Towards Nominal AC-Unification

Gabriel Ferreira Silva

Advisor: Mauricio Ayala-Rincón

Co-Advisor: Maribel Fernández

Talk presented as a partial requirement for the PhD in Informatics

Jury: José Meseguer, Christian Urban, César Muñoz, Vander Alves

Department of Computer Science - Universidade de Brasília

January 26, 2024

During My PhD I Worked With

Mauricio Ayala-Rincón

Maribel Fernández

Daniele Nantes

Temur Kutsia

Overview

System 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, allowing us to nicely represent systems with bindings.

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

Unification

Unification consists of making two terms equivalent by instantiating variables. For instance, the unification problem $f(a, X) \approx^? f(Y, c)$ is solved by the substitution $\sigma = \{X \mapsto c, Y \mapsto a\}$.

Unification has applications in narrowing, type inference algorithms, theorem provers, ...

Matching

Matching can be seen as a simpler version of unification, where we cannot instantiate the variables of the term on the right-hand side.

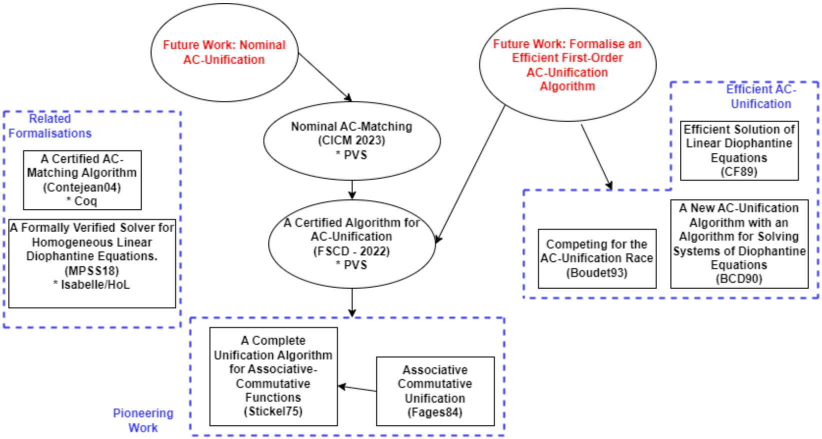
$$s \approx^? t \longrightarrow \sigma s \approx t$$

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

Our Work in a Nutshell

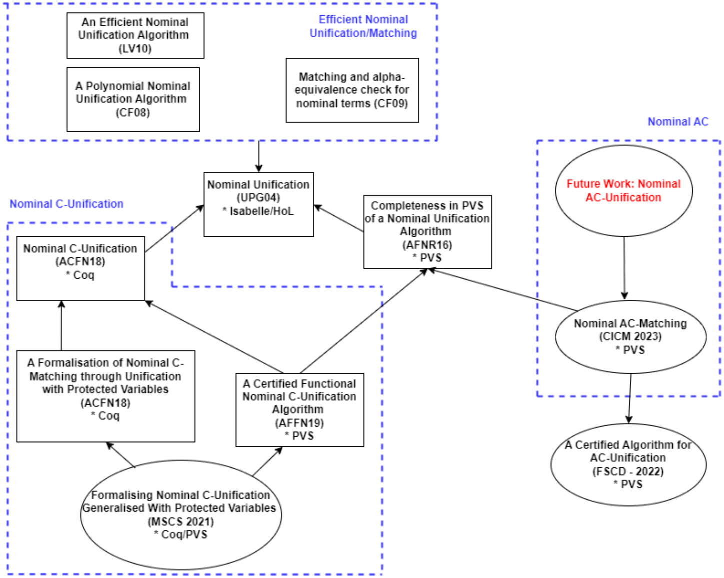
In a nutshell, our work is about nominal unification/matching in the presence of an equational theory E.

