# Computing Generalizers over Intersection and Union Type Theories

Gabriela Ferreira[1], David M. Cerna[2], Mauricio Ayala-Rincón[1], and Temur Kutsia[3]

[1] University of Brasilia (UnB), Brasília D.F., Brazil
222114451@aluno.unb.br, ayala@unb.br
[2] Czech Academy of Sciences Institute of Computer Science (CAS ICS), Prague, Czech Republic
dcerna@cs.cas.cz
[3] Research Institute for Symbolic Computation,
Johannes Kepler University Linz, Austria
kutsia@risc.jku.at

**Abstract**

We discuss how to extend anti-unification to generalize higher-order terms with different types by extending the type system used within an existing framework with union and intersection types. We provide examples illustrating desirable properties of the corresponding least general generalizers.

Keywords: Anti-Unification, higher-order terms, intersection and union types.

## 1 Introduction

Anti-unification was introduced and studied by Plotkin [18] and Reynolds [19] in the 1970s. It requires identifying similarities between two symbolic expressions and retaining them as a new symbolic expression, called a generalizer of the given ones. At the same time, the differences between the given expressions are also reflected in their generalization in the form of new variables. This new symbolic expression is referred to as *least general* (or most particular) when it maximally captures the structure of the input expressions and abstracts the differences by new variables uniformly. For instance, a least general generalizer (lgg) of two first-order terms $f(a, g(a))$ and $f(b, g(b))$ is $f(x, g(x))$. In the first-order syntactic case, the least general generalizer (lgg) is unique. But there are theories and problems for which there exist more than one (even infinitely many) lggs, or lggs do not exist at all.

In recent years, research on anti-unification has intensified, mainly due to its various applications. Questions about anti-unification have been studied in different syntactic and semantic frameworks; for example, Baader [4] studied it for a class called commutative theories, Alpuente *et al.* [2] considered anti-unification over equational theories with associative (A), and commutative (C) operators, Cerna and Kutsia studied theories with idempotent operators [8] and most recently, Ayala-Rincón *et al.* studied theories with absorbing operators [3]. Concerning higher-order anti-unification for simply-typed $\lambda$-terms, Cerna and Kutsia [11] proposed a generic framework of algorithms producing top-maximal (i.e., retaining maximal common top part of the given terms) and shallow (i.e., forbidding nested generalization variables) generalization variants, while Cerna and Buran [9] proved nullarity of anti-unification in this calculus for the unrestricted case. Some of those equational and higher-order anti-unification algorithms have been implemented and are accessible online [1, 6]. The recent survey [12] gives more detailed information about equational and higher-order anti-unification.

As an example of one of the variants included in the framework presented in [11], namely, the *common subterms variant* or, shortly, CS-variant, consider terms $s = \lambda x.f(g(x), g(g(g(x))))$ and $t = \lambda x.f(g(x), h(h(g(x))))$. They have unique top-maximal shallow CS-lgg $r = \lambda x.f(g(x), X(g(x)))$, which retains not only the topmost maximal common structure of $s$ and $t$, but also keeps the common subterm $g(x)$ that appears under distinct symbols in them. This common subterm appears in $r$ under the generalization variable $X$. An example of a non-shallow top-maximal lgg of $s$ and $t$ is $\lambda x.f(g(x), X(X(g(x))))$, where generalization variables appear nested.

Both mentioned papers [9, 11], as well as some other ones (e.g., [7, 13, 16]) assume that in generalization problems, the input terms have the same type. Relaxing this restriction would widen the practical application area of anti-unification techniques. This abstract presents work in progress towards lifting this restriction in the context of intersection and union types. We employ these types to capture the semantics of generalizing terms of different types and illustrate our approach through examples.

## 2 Preliminaries

*Types* are constructed from a set of *base types* $\pi$ using the grammar $\tau ::= \pi \mid \tau \to \tau \mid \tau \wedge \tau \mid \tau \vee \tau$, where $\wedge$ stands for type *intersection* and $\vee$ for type *union*. We use Greek letters $\tau$, $\sigma$ and $\rho$ to denote types.

