Nominal Unification with Commutative and Associative-Commutative Function Symbols

Gabriel Ferreira Silva

July 26, 2019

Advisor: Mauricio Ayala-Rincón
Talk presented as a partial requirement for the MSc in Mathematics
Jury: Professors Edward Haeusler, Daniele Sobrinho and Flávio de Moura
Department of Mathematics - University of Brasília

Table of contents

Introduction

Nominal Syntax

Nominal syntax extends first-order syntax by bringing mechanisms to deal with bound and free variables in a natural manner.

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

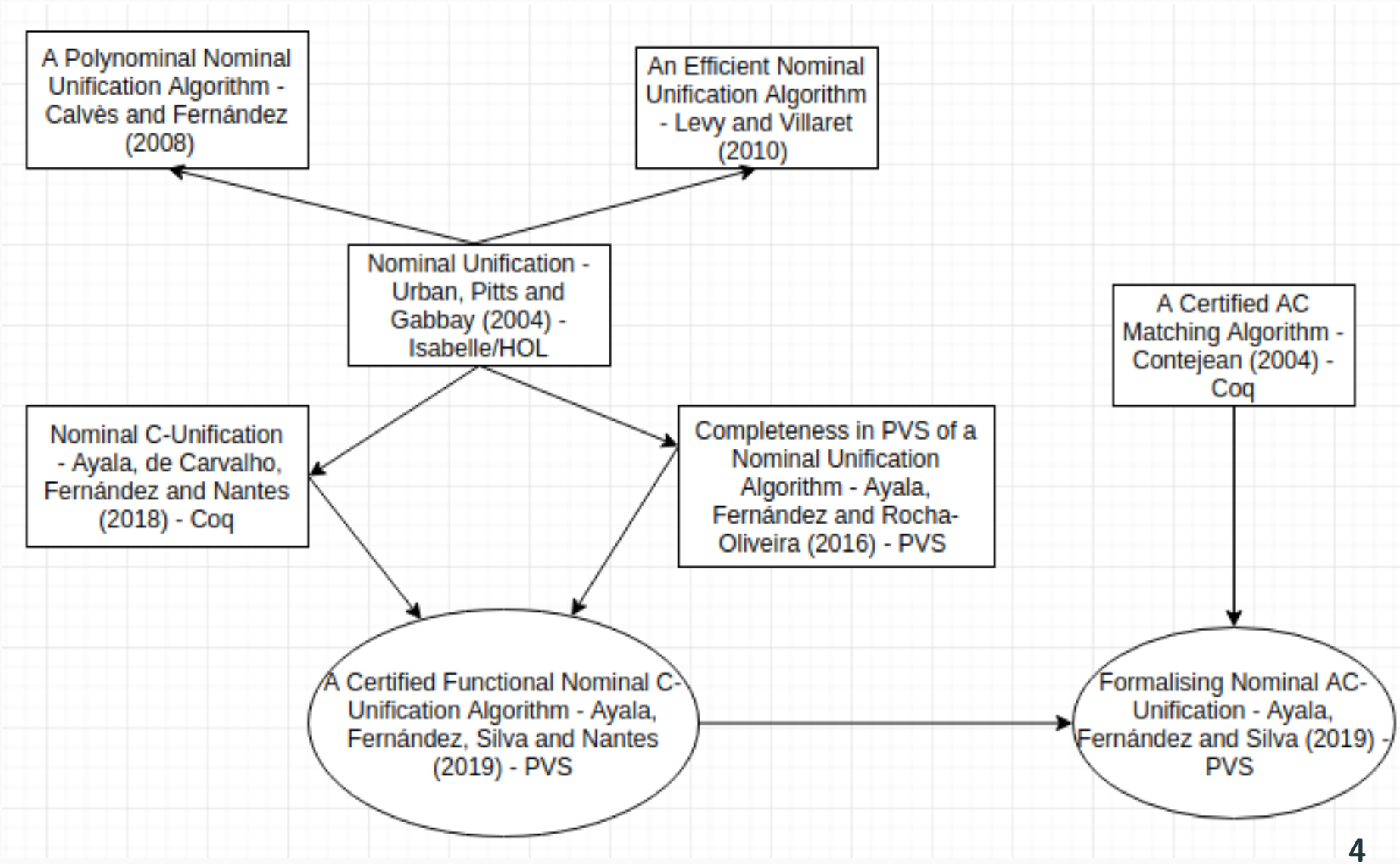
Unification

Unification is about "finding a way" to make two terms equal:

- $h\langle W, d\rangle$ and $h\langle c, Z\rangle$ can be made equal by "sending" W to c and Z to d, as they both become $h\langle c, d\rangle$.

