Higher-Order Unification with Definition by Cases

Chad E. Brown[1] and David Cerna[2,3]

[1] Czech Technical University in Prague
Czech Institute of Informatics, Robotics and Cybernetics
Prague, Czech Republic
[2] Czech Academy of Sciences Institute of Computer Science (CAS ICS), Prague, Czechia
[3] Research Institute for Symbolic Computation (RISC), Johannes Kepler University, Linz, Austria

Abstract

We discuss unification within the simply-typed λ-calculus extended by a definition by cases operator (denoted by d) slightly differing from similar operators introduced by earlier investigations. Such operators may be thought of as restrictions of *Hilbert's choice operator*. We provide several non-trivial examples which illustrate the benefits of introducing such an operator.

1 Introduction

Higher-order automated theorem provers often need to instantiate higher-order variables, i.e., variables of function type. It is common for provers to obtain such instantiations using Huet's preunification procedure for simply typed λ-calculus [5]. A recent procedure for unification (not just preunification) was described by Vukmirovic, et. al., [9] is implemented in Zipperposition [8], the winner of the higher-order division of the CASC competition for 2020 and 2021. There are differences between the problem of unifying terms in a simply-typed λ-calculus and unifying terms in a simply-typed higher order logic [4]. Typically in simply-typed λ-calculus one has (at least) a base type ι about which one makes no special assumptions. In simply-typed higher order logic there is a base type o of propositions and typically it is assumed o is a two element type with two distinct elements we call $\bot$ and $\top$. Clearly unifying two simply-typed λ-terms will ensure they are equal in the simply-typed higher order logic, however, there may be many ways to make two terms of higher order logic semantically (and provably) equal without making the terms $\beta\eta$-convertible. As a simple example, the two propositions s and $\neg\neg s$ are semantically equal, though they are not $\beta\eta$-convertible.

It is common for higher-order logics to include a Hilbert choice operator at every type α, i.e., $\varepsilon^\alpha : (\alpha \to o) \to \alpha$. For example, the language used by the TH0 problem suite of TPTP [7] includes a choice operator. For each type α this satisfies the axiom $\forall p : \alpha \to o.\forall x : \alpha.(p\ x \Rightarrow p\ (\varepsilon^\alpha p))$, where $\Rightarrow$ is logical implication. Essentially, for every type and every predicate which takes an argument of that type there is an operator which when given the predicate returns an element for which the predicate holds.

Having such an operator makes many more solutions possible. For example, suppose α is a type and a and b are distinct constants of type α. In simply typed λ-calculus, there is no solution for X satisfying $X\ a = b$ and $X\ b = a$. However, with ε^α there are such solutions as one can define *if-then-else*$^\alpha$ using ε^α. It is unclear how to obtain a unification procedure for simply typed higher order logic with a choice operator. However, we could consider a simpler problem. From a choice operator at α, an if-then-else operator is constructed as $\mathsf{if}^\alpha : o \to \alpha \to \alpha \to \alpha$ satisfying $\forall p : o.p \to \mathsf{if}\ p\ x\ y = x$ and $\forall p : o.\neg p \to \mathsf{if}\ p\ x\ y = y$. We could consider unification in a weakened logic without the choice operator but still with such an if-then-else operator. This would still be a difficult problem because a partial instantiation of the form $\mathsf{if}^\alpha PXY$ would require us to synthesize a proposition P. The problem of needing to synthesize propositions can

be avoided if we further weaken the logic to include the special if-then-else operator considered by Beeson [1], a definition-by-cases operator. Beeson introduced his d operator with reduction rules, e.g., $\mathsf{d}\ s\ s\ u\ v \Rightarrow u$, but the resulting system has undesirable properties (Church-Rosser theorem does not hold). We will instead use a semantic characterization of the d operator. Let us assume we have an operator $\mathsf{d}^\alpha : \iota \to \iota \to \alpha \to \alpha \to \alpha$ satisfying