$\lambda$-*terms* (typically $s, t, r$) are built using the grammar $t ::= x \mid c \mid \lambda x.t \mid t\ t$, where $x$ is a variable and $c$ is a constant. Notions as $\alpha$-conversion, $\beta$-reduction, $\eta$-long, and $\beta$-normal forms are defined as usual (e.g., [14]). Unless otherwise stated, we only consider $\lambda$-terms in $\beta$-normal $\eta$-long form and use *term* and $\lambda$-term synonymously. A complete system for typing terms is presented in [5] (see also [17]).

The *subtype relation* is formalized as the set of valid consequences derived using the inference rules presented below (S-ref, S-tran, and S-arrow); these rules derive statements of the form $\tau_1 \leq \tau_2$, read as "$\tau_1$ is a subtype of $\tau_2$" or "$\tau_2$ is a supertype of $\tau_1$".

$$\text{(S-ref)}\ \ \sigma \leq \sigma \qquad \text{(S-tran)}\ \frac{\sigma_1 \leq \sigma_2 \qquad \sigma_2 \leq \sigma_3}{\sigma_1 \leq \sigma_3} \qquad \text{(S-arrow)}\ \frac{\sigma \leq \sigma' \qquad \tau' \leq \tau}{\sigma' \to \tau' \leq \sigma \to \tau}$$

The subtyping relation with respect to a given type system $\mathcal{T}$ has the following property: If the judgment $\Gamma \vdash_{\mathcal{T}} t : \sigma$ holds and $\sigma \leq \tau$, then $\Gamma \vdash_{\mathcal{T}} t : \tau$ holds:

$$\text{(T-Sub)}\ \frac{\Gamma \vdash t : \sigma \qquad \sigma \leq \tau}{\Gamma \vdash t : \tau}$$

Properties of subtyping over the intersection and union types relevant to our work are presented below. Note, $\sigma \sim \tau$ denotes that both $\sigma \leq \tau$ and $\tau \leq \sigma$ hold.

$$1.\ \sigma \wedge \sigma \sim \sigma \sim \sigma \vee \sigma, \qquad 2.\ \sigma_i \leq \sigma_1 \vee \sigma_2,\ i = 1, 2, \qquad 3.\ \sigma_1 \wedge \sigma_2 \leq \sigma_i,\ i = 1, 2.$$

This abstract assumes the following:

**A1.** $\sigma \vee \tau$ is the least upper bound of $\sigma$ and $\tau$ w.r.t. the subtyping relation,

**A2.** $\sigma \wedge \tau$ is the greatest lower bound of $\sigma$ and $\tau$ w.r.t. the subtyping relation.

A *substitution* (typically $\theta$) is a finite set of pairs $\{X_1 \mapsto t_1, \cdots, X_n \mapsto t_n\}$, where $X_i \neq X_j$ if $i \neq j$. Postfix notation, e.g., $t\theta$, denotes substitution application.

Let the relation $\preceq$ ($\prec$ is the strict part) be a preorder over terms defined as follows: $r \preceq t$ if a substitution $\theta$ exists such that $r\theta = t$. A term $r$ is a *generalizer* of two terms $s$ and $t$ if $r \preceq s$ and $r \preceq t$. A term $r$ is a *least general generalizer* (lgg) of $s$ and $t$ if there is no term $r'$ such that $r'$ is a generalizer of $s$ and $t$ and $r \prec r'$. The *anti-unification problem* for $s$ and $t$, denoted as $s \triangleq t$, is defined as

**Given:** terms $t : \tau_1$ and $s : \tau_2$ in $\eta$-long $\beta$-normal form.

**Find:** an lgg $r : \tau$ of $s$ and $t$ such that $\tau_1 \vee \tau_2 = \tau$.

The intuition behind the lgg is that it should express the common structure of typed input expressions as much as possible while "minimizing the subtyping distance" between its type and the input types. Generalizers represent the divergences in the input term structures by variables. Thus, generalizers can be instantiated into the input terms of the problem. As for the generalizer type, it should be the least possible common supertype of the original ones. From the generalization definition, it follows that there should exist substitutions $\theta_1$ and $\theta_2$ such that $r\theta_1 = s : \tau$ and $r\theta_2 = t : \tau$, where $\tau$ is a supertype of both $\tau_1$ and $\tau_2$. There can be many supertypes of $\tau_1$ and $\tau_2$. However, $\tau$ is selected as the most specific supertype of $\tau_1$ and $\tau_2$, i.e., $\tau$ must be the least upper bound of the types of $s$ and $t$, which, by assumption 2, is exactly $\tau_1 \vee \tau_2$.

