Eberhard Karls Universität Tübingen
Mathematisch-Naturwissenschaftliche Fakultät
Wilhelm-Schickard-Institut für Informatik

Bachelor Thesis Computer Science

# Higher-Order Unification for Data and Codata Types

Julia Wegendt

Date

**Reviewer**

Prof. Dr. Klaus Ostermann
Department of Computer Science
University of Tübingen

**Wegendt, Julia**
*Higher-Order Unification for Data and Codata Types*
Bachelor Thesis Computer Science
Eberhard Karls Universität Tübingen
Period: from-till

# Abstract

Write here your abstract.

# Acknowledgements

Write here your acknowledgements.

# Contents



# List of Theorems

# 1 Introduction

# 2 The Untyped Calculus ND

I will be introducing the Untyped Calculus ND, based on ...

## 2.1 Syntax of the Untyped Calculus ND

Some knowledge of notation is necessary to familiarize oneself with the syntax of the Untyped Calculus. $X$ represents a (possibly empty) sequence $X_1, ...X_i, ...X_n$.
A pattern match $e.\mathbf{case}\{\overline{K(\overline{x}) \Rightarrow e}\}$ matches a term $e$ against a sequence of clauses, each clause consisting of a constructor and an expression. The expression associated with first constructor that matches the term is the result of the pattern match. For a copattern match $\mathbf{cocase}\{\overline{d(\overline{x}) \Rightarrow e}\}$ the same rules apply, but instead of constructors, the term is matched against destructors.

$$
\begin{array}{rcll}
e & ::= & x & \text{Variable} \\
 & | & K(\overline{e}) & \text{Constructor} \\
 & | & e.d(\overline{e}) & \text{Destructor} \\
 & | & e.\mathbf{case}\{\overline{K(\overline{x}) \Rightarrow e}\} & \text{Pattern match} \\
 & | & \mathbf{cocase}\{\overline{d(\overline{x}) \Rightarrow e}\} & \text{Copattern match}
\end{array}
$$

**Definition 2.1** (Terms of the Calculus ND)**.**

In pattern and copattern matches, every con- or destructor may occur no more than once.

Let's look at some examples to ... All rudimentary datatypes can be constructed through constructors:
The Booleans `True` and `False` are constructors on an empty sequence.

Abstract data types like lists, arrays, records etc. can similarly be defined through constructors:
`Cons(True, Cons(False, Nil))` and `Date(...)`

We can use Pattern matching to represent conditionals: Lik this if statement: "$\mathbf{if} e_1 \mathbf{then} e_2 \mathbf{else} e_3$" which is analogous to: $e_1.\mathbf{case}\{\texttt{True} \Rightarrow e_2, \texttt{False} \Rightarrow e_3\}$ Or the expression which tests whether a given list contains the number 5: $e.\mathbf{case}\{\}$

## 2.2 Free Variables, Substitutions, Contexts

**Definition 2.2** (Free Variables)**.** The set of free variables of a term $e$ is $\mathrm{FV}(e)$. A term is closed if this set is empty. Free Variables are defined recursively over the structure of terms as follows:

$$
\begin{aligned}
\mathrm{FV}(x) &:= \{x\} \\
\mathrm{FV}(K(e_1, ...e_n)) &:= \mathrm{FV}(e_1) \cup \cdots \cup \mathrm{FV}(e_n) \\
\mathrm{FV}(e.d(e_1, ...e_n)) &:= \mathrm{FV}(e) \cup FV(e_1) \cup \cdots \cup \mathrm{FV}(e_n) \\
\mathrm{FV}(e.\mathbf{case}\{\overline{K(\overline{x}) \Rightarrow e}\}) &:= \mathrm{FV}(e) \cup (\mathrm{FV}(e_1) \backslash \overline{x}) \cup \cdots \cup (\mathrm{FV}(e_n) \backslash \overline{x}) \\
\mathrm{FV}(\mathbf{cocase}\{\overline{d(\overline{x}) \Rightarrow e}\}) &:= (\mathrm{FV}(e_1) \backslash \overline{x}) \cup \cdots \cup (\mathrm{FV}(e_n) \backslash \overline{x})
\end{aligned}
$$