$$\forall x : \iota.\forall uv : \alpha.\mathsf{d}^\alpha\ x\ x\ u\ v = u$$

and

$$\forall xy : \iota.\forall uv : \alpha.x \neq y \to \mathsf{d}^\alpha\ x\ y\ u\ v = v$$

for each type α. By equality, we are referring to equality within the the model (see below). Clearly such a d^α could be defined from if^α by using an equation for the proposition giving the condition. Such a d^α is also sufficient to solve a unification problem of the form $X\ a = b \wedge X\ b = a$. Note that using the d operator will never require synthesizing a proposition, unless perhaps if the type α makes use of the type o. We will not attempt to fully solve the problem of unification in simply typed λ-calculus with a d^α here, but will give a semantic description of the problem and consider a few examples.

2 First Order Unification with Definition by Cases

Before considering the higher order case we briefly consider the first order case. Let us assume a first order signature consisting of two constants a and b and two unary functions f and g. The set of first-order terms (denoted by s, t,$s_1, t_1, \cdots$) are defined inductively as usual. a unification problem is of the form $s_1 = t_1 \wedge \cdots \wedge s_n = t_n$ and a solution is a substitution θ with domain V such that $\theta(s_i)$ is the same as $\theta(t_i)$ for $i \in \{1, \ldots, n\}$. Once we add a d operator, we want to consider terms the same up to the semantics of d. To prepare for this, we say θ is a solution of the problem $(V, (s_1 = t_1 \wedge \cdots \wedge s_n = t_n))$ if θ has domain V and $\theta(s_1 = t_1 \wedge \cdots \wedge s_n = t_n)$ is valid in every interpretation.

We now extend the first-order terms to include the 4-ary operator d. That is, if s, s', t, t' are terms, then $\mathsf{d}\ s\ s'\ t\ t'$ is a term. We say an interpretation is a d-interpretation if it satisfies the sentences

$$\forall xuv.\mathsf{d}\ x\ x\ u\ v = u \quad \text{and} \quad \forall xyuv.x \neq y \to \mathsf{d}\ x\ y\ u\ v = v.$$

In other words, d-interpretations are those that give the intended semantics to d. We now say θ is a solution to a unification problem $(V, (s_1 = t_1 \wedge \cdots \wedge s_n = t_n))$ if θ has domain V and $\theta(s_1 = t_1 \wedge \cdots \wedge s_n = t_n)$ is valid in every d-interpretation.

Example 2.1. *Let $V = \{x, y\}$. Even though an occurs check would would lead to failure, there is a solution to the unification problem*

$$(V, (x = \mathsf{d}\ y\ a\ b\ (f x))).$$

In particular, θ with $\theta x = b$ and $\theta y = a$ is a solution.

There is a relatively straightforward algorithm to solve such unification problems. Let Φ be a quantifier-free formula with at least one occurrence of $\mathsf{d}\ s\ s'\ t\ t'$. Let us write $\Phi[\mathsf{d}\ s\ s'\ t\ t']$ to denote the existence of an occurrence of $\mathsf{d}\ s\ s'\ t\ t'$ in Φ. Note that $\theta\Phi$ is valid in all d-interpretations if and only if

$$\theta(((s = s') \to \Phi[t]) \wedge ((s \neq s') \to \Phi[t']))$$

is valid in all d-interpretations. Continuing in this manner, all references to d can be eliminated, yielding a quantifier-free formula Φ' with no references to d. However, this formula Φ' is no longer of the form $s_1 = t_1 \wedge \cdots \wedge s_n = t_n$, i.e. this is no longer a unification problem. If Φ' is transformed into conjunctive normal form, then it will be a disjunction of formulas of the form

$$s_1^1 \neq t_1^1 \wedge \cdots \wedge s_m^1 \neq t_m^1 \wedge s_1^2 = t_1^2 \wedge \cdots \wedge s_n^2 = t_n^2.$$

Such problems fall into the class of disunification problems [6, 3]. While,at the current state of the investigation, it is clear that such methods are needed, further work is needed to provide precises uses and . For our purposes it is sufficient to note that this motivates considering the more general problem of finding solutions for problems of the form (V, Φ) where Φ is a quantifier-free formula with $=$ as the only predicate symbol. In fact, our definitions will make sense for arbitrary predicates Φ, though once quantifiers are permitted even checking if a candidate θ is a solution is a general theorem proving problem.