Related Work - AC-Unification



Future Work: Nominal AC-Unification
Future Work: Formalise an Efficient First-Order AC-Unification Algorithm
Related Formalisations
A Certified AC-Matching Algorithm (Contejean04) * Coq
A Formally Verified Solver for Homogeneous Linear Diophantine Equations. (MPSS18) * Isabelle/HoL
Nominal AC-Matching (CICM 2023) * PVS
A Certified Algorithm for AC-Unification (FSCD - 2022) * PVS
Efficient AC-Unification
Efficient Solution of Linear Diophantine Equations (CF89)
Competing for the AC-Unification Race (Boudet93)
A New AC-Unification Algorithm with an Algorithm for Solving Systems of Diophantine Equations (BCD90)
Pioneering Work
A Complete Unification Algorithm for Associative-Commutative Functions (Stickel75)
Associative Commutative Unification (Fages84)

Related Work - Nominal



Efficient Nominal Unification/Matching
An Efficient Nominal Unification Algorithm (LV10)
A Polynomial Nominal Unification Algorithm (CF08)
Matching and alpha-equivalence check for nominal terms (CF09)
Nominal Unification (UPG04) * Isabelle/HoL
Completeness in PVS of a Nominal Unification Algorithm (AFNR16) * PVS
Nominal AC
Future Work: Nominal AC-Unification
Nominal AC-Matching (CICM 2023) * PVS
A Certified Algorithm for AC-Unification (FSCD - 2022) * PVS
Nominal C-Unification
Nominal C-Unification (ACFN18) * Coq
A Formalisation of Nominal C-Matching through Unification with Protected Variables (ACFN18) * Coq
A Certified Functional Nominal C-Unification Algorithm (AFFN19) * PVS
Formalising Nominal C-Unification Generalised With Protected Variables (MSCS 2021) * Coq/PVS

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

How do we find all solutions to

$$f(X, X, Y, a, b, c) \approx^? f(b, b, b, c, Z)?[1].$$

[1] This example was taken from [1]

Eliminate Common Arguments

1. Eliminate common arguments in the term we are trying to unify.

We obtain $f(X, X, Y, a) \approx^? f(b, b, Z)$.

The Connection with Linear Equations in $\mathbb{N}$

2. According to the number of times each argument appear in the terms, transform the unification problem into a linear equation on $\mathbb{N}$.

After this step, our equation is:

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

where variable X_1 corresponds to argument X, variable X_2 corresponds to argument Y and so on.

Basis of Solution and New Variables

3. Generate a basis of solutions to the linear equation and associate a new variable with each solution.

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

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

Relating Old and New Variables

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

After this step, we obtain:

$$X_1 \approx^? Z_6 + Z_7$$
$$X_2 \approx^? Z_2 + Z_4 + 2Z_5$$
$$X_3 \approx^? Z_1 + 2Z_3 + Z_4$$
$$Y_1 \approx^? Z_3 + Z_4 + Z_5 + Z_7$$
$$Y_2 \approx^? Z_1 + Z_2 + 2Z_6$$

All Possible Cases

4. Decide whether we will include (set to 1) or not (set to 0) every "new" variable. Observe that every "old" variable must be different than zero.

In our example, we have $2^7 = 128$ 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, we have 69 cases.

Dropping Impossible Cases

5. Drop the cases where the variables that in fact represent 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 $f(Z_1, Z_4)$.

Replacing Variables and Proceeding

6. Replace 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:

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

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

The solutions are:

$$\left\{ \begin{array}{c} \{Y \to f(b,b), Z \to f(a, X, X)\} \\ \{Y \to f(Z_2, b, b), Z \to f(a, Z_2, X, X)\} \\ \{X \to b, Z \to f(a, Y)\} \\ \{X \to f(Z_6, b), Z \to f(a, Y, Z_6, Z_6)\} \end{array} \right\}$$

Nominal

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

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.

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 ∇.

α-equality

Our equality constraints $s \approx t$ take into account renaming of bound names.

For instance, let x and y be atoms, 0 and 1 be 0-ary function symbols (constants), the symbol $\forall$ be a unary function symbol, and $+, >$ be binary function symbols that we write infix. It is possible to prove that:

$$\forall[x](x + 1 > 0) \approx \forall[y](y + 1 > 0)$$

Some Tools We Used

We used the **PVS** proof assistant to do the formalisations of nominal C-unification, first-order AC-unification and nominal AC-matching.

We found useful to write structured proofs filled with details before proceeding to formalise a proof in PVS and for that we are using Leslie Lamport's `pf2` LaTeX package.

Figure 1: PVS logo

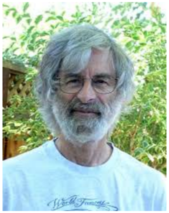


Figure 2: Leslie Lamport

NASALib

NASALib is the main repository for PVS formalisations. The formalisations described in this work are part of the **nominal** library of NASALib.