**Definition 2.3** (Substitution)**.** A simultaneous substitution $\sigma$ of the terms $e_1, ...e_n$ for the distinct variables $x_1, ...x_n$ is defined as follows:

$$\sigma ::= [e_1, ...e_n \backslash x_1, ...x_n]$$

The set of variables for which the substitution is defined is called the domain. The set of free variables which appear in the substitution is called the range.

**Definition 2.4** (Domain and Range of a Substitution)**.** The definitions of Domain and Range of a Substitution are as follows:

$$
\begin{aligned}
\mathtt{dom}([e_1, ...e_n/x_1, ...x_n]) &:= \{x_1, ...x_n\} \\
\mathtt{rng}([e_1, ...e_n/x_1, ...x_n]) &:= \mathrm{FV}(e_1) \cup \cdots \cup \mathrm{FV}(e_n)
\end{aligned}
$$

What is actually interesting is what happens when we appy a substitution to an expression

**Definition 2.5** (Action of a Substitution)**.** The action of a substitution $\sigma$ on a term $e$, written as $e\sigma$ and is defined as follows:

$$
\begin{aligned}
x[e_1, ...e_n/x_1, ...x_n] :=& e_i \quad (\text{if } x = x_i) \\
y\sigma :=& y \quad (\text{if } y \notin \mathtt{dom}(\sigma)) \\
(K(e_1, ...e_n))\sigma :=& K(e_1\sigma, \ldots, e_n\sigma) \\
(e.d(e_1, ...e_n))\sigma :=& (e\sigma).d(e_1\sigma, \ldots, e_n\sigma) \\
(e.\mathbf{case}\{\overline{K(\overline{x}) \Rightarrow e}\})\sigma :=& (e\sigma).\mathbf{case}\{\overline{K(\overline{y} \Rightarrow (e\sigma')\sigma)}\} \\
(\mathbf{cocase}\{\overline{d(\overline{x}) \Rightarrow e}\})\sigma :=& \mathbf{cocase}\{\overline{(d(\overline{y}) \Rightarrow (e\sigma')\sigma)}\}
\end{aligned}
$$

Where $\sigma'$ is a substitution that ensures that we don't bind new variables: $\sigma'$ has the form $[y_1, \ldots, y_n/x_1, ...x_n]$ and all $y_i$ are fresh for both the domain and the range of $\sigma$.

The composition of two substitutions $\sigma_2 \circ \sigma_1$ which is equivalent to first applying the substitution $\sigma_1$, then the substitution $\sigma_2$.

**Definition 2.6** (Composition of Subsitutions)**.** Given two substitutions

$$\sigma_1 := [e_1, ...e_n/x_1, ...x_n], \qquad \sigma_2 := [t_1, \dots, t_m/y_1, \dots, y_m]$$

Composition is defined as:

$$\sigma_2 \circ \sigma_1 := [e_1\sigma_2, \dots, e_n\sigma_2, t_j, \dots, t_k/x_1, ...x_n, y_j, \dots, y_k]$$

Where $j, \dots, k$ ist the greatest sub-range of indices $1, \dots, m$ such that none of the variables $y_j$ to $y_k$ is in the domain of $\sigma_1$

**Definition 2.7** (Idempotency)**.** A substitution $\sigma$ is idempotent, iff. $\sigma \circ \sigma = \sigma$. Concretely, this means that it doesn't matter how often we apply a substitution to a given problem.

