text{ such that } \sigma\delta = \gamma;$$

$$\delta = \{y/b\}.$$

Equational Problems - Syntactic Unification

Goal: *algorithm* that *unifies* terms.

Example:

- $h(\underbrace{x}, y, z) = h(\underbrace{f(w, w)}, f(x, x), f(y, y))$

Goal: *algorithm* that *unifies* terms.

Example:

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

Equational Problems - Syntactic Unification

Goal: *algorithm* that *unifies* terms.

Example:

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

Equational Problems - Syntactic Unification

Goal: *algorithm* that *unifies* terms.

Example:

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

Equational Problems - Unification Modulo

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*.

Equational Problems - Unification Modulo

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*.

Equational Problems - Unification Modulo

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), $f(x,x) \approx x$:

- $f(x, f(y,x)) = f(f(x,z),x)$?

The unification problem is of type *zero* (Schmidt-Schauß 1986, Baader 1986).

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), $h(x+y) \approx h(x) + h(y)$:

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

The unification problem is of type *zero* and undecidable (Narendran 1996). The same happens for ACUh (Nutt 1990, Baader 1993).

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 ACUh (Nutt 1990, Baader 1993).

Motivation

Synthesis on Unification modulo

Synthesis Unification modulo i

		Synthesis Unification modulo			
Theory	Unif. type	Equality-checking	Matching	Unification	Related work
Syntactic	1	$O(n)$	$O(n)$	$O(n)$	R65 MM76 PW78
C	ω	$O(n^2)$	NP-comp.	NP-comp.	BKN87 KN87
A	∞	$O(n)$	NP-comp.	NP-hard	M77 BKN87
AU	∞	$O(n)$	NP-comp.	decidable	M77 KN87
AI	0	$O(n)$	NP-comp.	NP-comp.	Klíma02 SS86 Baader86

Synthesis Unification modulo

		Synthesis Unification modulo			
Theory	Unif. type	Equality-checking	Matching	Unification	Related work
AC	ω	$O(n^3)$	NP-comp.	NP-comp.	BKN87 KN87 KN92
ACU	ω	$O(n^3)$	NP-comp.	NP-comp.	KN92
AC(U)I	ω	-	-	NP-comp.	KN92 BMMO20
D	ω	-	NP-hard	NP-hard	TA87
ACh	0	-	-	undecidable	B93 N96 EL18
ACUh	0	-	-	undecidable	B93 N96

Bindings and Nominal Syntax

Systems with Bindings

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.

Nominal

The nominal setting extends first-order syntax, replacing the concept of syntactical equality by α-equivalence, which let us represent smoothly those systems.

Profiting from the nominal paradigm implies adapting basic notions (substitution, rewriting, equality) to it.

Atoms and Variables

Consider a set of variables $\mathbb{X} = \{X, Y, Z, \ldots\}$ and a set of atoms $\mathbb{A} = \{a, b, c, \ldots\}$.

Nominal Terms

Definition 1 (Nominal Terms)
Nominal terms are inductively generated according to the grammar:

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

where π 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

Freshness predicate

$a\#t$ 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 Δ, Γ or ∇.

Permutations

An atom permutation π represents an exchange of a finite amount of atoms in $\mathbb{A}$ and is presented by a list of swappings:

$$\pi = (a_1\ b_1) :: ... :: (a_n\ b_n) :: nil$$

Examples of Permutation Actions

Permutations act on atoms and terms:

- $(a\ b) \cdot a = b$;
- $(a\ b) \cdot b = a$;
- $(a\ b) \cdot f(a, c) = f(b\ c)$;
- $(a\ b) :: (b\ c) \cdot [a]\langle a, c\rangle = (b\ c)[b]\langle b, c\rangle = [c]\langle c, b\rangle$.

Intuition Behind the Concepts

Two important predicates are the *freshness* predicate $\#$, and the α-*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 α-equivalent.

Contexts

A *context* is a set of constraints of the form $a\#X$. Contexts are denoted by the letters Δ, ∇ or Γ.