Unification has a lot of applications: logic programming, theorem proving and so on.

Related Work



Contributions

- A correct and complete functional algorithm for nominal C-Unification.
- Work in progress on the formalisation of nominal AC-Unification.

Preliminaries

Preliminaries

Nominal Terms, Permutations and Substitutions

Atoms and Variables

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

Permutations

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

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

Nominal Terms

Definition (Nominal Terms)

Nominal terms are inductively generated according to the grammar:

$$s, t \quad ::= \quad \langle\rangle \;\mid\; a \;\mid\; \pi \cdot X \;\mid\; [a]t \;\mid\; \langle s, t\rangle \;\mid\; f\ t$$

The symbols denote respectively: unit, atom, suspended variable, abstraction, pair and function application.

Examples of Permutation Actions

Permutations act on atoms and terms:

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

Substitution

Definition (Substitution)

A substitution σ is a mapping from variables to terms, such that $\{X \mid X \neq X\sigma\}$ is finite.

Examples of Substitutions Acting on Terms

Substitutions also act on terms:

- $\sigma = \{Y \to c\}$, $t = f(X, Y)$, $t\sigma = f(X, c)$.
- $\sigma = \{X \to \langle a, b\rangle\}$, $t = \langle X, X\rangle$, $t\sigma = \langle\langle a, b\rangle, \langle a, b\rangle\rangle$.

Preliminaries

Freshness and α-Equality

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.

Derivation Rules for Freshness

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

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

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

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

Derivation Rules for α-Equivalence

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

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

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

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

Derivation Rules as a Sequent Calculus

The derivation rules for freshness and α-equivalence are a sequent calculus.

Deriving $a\#f\langle[a]X, Y\rangle$ with $\Delta = \{a\#Y\}$:

$$\dfrac{\dfrac{\dfrac{}{a\#[a]X}\,(\#[a]a) \qquad a\#Y}{a\#\langle[a]X, Y\rangle}\,(\#\text{pair})}{a\#f\langle[a]X, Y\rangle}\,(\#\text{app})$$

Derivation Rules as a Sequent Calculus

Deriving $[a]a \approx_\alpha [b]b$:

$$\dfrac{\dfrac{}{a \approx_\alpha (a\ b) \cdot b}\,(\approx_\alpha \mathit{atom}) \qquad \dfrac{}{a\#b}\,(\#\mathit{atom})}{[a]a \approx_\alpha [b]b}\,(\approx_\alpha [a]b)$$

Preliminaries

Adding C and AC function symbols

Extending the Grammar

The grammar of nominal terms is extended:

$$s, t ::= \text{old rules} \mid f^{C}\langle t_1, t_2\rangle \mid f^{AC}\ t$$

We impose that commutative functions receive a pair as their argument.

Extending Freshness

$$\frac{\Delta \vdash a\#s \quad \Delta \vdash a\#t}{\Delta \vdash a\#f^{C}\ \langle s, t\rangle}\,(\#c - app)$$

$$\frac{\Delta \vdash a\#t}{\Delta \vdash a\#f^{AC}\ t}\,(\#ac - app)$$

Additional α-Equivalence Rule for Commutative Symbols

$$\frac{\Delta \vdash s_0 \approx_\alpha t_0,\ \ \Delta \vdash s_1 \approx_\alpha t_1}{\Delta \vdash f^C\langle s_0, s_1\rangle \approx_\alpha f^C\langle t_0, t_1\rangle}\ (\approx_\alpha C - app)$$

$$\frac{\Delta \vdash s_0 \approx_\alpha t_1,\ \ \Delta \vdash s_1 \approx_\alpha t_0}{\Delta \vdash f^C\langle s_0, s_1\rangle \approx_\alpha f^C\langle t_0, t_1\rangle}\ (\approx_\alpha C - app)$$

Additional α-Equivalence Rule for Associative-Commutative Symbols

Rule ($\approx_\alpha AC - app$):

$$\frac{\Delta \vdash S_1(f^{AC}\, s) \approx_\alpha S_i(f^{AC}\, t) \quad \Delta \vdash D_1(f^{AC}\, s) \approx_\alpha D_i(f^{AC}\, t)}{\Delta \vdash f^{AC}\, s \approx_\alpha f^{AC}\, t}$$

$S_n(f*)$ selects the nth argument of the flattened subterm $f*$.
$D_n(f*)$ deletes the nth argument of the flattened subterm $f*$.

