A Certified Functional Nominal C-Unification Algorithm

29th International Symposium on Logic-based Program Synthesis and Transformation (LOPSTR 2019)

Mauricio Ayala-Rincón[1], Maribel Fernández[2], **Gabriel Silva**[1], Daniele Nantes-Sobrinho[1]

October 9, 2019

1 - Universidade de Brasília

2 - King's College London

Acknowledgments: we thank FAPDF and CAPES for funding our participation in this event.

Outline of Presentation

1. Introduction
2. Preliminaries
3. Functional Nominal C-Unification
4. Work in Progress, Conclusion

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 commutative operators and comment about a certified **functional** algorithm for nominal C-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:

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

Nominal Terms

Definition (Nominal Terms)

Nominal terms are inductively generated according to the grammar:

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

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

Examples of Permutation Actions

Permutations act on atoms and terms:

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

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:

- $\sigma = \{Y \rightarrow c\}$, $t = f\langle X, Y\rangle$, $t\sigma = f\langle X, c\rangle$.

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

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

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

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

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

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 α-Equivalence Rule for Commutative Symbols

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

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

Preliminaries

The Problem of Nominal C-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 ($a \overset{?}{\#} s$).

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 $\nabla \vdash \delta\lambda \approx_\alpha \sigma$

Preliminaries

Differences from Nominal Syntactic Unification

Difference from Syntactic Unification

Nominal C-unification has 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-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

Novel Features

In relation to the other work in nominal C-unification:

- Functional algorithm that can be directly executed, instead of a set of non-deterministic inference rules.
- Reduction (from 4 to 2) in the number of parameters of the lexicographic measure.

The Parameters of the Algorithm

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 $(\Delta, \sigma, \mathit{UnPrb}, \mathit{FxPntEq})$.

Calling the Algorithm; Form of the Output

Call to unify terms t and s:

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

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

- Example: $[(\Delta_1, \sigma_1, \mathit{FxPntEq}_1)$, ..., $(\Delta_n, \sigma_n, \mathit{FxPntEq}_n)]$

Algorithm Pseudocode - An Overview

```
procedure UNIFY(Δ, σ, UnPrb, FxPntEq)
    if null(UnPrb) then
        return list((Δ, σ, FxPntEq))
    else
        (t, s) ⊕ UnPrb' = UnPrb
        [Code that analyses according to t and s]
    end if
end procedure
```

Functional Nominal C-Unification

Example

Example of the Algorithm

$$\text{unify}(\emptyset,\ \text{id},\ [f^C\langle (a\ b)\cdot X, a\rangle \approx_\alpha f^C\langle a, b\rangle],\ \emptyset)$$

$$\text{unify}(\emptyset,\ \text{id},\ [(a\ b)\cdot X \approx_\alpha a,\ a \approx_\alpha b],\ \emptyset)$$

$$\text{unify}(\emptyset,\ \{X \to b\},\ [a \approx_\alpha b],\ \emptyset)$$

$$\text{null}$$

$$\text{unify}(\emptyset,\ \text{id},\ [(a\ b)\cdot X \approx_\alpha b,\ a \approx_\alpha a],\ \emptyset)$$

$$\text{unify}(\emptyset,\ \{X \to a\},\ [a \approx_\alpha a],\ \emptyset)$$

$$\text{unify}(\emptyset,\ \{X \to a\},\ \text{null},\ \emptyset)$$

$$(\emptyset, \{X \to a\}, \emptyset)$$

Functional Nominal C-Unification

Formalisation

Specification and Formalisation

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

Functional Nominal C-Unification

Implementation

Implementation

The PVS specification was manually translated to a Python 3 implementation.

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.

Work in Progress, Conclusion

Work in Progress, Conclusion

Work in Progress - Implementation

Implementation - Idea

Compare the manual Python code with extracted verified code from PVS and with extracted verified code from Coq.

How to compare?

- First, guarantee that all 3 programs give the same output.
- Then, analyse the time performance of the 3 programs.

Implementation - Components

Components:

- Example generator - Done
- Python code - Done
- PVS verified code - Working now
- Coq verified code - To Be Done

Implementation - Example Generator

1. Generate randomly a nominal term t.

2. Make small modifications in t, obtaining a different term s.
According to predefined probabilities:

- Substitute part of the term t by a suspended variable.
- When dealing with a commutative function application, change the order of the two arguments.
- When dealing with an abstraction, "change" the atom being abstracted.
- When an atom a is encountered, change the atom to a different atom b.

3. Run algorithm to unify (if possible) t and s.

Implementation - Python Code

The Python code is a manual translation from the PVS specification.

Implementation - PVS extracted Code

PVSIO functionality: let us execute functions that were specified in PVS.

Currently: trying to use the PVSIO to read terms from a file and write the output to a different file.

Implementation - Coq extracted Code

Transform the set of inference rules of Ayala et. al. (LOPSTR 2017) in an algorithm (perhaps by giving a heuristic on how to apply the rules), formalise its correctness and then use the Coq feature of code extraction.

Work in Progress, Conclusion

Conclusion

Conclusion

- Nominal C-Unification was (hopefully) explained.
- Our work on a certified functional algorithm for the task was discussed.

Thank You

Thank you! Any questions?

References I

M. Ayala-Rincón, W. de Carvalho-Segundo, M. Fernández, and D. Nantes-Sobrinho.
Nominal C-Unification.
In *LOPSTR 2017, Revised Selected Papers*, volume 10855 of *LNCS*, 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.
Theoretical Computer Science, 781:3–23, 2019.

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

References II

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.

E. Contejean.
A certified AC matching algorithm.
In *Rewriting Techniques and Applications, 15th International Conference, RTA 2004, Proceedings*, volume 3091 of *LNCS*, 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.

Appendix I - Functional Nominal C-Unification Algorithm

Functional Nominal C-Unification Algorithm I

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

  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**

      $\sigma' = \{X \to \pi^{-1} \cdot t\}$

      $\sigma'' = \sigma' \circ \sigma$

      $(\Delta', \text{bool1}) = \text{appSub2Ctxt}(\sigma', \Delta)$

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

      $\mathit{UnPrb}'' = (\mathit{UnPrb}')\sigma' + (\mathit{FxPntEq})\sigma'$

Functional Nominal C-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 C-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 C-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 C-Unification Algorithm V

```
else if t == [a]t_1 then
    if s == [a]s_1 then
        UnPrb'' = [(t_1, s_1)] + UnPrb'
        return UNIFY(Δ, σ, UnPrb'', FxPntEq)
    else if s == [b]s_1 then
        (Δ', bool1) = fresh(a, s_1)
        Δ'' = Δ ∪ Δ'
        UnPrb'' = [(t_1,  (a b) s_1)] + UnPrb'
        if bool1 then
            return UNIFY(Δ'', σ, UnPrb'', FxPntEq)
        else return null
        end if
    else return null
```

Functional Nominal C-Unification Algorithm VI

end if

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

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 C-Unification Algorithm VII

else $\triangleright$ t is of the form $f(t_1, t_2)$
  if s != $f(s_1, s_2)$ **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
end if
end if
end if
end procedure

Appendix II - Related Work

Related Work