Nominal AC-Matching

KCL - UIUC 7th Meeting

Gabriel Ferreira Silva
Presented by: Mauricio Ayala-Rincón

(Universidade de Brasília - UnB)

May 12, 2023

Joint Work With

Figure: Maribel Fernández

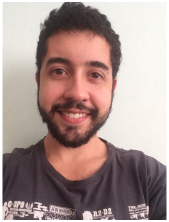
Figure: Gabriel Ferreira Silva

Figure: Daniele Nantes

Figure: Temur Kutsia

Outline

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:

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

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 ∇.

Unification

Unification consists of “finding a way” to equal two terms by instantiating variables.

$$\boxed{s \approx_? t \quad \rightsquigarrow \quad s\sigma \approx t\sigma}$$

- $f(a, X)$ and $f(Y, b)$ can be made equal by “sending” X to b and Y to a, as they both become $f(a, b)$.

Matching

Matching can be seen as a simpler version of unification, where the terms on the right-hand side do not contain variables that can be instantiated.

$$\boxed{s \approx_? t \quad \rightsquigarrow \quad s\sigma \approx t}$$

Matching has applications in rewriting, functional programming, and metaprogramming.

In this Talk

1. Discuss first-order AC-unification briefly.
2. Describe how we adapted our first-order AC-unification formalisation ([AFSS22]) to give a formalised algorithm for nominal AC-matching.
3. Discuss possible future work on nominal AC-unification.

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

We modified Stickel'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].

What is Tricky About AC? 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?

What is Tricky About AC? An Example

Yes!

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

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 via $\mathcal{X}$

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 $(\Gamma, P, \sigma, V, \mathcal{X})$.

$Vars(rhs(P)) \subseteq \mathcal{X}$

We assume the input satisfies $Vars(rhs(P)) \subseteq \mathcal{X}$ (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 (Γ_1, σ_1).

`applyACStep`

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.

Termination

Idea: for the particular case of matching (unlike unification) all the new moderated variables introduced by `solveAC` are instantiated by `instantiateStep`.

Termination is Easier

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

Notation

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

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

Solution to a Quintuple I

Our algorithm receives as input quintuples. Hence, to state the theorems of soundness and completeness, we need the definition of a solution (Δ, δ) to a quintuple $(\Gamma, P, \sigma, V, \mathcal{X})$.

Solution to a Quintuple II

Definition 2 (Solution for a Quintuple ↗)

A solution to a quintuple $(\Gamma, P, \sigma, V, \mathcal{X})$ is a pair (Δ, δ), where the following conditions are satisfied:

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

Solution to a Quintuple III

Note that if (Δ, δ) is a solution of $(\Gamma, \emptyset, \sigma, \mathbb{X}, \mathcal{X})$ this corresponds to the notion of (Δ, δ) being an instance of (Γ, σ) that does not instantiate variables in $\mathcal{X}$.

Soundness

Theorem 3 (Soundness for AC-Matching ↗)

Let the pair (Γ_1, σ_1) *be an output of* $\mathtt{ACMatch}(\emptyset, \{t \approx_? s\}, id, Vars(t, s), Vars(s))$.

If (Δ, δ) *is an instance of* (Γ_1, σ_1) *that does not instantiate the variables in* s*, then*

then (Δ, δ)*is a solution to* $(\emptyset, \{t \approx_? s\}, id, \mathbb{X}, Vars(s))$.

Interpretation for Soundness

An interpretation of the previous Theorem is that if (Δ, δ) is an AC-matching instance to one of the outputs of `ACMatch`, then (Δ, δ) is an AC-matching solution to the original problem.

Completeness

Theorem 4 (Completeness for AC-Matching ↗)
Suppose that (Δ, δ) is a solution to $(\emptyset, \{t \approx_? s\}, id, \mathbb{X}, Vars(s))$, that $\delta \subseteq V$ and that $Vars(\Delta) \subseteq V$.

Then, there exists

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

such that (Δ, δ) is an instance (restricted to the variables of V) of (Γ, σ) that does not instantiate the variables of s.

Interpretation for Completeness

An interpretation of the previous Theorem is that if (Δ, δ) is an AC-matching solution to the initial problem, then (Δ, δ) is an AC-matching instance of one of the outputs of `ACMatch`.

The hypotheses $\delta \subseteq V$ and $Vars(\Delta) \subseteq V$

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 $Vars(\Delta)$.

The Loop in Nominal AC-Unification

If we apply `ACMatch` to $f(X, W) \approx_? f(\pi \cdot X, \pi \cdot Y)$, where $X \notin \mathcal{X}$, we obtain a loop (more details in Appendix).

The problem happens when the same variable occurs as an argument of an AC operator multiple times with **different** suspended permutations.

A Different Approach to Nominal AC-Unification I

Idea: Explore the connection between nominal unification and higher-order pattern unification and the work of Boudet and Contejean in AC higher-order pattern unification [BC97].

Work in Progress I

Removing the hypotheses $\delta \subseteq V$ and $Vars(\Delta) \subseteq V$ in the statement of completeness.

Table: 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 [AFSS22].

Future Work

1. 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.
2. Explore the connection between nominal and higher order patterns to obtain a nominal AC-unification algorithm.

Thank You

Thank you! Any comments/suggestions/doubts?

References I

- 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 and Evelyne Contejean, *AC-Unification of Higher-Order Patterns*, Third International Conference on Principles and Practice of Constraint Programming CP97, 1997.
- 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).

The loop in $f(X, W) \approx_? f(\pi \cdot X, \pi \cdot Y)$

We found a loop while solving $f(X, W) \approx_? f(\pi \cdot X, \pi \cdot Y)$.

Table of Solutions

The Diophantine equation associated[1] 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: 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

[1]variable U_1 is associated with argument X, variable U_2 is associated with argument W, variable V_1 is associated with argument $\pi \cdot X$ and variable V_2 is associated with argument $\pi \cdot Y$.

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`

7 branches are generated:

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

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

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

$B4 - \textit{No solution}$

$B5 - \textit{No solution}$

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

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

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

The Loop

Focusing on *Branch7*, notice that the problem before the AC Step and the problem after the AC Step and instantiating the variables are:

$$P = \{f(X, W) \approx_? f(\pi \cdot X, \pi \cdot Y)\}$$
$$P_1 = \{f(X_1, W_1) \approx_? f(\pi \cdot X_1, \pi \cdot Y_1)\}$$