Nominal C-Unification and Nominal C-matching

Gabriel Ferreira Silva[1]

February 11, 2020

Advisor: Mauricio Ayala-Rincón
XII Summer Workshop - Department of Mathematics - University of Brasília (UnB)
XVII Seminário Informal(, mas Formal) do GTC da UnB
1 - Department of Computer Science - University of Brasília (UnB)

Table of contents

1. Introduction

2. Preliminaries

3. Functional Nominal C-Unification

4. Nominal C-Matching through Nominal C-Unification with Protected Variables

5. Experiments

6. Conclusion and Future Work

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.

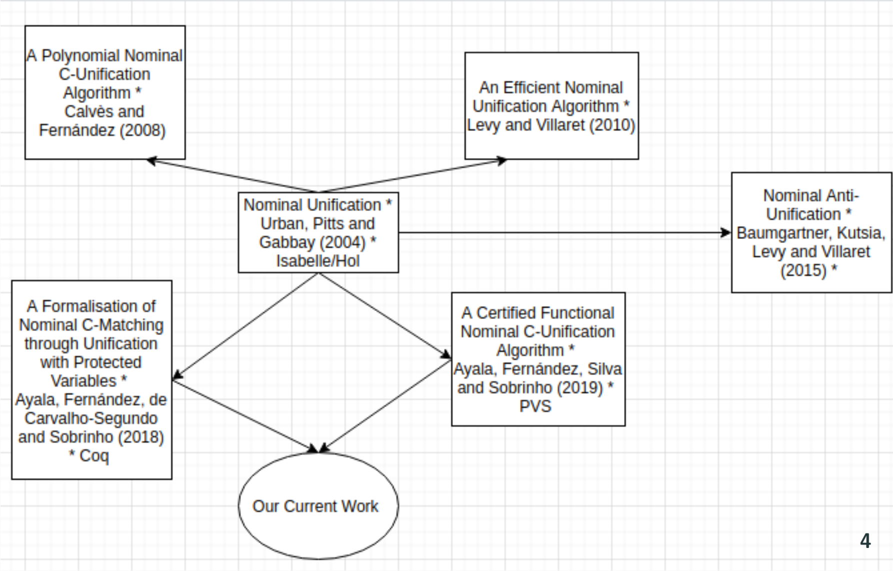
Unification

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



Purpose of Talk

- Briefly explain nominal C-unification.
- Discuss modifications needed and lessons learned in order to adapt the nominal C-unification algorithm to handle C-matching.
- Present experiments with implementations of the nominal C-unification algorithm.

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:

$$s, t \quad ::= \quad \langle\rangle \;\mid\; a \;\mid\; \pi \cdot X \;\mid\; [a]t \;\mid\; \langle s, t\rangle \;\mid\; f\ t \;\mid f^{C}\langle t_1, t_2\rangle$$

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:

- $t = b$, $\pi = (a\ b)$, $\pi \cdot t = a$.

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:

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

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

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

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

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

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

Derivation Rules for α-Equivalence

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

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

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

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

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

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

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

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

Differences from Syntactic Unification

Nominal C-Unification has 2 main differences when compared with syntactic nominal unification, related to fixpoint equations and set of solutions.

Differences from Syntactic Unification

A fixpoint equation is an equation of the form $\pi \cdot X \approx_\alpha \gamma \cdot X$.

A fixpoint equation is solved in syntactic unification by adding $ds(\pi, \gamma)\#X$ to the context.

This approach is not complete in nominal C-unification: consider for instance the fixpoint equation $(a\ b) \cdot X \approx_\alpha X$ and the instantiation $X \to a + b$.

Fixpoint equations are carried on as part of the solution to unification problems.

Differences from Syntactic Unification

In nominal C-unification, we obtain a set of solutions, not just one, because commutativity introduces branches, as will be shown in an example.

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 allows 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:

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

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

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

The First Part of the Functional Nominal C-Unification Algorithm

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

Functional Nominal C-Unification

Example

Example of the Algorithm

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

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

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

null

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

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

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

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

Nominal C-Matching through Nominal C-Unification with Protected Variables

Nominal C-Matching

Matching can be seen as unification, but the variables that occur in the right hand side of *UnPrb* cannot be instantiated.

