Anti-Unification in λP

Gabriela Ferreira
Universidade de Brasília - PPG MAT
G.S.Ferreira@mat.unb.br

1 Introduction

The predicate calculus, in short λP, is a component of the Barendregt cube and is the typing system that allows types to depend on terms. It is a powerful tool for expressing lists, arrays and other structures used daily for typing structures. Moreover, dependent types are implemented in functional programs like Agda and modern proof assistants like Coq.

The anti-unification problem consists of comparing two structures t_1 and t_2, finding a new one that represents the commonalities of t_1 and t_2. This problem was originally investigated by Plotkin in [4]. Later on, questions about anti-unification have been explored in high-order terms, for example, Cerna and Kutsia [2] proposed a generic framework for computing solutions that have no nested generalization variables and preserve the top maximal communalities of the structures. The survey [3] gives more information about equational and simply-typed anti-unification, including classification of the problem with different variants.

The objective is to extend the frameworks of term generalization in high-order theories, in special [2], to deal with dependent types. Since in this system we have no type-variables, the proposal is to treat type-constants as rigid parts, allowing divergences only in the terms that they depend on.

2 Preliminaries

The system λP is based on a set of pseudo-expressions $\mathcal{T}$ defined by the syntax

$$\mathcal{T} = V \mid C \mid \mathcal{T}\mathcal{T} \mid \lambda V : \mathcal{T}.\mathcal{T}\mathcal{T} \mid \Pi V : \mathcal{T}\mathcal{T}$$

Where V and C are infinite collections of variables and constants, respectively. Among the constants C, two elements are selected and given names $\star$ and $\Box$, such elements are called *sorts*.

A *statement* of λP is of the form $t : \tau$, where $t, \tau \in \mathcal{T}$. A *context* (typically Γ) is a finite linearly ordered set of statements with distinct variables subjects, if $\Gamma =< x_1 : \tau_1, \ldots, x_n : \tau_n >$ then $\Gamma, t : \tau :=< x_1 : \tau_1, \ldots, x_n : \tau_n, t : \tau >$. A *signature* (typically Σ) is a finite set of statements that assigning types and kinds to constants. The rules for construct statements are given in Table 1, where the letter s ranges over $\{\star, \Box\}$. We omitted well-formedness of signatures. Moreover, $\vdash t : \tau$ denotes that the statement $t : \tau$ is derived in an empty context.

(cons) $\Gamma \vdash_\Sigma t : \tau$, where $t : \tau \in \Sigma$ (var) $\dfrac{\Gamma \vdash_\Sigma \tau : \star}{\Gamma, x : \tau \vdash_\Sigma x : \tau}$, where x is fresh variable in Γ

(weak) $\dfrac{\Gamma \vdash_\Sigma t : \tau \quad \Gamma \vdash_\Sigma \tau' : \star}{\Gamma, x : \tau' \vdash_\Sigma t : \tau}$ (form) $\dfrac{\Gamma \vdash_\Sigma \tau : \star \quad \Gamma, x : \tau \vdash_\Sigma \tau' : \mathrm{s}}{\Gamma \vdash_\Sigma (\Pi x : \tau.\tau') : \mathrm{s}}$

(app) $\dfrac{\Gamma \vdash_\Sigma F : (\Pi x : \tau.\tau') \quad \Gamma \vdash_\Sigma a : \tau}{\Gamma \vdash_\Sigma Fa : \tau'\{x \mapsto a\}}$

(abs) $\dfrac{\Gamma, x : \tau \vdash_\Sigma b : \tau' \quad \Gamma \vdash_\Sigma (\Pi x : \tau.\tau') : \mathrm{s}}{\Gamma \vdash_\Sigma (\lambda x : \tau.b) : (\Pi x : \tau.\tau')}$

Table 1: Rules for type-statements.

We are supposing that pseudo-expressions t are in η-long β-normal form, and we are considering equality modulo α-equivalence.

Definition 2.1. The set $\Theta_\Gamma^{\Gamma'}$ of well-typed substitutions from Γ to Γ' is the set of those substitutions θ such that for every $v : \nu \in \Gamma$, we have $\Gamma' \vdash_\Sigma v\theta : \nu\theta$. The set Θ_Γ of well-typed substitutions over Γ is the union over all well-formed contexts Γ' of $\Theta_\Gamma^{\Gamma'}$.