The Operators S_n and D_n

Let f be an AC function:

- $S_2(f\langle\langle a, b\rangle, c\rangle)$ is equal to b.
- $D_2(f\langle\langle a, b\rangle, c\rangle)$ is equal to $f\langle a, c\rangle$.

Preliminaries

The Problem of Nominal C and AC-Unification

Unification Problem

Definition (Unification Problem)

A unification problem is a pair $\langle \Delta, P \rangle$, where Δ is a freshness context and P is a finite set of equations $(s \overset{?}{\approx}_\alpha t)$ and freshness constraints $(a \overset{?}{\#} s)$.

Example

f is an AC function symbol.

$\langle \Delta, P \rangle = \langle \emptyset,\ f\langle f\langle [b]X, Y\rangle, b\rangle\rangle \ \approx_?\ f\langle c, f\langle [a]a, b\rangle\rangle\rangle$.

Solution to a Unification Problem

Definition (Solution to a Unification Problem)

The unification problem $\langle \Delta, P \rangle$ is associated with the triple $\langle \Delta, id, P \rangle$.

The pair $\langle \nabla, \sigma \rangle$ is a solution for a triple $\mathcal{P} = \langle \Delta, \delta, P \rangle$ when

- $\nabla \vdash \Delta\sigma$
- $\nabla \vdash a\#t\sigma$, if $a \overset{?}{\#} t \in P$
- $\nabla \vdash s\sigma \approx_\alpha t\sigma$, if $s \approx_? t \in P$
- There exists λ such that $\nabla \vdash \delta\lambda \approx_\alpha \sigma$

Example of Unification Problem and Solution

f is an AC function symbol.

One possible solution for
$\langle \emptyset,\ f\langle f\langle [b]X, Y\rangle, b\rangle\rangle \approx_?\ f\langle c, f\langle [a]a, b\rangle\rangle\rangle$ is:

$\langle \emptyset, \{X \to b, Y \to c\}\rangle$

Preliminaries

Differences from Nominal Syntactic Unification

Difference from Syntactic Unification

Nominal C and AC-Unification have 2 main differences when compared with syntactic nominal unification:

- A fixpoint equation is of the form $\pi \cdot X \approx_\alpha \gamma \cdot X$. Fixpoint equations are not solved in C and AC-unification. Instead, they are carried on, as part of the solution.
- We obtain a set of solutions, not just one.

Functional Nominal C-Unification

Functional Nominal C-Unification

The Algorithm

General Comments About the Functional Nominal C-Unification Algorithm

- We will present the pseudocode of a **functional** nominal C-unification algorithm, that allow us to unify two terms t and s.
- Since the algorithm is recursive and needs to keep track of the current context, the substitutions made so far, the remaining terms to unify and the current fixpoint equations, the algorithm receives as input a quadruple $(\Delta, \sigma, \mathit{UnPrb}, \mathit{FxPntEq})$.

General Comments About the Functional Nominal C-Unification Algorithm

Call to unify terms t and s:

$$\textsc{unify}(\emptyset, id, [(t, s)], \emptyset).$$

The algorithm returns a list (possibly empty) of solutions. Each solution is of the form $(\Delta, \sigma, FxPntEq)$.

- Example: $[(\Delta_1, \sigma_1, FxPntEq_1), \ldots, (\Delta_n, \sigma_n, FxPntEq_n)]$

The First Part of the Functional Nominal C-Unification Algorithm

1: **procedure** UNIFY$(\Delta, \sigma, \mathit{UnPrb}, \mathit{FxPntEq})$
2: **if** null(UnPrb) **then**
3: **return** list$((\Delta, \sigma, \mathit{FxPntEq}))$
4: **else**
5: $(t, s) \oplus \mathit{UnPrb}' = \mathit{UnPrb}$
6: [Code that analyses according to t and s]
7: **end if**
8: **end procedure**