# 3 Extending Generalization to Types

**Generalizing applications.** Consider an anti-unification problem $f(a) \triangleq g(b)$ with $f : \sigma \to \tau$, $g : \sigma' \to \tau'$, $a : \rho_a \leq \sigma$ and $b : \rho_b \leq \sigma'$. It is straightforward that the term $XY$ with an adequate type is its solution. The main problem is how to systematically build the adequate generalizer and its type. Such a mechanism is still a work in progress, not addressed in this abstract. Instead, the focus is on the desired properties of such generalizations regarding works on HO-generalization as [10].

Looking at $XY$ above, we see that $X$ must be of function type, i.e., $X : \gamma_1 \to \gamma_2$ because it applies to $Y$. Furthermore, $X$ is a generalizer of $f$ and $g$. Thus, its type must be a supertype of both types of $f$ and $g$: $\sigma \to \tau \leq \gamma_1 \to \gamma_2$ and $\sigma' \to \tau' \leq \gamma_1 \to \gamma_2$. To satisfy both relations, it is necessary that $\gamma_1 \leq \sigma$ and $\gamma_1 \leq \sigma'$, and that $\tau \leq \gamma_2$ and $\tau' \leq \gamma_2$. Choose $\gamma_1 = \sigma \wedge \sigma'$ and $\gamma_2 = \tau \vee \tau'$. Then, the subtyping statements $\sigma \to \tau \leq \gamma_1 \to \gamma_2$ and $\sigma' \to \tau' \leq \gamma_1 \to \gamma_2$ follow from a derivation by the (S-arrow) rule:

$$\frac{\sigma \wedge \sigma' \leq \sigma \qquad \tau \leq \tau \vee \tau'}{\sigma \to \tau \leq (\sigma \wedge \sigma') \to (\tau \vee \tau')} \qquad \frac{\sigma \wedge \sigma' \leq \sigma' \qquad \tau' \leq \tau \vee \tau'}{\sigma' \to \tau' \leq (\sigma \wedge \sigma') \to (\tau \vee \tau')}$$

Next, since $Y$ generalizes $a$ and $b$, the type of $Y$, say $\rho_y$, must be a supertype of both $\rho_a$ and $\rho_b$. Also, $\rho_y \leq \gamma_1$ since $XY$ should be well-typed. It implies that

$$\rho_a \vee \rho_b \leq \rho_y \leq \sigma \wedge \sigma'. \tag{1}$$

Suppose that condition (1) holds, and select the lower bounds of the supertypes obtained above, i.e.,

$$X : \gamma_1 \to \gamma_2 = (\sigma \wedge \sigma') \to (\tau \vee \tau')$$
$$Y : \rho_y = \rho_a \vee \rho_b.$$

Then, it follows that we have $f(a) : \tau$, $g(b) : \tau'$ and their generalizer $XY : \tau \vee \tau'$.

The generalizer $XY$ of $f(a)$ and $g(b)$ is not a shallow term. It was chosen to facilitate readers's comprehension since it is an application (as the input terms). If $a \neq b$, then the unique top-maximal shallow generalizer of $f(a)$ and $g(b)$ is just $X$.

Cerna and Kutsia introduced a so-called common-subterm variant for HO-generalization (CS-variant, see [11]). It is one of the special cases of top-maximal shallow generalization. In this variant, every free generalization variable occurring in an lgg of two terms $s$ and $t$ and generalizing their subterms $s'$ and $t'$, should apply to maximal common subterms that appear in $s'$ and $t'$. For example, $X(h(a))$ is a CS-generalizer of $f(h(a), a) \triangleq g(b, h(a))$ while $X(Y, Z)$ is not (although it is their generalizer).

Now consider $f(a) \triangleq g(a)$ with $f : \sigma \to \tau$, $g : \sigma' \to \tau'$, $a : \rho_1 \leq \sigma$ and $a : \rho_2 \leq \sigma'$. The CS-generalizer of this problem is $X(a)$ with the types $X : \gamma_1 \to \gamma_2 = (\sigma \wedge \sigma') \to (\tau \vee \tau)'$ and $a : \rho_1 \vee \rho_2$, subject of an additional constraint $\rho_1 \vee \rho_2 \leq \sigma \wedge \sigma'$.

