Functional Nominal C-Unification

XVI Seminário Informal (, mas Formal!) - GTC - UnB

Gabriel Silva

February 21, 2019

Advisor: Mauricio Ayala-Rincón
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.

Purpose of Presentation

We revisit the problem of nominal unification with commutative operators and briefly comment about a **functional** algorithm for nominal C-unification and its formalisation.

Background

Background

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:

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

The symbols denote respectively: unit, atom term, suspended variable, abstraction, pair and function application.
We impose a restriction on the syntax of commutative function symbols: they must receive pairs.

Examples of Permutation Actions

Permutations act on atoms and terms:

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

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:

- $\sigma = \{X \rightarrow f(a, b)\}$, $t = f(X, c)$, $t\sigma = f(f(a, b), c)$.

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

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

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

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

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

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 Commutative Symbols

We need to add a rule to take into account commutative function symbols. Therefore, if a function symbol is commutative, the following rule can be applied:

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

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 C-Unification

Nominal C-Unification

Definition of the Problem

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 ($s \stackrel{?}{\approx}_{\alpha} t$) and freshness constraints ($a \stackrel{?}{\#} s$).

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 $s \approx_\alpha t \in P$
- There exist λ such that $\nabla \vdash \delta\lambda \approx_\alpha \sigma$

Nominal C-Unification

Differences from Nominal Syntactic Unification

Difference from Syntactic Unification

C-Unification has 2 main differences when compared with 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.

Nominal C-Unification

Comments About Functional Nominal C-Unification Algorithm

General Comments About the Functional Nominal C-Unification Algorithm

- We will talk about the main points behind 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 $(\Delta, \sigma, \mathit{FxPntEq})$.
  - Example: $[(\Delta_1, \sigma_1, \mathit{FxPntEq}_1)$, ..., $(\Delta_n, \sigma_n, \mathit{FxPntEq}_n)]$

Example of the Algorithm

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

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

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

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

$$(\emptyset, id, (a\ b)\cdot X \approx_\alpha X)$$

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

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

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

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

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

Soundness of Algorithm - an Example

Considering the last example, soundness guarantees that if $(\emptyset, X \to a+b)$ is a solution to $(\emptyset, id, (a\ b) \cdot X \approx_\alpha X)$ then $(\emptyset, X \to a+b)$ is a solution to $(\emptyset, id, f((a\ b) \cdot X, c)\ \approx_\alpha\ f(X, c))$.

Completeness of the Algorithm

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

Comments About the Formalisation

- Almost 200 lemmas were specified and proved in order to get soundness and completeness of the nominal C-unification algorithm.
- Completeness was harder to formalise than soundness.

Comments About the Formalisation

- The proof of both theorems was by induction on the lexicographic measure:

$$\langle |\mathit{Var}(\mathit{UnPrb} \cup \mathit{FxPntEq})|, \mathit{size}(\mathit{UnPrb})\rangle$$

- The hardest case happened when dealing with suspended variables.

Comments About the Formalisation

- Working modulo commutativity, we had to:
    - Unify commutative function symbols - easy
    - Handle fixpoint equations - easy
    - Deal with the appropriate data structure for the unification problems and the solutions to be obtained - hard

Related Work and Contribution

Related Work and Contribution

This works extends the work of [2]:

- [2] proved, using PVS, that a nominal unification algorithm is sound and complete.
- We extended the specification of [2] to prove that a nominal C-unification algorithm is sound and complete.

Related Work and Contribution

This work is similar, but not equal, to the work of [1]:

- [1] proposes a set of rules for nominal C-unification and, using Coq, shows this set of rules is sound and complete.
- We propose an algorithm for C-unification, not a set of rules to be applied to a unification problem.
- [1] uses a lexicographic order with 4 parameters. We were able to reduce the lexicographic order to 2 parameters.

Conclusion and Future Work

Conclusion

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

Future Work

Future work:

- Extend algorithm for A and AC-unification and verify its correctness in PVS.
- Work with other equational theories.

References I

M. Ayala-Rincón, W. de Carvalho-Segundo, M. Fernández, and D. Nantes-Sobrinho.
Nominal C-unification.
arXiv preprint arXiv:1709.05384, 2017.

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.

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

References II

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 C-Unification Algorithm

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 C-Unification Algorithm I

procedure $\text{UNIFY}(\Delta, \sigma, \mathit{UnPrb}, \mathit{FxPntEq})$
    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**

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

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

  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($\Delta, \sigma, UnPrb'', FxPntEq$)
    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 UNIFY($\Delta'', \sigma, UnPrb'', FxPntEq$)
        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                                        ▷ 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 = UNIFY(Δ, σ, UnPrb_1, FxPntEq)
                    UnPrb_2 = [(s_1, t_2)] + [(s_2, t_1)] + UnPrb'
                    sol_2 = UNIFY(Δ, σ, UnPrb_2, FxPntEq)
                    return APPEND(sol_1, sol_2)
                end if
            end if
        end if
    end if
end procedure
```