Formalising Nominal AC-Unification

UNIF - 33nd International Workshop on Unification

Mauricio Ayala-Rincón[†,‡], Maribel Fernández[*] and **Gabriel Ferreira Silva**[‡]

Dortmund, 24 June 2019

Departments of [†]Computer Science and ‡Mathematics, Universidade de Brasília
[*]Department of Informatics, Kings College London

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.

Purpose of Presentation

- We revisit the problem of nominal unification with associative-commutative (AC) operators
- We briefly comment about a functional algorithm for nominal AC-unification and our work in progress on its formalisation.

Background

Background

Nominal Terms, Permutations and Substitutions

Atoms and Variables

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

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:

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

Nominal Terms

Definition (Nominal Terms)

Nominal terms are inductively generated according to the grammar:

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

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

Examples of Permutation Actions

Permutations act on atoms and terms:

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

Background

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

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

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

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

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

Additional Rule for Freshness with AC Functions

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

Derivation Rules for α-Equivalence

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

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

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

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

Additional Rule for α-Equivalence with AC Functions

Let f be an AC function symbol.

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

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

$S_n(f*)$ 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:

- $S_2(f\langle f\langle a, b\rangle, f\langle [a]X, \pi \cdot Y\rangle\rangle)$ is equal to b.
- $D_2(f\langle f\langle a, b\rangle, f\langle [a]X, \pi \cdot Y\rangle\rangle\rangle)$ is equal to $f\langle f\ \ a, f\langle [a]X, \pi \cdot Y\rangle\rangle\rangle)$.

Derivation Rules as a Sequent Calculus

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

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

Nominal AC-Unification

Nominal AC-Unification

Definition of the 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.

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

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 $\langle\emptyset,\ f\langle f\langle X, Y\rangle, c\rangle\rangle \ \approx_? \ f\langle c, f\langle a, b\rangle\rangle\rangle$ is:

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

Nominal AC-Unification

Differences from Nominal Syntactic Unification

Difference from Syntactic Unification

AC-Unification has 2 main differences when compared with nominal unification:

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

Nominal AC-Unification

Comments About the Specification

Comments About the Specification

- We will talk about the main points behind a **functional** nominal AC-unification algorithm, that allows us to unify two terms t and s, focusing on the case of AC function symbols.

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.

The function `gen_unif_prb` generates and combines those pairings.

Example of Pairings Generated

Let f be an AC function symbol.

Suppose trying to unify $f\langle f\langle X, Y\rangle, c\rangle$ with $f\langle c, f\langle a, b\rangle\rangle$. Then:

- The two generated pairings of $f\langle f\langle X, Y\rangle, c\rangle$ in the order (X, Y, c) are: $\langle X, \langle Y, c\rangle\rangle$ and $\langle\langle X, Y\rangle, c\rangle$. Twelve pairings would be generated for this term.
- The twelve pairings generated for the term $f\langle c, f\langle a, b\rangle\rangle$ include, for instance: $\langle c, \langle a, b\rangle\rangle$, $\langle\langle c, a\rangle, b\rangle$, $\langle\langle b, c\rangle, a\rangle$ and $\langle a, \langle b, c\rangle\rangle$.

A Problematic Approach - I

Why not generate all pairings of t **preserving the order** and generate the pairings of s **in any order**?

This is not complete however ...

A Problematic Approach - II

Counterexample: $t = f\langle a, \langle b, c\rangle\rangle$ and $s = f\langle X, b\rangle$.

The substitution $\sigma = \{X \to \langle a, c\rangle\}$ would not be found had we generated the pairings of t **preserving the order**, but it can be found by our approach.

This is because σ is found when trying to unify $\langle\langle a, c\rangle, b\rangle$ with $\langle X, b\rangle$

Efficiency vs Difficulty of the Proof

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

Formalisation

Theorem (Soundness of Unifying AC functions)

Suppose that $(t_1, s_1) \in \texttt{gen\_unif\_prb}(ft, fs)$ *and that* $\nabla \vdash t_1\sigma \approx_\alpha s_1\sigma$. *Then,* $\nabla \vdash (ft)\sigma \approx_\alpha (fs)\sigma$.

Theorem (Completeness of Unifying AC functions)

Suppose that $\nabla \vdash (ft)\sigma \approx_\alpha (fs)\sigma$. *Then,* **there exists** $(t_1, s_1) \in \texttt{gen\_unif\_prb}(ft, fs)$ *such that* $\nabla \vdash t_1\sigma \approx_\alpha s_1\sigma$.

The Analysis That Follows

The analysis that follows is for the proof of soundness. A similar analysis, however, could be done to prove completeness.

Another Problematic Approach

Proving the soundness theorem directly, by induction on the size of the term?