Advantages of the name binding nominal approach

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

Example

β and η rules as nominal rewriting rules:

$$\boxed{\begin{array}{ll} app\langle lam[a]M, N\rangle \to subs\langle [a]M, N\rangle & (\beta) \\ a\#M \vdash lam[a]app\langle M, a\rangle \to M & (\eta) \end{array}}$$

Some substitution rules:

$$\boxed{\begin{array}{l} b\#M \vdash subs\langle [b]M, N\rangle \to M \\ a\#N \vdash subs\langle [b]lam[a]M, N\rangle \to lam[a]sub\langle [b]M, N\rangle \\ c\#M, c\#N \vdash subs\langle [b]lam[a]M, N\rangle \to lam[c]sub\langle [b](a\ c) \cdot M, N\rangle \end{array}}$$

Advantages of the name binding nominal approach

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

Derivation Rules for Freshness

$$\frac{}{\Delta \vdash a\#\langle\rangle}\,(\#\langle\rangle) \qquad \frac{}{\Delta \vdash a\#b}\,(\#atom)$$

$$\frac{(\pi^{-1}(a)\#X) \in \Delta}{\Delta \vdash a\#\pi \cdot X}\,(\#X) \qquad \frac{}{\Delta \vdash a\#[a]t}\,(\#[a]a)$$

$$\frac{\Delta \vdash a\#t}{\Delta \vdash a\#[b]t}\,(\#[a]b) \qquad \frac{\Delta \vdash a\#s \quad \Delta \vdash a\#t}{\Delta \vdash a\#\langle s, t\rangle}\,(\#pair)$$

$$\frac{\Delta \vdash a\#t}{\Delta \vdash a\#f\ \ t}\,(\#app)$$

Derivation Rules for alpha-Equivalence

$$\frac{}{\Delta \vdash \langle\rangle \approx_\alpha \langle\rangle}\,(\approx_\alpha \langle\rangle) \qquad \frac{}{\Delta \vdash a \approx_\alpha a}\,(\approx_\alpha \mathit{atom})$$

$$\frac{\Delta \vdash s \approx_\alpha t}{\Delta \vdash fs \approx_\alpha ft}\,(\approx_\alpha \mathit{app}) \qquad \frac{\Delta \vdash s \approx_\alpha t}{\Delta \vdash [a]s \approx_\alpha [a]t}\,(\approx_\alpha [a]a)$$

$$\frac{\Delta \vdash s \approx_\alpha (a\ b)\cdot t,\ a\#t}{\Delta \vdash [a]s \approx_\alpha [b]t}\,(\approx_\alpha [a]b) \qquad \frac{ds(\pi,\pi')\#X \subseteq \Delta}{\Delta \vdash \pi\cdot X \approx_\alpha \pi'\cdot X}\,(\approx_\alpha \mathit{var})$$

$$\frac{\Delta \vdash s_0 \approx_\alpha t_0,\ \ \Delta \vdash s_1 \approx_\alpha t_1}{\Delta \vdash \langle s_0, s_1\rangle \approx_\alpha \langle t_0, t_1\rangle}\,(\approx_\alpha \mathit{pair})$$

Additional Rule for alpha-Equivalence with C Functions

Let f be a C function symbol.

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

$$\frac{\Delta \vdash s_2 \approx_\alpha t_1 \qquad \Delta \vdash s_1 \approx_\alpha t_2}{\Delta \vdash f^C\langle s_1, s_2\rangle \approx_\alpha f^C\langle t_1, t_2\rangle}$$

Additional Rule for alpha-Equivalence with AC Functions

Let f be an AC function symbol.

We add rule ($\approx_\alpha$ *ac-app*) for dealing with AC functions:

$$\frac{\Delta \vdash S_i(f^{AC}s) \approx_\alpha S_j(f^{AC}t) \quad \Delta \vdash D_i(f^{AC}s) \approx_\alpha D_j(f^{AC}t)}{\Delta \vdash f^{AC}s \approx_\alpha f^{AC}t}$$

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

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

$$
\dfrac{\dfrac{\dfrac{}{a \approx_\alpha a}\,(\approx_\alpha \mathit{atom}) \qquad \dfrac{\dfrac{}{a \approx_\alpha a}\,(\approx_\alpha \mathit{atom})}{fa \approx_\alpha fa}\,(\approx_\alpha \mathit{app})}{\oplus\langle a, fa\rangle \approx_\alpha (a\ b) \cdot \oplus\langle fb, b\rangle}\,(\approx_\alpha \mathit{c\text{-}app}) \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{a\#b}\,(\#\mathit{atom})}{a\#fb}\,(\#\mathit{app}) \qquad \dfrac{}{a\#b}\,(\#\mathit{atom})}{a\#\langle fb, b\rangle}\,(\#\mathit{pair})}{a\# \oplus \langle fb, b\rangle}\,(\#\mathit{app})}{\dfrac{[a] \oplus \langle a, fa\rangle \approx_\alpha [b] \oplus \langle fb, b\rangle}{\forall[a] \oplus \langle a, fa\rangle \approx_\alpha \forall[b] \oplus \langle fb, b\rangle}\,(\approx_\alpha \mathit{app})}\,(\approx_\alpha [a]b)
$$

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 $\pi \cdot X \approx_\alpha{}^? X$ may have infinite independent solutions.