Functional Nominal C-Unification Algorithm I

1: **procedure** UNIFY$(\Delta, \sigma, \mathit{UnPrb}, \mathit{FxPntEq})$
2: **if** null(UnPrb) **then**
3: **return** list$((\Delta, \sigma, \mathit{FxPntEq}))$
4: **else**
5: $(t, s) \oplus \mathit{UnPrb}' = \mathit{UnPrb}$
6: **if** $(s == \pi \cdot X)$ and $(X$ not in $t)$ **then**
7: $\sigma' = \{X \to \pi^{-1} \cdot t\}$
8: $\sigma'' = \sigma' \circ \sigma$
9: $(\Delta'$, bool1$) =$ fresh_subs?(σ', Δ)
10: $\Delta'' = \Delta \cup \Delta'$
11: $\mathit{UnPrb}'' = (\mathit{UnPrb}')\sigma' + (\mathit{FxPntEq})\sigma'$

Functional Nominal C-Unification Algorithm II

12: **if** bool1 **then return** $\text{UNIFY}(\Delta'', \sigma'', \mathit{UnPrb}'', \mathit{null})$
13: **else return** null
14: **end if**
15: **else**
16: **if** $t == a$ **then**
17: **if** s == a **then**
18: **return** $\text{UNIFY}(\Delta, \sigma, \mathit{UnPrb}', \mathit{FxPntEq})$
19: **else**
20: **return** null
21: **end if**

Functional Nominal C-Unification Algorithm III

22: **else if** $t == \pi \cdot X$ **then**
23: **if** (X not in s) **then**
24: ▷ Similar to case above where
25: ▷ s is a suspension
26: **else if** $(s == \pi' \cdot X)$ **then**
27: $FxPntEq' = FxPntEq \cup \{((\pi')^{-1} \oplus \pi) \cdot X\}$
28: **return** $\text{UNIFY}(\Delta, \sigma, UnPrb', FxPntEq')$
29: **else return** null
30: **end if**

Functional Nominal C-Unification Algorithm IV

31: **else if** $t == \langle\rangle$ **then**
32: **if** $s == \langle\rangle$ **then**
33: **return** $\text{UNIFY}(\Delta, \sigma, \mathit{UnPrb}', \mathit{FxPntEq})$
34: **else return** null
35: **end if**
36: **else if** $t == \langle t_1, t_2\rangle$ **then**
37: **if** $s == \langle s_1, s_2\rangle$ **then**
38: $\mathit{UnPrb}'' = [(s_1, t_1)] + [(s_2, t_2)] + \mathit{UnPrb}'$
39: **return** $\text{UNIFY}(\Delta, \sigma, \mathit{UnPrb}'', \mathit{FxPntEq})$
40: **else return** null
41: **end if**

Functional Nominal C-Unification Algorithm V

42: **else if** $t == [a]t_1$ **then**
43: **if** $s == [a]s_1$ **then**
44: $UnPrb'' = [(t_1, s_1)] + UnPrb'$
45: **return** $\text{UNIFY}(\Delta, \sigma, UnPrb'', FxPntEq)$
46: **else if** $s == [b]s_1$ **then**
47: $(\Delta', bool1) = fresh?(a, s_1)$
48: $\Delta'' = \Delta \cup \Delta'$
49: $UnPrb'' = [(t_1, \quad (a\ b) \cdot s_1)] + UnPrb'$
50: **if** bool1 **then**
51: **return** $\text{UNIFY}(\Delta'', \sigma, UnPrb'', FxPntEq)$
52: **else return** null
53: **end if**
54: **else return** null