**Generalizing abstractions.** Consider an anti-unification problem where both input terms are different identity functions: $\lambda x.x \triangleq \lambda y.y$ where $x : \sigma$ and $y : \tau$. It is straightforward that $\lambda z.z$ with the appropriated type will be the generalizer of the input terms.

To see what should be the generalization type, first notice that $r = \lambda z.z$ must have a function type, i.e., $r : \gamma_1 \to \gamma_2$. Also, this type must be a supertype of both input types: $\sigma \to \sigma \leq \gamma_1 \to \gamma_2$ and $\tau \to \tau \leq \gamma_1 \to \gamma_2$. To satisfy both relations, it is necessary that $\gamma_1 \leq \sigma$, $\gamma_1 \leq \tau$, $\sigma \leq \gamma_2$ and $\sigma \leq \gamma_2$. By a reasoning analogous to the application case above, taking into account that $z : \sigma \wedge \tau$ implies $z : \sigma \vee \tau$, we get that

$$r = \lambda z.z : (\sigma \wedge \tau) \to (\sigma \vee \tau). \tag{2}$$

Observe in (2) that the typing condition for the generalizer decreased the domain and increased the range of those input terms. Otherwise, it would not satisfy the requirement that the type of the generalizer must be a supertype of the types of the input terms. Obviously, $(\sigma \wedge \tau) \to (\sigma \wedge \tau)$ is a supertype neither of $\sigma \to \sigma$ nor of $\tau \to \tau$. The same is true for $(\sigma \vee \tau) \to (\sigma \vee \tau)$. This is a consequence of the contravariance in the rule (S-arrow).

**Example 1.** Now, consider the identity function defined in different sets: $\mathtt{ID}_{\mathbb{N}}(n) = n$ and $\mathtt{ID}_{\mathbb{Z}}(z) = z$. They are expressed in $\lambda$-calculus by $\lambda(x : \mathbb{N}).(x : \mathbb{N})$ and $\lambda(y : \mathbb{Z}).(y : \mathbb{Z})$, then the generalizer of those functions will be $\lambda(k : \mathbb{N} \wedge \mathbb{Z}).(k : \mathbb{N} \vee \mathbb{N})$. Therefore, the generalizer is $\mathtt{ID}_{\mathtt{gen}}(k) = k$ with domain $\mathbb{N} \cap \mathbb{Z}$ and range $\mathbb{N} \cup \mathbb{Z}$ which means that the computed generalizer is a non-surjective identity function from $\mathbb{N}$ to $\mathbb{Z}$.

This abstract does not discuss the inhabitation of intersection types. It is clear that by interpreting types as sets, some intersection types may get uninhabited. In these special cases, some generalizations will not have semantic meaning, suggesting that generalizations of the input terms do not exist. (A similar phenomenon was observed in the calculus of constructions [15] where there is no semantic interpretation of the generalization of abstract kinds, Set, Prop, Type, etc.) For instance, consider two identity functions: one defined on the set of rational numbers and the other one on the set of irrational numbers, respectively: $\mathtt{ID}_{\mathbb{Q}}(q) = q$ and $\mathtt{ID}_{\mathbb{I}}(i) = i$. The generalizer should be $\mathtt{ID}_{\mathtt{gen}}(g) = g$ with domain $\mathbb{Q} \cap \mathbb{I}$ and range $\mathbb{Q} \cup \mathbb{I}$; however, this domain set is empty. Consequently, the generalizer does not exist.

Now, consider a generalization problem $\lambda x.f(x, a) \triangleq \lambda y.g(b, y)$ with $x : \sigma_x \leq \sigma$, $f : \sigma \to \tau$, $y : \sigma_y \leq \sigma'$ and $g : \sigma' \to \tau'$, and the requirement that a solution of this problem should retain the common top-maximal structure of the given terms. Since both terms are abstractions, the desired generalizer should be an abstraction, too. Also, since the scopes of the input terms

have different heads $f$ and $g$, the scope of the generalizer must be a free variable. Therefore, $\lambda z.X(z)$ is the lgg that can be transformed to the original expressions via the substitutions $\theta_1 = \{X \mapsto \lambda u.f(u,a)\}$ and $\theta_2 = \{X \mapsto \lambda u.g(b,u)\}$:

$$\lambda z.X(z)\theta_1 = \lambda z.(\lambda u.f(u,a))(z) =_\beta \lambda z.f(z,a) =_\alpha \lambda x.f(x,a)$$
$$\lambda z.X(z)\theta_2 = \lambda z.(\lambda u.f(b,u))(z) =_\beta \lambda z.g(b,z) =_\alpha \lambda y.g(b,y).$$