Nominal C-Unification With Protected Variables

We extended a functional nominal C-unification algorithm (`CUnif`) by adding a parameter $\mathcal{X}$ of protected variables, i.e., variables that cannot be instantiated.

Let P be the equational constraints.

- When $\mathcal{X} = \emptyset$ then `CUnif` performs unification.
- When $\mathcal{X} = \mathit{Vars}(\mathit{rhs}(P))$ then `CUnif` performs matching.

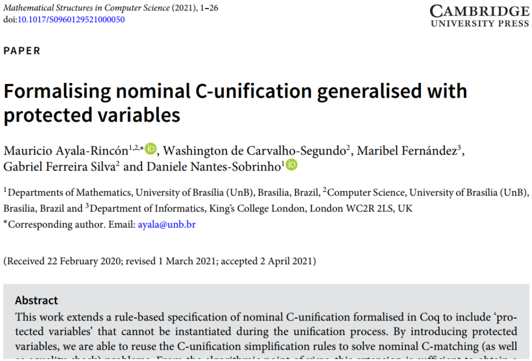
PVSio

PVSio is a PVS package that extends the ground evaluator with a predefined library of imperative programming language features, among them input and output operators.

We used the input/output capabilities of PVSio to test our manual Python implementation of the algorithm.

Nominal C-Unification With Protected Variables

This part of our work was published in the journal Mathematical Structures in Computer Science.



Mathematical Structures in Computer Science (2021), 1–26
doi:10.1017/S0960129521000050

CAMBRIDGE
UNIVERSITY PRESS

PAPER

Formalising nominal C-unification generalised with protected variables

Mauricio Ayala-Rincón[1,2,*], Washington de Carvalho-Segundo[2], Maribel Fernández[3], Gabriel Ferreira Silva[2] and Daniele Nantes-Sobrinho[1]

[1]Departments of Mathematics, University of Brasília (UnB), Brasilia, Brazil, [2]Computer Science, University of Brasília (UnB), Brasilia, Brazil and [3]Department of Informatics, King's College London, London WC2R 2LS, UK
*Corresponding author. Email: ayala@unb.br

(Received 22 February 2020; revised 1 March 2021; accepted 2 April 2021)

Abstract
This work extends a rule-based specification of nominal C-unification formalised in Coq to include 'protected variables' that cannot be instantiated during the unification process. By introducing protected variables, we are able to reuse the C-unification simplification rules to solve nominal C-matching (as well

Certified First Order AC-Unification

We modified Stickel's seminal AC-unification algorithm to avoid mutual recursion and formalised it in the PVS proof assistant. We proved the adjusted algorithm's termination, soundness and completeness.

Input and Output of Algorithm

The algorithm `ACUnif` is recursive and receives as input a triple (P, σ, V), where P is the current unification problem, σ is the substitution computed so far and V are the variables that are/were in the problem.

To unify two terms t and s the initial call is with $P = \{t \approx^? s\}$, $\sigma = Id$ and $V = Vars(t, s)$.

The output is a list of substitutions, where each substitution δ in this list is an AC-unifier of P.

Mutual Recursion

When specifying Stickel's algorithm we tried to follow closely the pseudocode from Fages' paper (see [2]) [2].

In [2] there are two functions:

- `uniAC` - unifies two terms t and s
- `unicompound` - unifies a list of terms $(t_1, \ldots, t_n)$ with a list of terms $(s_1, \ldots, s_n)$.

Those function are mutually recursive, something not allowed in PVS. We have adapted the algorithm to use only one main function.

[2] Stickel's paper [1] and [3] give a higher-level description

Termination

Almost a decade after Stickel's paper on AC-unification, Fages discovered and fixed a flaw in the proof of termination. Our proof of termination is based on Fages' proof.

Notation: $\delta \subseteq V$

Let V be a set of variables and δ a substitution. We say that $\delta \subseteq V$ if $dom(\delta) \subseteq V$ and $Vars(im(\delta)) \subseteq V$.

Completeness of ACUnif with $\delta \subseteq V$

Lemma 2 (Completeness of ACUnif with $\delta \subseteq V$)

Let V be a set of variables such that $\delta \subseteq V$ and $Vars(t, s) \subseteq V$. If δ unifies $t \approx^? s$, then there is a substitution $\gamma \in ACUnif(\{t \approx^? s\}, Id, V)$ such that $\gamma \leq_V \delta$.