How To do It

Add a set of protected variable ($\mathcal{X}$) as another parameter to the algorithm, and prove soundness and completeness of the algorithm for an arbitrary $\mathcal{X}$.

- If $\mathcal{X} = \emptyset$ then we have a nominal C-unification algorithm.
- If $\mathcal{X}$ is the set of variables in the right hand side of *UnPrb* then we have a nominal C-matching algorithm.
- If $\mathcal{X}$ is the set of variables in *UnPrb* we have a nominal C-equality checking algorithm.

Lessons Learned - A First Attempt

Preliminary attempt: when the algorithm would instantiate a variable X, first check if it is in $\mathcal{X}$. If it is not, proceed normally. If it is, return an empty list, as there cannot be a solution. Is this sufficient?

Lessons Learned - A First Attempt

No. Consider for instance that we are trying to match $t = \pi \cdot Y$ with $s = \pi' \cdot X$ and we have $X \in \mathcal{X}$ but $Y \notin \mathcal{X}$. Instantiate X to solve the matching problem is not possible, but the algorithm should not return an empty list, as it is possible to instantiate Y.

Lessons Learned - Proving Soundness and Completeness

Let $Rvar(UnPrb)$ be the set of variables occuring in the right hand side of $UnPrb$. Can we specify the main theorems of soundness and completeness of nominal C-matching by passing $Rvar(UnPrb)$ as $\mathcal{X}$?

Lessons Learned - Proving Soundness and Completeness

No. Because the proofs of the main theorems for soundness and completeness are done by induction and from one recursive call to the other $Rvar(UnPrb)$ may change, while $\mathcal{X}$ stays constant during the whole execution of the algorithm.

Lessons Learned - Proving Soundness and Completeness

We must prove the correctness of the algorithm for an arbitrary $\mathcal{X}$ and then obtain as corollaries the correctness of it for nominal C-unification and nominal C-matching by suitable instantiation of $\mathcal{X}$.

Experiments

Experiments

Work in Progress - Implementation and Experiments

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 - Done
- Coq verified code - To Be Done
- Experiments - Working now

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 (p_{var}).
- When dealing with a commutative function application, change the order of the two arguments (p_C).
- When dealing with an abstraction, "change" the atom being abstracted (p_{abs}).
- When an atom a is encountered, change the atom to a different atom b (p_{atom}).

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

Example Generator - Probability of Each Type of Term

Table 1: Probability of Generating Each Type of Term.

Type of the term	Probability
Atom	0.1
Suspended Variable	0.2
Unit	0.1
Abstraction	0.2
Pair	0.1
Function Application	0.2
Commutative Function Application	0.1

Example Generator - Number of Different Terms in the Domain

Table 2: Number of Different Atoms, Variables, Function Applications and Commutative Function Applications in the Domain.

Type of Term	Different Terms
Atom	10
Variable	10
Function Application	5
Commutative Function Application	5

Example Generator - Constructing s from the Term t

Table 3: Probability of Making Modifications in the Term s when Constructing It From the Term t.

Type of Modification	Probability
p_{var}	0.05
p_C	0.5
p_{abs}	0.5
p_{atom}	0.1

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: We have the Python implementation and the PVSIO implementation and we are running experiments to compare them.

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.

Preliminary Results

The two implementations gave the same output.

Table 4: Time Each Implementation Took to Unify a Given Number of Terms.

Unification Problems	Python - Time	PVS - Time
1000	< 1s	43s
2000	< 1s	1min24s
10000	3s	Error - Stack Overflow

Conclusion and Future Work

Conclusion

- Nominal C-Unification was (hopefully) explained.
- Our work on adapting the algorithm of nominal C-unification to the task of matching has been discussed.
- Our preliminary experiments were described.

Future Work

- Finish experiments.
- Work with AC-Unification.

Thank You

Thank you! Any questions?