Definition 2.2. Given a signature Σ, two well-formed and consistent contexts Γ_1, Γ_2, a statement $\Gamma_1 \vdash_\Sigma t_1 : \tau_1$ is *more general* than $\Gamma_2 \vdash_\Sigma t_2 : \tau_2$ if there exists a substitution $\theta \in \Theta_{\Gamma_1}^{\Gamma_2}$ such that $t_1\theta = t_2$ and $\tau_1\theta = \tau_2$. This relation is denoted by $(\Gamma_1 \vdash_\Sigma t_1 : \tau_1) \preceq (\Gamma_2 \vdash_\Sigma t_2 : \tau_2)$, and its strict part by $\prec$. When no confusion is created, Σ is omitted. The *anti-unification problem* in λP is defined as:

Given a signature Σ such that $\vdash_\Sigma t_1 : \tau_1$ and $\vdash_\Sigma t_2 : \tau_2$,

Find a context Γ and a statement $r : \rho$ such that

- $(\Gamma \vdash_\Sigma r : \rho) \preceq (\vdash_\Sigma t_i : \tau_i)$, with $i \in \{1, 2\}$, and
- there is no generalization $\Gamma' \vdash r' : \rho'$ of $\vdash_\Sigma t_i : \tau_i$ such that $(\Gamma \vdash_\Sigma r' : \rho') \prec (\Gamma' \vdash_\Sigma r : \rho)$.

Moreover, this statement $\Gamma \vdash r : \rho$ is called a *least general generalization* (shortly, lgg) of $t_i : \tau_i$.

Example 1. Consider the anti-unification problem for $\vdash_\Sigma t_1 : \tau_1$ and $\vdash_\Sigma t_2 : \tau_2$, where

$$t_1 := (\lambda x : \mathtt{Nat}.\lambda y : (\Pi a : \mathtt{Nat}.\mathtt{Leq}(a, x)).-(x, y))$$
$$\tau_1 := (\Pi x : \mathtt{Nat}.\Pi y : (\Pi a : \mathtt{Nat}.\mathtt{Leq}(a, x)).\mathtt{Nat})$$

$$t_2 := (\lambda x : \mathtt{Nat}.\lambda y : (\Pi a : \mathtt{Nat}.\mathtt{Leq}(a, +(x, x))).-(+(x, x), y))$$
$$\tau_2 := (\Pi x : \mathtt{Nat}.\Pi y : (\Pi a : \mathtt{Nat}.\mathtt{Leq}(a, +(x, x))).\mathtt{Nat})$$

for a suitable Σ, it follows that a lgg is $\langle X : \Pi n : \mathtt{Nat}.\mathtt{Nat}\rangle \vdash_\Sigma r : \rho$, where

$$r := \lambda x : \mathtt{Nat}.\lambda y : (\Pi a : \mathtt{Nat}.\mathtt{Leq}(a, X(x))).-(X(x), y))$$
$$\rho := \Pi x : \mathtt{Nat}.\Pi y : (\Pi a : \mathtt{Nat}.\mathtt{Leq}(a, X(x))).\mathtt{Nat}$$

The colours red, orange and pink represent the respective divergence points in the compared objects. Observe that divergences were the same, that is, all arise from comparing x with $+(x, x)$. Hence, they are represented by the same generalization $X(x)$. Concerning generalizers substitutions, they are $\theta_1 = \{X \mapsto \lambda : \mathtt{Nat}.x\}$ and $\theta_2 = \{X \mapsto \lambda x : \mathtt{Nat}.+(x, x)\}$.

Example 2. Consider the anti-unification problem for

$$\vdash_\Sigma (\lambda x : \mathtt{Nat}.x) : (\Pi x : \mathtt{Nat}.\mathtt{Nat}), \quad \vdash_\Sigma \quad (\lambda y : \mathtt{Real}.y) : (\Pi y : \mathtt{Real}.\mathtt{Real})$$

with a suitable syntax Σ. Both have a similar structure; in fact, they are identity functions of different types, the first one mapping naturals into naturals, and the second one reals into reals. However, despite the commonalities, there is no answer for this problem. It happens because λP is a non-polymorphic calculus (observe that the rule (var) in Table 1 forbids elements of sort $\star$ and $\Box$ from being variables). Hence, when the differences are headed by type, there is no solution in our approach.