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

has solutions: $\begin{cases} \langle\{a\#X, b\#X\}, id\rangle, \\ \langle\emptyset, \{X/a \oplus b\}\rangle, \langle\emptyset, \{X/a \star b\}\rangle, \ldots \\ \langle\{a\#Z, b\#Z\}, \{X/(a \oplus b) \oplus Z\}\rangle, \ldots \\ \langle\emptyset, \{X/(a \oplus b) \star (b \oplus a)\}\rangle, \ldots \end{cases}$

Issues Adapting First-Order to Nominal AC-Unification

Our Work in First-Order AC-Unification in a Nutshell

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].

An Example

Let f be an AC function symbol. The solutions that come to mind when unifying:

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

are:

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

Are there other solutions?

An Example

Yes!

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

Example

the **AC Step** for AC-unification.

How do we generate a complete set of unifiers for:

$$f(X, X, Y, a, b, c) \approx^{?} f(b, b, b, c, Z)$$

Stickel-Fages AC-unification - eliminating Common Arguments

Eliminate common arguments in the terms we are trying to unify.

Now, we must unify

$$f(X, X, Y, a) \approx^{?} f(b, b, Z)$$

Stickel-Fages AC-unification - introducing a Linear equation

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.

Stickel-Fages AC-unification - building a basis of solutions

Generate a basis of solutions to the linear equation.

Table 1: Solutions for the Equation $2X_1 + X_2 + X_3 = 2Y_1 + Y_2$

X_1	X_2	X_3	Y_1	Y_2	$2X_1 + X_2 + X_3$	$2Y_1 + Y_2$
0	0	1	0	1	1	1
0	1	0	0	1	1	1
0	0	2	1	0	2	2
0	1	1	1	0	2	2
0	2	0	1	0	2	2
1	0	0	0	2	2	2
1	0	0	1	0	2	2

Stickel-Fages AC-unification - associating new variables

Associate new variables with each solution.

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

X_1	X_2	X_3	Y_1	Y_2	$2X_1 + X_2 + X_3$	$2Y_1 + Y_2$	**New Variables**
0	0	1	0	1	1	1	Z_1
0	1	0	0	1	1	1	Z_2
0	0	2	1	0	2	2	Z_3
0	1	1	1	0	2	2	Z_4
0	2	0	1	0	2	2	Z_5
1	0	0	0	2	2	2	Z_6
1	0	0	1	0	2	2	Z_7

Stickel-Fages AC-unification - old and new variables

Observing the previous Table, relate the "old" variables and the "new" ones:

$$
\begin{aligned}
X_1 &\approx^? Z_6 + Z_7 \\
X_2 &\approx^? Z_2 + Z_4 + 2Z_5 \\
X_3 &\approx^? Z_1 + 2Z_3 + Z_4 \\
Y_1 &\approx^? Z_3 + Z_4 + Z_5 + Z_7 \\
Y_2 &\approx^? Z_1 + Z_2 + 2Z_6
\end{aligned}
$$

Stickel-Fages AC-unification - all the possible cases

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.

Stickel-Fages AC-unification - dropping impossible cases

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 $f(Z_1, Z_4)$.

Stickel-Fages AC-unification - dropping more cases

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:

$$\{X \approx^? Z_6, Y \approx^? Z_4, a \approx^? Z_4, b \approx^? Z_4, Z \approx^? f(Z_6, Z_6)\}$$

Stickel-Fages AC-unification - solutions

In our example,

$$f(X, X, Y, a, b, c) \approx^? f(b, b, b, c, Z)$$

the solutions are:

$$\left\{ \begin{array}{l} \sigma_1 = \{Y \to f(b, b), Z \to f(a, X, X)\} \\ \sigma_2 = \{Y \to f(Z_2, b, b), Z \to f(a, Z_2, X, X)\} \\ \sigma_3 = \{X \to b, Z \to f(a, Y)\} \\ \sigma_4 = \{X \to f(Z_6, b), Z \to f(a, Y, Z_6, Z_6)\} \end{array} \right\}$$

Adapting first-order AC-unification to nominal AC-unification

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 $U_1 + U_2 = V_1 + V_2$

U_1	U_2	V_1	V_2	$U_1 + U_2$	$V_1 + V_2$	New variables
0	1	0	1	1	1	Z_1
0	1	1	0	1	1	W_1
1	0	0	1	1	1	Y_1
1	0	1	0	1	1	X_1