1. find the i that makes $\nabla \vdash S_1((ft)\sigma) \approx_\alpha S_i((fs)\sigma)$
2. use I.H. to prove that $\nabla \vdash D_1((ft)\sigma) \approx_\alpha D_i((fs)\sigma)$

The term being deleted of $(ft)\sigma$ could be the first term of ft but it could also have being introduced by the substitution σ. We cannot apply the I.H.!

Our Solution

First eliminate the substitution from our problem and then solve a simplified version of the problem.

Yet Another Problematic Approach - I

Proving that:

$(t_1, s_1) \in \texttt{gen\_unif\_prb}(ft, fs) \implies (t_1\sigma, s_1\sigma) \in \texttt{gen\_unif\_prb}((ft)\sigma, (fs)\sigma)$

would let us, by a renaming of variables, solve a version of the problem without a substitution.

Yet Another Problematic Approach - II

This, however, does not work: the substitution σ can reintroduce AC function symbols f into the terms $t_1\sigma$ and $s_1\sigma$, but an output of `gen_unif_prb` cannot have the AC function symbol.

Yet Another Problematic Approach - Example

Let f be an AC symbol, $t = f\ X$, $s = f\ Y$ and $\sigma = \{X \to f\langle a, b\rangle, Y \to f\langle b, a\rangle\}$. Then:

- $t_1 = X$ and $s_1 = Y$ do not contain the AC function symbol
- σ reintroduces the AC function symbol and we have $t_1\sigma = f\langle a, b\rangle$ and $s_1\sigma = f\langle b, a\rangle$

The Operator F_{AO}

$F_{AO}(s)$ generates all possible flattened versions of a term s, **in any order**.

Auxiliar Lemma 1 for Eliminating Substitutions Out of Equation

Lemma

Suppose that $(t_1, s_1) \in \texttt{gen\_unif\_prb}(ft, fs)$*. Then,* $\forall t_1' \in F_{AO}(t_1\sigma)$, $s_1' \in F_{AO}(s_1\sigma)$*:* $(t_1', s_1') \in \texttt{gen\_unif\_prb}(ft\sigma, fs\sigma)$*.*

Auxiliar Lemma 2 for Eliminating Substitution Out of Equation

Lemma

Suppose that $(t_1, s_1) \in \texttt{gen\_unif\_prb}(ft, fs)$ *and that* $\nabla \vdash t_1\sigma \approx_\alpha s_1\sigma$*. Then,* $\exists t_1' \in F_{AO}(t_1\sigma), s_1' \in F_{AO}(s_1\sigma) : \nabla \vdash t_1' \approx_\alpha s_1'$*.*

Version of the Problem without Substitution

Lemma

Suppose that $(t_1, s_1) \in \texttt{gen\_unif\_prb}(ft, fs)$ *and that* $\nabla \vdash t_1 \approx_\alpha s_1$
Then, $\nabla \vdash (ft) \approx_\alpha (fs)$.

Proof.

We plan an induction on the size of ft. $\square$

A Loose End

Let f be an AC function symbol.

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

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

Related Work and Contribution

Related Work

Ayala et al. (2019) presents a correct and complete algorithm for nominal C-unification. This work is a continuation of that work.

Contejean (2004) gave the first formalisation of AC-matching, opening the way for a formalisation of AC-unification.

Contribution

If completed, the work here presented would give not only the first formalisation of nominal AC-unification, but also, as far as we know, the first formalisation of first-order AC-unification.

Conclusion and Future Work

Conclusion

- Nominal AC-unification was (hopefully) explained.
- A functional algorithm and aspects of its formalisation were commented.

Future Work

Future work:

- Complete the formalisation
- Work with other equational theories.

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, M. Fernández, G. Ferreira Silva, and D. Nantes-Sobrinho.
Soundness and completeness in PVS of a functional nominal C-unification algorithm.
Available at `http://ayala.mat.unb.br/publications.html`, 2019.

M. Ayala-Rincón, M. Fernández, and D. Nantes-Sobrinho.
Nominal narrowing.
In *1st International Conference on Formal Structures for Computation and Deduction (FSCD)*, volume 52 of *LIPIcs*, pages 11:1–11:17. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

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.

F. Baader and T. Nipkow.
Term rewriting and all that.
Cambridge University Press, 1999.

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.

J. Levy and M. Villaret.
Nominal unification from a higher-order perspective.
In *19th International Conference on Rewriting Techniques and Applications (RTA)*, volume 5117 of *Lecture Notes in Computer Science*, pages 246–260. Springer, 2008.

A. Pitts.
Nominal sets: Names and symmetry in computer science.
Cambridge University Press, 2013.

M. Schmidt-Schauß, T. Kutsia, J. Levy, and M. Villaret.
Nominal unification of higher order expressions with recursive let.
In *26th International Symposium on Logic-Based Program Synthesis and Transformation (LOPSTR 2016), Revised Selected Papers*, volume 10184 of *Lecture Notes in Computer Science*, pages 328–344. Springer, Springer, 2017.

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?