For example, $[\textbf{cocase}\{\texttt{Ap}(x) \Rightarrow x\}/y]$ is idempotent, since:

$$\begin{aligned}&[\textbf{cocase}\{\texttt{Ap}(x) \Rightarrow x\}/y] \circ [\textbf{cocase}\{\texttt{Ap}(x) \Rightarrow x\}/y] \\ &= [\textbf{cocase}\{\texttt{Ap}(x) \Rightarrow x\}[\textbf{cocase}\{\texttt{Ap}(x) \Rightarrow x\}/y]/y] \\ &= [\textbf{cocase}\{\texttt{Ap}(x) \Rightarrow x\}/y]\end{aligned}$$

On the other hand, the substitution $[\textbf{cocase}\{\texttt{Ap}(y) \Rightarrow x\}/x]$ is not idempotent, since:

$$\begin{aligned}&[\textbf{cocase}\{\texttt{Ap}(y) \Rightarrow x\}/x] \circ [\textbf{cocase}\{\texttt{Ap}(y) \Rightarrow x\}/x] \\ &= [\textbf{cocase}\{\texttt{Ap}(y) \Rightarrow x\}[\textbf{cocase}\{\texttt{Ap}(y) \Rightarrow x\}/x]/x] \\ &= \textbf{cocase}\{\texttt{Ap}(y) \Rightarrow (\texttt{Ap}(y) \Rightarrow x)\} \neq [\textbf{cocase}\{\texttt{Ap}(y) \Rightarrow x\}/x]\end{aligned}$$

**Definition 2.8** (More General)**.** A substitution $\sigma$ is more general than a substitution $\theta$, iff. there exists a mapping $\tau$, such that: $\theta = \tau \circ \sigma$.

For example, for the problem $\alpha x = zx$, both $\sigma_1 = [z/\alpha]$ and $\sigma_2 = [\lambda y.zx, \lambda y.zx/\alpha, z]$ are solutions, but $\sigma_1$ is more general than $\sigma_2$, since there exists a substitution $\sigma_3 = [\lambda y.zx/z]$, and:

$$\sigma_3 \circ \sigma_1 = [z[\lambda y.zx/z], \lambda y.zx/\alpha, z] = [(\lambda y.zx), \lambda y.zx/\alpha, z] = \sigma_2$$

## 2.3 Conversion

**Definition 2.9** (Beta-Conversion)**.** A single step of beta-conversion $e_1 \equiv^1_\beta e_2$ is defined as follows:

$$K(\overline{e}).\textbf{case}\{\dots, K(\overline{x}) \Rightarrow e\} \equiv^1_\beta e[\overline{e}/x] \qquad (\beta\text{-Data})$$

$$\textbf{cocase}\{\dots, d(\overline{x}) \Rightarrow e, \dots\}.d(\overline{e}) \equiv^1_\beta e[\overline{e}/x] \qquad (\beta\text{-Codata})$$

We require that the constructor $K(\overline{e})$ and the constructor $K(\overline{x})$ have the same number of arguments. This, in short ensures that we don't generate stuck terms, i.e. Terms that can't be evaluated.

**Definition 2.10** (Eta-Conversion for Codata)**.** A single step of eta-conversion $e_1 \equiv_\eta^1 e_2$ is defined as follows:

$$\mathbf{cocase}\{d(\overline{x}) \Rightarrow e.d(\overline{x})\} \equiv_\eta^1 e \quad (\text{if } \overline{x} \notin \mathrm{FV}(e)) \qquad (\eta\text{-Codata})$$

# 3 The Unification Problem

The Unification Problem is described by a set of equations with expressions on each side $\overline{e = e}$ containing unknown unification variables $\alpha^?$, where our goal is to find a simultaneous substitution $[e_1, ...e_n/\alpha_1^?, \ldots, \alpha_n^?]$ which substitutes expressions for unification variables, such that the sides of the given equations are the same (respectively).
To describe this formally, we need to expand our syntax to be able to represent unification variables.

$$\begin{array}{rcll} e & ::= & \alpha^? & \text{Unification variable} \\ & | & x & \text{Variable} \\ & | & K(\overline{e}) & \text{Constructor} \\ & | & e.d(\overline{e}) & \text{Destructor} \\ & | & e.\mathbf{case}\{\overline{K(\overline{x}) \Rightarrow e}\} & \text{Pattern match} \\ & | & \mathbf{cocase}\{\overline{d(\overline{x}) \Rightarrow e}\} & \text{Copattern match} \end{array}$$

In unification, we are not concerned with syntactic equality, but want a broader set of terms to be equal to one another. Depending on the type of unification problem one wants to solve, they may want to only include beta-equality or both beta- and eta-equality.