After `solveAC`

$$\{X \approx^? X_1, W \approx^? Z_1, \pi \cdot X \approx^? X_1, \pi \cdot Y \approx^? Z_1\}$$
$$\{X \approx^? Y_1, W \approx^? W_1, \pi \cdot X \approx^? W_1, \pi \cdot Y \approx^? Y_1\}$$
$$\{X \approx^? Y_1 + X_1, W \approx^? W_1, \pi \cdot X \approx^? W_1 + X_1, \pi \cdot Y \approx^? Y_1\}$$
$$\{X \approx^? Y_1 + X_1, W \approx^? Z_1, \pi \cdot X \approx^? X_1, \pi \cdot Y \approx^? Z_1 + Y_1\}$$
$$\{X \approx^? X_1, W \approx^? Z_1 + W_1, \pi \cdot X \approx^? W_1 + X_1, \pi \cdot Y \approx^? Z_1\}$$
$$\{X \approx^? Y_1, W \approx^? Z_1 + W_1, \pi \cdot X \approx^? W_1, \pi \cdot Y \approx^? Z_1 + Y_1\}$$
$$\{X \approx^? Y_1 + X_1, W \approx^? Z_1 + W_1, \pi \cdot X \approx^? W_1 + X_1, \pi \cdot Y \approx^? Z_1 + Y_1\}$$

After `instantiateStep`

Seven branches are generated:

$B1 - \{\pi \cdot X \approx^? X\}, \sigma = \{W \mapsto \pi \cdot Y\}$

$B2 - \sigma = \{W \mapsto \pi^2 \cdot Y, X \mapsto \pi \cdot Y\}$

$B3 - \{f(\pi^2 \cdot Y, \pi \cdot X_1) \approx^? f(W, X_1)\}, \sigma = \{X \mapsto f(\pi \cdot Y, X_1)\}$

$B4 -$ *No solution*

$B5 -$ *No solution*

$B6 - \sigma = \{W \mapsto f(Z_1, \pi \cdot X), Y \mapsto f(\pi^{-1} \cdot Z_1, \pi^{-1} \cdot X)\}$

$B7 - \{f(\pi \cdot Y_1, \pi \cdot X_1) \approx^? f(W_1, X_1)\},$

$\sigma = \{X \mapsto f(Y_1, X_1),\ W \mapsto f(Z_1, W_1), Y \mapsto f(\pi^{-1} \cdot Z_1, \pi^{-1} \cdot Y_1)\}$

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

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]).

From unification to matching using protected variables

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.
- α-**Equivalence** - By putting $\mathcal{X}$ as the set of variables that appear in the problem.

From First-Order AC-Unification to Nominal AC-Matching

We modify our first-order AC-unification formalisation to obtain a formalised algorithm for nominal AC-matching.

Input

The algorithm is recursive and needs to keep track of

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

Hence, it's input is a quintuple $\langle \Gamma, P, \sigma, V, \mathcal{X} \rangle$.

Set of protected variables for matching problems

We assume the input satisfies $\mathit{Vars}(\mathit{rhs}(P)) \subseteq \mathcal{X}$ (notice that to obtain a nominal AC-unification algorithm, we would have to eliminate this hypothesis from the proofs).

Output

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 `solveAC` are instantiated by `instantiateStep`.

Formalisation Nominal AC-matching - Termination is Easier

Hence, termination is much easier in nominal AC-matching than in first-order AC-unification.

Notation

$\nabla' \vdash \nabla\sigma$ denotes that $\nabla' \vdash a\#X\sigma$ holds for each $(a\#X) \in \nabla$.

$\nabla \vdash \sigma \approx_V \sigma'$ denotes that $\nabla \vdash X\sigma \approx_\alpha X\sigma'$ for all X in V. When V is the set of all variables $\mathbb{X}$, we write $\nabla \vdash \sigma \approx \sigma'$.

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 $a\#_{?}t \in P$ then $\Delta \vdash a\#t\delta$.
3. if $t \approx_{?} s \in P$ then $\Delta \vdash t\delta \approx_{\alpha} s\delta$.
4. there exists λ such that $\Delta \vdash \lambda \circ \sigma \approx_{V} \delta$.
5. $dom(\delta) \cap \mathcal{X} = \emptyset$.

Note that if $\langle \Delta, \delta \rangle$ is a solution of $\langle \Gamma, \emptyset, \sigma, \mathbb{X}, \mathcal{X} \rangle$ this corresponds to the notion of $\langle \Delta, \delta \rangle$ being an instance of $\langle \Gamma, \sigma \rangle$ that does not instantiate variables in $\mathcal{X}$.