Appendix - Functional Nominal AC-Unification Algorithm

Functional Nominal AC-Unification Algorithm - Sketch

procedure UNIFY($\Delta, \sigma, \mathit{UnPrb}, \mathit{FxPntEq}$)
    if null(UnPrb) **then**
        return list(($\Delta, \sigma, \mathit{FxPntEq}$))
    else
        $(t, s) \oplus \mathit{UnPrb}' = \mathit{UnPrb}$
        [Code that analyses according to t and s]
    end if
end procedure

Functional Nominal AC-Unification Algorithm i

procedure UNIFY($\Delta, \sigma, \mathit{UnPrb}, \mathit{FxPntEq}$)
    if null(UnPrb) **then**
        return list(($\Delta, \sigma, \mathit{FxPntEq}$))
    else
        $(t, s) \oplus \mathit{UnPrb}' = \mathit{UnPrb}$
        if $(s == \pi \cdot X)$ and (X not in t) **then**
            $\sigma' = \{X \to \pi^{-1} \cdot t\}$
            $\sigma'' = \sigma' \circ \sigma$
            (Δ', bool1) = appSub2Ctxt(σ', Δ)
            $\Delta'' = \Delta \cup \Delta'$
            $\mathit{UnPrb}'' = (\mathit{UnPrb}')\sigma' + (\mathit{FxPntEq})\sigma'$

Functional Nominal AC-Unification Algorithm ii

if bool1 **then return** $\textsc{unify}(\Delta'', \sigma'', \mathit{UnPrb}'', \mathit{null})$
else return null
end if
else
if $t == a$ **then**
if s == a **then**
return $\textsc{unify}(\Delta, \sigma, \mathit{UnPrb}', \mathit{FxPntEq})$
else
return null
end if

Functional Nominal AC-Unification Algorithm iii

else if $t == \pi \cdot X$ **then**

  if (X not in s) **then**

    ▷ Similar to case above where

    ▷ s is a suspension

  else if $(s == \pi' \cdot X)$ **then**

    $\mathit{FxPntEq}' = \mathit{FxPntEq} \cup \{((\pi')^{-1} \oplus \pi) \cdot X\}$

    return $\textsc{unify}(\Delta, \sigma, \mathit{UnPrb}', \mathit{FxPntEq}')$

  else return null

  end if

Functional Nominal AC-Unification Algorithm iv

else if $t == \langle\rangle$ **then**
  if $s == \langle\rangle$ **then**
    return $\textsc{unify}(\Delta, \sigma, \mathit{UnPrb}', \mathit{FxPntEq})$
  else return null
  end if
else if $t == \langle t_1, t_2 \rangle$ **then**
  if $s == \langle s_1, s_2 \rangle$ **then**
    $\mathit{UnPrb}'' = [(s_1, t_1)] + [(s_2, t_2)] + \mathit{UnPrb}'$
    return $\textsc{unify}(\Delta, \sigma, \mathit{UnPrb}'', \mathit{FxPntEq})$
  else return null
  end if

Functional Nominal AC-Unification Algorithm v

else if $t == [a]t_1$ **then**

  if $s == [a]s_1$ **then**

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

    return $\textsc{unify}(\Delta, \sigma, UnPrb'', FxPntEq)$

  else if $s == [b]s_1$ **then**

    $(\Delta', bool1) = fresh(a, s_1)$

    $\Delta'' = \Delta \cup \Delta'$

    $UnPrb'' = [(t_1, \ (a\ b)\ s_1)] + UnPrb'$

    if bool1 **then**

      return $\textsc{unify}(\Delta'', \sigma, UnPrb'', FxPntEq)$

    else return null

    end if

  else return null

Functional Nominal AC-Unification Algorithm vi

end if

else if $t == f\ t_1$ **then**

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

else

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

return $\textsc{unify}(\Delta, \sigma, UnPrb'', FxPntEq)$

end if

Functional Nominal AC-Unification Algorithm vii

else if $t == f^C(t_1, t_2)$ **then**

if $s! = f^C(s_1, s_2)$ **then return** null

else

$UnPrb_1 = [(s_1, t_1)] + [(s_2, t_2)] + UnPrb'$

$sol_1 = \textsc{unify}(\Delta, \sigma, UnPrb_1, FxPntEq)$

$UnPrb_2 = [(s_1, t_2)] + [(s_2, t_1)] + UnPrb'$

$sol_2 = \textsc{unify}(\Delta, \sigma, UnPrb_2, FxPntEq)$

return $\textsc{append}(sol_1, sol_2)$

end if

```
            else                                        ▷ t is of the form f^AC t'
                if s != f^AC s' then return null
                else
                    UnPrb'' = gen_unif_prb(t, s)
                    LQ = GLQ(UnPrb'', Δ, σ, UnPrb', FxPntEq)
                    LstLstSol = map(UNIFY, LQ)
                    return FLATTEN(LstLstSol)
                end if
            end if
        end if
    end if
end procedure
```