The hypothesis $\delta \subseteq V$

The hypothesis $\delta \subseteq V$ in the lemma of completeness guarantees that the new variables introduced by `ACUnif` do not clash with the variables in $dom(\delta)$ or the variables in the terms in $im(\delta)$.

Completeness

Then, we removed hypothesis $\delta \subseteq V$ and proved:

Theorem 3 (Completeness of `ACUnif`)

If δ unifies $t \approx^{?} s$, then there is a substitution $\gamma \in \mathtt{ACUnif}(\{t \approx^{?} s\}, Id, Vars(t, s))$ such that $\gamma \leq_{Vars(t,s)} \delta$.

Applications

- Use our formalisation as a starting point to formalise more efficient first-order AC-unification algorithms.
- Use PVSio to test implemented AC-unification algorithms.

A Small Problem with PVSio

ACUnif relies on some functions that cannot be evaluated by PVSio. For instance,

```
divides(n, m): bool = EXISTS x : m = n * x
```

PVSio cannot be immediately used when the algorithm relies on code fragments such as divides that use the PVS reserved word EXISTS.

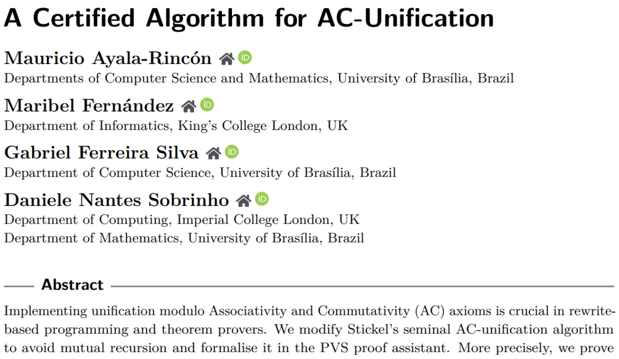
Solution: Semantic Attachments

We can use **semantic attachments** to execute some LISP code for every code fragment that cannot be evaluated by PVSio[3].

[3] Thanks César Muñoz for pointing this.

Certified First-Order AC-Unification and Applications

This part of our work was published in the FSCD 2022 conference and an extended version was submitted to the Journal of Automated Reasoning.



A Certified Algorithm for AC-Unification

Mauricio Ayala-Rincón
Departments of Computer Science and Mathematics, University of Brasília, Brazil

Maribel Fernández
Department of Informatics, King's College London, UK

Gabriel Ferreira Silva
Department of Computer Science, University of Brasília, Brazil

Daniele Nantes Sobrinho
Department of Computing, Imperial College London, UK
Department of Mathematics, University of Brasília, Brazil

Abstract

Implementing unification modulo Associativity and Commutativity (AC) axioms is crucial in rewrite-based programming and theorem provers. We modify Stickel's seminal AC-unification algorithm to avoid mutual recursion and formalise it in the PVS proof assistant. More precisely, we prove

From F.O. AC-Unification to Nominal AC-Matching

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

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

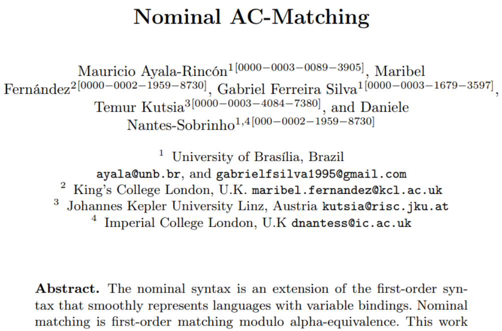
Idea: for the particular case of matching (unlike unification) all the new moderated variables introduced by `solveAC` are instantiated by `instantiateStep`.

Termination is Easier

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

Nominal AC-Matching

Our work on nominal AC-matching received the best paper award at the CICM 2023 conference.



Nominal AC-Matching

Mauricio Ayala-Rincón$^{1[0000-0003-0089-3905]}$, Maribel Fernández$^{2[0000-0002-1959-8730]}$, Gabriel Ferreira Silva$^{1[0000-0003-1679-3597]}$, Temur Kutsia$^{3[0000-0003-4084-7380]}$, and Daniele Nantes-Sobrinho$^{1,4[000-0002-1959-8730]}$

[1] University of Brasília, Brazil
ayala@unb.br, and gabrielfsilva1995@gmail.com
[2] King's College London, U.K. maribel.fernandez@kcl.ac.uk
[3] Johannes Kepler University Linz, Austria kutsia@risc.jku.at
[4] Imperial College London, U.K dnantess@ic.ac.uk

Abstract. The nominal syntax is an extension of the first-order syntax that smoothly represents languages with variable bindings. Nominal matching is first-order matching modulo alpha-equivalence. This work

Main Challenges in Nominal AC-Unification

We believe the two main challenges in nominal AC-unification will be:

- Solving fixpoint equations such as $\pi \cdot X \approx^? X$.
- Proving termination of problems such as $f(X, W) \approx^? f(\pi \cdot X, \pi \cdot Y)$.