3 Higher Order Unification with Definition by Cases

We now turn to the main topic: higher-order unification with Beeson's definition-by-cases operator. We start with a version of Church's simple type theory as presented in [2]. For types we consider those generated by the grammar $\tau ::= o|\iota|\tau \to \tau$. For terms we take simply typed λ-terms with at least the logical constants $\bot : o$, $\neg : o \to o$, $\wedge : o \to o \to o$, $=_\sigma$: $\sigma \to \sigma \to o$ and $\forall_\sigma : (\sigma \to o) \to o$. The notion of a Henkin interpretation (or simply *interpretation*) is given in [2].

We say a type is *pure* if it has no occurrence of o. A term is *pure* if every constant and variable in the term is of a pure type. We next additionally consider constants $\mathsf{d}^\sigma : \iota \to \iota \to \sigma \to \sigma \to \sigma$ for each pure type σ. Note that each d^σ is a pure term. (This would not be true with a more general if-then-else operator of type $o \to \sigma \to \sigma \to \sigma$.) Let D be the set

$$\{\forall x : \iota.\forall uv : \sigma.\mathsf{d}^\sigma\ x\ x\ u\ v = u | \sigma \text{ is a pure type}\}$$
$$\cup\{\forall xy : \iota.\forall uv : \sigma.x \neq y \to \mathsf{d}^\sigma\ x\ y\ u\ v = v | \sigma \text{ is a pure type}\}$$

of sentences. A Henkin model of D is a Henkin interpretation in which every sentence in D evaluates to 1, i.e. we are restricting the Henkin interpretations of simple type theory to those that also statisfy the above statements concerning d.

The problems we consider will be given as (V, Φ) where V is a finite set of variables of pure type and Φ in which the only constants and constants not of pure type allowed to occur in Φ are $\neg$, $\wedge$ and $=_\sigma$ where σ is a pure type. That is, Φ is quantifier-free and every atom in Φ is of the form $s =_\sigma t$ where σ, s and t are pure. If Φ is converted to conjunctive normal form, each conjunct can be assumed to have the form

$$s_1^1 \neq_{\sigma_1^1} t_1^1 \wedge \cdots \wedge s_m^1 \neq_{\sigma_m^1} t_m^1 \wedge s_1^2 =_{\sigma_1^2} t_1^2 \wedge \cdots \wedge s_n^2 =_{\sigma_n^2} t_n^2$$

where each σ_j^i, s_j^i and t_j^i is pure. Such problems (V, Φ) generalize Huet's unification problem while still avoiding the complications arising from occurrences of the type o.

Let V be a finite set of variables. For a proposition Φ and substitution θ we say θ is a solution for (V, Φ) if θ has domain V and $\theta\Phi$ is valid in all Henkin models of D. We say θ is more general than θ' (written $\theta \preceq_V \theta'$) if there is a substitution τ such that $\tau(\theta X) = \theta' X$ is valid in all Henkin models of D for all $X \in V$.

Consider the following example one can find in Beeson [1].

Example 3.1. *Let $V = \{X\}$ and Φ be $X\ a = a$ where X is a variable of type $\iota\iota$ and a is a constant of type ι. Huet's rules yield two incomparable solutions: θ_I and θ_K with $\theta_I X = \lambda z.z$ and $\theta_K X = \lambda z.a$. Beeson's use of d yields the less committed solution θ_B with $\theta_B X = \lambda z.\mathsf{d}^\iota\ z\ a\ a\ (Y\ z)$ where Y is a variable of type $\iota\iota$. It is obvious all three are solutions. The fact that θ_B is more general than θ_I follows by taking τY to be $\lambda z.z$. The fact that θ_B is more general than θ_K follows by taking τY to be $\lambda z.a$.*

We also consider our original motivating example.

