Functional Nominal C-Unification

Seminário de Computação - UnB

Gabriel Silva

November 22, 2018

Advisor: Mauricio Ayala-Rincón
Department of Mathematics - University of Brasília

Table of contents

Introduction

Nominal Syntax

Nominal syntax extends first-order syntax 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

The problem of nominal unification with commutative operators is revisited, and a **functional** algorithm for nominal C-unification is presented.

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:

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

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:

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

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:

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

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

$$\frac{}{\Delta \vdash \langle\rangle \approx_\alpha \langle\rangle}\ (\approx_\alpha \langle\rangle)$$

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

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

$$\frac{\Delta \vdash s \approx_\alpha t}{\Delta \vdash [a]s \approx_\alpha [a]t}\ (\approx_\alpha [a]a)$$

$$\frac{\Delta \vdash s \approx_\alpha (a\ b) \cdot t,\ a\#t}{\Delta \vdash [a]s \approx_\alpha [b]t}\ (\approx_\alpha [a]b)$$

$$\frac{ds(\pi, \pi')\#X \subseteq \Delta}{\Delta \vdash \pi \cdot X \approx_\alpha \pi' \cdot X}\ (\approx_\alpha \mathit{var})$$

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

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

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

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

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

Derivation Rules as a Sequent Calculus

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

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

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

Nominal C-Unification

Differences from Nominal Syntactic Unification

Difference from Syntactic Unification - Fixpoint Equations

A fixpoint equation is of the form $\pi \cdot X \approx_{\alpha} \gamma \cdot X$. In syntactic nominal unification, this is resolved by adding to the context the constraints that guarantee that atoms that are affected in different ways by π and γ must be fresh in X.

Difference from Syntactic Unification - Fixpoint Equations

However, this approach is not complete in C-unification, because of commutativity.

Take for example $\pi = (a\ b)$, $\gamma = nil$ and the substitution $X \to a + b$. In this case, $b + a \approx_\alpha a + b$. However, the treatment of syntactic unification would lose this solution, since neither a or b are fresh in $a + b$.

Because of that, fixpoint equations are part of the solution for a C-unification problem.

Difference from Syntactic Unification - Set of Solutions

Let f be a commutative symbol. The equation $f(t_0, t_1) \approx_\alpha f(s_0, s_1)$ can be solved by solving $(t_0 \approx_\alpha s_0$ and $t_1 \approx_\alpha s_1)$ OR by solving $(t_0 \approx_\alpha s_1$ and $t_1 \approx_\alpha s_0)$.

This means we need to consider two branches. Since each branch can generate solutions, there is now a set of solutions to be obtained, instead of only one.

Nominal C-Unification

A Functional Nominal C-Unification Algorithm

General Comments About the Functional Nominal C-Unification Algorithm

- We will show 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 $(\Delta, \sigma, FxPntEq)$.

Resume of Algorithm I

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

A 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**
            $\sigma' = \{X \to \pi^{-1} \cdot t\}$
            $\sigma'' = \sigma' \cup \sigma$
            $(\Delta',$ bool1$) =$ appSub2Ctxt(σ', Δ)
            $\mathit{UnPrb}'' = (\mathit{UnPrb}')\sigma' + (\mathit{FxPntEq})\sigma'$

A Functional Nominal C-Unification Algorithm II

```
        if bool1 then return UNIFY(Δ′, σ″, UnPrb″, null)
        else return null
        end if
    else
        if t == a then
            if s == a then
                return UNIFY(Δ, σ, UnPrb′, FxPntEq)
            else
                return null
            end if
```

A Functional Nominal C-Unification Algorithm III

else if $t == \pi \cdot X$ **then**
    if (X not in s) **then**
        $\triangleright$ Similar to case above where
        $\triangleright$ 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

A 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

A Functional Nominal C-Unification Algorithm V

```
        else if t == [a]t1 then
            if s == [a]s1 then
                UnPrb'' = [(t1, s1)] + UnPrb'
                return UNIFY(Δ, σ, UnPrb'', FxPntEq)
            else if s == [b]s1 then
                (Δ', bool1) = fresh(a, s1)
                Δ'' = Δ ∪ Δ'
                UnPrb'' = [(t1,  (a b) s1)] + UnPrb'
                if bool1 then
                    return UNIFY(Δ'', σ, UnPrb'', FxPntEq)
                else return null
                end if
            else return null
```

A 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

```
            else                                        ▷ t is of the form f(t1, t2)
                if s != f(s1, s2) then return null
                else
                    UnPrb1 = [(s1, t1)] + [(s2, t2)] + UnPrb'
                    sol1 = UNIFY(Δ, σ, UnPrb1, FxPntEq)
                    UnPrb2 = [(s1, t2)] + [(s2, t1)] + UnPrb'
                    sol2 = UNIFY(Δ, σ, UnPrb2, FxPntEq)
                    return APPEND(sol1, sol2)
                end if
            end if
        end if
    end if
end procedure
```

Example of the Algorithm

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

$$\begin{array}{cc}
\text{unify}(\emptyset,\ \text{id},\ [(a\ b)\cdot X \approx_\alpha X,\ c \approx_\alpha c],\ \emptyset) & \text{unify}(\emptyset,\ \text{id},\ [(a\ b)\cdot X \approx_\alpha c,\ X \approx_\alpha c],\ \emptyset) \\
| & | \\
\text{unify}(\emptyset,\ \text{id},\ [c \approx_\alpha c],\ (a\ b)\cdot X \approx_\alpha X) & \text{unify}(\emptyset,\ \{X \to c\},\ [c \approx_\alpha c],\ \emptyset) \\
| & | \\
\text{unify}(\emptyset,\ \text{id},\ \text{null},\ (a\ b)\cdot X \approx_\alpha X) & \text{unify}(\emptyset,\ \{X \to c\},\ \text{null},\ \emptyset) \\
| & | \\
(\emptyset, id, (a\ b)\cdot X \approx_\alpha X) & (\emptyset, \{X \to c\}, \emptyset)
\end{array}$$

Sketch of a Bigger Example

Let t and s be the two terms we unified in the previous example. Imagine we are trying to unify $f(t, Y) = f(s, \pi \cdot Y)$ where f is a commutative operator:

- In one branch, first we unify t and s obtaining the solutions of the previous slide. We add to these 2 solutions the fixpoint equation $Y \approx_\alpha \pi \cdot Y$.
- In the other branch, we unify t with $\pi \cdot Y$ and then proceed to unify s with $\pi^{-1} \cdot t$.

Conclusion and Future Work

Conclusion

Nominal C-unification was explained (hopefully) and a functional algorithm was presented.

Future Work

Future work:

- Use a proof verification system to verify soundness and completeness of the algorithm.
- Extend algorithm for A and AC-unification and verify its correctness in a proof verification system.

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?