What about the type of this generalizer? Let $\lambda z.X(z) : \gamma_1 \to \gamma_2$. Again, to infer the conditions for this to be an adequate type, we have $\sigma_x \to \tau \leq \gamma_1 \to \gamma_2$ and $\sigma_y \to \tau' \leq \gamma_1 \to \gamma_2$, which imply

$$\lambda z.X(z) : (\sigma_x \wedge \sigma_y) \to (\tau \vee \tau').$$

Then $z : \sigma_x \wedge \sigma_y$ and $X(z) : \tau \vee \tau'$. Since $X$ applies $z$ and has range $\tau \vee \tau'$, it follows that $X : (\sigma_x \wedge \sigma_y) \to (\tau \vee \tau')$.

**Example 2.** With this example, we illustrate how generalization with intersection and union types can be used to synthesize a generic function from two concrete instances. As the given concrete ones, consider two functions, congruences modulo 3 and 5, defined respectively as

$$\texttt{MOD}_3(n : \mathbb{N}) : \{0,\ldots,2\} = \text{if } n < 3 \text{ then } n \text{ else } \texttt{MOD}_3(n-3) \tag{3}$$
$$\texttt{MOD}_5(n : \mathbb{N}) : \{0,\ldots,4\} = \text{if } n < 5 \text{ then } n \text{ else } \texttt{MOD}_5(n-5), \tag{4}$$

where the explicit types indicate that $\texttt{MOD}_3 : \mathbb{N} \to \{0,\ldots,2\}$ and $\texttt{MOD}_5 : \mathbb{N} \to \{0,\ldots,4\}$.

We aim to synthesize a generic function for congruence modulo, from which proper instantiations are used to obtain these concrete ones. This we do in two steps, where only the first one concerns generalizer computation:

**Step 1.** Anti-unify (3) and (4). It will give

$$X(n) = \text{if } n < k \text{ then } n \text{ else } X(n-k) \tag{5}$$

where $X$ and $k$ are generalization variables of types respectively $X : (\mathbb{N} \wedge \mathbb{N}) \to (\{0,\ldots,2\} \vee \{0,\ldots,4\}) = \mathbb{N} \to \{0,\ldots,4\}$ and $k : \mathbb{N}$. The original expressions (3) and (4) can be obtained by the substitutions, respectively:

$$\{X \mapsto \lambda x.\texttt{MOD}_3(x),\ k \mapsto 3\}$$
$$\{X \mapsto \lambda x.\texttt{MOD}_5(x),\ k \mapsto 5\}.$$

**Step 2.** Notice that although (5) is a (least general) generalizer of two function definitions (3) and (4), it is not a function definition itself because of those free generalization variables. Now, we turn it into such a definition by introducing a new function name `GENMOD` (meaning generic `MOD`) instead of $X$. It has both $n$ and $k$ as its arguments:

$$\texttt{GENMOD}(n,k) = \text{if } n < k \text{ then } n \text{ else } \texttt{GENMOD}(n-k,k) \tag{6}$$

With some further processing that is beyond the scope of this abstract, one can connect the type to `GENMOD` to $k$ as, e.g., $\mathbb{N} \to \mathbb{N} \to \{0,\ldots,k-1\}$.

# 4 Final Remarks

This abstract presented preliminary investigations about higher-order generalization with intersection and union types, aiming at extending existing HO-generalization problems (e.g., [7, 10, 11]) to this setting. We illustrated some desirable properties of such generalizers. An anti-unification algorithm to construct generalizers and an algorithm to compute their (minimal) types are subjects of ongoing work. Future research should cover anti-unification with abstractions and terms of different structures.

**Acknowledgments.** The Brazilian Higher Education Council (CAPES) supported the Brazilian-Austrian cooperation through the PrInt program finance code 001. Funded by the Austrian Science Fund (FWF), under project P 35530; Czech Science Foundation, grant No. 22-06414L; Cost Action CA20111 EuroProofNet; Brazilian Research Council (CNPq), grants Universal 409003/21-2, RG 313290/21-0; and the Brazilian Federal District Research Foundation FAPDF, grant DE 00193-00001175/2021-11.