Example 3.2. *Let $V = \{X\}$ and Φ be $X\ a = b \wedge X\ b = a$ where X is a variable of type $\iota\iota$ and a and b are constants of type ι. Consider the following solutions:*

- *θ_1 with $\theta_1 X = \lambda z.\mathsf{d}^\iota\ z\ a\ b\ a$.*
- *θ_2 with $\theta_2 X = \lambda z.\mathsf{d}^\iota\ z\ b\ a\ b$.*
- *θ_3 with $\theta_3 X = \lambda z.\mathsf{d}^\iota\ z\ a\ b\ (\mathsf{d}^\iota\ z\ b\ a\ (Y\ z))$.*
- *θ_4 with $\theta_4 X = \lambda z.\mathsf{d}^\iota\ z\ b\ a\ (\mathsf{d}^\iota\ z\ a\ b\ (Y\ z))$.*

It is easy to see these are all solutions, with θ_3 and θ_4 being most general solutions. To see that, for example, $\theta_3 \preceq_X \theta_4$, take τ to be the identity substitution and note that in every Henkin model of D the following equation is satisfied

$$(\lambda z.\mathsf{d}^\iota\ z\ a\ b\ (\mathsf{d}^\iota\ z\ b\ a\ (Y\ z))) =_{\iota\iota} (\lambda z.\mathsf{d}^\iota\ z\ b\ a\ (\mathsf{d}^\iota\ z\ a\ b\ (Y\ z))).$$

While the use of d can make some unification problems solvable that were not solvable without d, there are, of course, many examples of unsolvable problems. The next example demonstrates this.

Example 3.3. *Let $V = \{X\}$ and Φ be $f\ (X\ a)\ (X\ a) = f\ b\ c$ where X has type $\iota\iota$, f has type $\iota\iota\iota$ and a, b, c have type ι. The problem (V, Φ) has no solution. Consider a Henkin model of D in which b and c are interpreted differently. This model could be used to prove any candidate θ is not a solution.*

The following example demonstrates that our unification problem differs from Beeson's as it is not unitary.

Example 3.4. *Let $V = \{X\}$ and Φ be $(\lambda z.Xzz) = (\lambda z.z)$ where X has type $\iota\iota\iota$. Huet's procedure gives the following two solutions (via projection):*

$$\theta_1 \text{ with } \theta_1 X = \lambda uv.u \qquad\qquad \theta_2 \text{ with } \theta_2 X = \lambda uv.v.$$

We claim conjecture that there is no solution more general than the two solutions above, furthermore that this In particular, a naive attempt to use the d operator to write a term like $\lambda uv.\mathsf{d}^\iota uzz(Yz)$ fails to provide a solution, since the z is unbound here.

The example solutions above using d all have the form $\cdots \mathsf{d}\ x\ t \cdots$ where x is a local bound variable and t has no reference to bound variables. We next consider examples where d is used differently.

Example 3.5. *Let $V = \{X\}$ and Φ be the conjunction*

$$(\lambda u.X\ u\ u) = (\lambda u.f\ (g\ u)\ a) \wedge (\lambda u.X\ u\ a) = (\lambda u.f\ (g\ u)\ u)$$

where X and f have type $\iota\iota$, g has type $\iota\iota$ and u and a have type ι. Without using d *the only solution for the second conjunct would be θ with $\theta\ X = \lambda z_1 z_2.f\ (g\ z_1)\ z_1$, which does not solve the first conjunct. The following solution makes use of* d *on the diagonal:*

$$\theta(X) = \lambda z_1 z_2.\mathsf{d}^{\iota}\ z_1\ z_2\ (f\ (g\ z_1)\ a)\ (f\ (g\ z_1)\ z_1).$$