Thus, I will use $\equiv$ for unification problems, where two terms are either syntactically equivalent, beta-equivalent or eta-equivalent. This essentially means that two terms are equivalent if they are equivalent after function application and/or are equivalent externally.

**Definition 3.1** ((Higher-Order) Unification Problem)**.** A unification problem is a set of equations consisting of expressions $\overline{e \equiv e}$ consisting of our expanded syntax.

**Definition 3.2** (Solution)**.** A solution to a given unification problem is described by a simultaneous substitution $[e_1, ...e_n/\alpha_1^?, \ldots, \alpha_n^?]$ which when applied to the problem solves it, i.e. makes the sides of the equations equal

For some unification problems, there exists a solution and it is obvious: The solution to $\alpha^? \equiv True$ is $[True/\alpha^?]$

For some unification problems, it is rather obvious that there is no solution on the other hand: There is no mapping of unification variables that make both sides of $True \equiv False$ the same.

Solutions aren't necessarily unique, either. The problem $\alpha^?.\texttt{Ap}(5) \equiv 5$ has multiple solutions:

$[\textbf{cocase}\{\texttt{Ap}(x) \Rightarrow x\}/\alpha^?]$ and $[\textbf{cocase}\{\texttt{Ap}(x) \Rightarrow 5\}/\alpha^?]$,

where Ap is a function applicator. Thus, the two given solutions are the identity function and the constant function 5, which when consuming the argument 5, both give back 5.

The interesting unification problems are those where it is not clear at first sight whether there exists a solution.

## 3.1 Types Of Solutions

To be able to discuss the algorithm for solving the unification problem, we need to define some helpful concepts first.

**Definition 3.3** (Most General)**.** A solution is the most general unifier (mgu), iff. it is more general than all other solutions.

There does not always exist a mgu, for instance in the problem provided above ($\alpha^?.\texttt{Ap}(5) \equiv 5$)

## 3.2 Fundamental Results

$$
\begin{array}{rcll}
e, t & ::= & \alpha^? & \text{Unification variable} \\
 & | & x & \text{Variable} \\
 & | & K(\overline{e}) & \text{Constructor}
\end{array}
$$

**Definition 3.4** (First order unification)**.**

**Theorem 3.1** (Decidability of First-Order Unification)**.** For first-order unification, there exists an algorithm on equations $E$, which always terminates, and returns the solution if there exists one. In particular, this solution is always a mgu (i.e. if there is a solution, then there always exists a most general one).

**Definition 3.5** (Unification algorithm for First-Order Unification )**.** $\perp$ is the symbol for fail. The algorithm is defined by non-deterministically applying the below rules:

$$E \cup \{e \equiv e\} \Rightarrow E \qquad \text{(delete)}$$

$$E \cup \{K(e_1 ... e_n) \equiv K(t_1 ... t_n)\} \Rightarrow E \cup \{e_1 \equiv t_1, ... e_n \equiv t_n\} \qquad \text{(decompose)}$$

$$E \cup \{K_1(e_1, ... e_n) \equiv K_2(t_1, ..., t_m)\} \Rightarrow \perp \quad \text{if } K_1 \neq K_2 \text{ or if } n \neq m \qquad \text{(conflict)}$$

$$E \cup \{e \equiv \alpha^?\} \Rightarrow E \cup \{\alpha^? \equiv e\} \qquad \text{(swap)}$$

$$E \cup \{\alpha^? \equiv e\} \Rightarrow E[e/\alpha^?] \cup \{\alpha^? \equiv e\} \quad \text{if } \alpha^? \in E \text{ and } \alpha^? \notin e \qquad \text{(eliminate)}$$

$$E \cup \{\alpha^? \equiv K(e_1, ... e_n)\} \Rightarrow \perp \quad \text{if } \alpha^? \in e_1, ... e_n \qquad \text{(check)}$$

This algorithm is based on the version presented by Martelli and Montanari [2], adapted to our syntax.