Functional Nominal C-Unification Algorithm VI

55: **end if**

56: **else if** $t == f\ t_1$ **then** ▷ f is not commutative

57: **if** $s\ != f\ s_1$ **then return** null

58: **else**

59: $UnPrb'' = [(t_1, s_1)] + UnPrb'$

60: **return** $\text{UNIFY}(\Delta, \sigma, UnPrb'', FxPntEq)$

61: **end if**

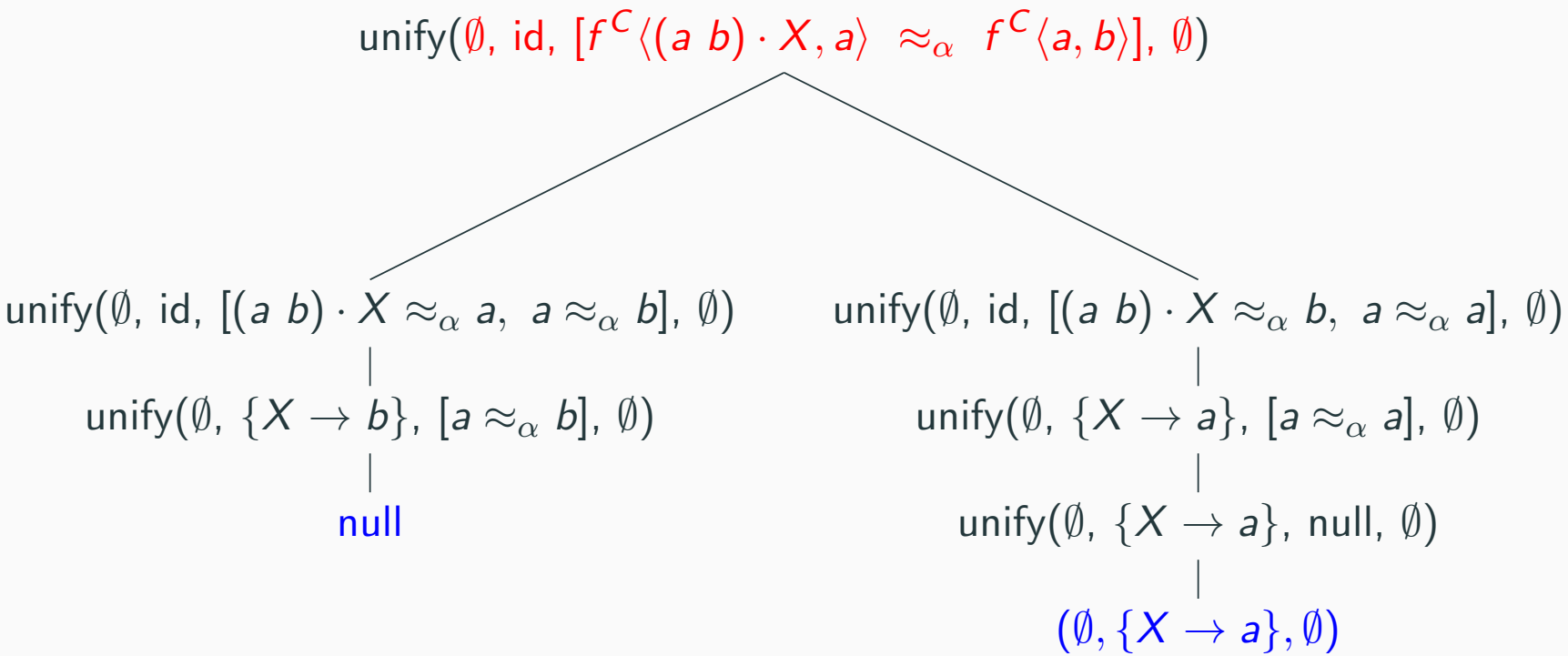
Functional Nominal C-Unification Algorithm VII

62: **else** ▷ t is of the form $f^C\langle t_1, t_2\rangle$
63: **if** s != $f^C\langle s_1, s_2\rangle$ **then return** null
64: **else**
65: $UnPrb_1 = [(s_1, t_1)] + [(s_2, t_2)] + UnPrb'$
66: $sol_1 = \text{UNIFY}(\Delta, \sigma, UnPrb_1, FxPntEq)$
67: $UnPrb_2 = [(s_1, t_2)] + [(s_2, t_1)] + UnPrb'$
68: $sol_2 = \text{UNIFY}(\Delta, \sigma, UnPrb_2, FxPntEq)$
69: **return** $\text{APPEND}(sol_1, sol_2)$
70: **end if**
71: **end if**
72: **end if**
73: **end if**
74: **end procedure**

Functional Nominal C-Unification

Example

Example of the Algorithm



Functional Nominal C-Unification

Formalisation

Specification and Formalisation

We proved soundness and completeness of the algorithm here described, using PVS (Prototype Verification System).