# References

[1] María Alpuente, Demis Ballis, Angel Cuenca-Ortega, Santiago Escobar, and José Meseguer. Acuos$^2$: A high-performance system for modular ACU generalization with subtyping and inheritance. In Francesco Calimeri, Nicola Leone, and Marco Manna, editors, *Logics in Artificial Intelligence - 16th European Conference, JELIA 2019, Rende, Italy, May 7-11, 2019, Proceedings*, volume 11468 of *Lecture Notes in Computer Science*, pages 171–181. Springer, 2019.

[2] María Alpuente, Santiago Escobar, Javier Espert, and José Meseguer. A modular order-sorted equational generalization algorithm. *Inf. Comput.*, 235:98–136, 2014.

[3] Mauricio Ayala-Rincón, David M. Cerna, Andrés Felipe Gonzáez Barragán, and Temur Kutsia. Equational anti-unification over absorption theories. In *Automated Reasoning - 12th International Joint Conference, IJCAR 2024, Nancy, France, Proceedings*, volume 14740 of *Lecture Notes in Computer Science*. Springer, 2024.

[4] Franz Baader. Unification, weak unification, upper bound, lower bound, and generalization problems. In *Rewriting Techniques and Applications, 4th International Conference, RTA-91, Como, Italy, April 10-12, 1991, Proceedings*, volume 488 of *Lecture Notes in Computer Science*, pages 86–97. Springer, 1991.

[5] Franco Barbanera and Mariangiola Dezani-Ciancaglini. Intersection and union types. In Takayasu Ito and Albert R. Meyer, editors, *Theoretical Aspects of Computer Software, International Conference TACS '91, Sendai, Japan, September 24-27, 1991, Proceedings*, volume 526 of *Lecture Notes in Computer Science*, pages 651–674. Springer, 1991.

[6] Alexander Baumgartner and Temur Kutsia. A library of anti-unification algorithms. In Eduardo Fermé and João Leite, editors, *Logics in Artificial Intelligence - 14th European Conference, JELIA 2014, Funchal, Madeira, Portugal, September 24-26, 2014. Proceedings*, volume 8761 of *Lecture Notes in Computer Science*, pages 543–557. Springer, 2014.

[7] Alexander Baumgartner, Temur Kutsia, Jordi Levy, and Mateu Villaret. Higher-order pattern anti-unification in linear time. *J. Autom. Reason.*, 58(2):293–310, 2017.

[8] David Cerna and Temur Kutsia. Idempotent anti-unification. *ACM Trans. Comput. Log.*, 21(2):10:1–10:32, 2020.

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

[10] David M. Cerna and Temur Kutsia. Higher-order equational pattern anti-unification. In Hélène Kirchner, editor, *3rd International Conference on Formal Structures for Computation and Deduc-*

*tion, FSCD 2018, July 9-12, 2018, Oxford, UK*, volume 108 of *LIPIcs*, pages 12:1–12:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[11] 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.

[12] 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.

[13] Cao Feng and Stephen H. Muggleton. Towards inductive generalization in higher order logic. In Derek H. Sleeman and Peter Edwards, editors, *Proceedings of the Ninth International Workshop on Machine Learning (ML 1992), Aberdeen, Scotland, UK, July 1-3, 1992*, pages 154–162. Morgan Kaufmann, 1992.

[14] J Roger Hindley. *Basic simple type theory*. Number 42 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1997.

[15] Frank Pfenning. Unification and anti-unification in the calculus of constructions. In *Proceedings of the Sixth Annual Symposium on Logic in Computer Science (LICS '91), Amsterdam, The Netherlands, July 15-18, 1991*, pages 74–85. IEEE Computer Society, 1991.

[16] Brigitte Pientka. Higher-order term indexing using substitution trees. *ACM Trans. Comput. Log.*, 11(1):6:1–6:40, 2009.

[17] Benjamin C. Pierce. *Types and programming languages*. MIT Press, 2002.

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

[19] John C. Reynolds. Transformational system and the algebraic structure of atomic formulas. *Machine Intelligence 5*, 5:135–151, 1970.