Notation

From now on we may write a moderated variable $\pi \cdot X$ simply as πX.

The approach of nominal unification does not work

In nominal syntactic unification, the solution to $\pi X \approx^? X$ is $(dom(\pi)\#X, id)$. For instance, if $\pi = (a\ b)$ then the solution is

$$(\{a\#X, b\#X\}, id).$$

This approach is not complete in nominal AC-unification. For instance, if we returned only the previous solution we would miss

$$(\emptyset, X \mapsto +(a, b)).$$

Enumeration Procedure for Fixpoint Equations

We have devised a non-deterministic enumeration procedure given as a set of rules to generate all the solutions to $\pi X \approx^? X$.

It operates on triples of the form (Γ, σ, FP), where Γ is the context already computed, σ represents the instantiations we have done so far and FP is the set of equations.

The Set of Rules for Fixpoint Equations

- `(Var)`
- `(Func)`
- `(Abs a)` and `(Abs b)`
- `(AC Func)`

The (Var) Rule

$$(\Gamma, \sigma, \{\pi X \approx^{?} X\} \cup FP) \xrightarrow{(Var)} (\Gamma \cup dom(\pi)\#X, \sigma, FP).$$

The (AC Func) Rule I

Let m be an arbitrary number and let ψ be an arbitrary permutation from $\{1, \ldots, m\}$ to $\{1, \ldots, m\}$, such that:

$$\psi = (x_1 x_2 \ldots x_{m_1})(x_{m_1+1} x_{m_1+2} \ldots x_{m_2}) \ldots (x_{m_{k-1}+1} x_{m_{k-1}+2} \ldots x_{m_k}),$$

where $l_1, \ldots, l_k$ are the length of the cycles.

Let

$$\sigma' = \{X \mapsto f(\underbrace{X_1, \pi^1 X_1 \ldots, \pi^{l_1 - 1} X_1}, \ldots, \underbrace{X_k, \pi^1 X_k, \ldots, \pi^{l_k - 1} X_k})\}$$

The (AC Func) Rule II

$$(\Gamma, \sigma, \{\pi X \approx^? X\} \cup FP) \xrightarrow{(AC)}$$
$$(\Gamma, \sigma'\sigma, \{\pi^{l_1} X_1 \approx^? X_1, ..., \pi^{l_k} X_k \approx^? X_k\} \cup \sigma' FP)$$

Example of Solving Fixpoint Equations I

Let $*$ and $+$ be AC-function symbols and $\pi = (123456)(7891011)$.
Consider the solution:

$$(\emptyset, X \mapsto *(+(1,4), +(2,5), +(3,6)))$$

How can we inductively generate this solution?

Example of Solving Fixpoint Equations II

1. Apply `(AC Func)` and consider $m = 3$ and the permutation $\psi = (123)$. Then, we would compute $\sigma'_1 = X \mapsto *(X_1, \pi^1 X_1, \pi^2 X_1)$ and proceed to solve $\pi^3 X_1 \approx^? X_1$.

2. Apply `(AC Func)` and consider $m = 2$ and the permutation $\psi = (12)$. Our algorithm would compute $\sigma'_2 = X_1 \mapsto +(X_2, \pi^3 X_2)$ and proceed to solve $(\pi^3)^2 X_2 \approx^? X_2$.

Example of Solving Fixpoint Equations III

3. Notice that $(\pi^3)^2 = \pi^6 = (7891011)$. Solve $(\pi^3)^2 X_2 \approx^? X_2$ by applying `(Var)` and obtain the final solution:

$$(\{7, 8, 9, 10, 11\}\#X_2, X \mapsto *(+(X_2, \pi^3 X_2), +(\pi X_2, \pi^4 X_2), +(\pi^2 X_2, \pi^5 X_2)))$$

Example of Solving Fixpoint Equations IV

The particular solution:

$$(\emptyset, X \mapsto *(+(1,4), +(2,5), +(3,6)))$$

can be obtained from:

$$(\{7,8,9,10,11\}\#X_2, X \mapsto *(+(X_2, \pi^3 X_2), +(\pi X_2, \pi^4 X_2), +(\pi^2 X_2, \pi^5 X_2)))$$

by instantiating $X_2 \mapsto 1$.