3 Generalization Rules

An *Anti-Unification Triple* (shortly, AUT) is of the form "$X(\vec{x}) \triangleq s \triangleq t$", where $X(\overline{x})$ is a tentative generalization of s and t, with $\vec{x}$ being a list of variables, and X a fresh variable called *generalizable variable* (shortly, genvar). Active AUTs represent problems of type statements that are pending to be solved, then they have the form "$X(\vec{x}) \triangleq s : \sigma \triangleq t : \tau$", where X is a meta-variable. Solved AUTs have the form '$X(\vec{x}) \triangleq s \triangleq t$" with s and t being terms, and they represent problems already solved.

A configuration is either $\bot$ or a triple $\langle P;\ S;\ \theta\rangle$ where P contains active AUTs, S stores solved AUTs, and θ is a substitution computed so far. In Table 2, a set of inference rules that operate in configuration is presented.

To solve the anti-unification problem for $\vdash_\Sigma t_1 : \tau_1$ and $\vdash_\Sigma t_2 : \tau_2$, the sketch of the proposal is:

1. Rewrite the statements in a way that the complete typing chains of terms appear when they are relevant for anti-unification analysis, obtaining the expressions t_1' and t_2' (this process was omitted due to lack of space).
2. with input $\langle\{X \triangleq t_1' \triangleq t_2'\};\ \emptyset;\ id\rangle$, apply the rules of Table 1, as far as possible.
3. if the last computed configuration is $\bot$, then no solution was found; otherwise, if the last computed solution is $\langle\emptyset;\ S', \theta\rangle$, then the given generalization is $X\theta$.

The procedure is terminating, thus it is an algorithm. Moreover, it is sound in the sense that it outputs a generalization of the input objects.

This algorithm is a generic framework for computing generalization, and more precise (SOL) rules need to be given to obtain solutions for specific variants of the problem.

Rules for Anti-Unification in λP
(Generic rules)

Strbx: **Starbox**

$$\frac{\langle\{X(\vec{x}) \triangleq \star : \Box \triangleq \star : \Box\} \uplus P;\ S;\ \theta\rangle}{\langle P;\ S;\ \theta\{X \mapsto \lambda\vec{x}.\star : \Box\}\rangle}$$

Dec: **Decomposition**

$$\frac{\langle\{X(\vec{x}) \triangleq h((s_1 : \sigma_1), \ldots, (s_m : \sigma_m)) : \sigma \triangleq h((t_1 : \tau_1), \ldots, (t_m : \tau_m)) : \tau\} \uplus P;\ S;\ \theta\rangle}{\langle\{Z(\vec{x}) \triangleq \sigma \triangleq \tau,\ Y_1(\vec{x}) \triangleq s_1 : \sigma_1 \triangleq t_1 : \tau_1,\ \ldots,\ Y_m(\vec{x}) \triangleq s_m : \sigma_m \triangleq t_m : \tau_m\} \cup P;\ S;\ \theta\theta'\rangle}$$

where $h \neq \star, Z, Y_1, \ldots, Y_m$ are fresh meta-variables, and $\theta' = \{X \mapsto \lambda\vec{x}.(h(Y_1(\vec{x}), \ldots, Y_m(\vec{x})) : Z(\vec{x}))\}$.

Abs: **Abstraction**

$$\frac{\langle\{X(\vec{x}) \triangleq \lambda y : \sigma.(s : \sigma') \triangleq \lambda z : \tau.(t : \tau')\} \uplus P;\ S;\ \theta\rangle}{\langle\{Y(\vec{x}) \triangleq \sigma \triangleq \tau,\ Z(\vec{x}, w) \triangleq (s : \sigma')\nu \triangleq (t : \tau')\nu,\} \cup P;\ S;\ \theta\{X \mapsto \lambda\vec{x}, w : Y(\vec{x}).Z(\vec{x}, w)\}\rangle}$$

where Y, Z, are fresh meta-variabes, w is a fresh variable, and $\nu = \{y \mapsto w, z \mapsto w\}$.

Prod: **Product**

$$\frac{\langle\{X(\vec{x}) \triangleq \Pi y : \sigma.(s : \sigma') \triangleq \Pi z : \tau.(t : \tau')\} \uplus P;\ S;\ \theta\rangle}{\langle\{Y(\vec{x}) \triangleq \sigma \triangleq \tau,\ Z(\vec{x}, w) \triangleq s\nu : \sigma' \triangleq t\nu : \tau'\} \cup P;\ S;\ \theta\{X \mapsto \lambda\vec{x}.\Pi w : Y(\vec{x}).Z(\vec{x}, w)\}\rangle}$$

where $\nu = \{y \mapsto w, z \mapsto w\}$, Y, Z are fresh meta-variables, w is a fresh variable.