Soundness of the Algorithm

Theorem (Soundness of Unify)

Suppose $(\Delta, \delta, \mathit{FxPntEq}) \in \textsc{unify}(\emptyset, \mathit{id}, [(t, s)], \emptyset)$ *and* (∇, σ) *is a solution to the unification problem* $(\Delta, \delta, \mathit{FxPntEq})$*. Then* (∇, σ) *is a solution to the unification problem* $(\emptyset, \mathit{id}, [(t, s)])$*.*

Completeness of the Algorithm

Theorem (Completeness of Unify)

Suppose (∇, σ) *is a solution to the unification problem* $(\emptyset, id, [(t, s)])$. *Then, there exists* $(\Delta, \delta, FxPntEq) \in$ $\textsc{unify}(\emptyset, id, [(t, s)], \emptyset)$ *such that* (∇, σ) *is a solution to* $(\Delta, \delta, FxPntEq)$.

Functional Nominal C-Unification

Implementation

Implementation

The algorithm has been implemented in Python 3.

Functional Nominal C-Unification

Possible Applications

Possible Applications

- The algorithm could be used on α-Prolog.
- The algorithm could be adapted to the task of matching.
- Nominal C-matching and nominal C-unification could be used in nominal rewriting and nominal narrowing.

Formalising AC-Unification

Cases to Consider When Dealing With AC Function Symbols

Three cases to consider:

- When t or s is an AC function application and the other term is a suspended variable: instantiate the variable appropriately.
- When t and s are both applications of the same AC function symbol: interesting case.
- Otherwise, no solution is possible.

When t and s are AC Functions

1. Extract all arguments of t and generate all pairings of those arguments, **in any order**.
2. Extract all arguments of s and generate all pairings of those arguments, **in any order**.
3. Try to unify every generated pairing of t with every generated pairing of s.

Efficiency vs Dificulty of the Proof

The algorithm explores the combinatory of the problem without considering efficiency, simplifying in this manner the formalisation.

A Loose End

Let g be an AC function symbol.

Suppose trying to unify $g\langle W, a\rangle$ with $g\langle Z, b\rangle$. The algorithm, after generating the pairings and combining them, would only find as substitution $\{W \to b, Z \to a\}$.

But the following substitution $\{W \to g\langle b, U\rangle, Z \to g\langle a, U\rangle\}$ is also correct. We are missing something!

Conclusion and Future Work

Conclusion

- The problem of nominal C and AC-Unification was explained.
- A correct and complete algorithm for nominal C-unification was presented.
- Our work in progress on formalising AC-Unification was discussed.

Future Work

- Finish formalisation of AC-Unification.
- Work with other equational theories such as A-unification.

References I

M. Ayala-Rincón, W. de Carvalho-Segundo, M. Fernández, and D. Nantes-Sobrinho.

Nominal C-Unification.

In *27th International Symposium on Logic-Based Program Synthesis and Transformation (LOPSTR 2017), Revised Selected Papers*, volume 10855 of *Lecture Notes in Computer Science*, pages 235–251. Springer, 2018.

M. Ayala-Rincón, W. de Carvalho Segundo, M. Fernández, D. Nantes-Sobrinho, and A. C. R. Oliveira.

A formalisation of nominal α-equivalence with A, C, and AC function symbols.

Theor. Comput. Sci., 781:3–23, 2019.

References II


M. Ayala-Rincón, M. Fernández, and G. Ferreira Silva.
Formalising Nominal AC-Unification.
2019.
Presented in the International Workshop on Unification UNIF 2019.


M. Ayala-Rincón, M. Fernández, G. Ferreira Silva, and D. Nantes-Sobrinho.
A Certified Functional Nominal C-Unification Algorithm.
Accepted at LOPSTR, 2019.


M. Ayala-Rincón, M. Fernández, and A. C. Rocha-Oliveira.
Completeness in PVS of a nominal unification algorithm.
Electronic Notes in Theoretical Computer Science, 323:57–74, 2016.

References III


E. Contejean.
A certified AC matching algorithm.
In *Rewriting Techniques and Applications, 15th International Conference, RTA 2004, Aachen, Germany, June 3-5, 2004, Proceedings*, volume 3091 of *Lecture Notes in Computer Science*, pages 70–84. Springer, 2004.


M. Fernández and M. J. Gabbay.
Nominal rewriting.
Information and Computation, 205(6):917–965, 2007.


C. Urban, A. M. Pitts, and M. J. Gabbay.
Nominal unification.
Theoretical Computer Science, 323(1-3):473–497, 2004.

Thank You

Thank you! Any questions?