The Loop in $f(X, W) \approx^? f(\pi X, \pi Y)$

We found a loop in 2 branches that are created while solving $f(X, W) \approx^? f(\pi X, \pi Y)$.

The Loop

In one branch, the problem before `applyACStep` and the problem after `applyACStep` are:

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

and we have

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

Bound on the Number of Times We Take the Loop

We proved that it is enough to take the branches that loop at most $2k$ times, where k is the order of the permutation π.

$f(2X_1, X_2, X_3) \approx^? f(2\pi X_2, Y_1)$

We could not generalise our reasoning for the problem

$$f(2X_1, X_2, X_3) \approx^? f(2\pi X_2, Y_1).$$

A Different Approach to Nominal AC-Unification

Idea: Explore the connection between nominal unification and higher-order pattern unification ([4], [5]) and the work of Boudet and Contejean in AC higher-order pattern unification [6].

Where to Start with Higher-Order Pattern AC-Unification

1. How do we translate $\pi X \approx^? X$ and $f(X, W) \approx^? f(\pi X, \pi Y)$ to higher-order patterns?
2. What answers are obtained when we perform AC-unification of those higher-order patterns?
3. How could those answers be translated back to nominal AC-unification?

$(ab)X \approx^? X$ in Higher-Order Patterns I

The translation of $(ab)X \approx^? X$ to higher-order pattern is:

$$\lambda ab.X(a, b) \approx^? \lambda ab.X$$

This kind of equation is called **flexible-flexible equation with the same head variable** and they are kept as part of the output in higher-order patterns [6].

$(ab)X \approx^? X$ in Higher-Order Patterns II

The problem corresponding with $\pi X \approx^? X$ is nullary in AC-unification of higher-order patterns ([7]), suggesting that an enumeration procedure for $\pi X \approx^? X$ in nominal AC-unification is indeed the best we can do.

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

Let $\pi = (ab)$. The translation of

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

is:

$$\lambda ab.f(X, W) \approx^? \lambda ab.f(X(a, b), Y(a, b))$$

$f(X, W) \approx^{?} f(\pi X, \pi Y)$ II

The algorithm employed is different than the one from Stickel, dividing this one equation into a system of linear Diophantine equations, with $n \, |\Pi|$ variables, where

- n - **number of coefficients** in the problem.
- $|\Pi|$ - **number of different permutations** of the atoms in the problem.

In our case $n = 4$ and $|\Pi| = 2$.

Conclusion

- We extended a functional nominal C-unification algorithm, adding a parameter $\mathcal{X}$ of protected variables and formalised it to be terminating, sound and complete.
- We gave the first certified first-order AC-unification algorithm.
- We proposed the first nominal AC-matching algorithm and formalised it to be terminating, sound and complete.
- We moved towards nominal AC-unification.

Thank You

Thank you! Any comments/feedback/doubts? [4]

[4] In case you want to see more of my work: https://gabriel951.github.io/

Bibliography I

[1] M. E. Stickel, "A Complete Unification Algorithm for Associative-Commutative Functions," in *Advanced Papers of the Fourth International Joint Conference on Artificial Intelligence*, 1975.

[2] F. Fages, "Associative-Commutative Unification," in *Proceedings 7th International Conference on Automated Deduction (CADE)*, 1984.

[3] M. E. Stickel, "A Unification Algorithm for Associative-Commutative Functions," *J. of the ACM*, 1981.

[4] J. Levy and M. Villaret, "Nominal Unification from a Higher-Order Perspective," *ACM Trans. Comput. Log.*, 2012.

[5] J. Cheney, "Relating Nominal and Higher-order Pattern Unification," in *Proc. of the 19th international workshop on Unification, (UNIF)*, 2005.

Bibliography II

[6] A. Boudet and E. Contejean, "AC-Unification of Higher-Order Patterns," in *3rd International Conference on Principles and Practice of Constraint Programming (CP)*, 1997.

[7] Z. Qian and K. Wang, "Modular AC Unification of Higher-Order Patterns," in *Constraints in Computational Logics (CCL)*, 1994.