Sol: **Solve**

$$\frac{\langle\{X(\vec{x}) \triangleq s : \sigma \triangleq t : \tau\} \uplus P;\ S;\ \theta\rangle}{\langle\{Z(\vec{x}) \triangleq \sigma \triangleq \tau\} \cup P;\ \{Y(y_1, \ldots, y_m) \triangleq (C_s\, y_1 \cdots y_m) \triangleq (C_t\, y_1 \cdots y_m)\} \cup S;\ \theta\theta'\rangle}$$

where $head(s) \neq head(t)$,$\sigma \neq (\star : \Box)$, $\tau \neq (\star : \Box)$, $\theta' = \{X \mapsto \lambda\vec{x}.(Y(q_1, \ldots, q_m) : Z(\vec{x}))\}$, $q_1, \ldots, q_m$ are distinct subterms of t or s, C_t and C_s are terms such that $(C_t\, q_1 \cdots q_m) = t$ and $(C_s\, q_1 \cdots q_m) = s$, C_t and C_s do not contain any $x \in \vec{x}$, and Z,Y, $y_1, \ldots, y_m$ are distinct fresh variables.

Mer: **Merge**

$$\frac{\langle\emptyset;\ \{X(\vec{x}) \triangleq s_1 \triangleq t_1, Y(\vec{y}) \triangleq s_2 \triangleq t_2\} \uplus S;\ \theta\rangle}{\langle\emptyset;\ \{X(\vec{x}) \triangleq s_1 \triangleq t_1\} \cup S;\ \theta\{Y \mapsto \lambda\vec{y}.X(\vec{x}\pi)\}\rangle,}$$

where $\pi : \{\vec{x}\} \to \{\vec{y}\}$ is a bijection, extended as a substitution, with $t_1\pi = t_2$ and $s_1\pi = s_2$.

Fail: **Fail**

$$\frac{\langle A;\ S;\ \theta\rangle}{\bot}$$ where $A \neq \emptyset$ and no other rules apply.

Table 2: Inference rules.

4 Work in Progress

We are investigating how to extend the notion of top maximal solutions for λP generalizations. It is crucial for the work because Cerna and Buran [1] proved the nullarity of anti-unification in the simply-typed lambda calculus for the unrestricted case. Since this calculus is embedded in λP, the result also holds for the system we are interested in now. Hence, restraining the form of the solutions is a way of ensuring that a minimal and complete set of generalization can be computed. The challenges faced in this extension arise from the dependence of the term trees on the definition of top-maximality. Since types and terms occur nested in the structures of this system, it requires a way of designing term trees in such a manner that terms, types, and type constructors can be identified just by looking at the representation. After concluding this extension, we aim to prove that our algorithm computes top-maximal solutions with no nested generalizable variables.

Another work in progress is the extension of the Common-Subterm variant (in short CS-variant, see [2]), for λP generalizations. After reaching a divergence in the compared objects, the procedure searches for subterms in common and represents them in the generalization. Therefore, with the extension being well established, we plan to adapt the current algorithm by changing the solve rule in such a manner that the procedure will produce CS-generalizations.

5 Future Work

Further on, we are planning to study the type of the problem (see [3]). The intuition says that it is of type at least finitary, but probably unary. However, it is necessary to formally prove it. Only them, we will investigate if the CS-variant procedure is complete.

References

[1] David M. Cerna and Michal Buran. One or nothing: Anti-unification over the simply-typed lambda calculus. *ACM Trans. Comput. Logic*, 2024. Accepted.

[2] David M. Cerna and Temur Kutsia. A generic framework for higher-order generalizations. In Herman Geuvers, editor, *4th International Conference on Formal Structures for Computation and Deduction, FSCD 2019, June 24-30, 2019, Dortmund, Germany*, volume 131 of *LIPIcs*, pages 10:1–10:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[3] David M. Cerna and Temur Kutsia. Anti-unification and generalization: A survey. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI 2023, 19th-25th August 2023, Macao, SAR, China*, pages 6563–6573. ijcai.org, 2023.

[4] Gordon D. Plotkin. A note on inductive generalization. *Machine Intelligence 5*, 5:153–163, 1970.

Acknoledments

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001.

About the author

Gabriela Ferreira is a PhD candidate in Mathematics at Universidade de Brasília, working under the supervision of Professor Mauricio Ayala-Rincón (supervisor, UnB) and Professor Temur Kutsia (cosupervisor, RISC/JKU).