To verify this is a solution one must note that the following proposition is valid in all Henkin interpretations: $\forall u.u = a \to f\ (g\ u)\ a = f\ (g\ u)\ u$. Consequently

$$(\lambda u.\mathsf{d}^{\iota}\ u\ a\ (f\ (g\ u)\ a)\ (f\ (g\ u)\ u)) = (\lambda u.f\ (g\ u)\ u)$$

is valid in all Henkin models of D. Another solution makes use of d *to check if the second argument is a:*

$$\theta(X) = \lambda z_1 z_2.\mathsf{d}^{\iota}\ z_2\ a\ (f\ (g\ z_1)\ z_1)\ (f\ (g\ z_1)\ a).$$

This uses d *as before, to check if an input is equal to a specific term.*

Example 3.6. *Let $V = \{X\}$ and Φ be the conjunction*

$$(\lambda u.X\ u\ u) = (\lambda u.f\ (g\ u)\ (h\ u)) \wedge (\lambda u.X\ u\ (h\ u)) = (\lambda u.f\ (g\ u)\ u)$$

where X and f have type $\iota\iota$, g and h have type $\iota\iota$ and u has type ι. The following solution makes use of d *in a way where both of the first two arguments refer to the inputs:*

$$\theta(X) = \lambda z_1 z_2.\mathsf{d}^{\iota}\ z_2\ (h\ z_1)\ (f\ (g\ z_1)\ z_1)\ (f\ (g\ z_1)\ (h\ z_2)).$$

Since $\forall u.u = h\ u \to f\ (g\ u)\ u = f\ (g\ u)\ (h\ u)$ is valid in all Henkin interpretations,

$$(\lambda u.\mathsf{d}^{\iota}\ u\ (h\ u)\ (f\ (g\ u)\ u)\ (f\ (g\ u)\ (h\ u))) = (\lambda u.f\ (g\ u)\ (h\ u))$$

is valid in all Henkin models of D.

These examples illustrate that the space of solutions has some important differences from the unification problem in the simply typed λ-calculus case as well as Beeson's **d**-operator. We leave it to future work to consider algorithmic approaches to solving problems in this extended language and resolving our conjecture concerning the unification type of this theory.

Acknowledgments

The results were supported by the Ministry of Education, Youth and Sports within the dedicated program ERC CZ under the project POSTMAN no. LL1902, the Math$_{LP}$ project (LIT-2019-7-YOU-213) of the Linz Institute of Technology and the state of Upper Austria, and Cost action CA20111 EuroProofNet.

References

[1] Michael Beeson. Unification in lambda calculus with if-then-else. In Claude Kirchner and Helene Kirchner, editors, *15th CADE*, pages 96–111, 1998.

[2] Chad E. Brown and Gert Smolka. Analytic tableaux for simple type theory and its first-order fragment. *Logical Methods in Computer Science*, 6(2), Jun 2010.

[3] Wray L. Buntine and Hans-Jürgen Bürckert. On solving equations and disequations. *Journal of the ACM*, 41:591–629, 1994.

[4] Alonzo Church. A formulation of the simple theory of types. *The Journal of Symbolic Logic*, 5:56–68, 1940.

[5] Gérard P. Huet. A unification algorithm for typed lambda-calculus. *Theor. Comput. Sci.*, 1(1):27–57, 1975.

[6] Claude Kirchner and Pierre Lescanne. Solving disequations. Research Report RR-0686, INRIA, 1987.

[7] G. Sutcliffe. The TPTP Problem Library and Associated Infrastructure. From CNF to TH0, TPTP v6. 4.0. *Journal of Automated Reasoning*, 59(4):483–502, 2017.

[8] P. Vukmirović, A. Bentkamp, J. Blanchette, S. Cruanes, V. Nummelin, and S. Tourret. Making higher-order superposition work. In A. Platzer and G. Sutcliffe, editors, *CADE 2021*, pages 415–432. Springer, 2021.

[9] Petar Vukmirovic, Alexander Bentkamp, and Visa Nummelin. Efficient full higher-order unification. In Zena M. Ariola, editor, *FSCD 2020*, volume 167 of *LIPIcs*, pages 5:1–5:17, 2020.