In Automated Reasoning (Volume 2) Chapter 16 [1], ...

**Theorem 3.2** (Decidability of Higher-Order Unification)**.** Higher-order unification includes unification problems containing higher-order terms (equivalent to lambda abstractions), and is covered by our introduced syntax. Higher-order unification is not decidable. This can be proven through reducing Hilbert's tenth problem to the unification problem.

# 4 Patterns Unification

Patterns Unification, also sometimes called the Pattern Fragment is a subsection of Higher-Order Unification. It is an extension of First-Order unification with its solution being similarly simple as the one to first-order unification. It was described first by Miller in ...

**Definition 4.1** (Normal Form)**.** The normal form **NF** is defined as follows:

$$n ::= x \mid \alpha^? \mid n.d(\overline{v}) \mid n.\mathbf{case}\overline{K(\overline{x} \Rightarrow v)}$$
$$v ::= n \mid K(\overline{v}) \mid \mathbf{cocase}\{\overline{d(\overline{x} \Rightarrow v)}\}$$

**Definition 4.2** (Pattern)**.** A pattern is any term $t$ where for its subterms of the form $(...(\alpha^?.\texttt{Ap}(e_1))...).\texttt{Ap}(e_n)$, it holds that all $e_1, ...e_n$ are distinct and bound variables in $t$ (NOT arbitrary expressions).

The intuition here is that in the above case, $\alpha^?$ is an unknown function applied to its arguments $e_1, ...e_n$, and anytime we apply an unknown function, we should read the problem as a definition for the function. Some examples of patterns:

$$t_1 = \mathbf{cocase}\{\texttt{Ap}(x) \Rightarrow \alpha^?.\texttt{Ap}(x)\}$$
$$t_2 = \mathbf{cocase}\{\texttt{Ap}(x) \Rightarrow \mathbf{cocase}\{\texttt{Ap}(y) \Rightarrow (\alpha^?.\texttt{Ap}(y)).\texttt{Ap}(x)\}\}$$

In each of these terms, our unification variable is applied to distinct variables which are bound through cocases.

However, these terms are not patterns:

$$t_3 = (\alpha^?.\texttt{Ap}(x)).\texttt{Ap}(y)$$
$$t_4 = \mathbf{cocase}\{\texttt{Ap}(y) \Rightarrow (\alpha^?.\texttt{Ap}(x))\}$$
$$t_5 = \mathbf{cocase}\{\texttt{Ap}(x) \Rightarrow (\alpha^?.\texttt{Ap}(x)).\texttt{Ap}(x)\}$$

as they all don't contain exactly one cocase per variable which would bind that variable distinctly.

**Theorem 4.1** (Decidability of Patterns Unification)**.** Patterns Unification is decidable. If there exists a solution, there also exists a mgu.

## References

[1] G. Dowek. Higher-order unification and matching. In J. A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning (in 2 volumes)*, pages 1009–1062. Elsevier and MIT Press, 2001.

[2] A. Martelli and U. Montanari. An efficient unification algorithm. *ACM Trans. Program. Lang. Syst.*, 4(2):258–282, Apr. 1982.

## Selbständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbständig und nur mit den angegebenen Hilfsmitteln angefertigt habe und dass alle Stellen, die dem Wortlaut oder dem Sinne nach anderen Werken entnommen sind, durch Angaben von Quellen als Entlehnung kenntlich gemacht worden sind. Diese Bachelorarbeit wurde in gleicher oder ähnlicher Form in keinem anderen Studiengang als Prüfungsleistung vorgelegt.

Ort, Datum Unterschrift