Formalisation Nominal AC-matching - Soundness

Theorem 3 (Soundness for AC-Matching)

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

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

$$\langle \Delta, \delta \rangle \textit{ is a solution to } \langle \emptyset, \{t \approx_? s\}, id, \mathbb{X}, Vars(s) \rangle.$$

Interpretation for Soundness

An interpretation of the previous Theorem is that if $\langle \Delta, \delta \rangle$ is an AC-matching instance to one of the outputs of `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* $\langle \emptyset, \{t \approx_? s\}, id, \mathbb{X}, Vars(s) \rangle$, *that* $\delta \subseteq V$ *and that* $Vars(\Delta) \subseteq V$.

Then, there exists

$$(\langle \Gamma, \sigma \rangle \in \mathtt{ACMatch}(\langle \emptyset, \{t \approx_? s\}, id, V, Vars(s) \rangle)$$

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.

Interpretation for Completeness

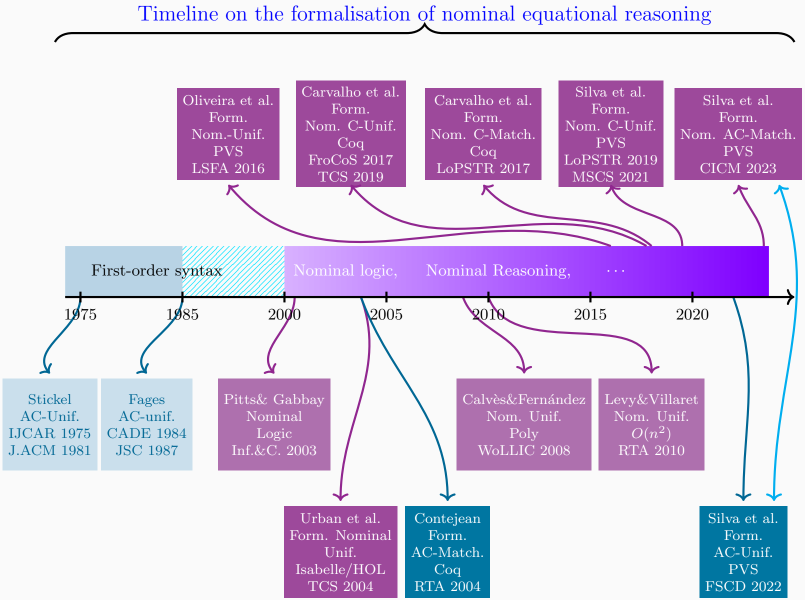
An interpretation of the previous Theorem is that if $\langle \Delta, \delta \rangle$ is an AC-matching solution to the initial problem, then $\langle \Delta, \delta \rangle$ is an AC-matching instance of one of the outputs of `ACMatch`.

Formalisation Nominal AC-matching - The hypotheses on variables

The hypotheses $\delta \subseteq V$ and $Vars(\Delta) \subseteq V$ 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 $dom(\delta)$ or in the terms in $im(\delta)$ or in $Vars(\Delta)$.

Synthesis on Nominal Equational Modulo

Synthesis on Nominal Equational Modulo



Synthesis of results on Nominal Unification Modulo

		Synthesis Unification Nominal Modulo			
Theory	Unif. type	Equality-checking	Matching	Unification	Related work
$\approx_\alpha$	1	$O(n \log n)$	$O(n \log n)$	$O(n^2)$	UPG04 LV10 CF08 CF10 LSFA2015
C	∞	$O(n^2 \log n)$	NP-comp.	NP-comp.	LOPSTR2017 FroCoS2017 TCS2019 LOPSTR2019 MSCS2021
A	∞	$O(n \log n)$	NP-comp.	NP-hard	LSFA2016 TCS2019
AC	ω	$O(n^3 \log n)$	NP-comp.	NP-comp.	LSFA2016 TCS2019 CICM2023

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 $Vars(\Delta) \subseteq V$ in the statement of completeness.

Table 4: Quantitative Data.

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

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

- 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

Thank you!

- Mauricio Ayala-Rincón, Maribel Fernández, Gabriel Ferreira Silva, and Daniele Nantes Sobrinho, *A Certified Algorithm for AC-Unification*, Formal Structures for Computation and Deduction, FSCD 2022 (2022).
- Alexandre Boudet, Evelyne Contejean, and Hervé Devie, *A New AC Unification Algorithm with an Algorithm for Solving Systems of Diophantine Equations*, Proceedings of the Fifth Annual Symposium on Logic in Computer Science, LICS, 1990.
- Alexandre Boudet, *Competing for the AC-Unification Race*, J. of Autom. Reasoning (1993).