Draft, manuscript No.
(will be inserted by the editor)

Functions-as-constructors Higher-order Unification: Extended Pattern Unification

Tomer Libal · Dale Miller

Received: date / Accepted: date

Abstract Unification is a central operation in constructing a range of computational logic systems based on first-order and higher-order logics. First-order unification has several properties that guide its incorporation in such systems. In particular, first-order unification is decidable, unary, and can be performed on untyped term structures. None of these three properties hold for full higher-order unification: unification is undecidable, unifiers can be incomparable, and term-level typing can dominate the search for unifiers. The so-called *pattern* subset of higher-order unification was designed to be a small extension to first-order unification that respects the laws governing λ-binding (i.e., the equalities for α, β, and η-conversion) but which also satisfied those three properties. While the pattern fragment of higher-order unification has been used in numerous implemented systems and in various theoretical settings, it is too weak for many applications. This paper defines an extension of pattern unification that should make it more generally applicable, especially in proof assistants that allow for higher-order functions. This extension's main idea is that the arguments to a higher-order, free variable can be more than just distinct bound variables. In particular, such arguments can be terms constructed from (sufficient numbers of) such bound variables using term constructors and where no argument is a subterm of any other argument. We show that this extension to pattern unification satisfies the three properties mentioned above.

Keywords Higher-order unification · Pattern unification · Deterministic unification algorithms · Type-free unification algorithms

T. Libal
University of Luxembourg, Belval, Luxembourg
E-mail: tomer@libal.info

D. Miller
Inria Saclay & LIX/École Polytechnique, Palaiseau, France
E-mail: dale.miller@inria.fr

1 Introduction

Unification is the process of solving equality constraints by the computation of substitutions. This process is used in computational logic systems ranging from automated theorem provers, proof assistants, type inference systems, and logic programming. First-order unification—that is, unification restricted to first-order terms—enjoys at least three important computational properties: decidability, determinacy, and type-freeness. These properties of unification shaped the way it can be used within computational logic systems. The first two of these properties ensures that unification—as a process—will either fail to find a unifier for a given set of pairs of term or will succeed and return the *most general unifier* (mgu). The notion of type-freeness means that unification can be performed independently of the possible typing discipline employed with terms. Thus, first-order unification can be performed on untyped first-order terms (as terms are usually considered in, say, Prolog). Since such unification is untyped, it is often possible to employ such unification within various typed disciplines. Since typing is usually an open-ended design issue (consider, for example, higher-order types, subtypes, dependent types, parametric types, linear types, etc.), the type-freeness of unification is one of its an important characteristics.

Of course, many syntactic objects are not represented well by purely first-order terms: this is the case when such objects contains bindings. Instead, many computational systems follow Church's approach used in his Simple Theory of Types [13]: there, terms, formulas, and equality are taken directly from the λ-calculus. All binding operations—e.g., quantification in first-order formulas, function arguments in functional programs, local variables, etc.—can be represented using the sole binder of the λ-calculus. Early papers showed that second-order pattern matching can support interesting program analysis and program transformation [24] and that a higher-order version of Prolog could be used to do more general manipulations of programs and formulas [28]. Today, there is a rich collection of computational logic systems that have moved beyond first-order term unification and rely on some form of unification of simply typed λ-terms (a.k.a. *higher-order unification*). These include the theorem provers TPS [3], Leo [9] and Satallax [12]; the proof assistants Isabelle [35], Coq [49], Matita [4], Minlog [40], Agda [11], Abella [6], and Beluga [38]; the logic programming languages λProlog [30] and Twelf [37]; and various application domains such as natural language processing [14] and proof checking [31].

The integration of full higher-order unification into computational logic systems is not as simple as it is in first-order systems since the three properties mentioned above do not hold. The unification of simply typed λ-terms is undecidable [17,22] and there can be incomparable unifiers, implying that no most general unifiers exist in the general situation. Also, types matter a great deal in determining the search space of unifiers. For example, let i and j be primitive types, let a be a constant of type i, and let F and X be variables of type $\alpha \to i$ and α, respectively, where α is a type variable. Consider the unification problem $(F\ X) = a$. If we set α to j, then there is an mgu for this problem, namely $[F \mapsto \lambda w.a]$. If we set α to i, then there are two incomparable solutions, namely $[F \mapsto \lambda w.a]$ and $[F \mapsto \lambda w.w,\ X \mapsto a]$. If we set α to $i \to i$, then there is an infinite number of incomparable solutions: $[F \mapsto \lambda f.a]$ and, for each natural number $n \geq 1$, $[F \mapsto \lambda f.f^n a,\ X \mapsto \lambda w.w]$. If we instantiate α with types of still higher order, the possibility of unifiers becomes dizzying. For these reasons, the integration of unification for simply typed λ-terms into computational logic systems is complex: most such integration efforts attempt to accommodate the (pre-)unification search procedure of Huet [23].

Instead of moving from first-order unification to full higher-order unification, it is possible to move to an intermediate space of unification problems. Given that higher-order unification is undecidable, there is an infinite number of decidable classes that one could consider. The setting of *higher-order pattern unification* (proposed in [27] and called L_λ-unification there) could be seen as the weakest extension of first-order unification in which the equations of α, β, and η conversion hold. In this fragment, a free variable cannot be applied to general terms but only to bound variables that cannot appear free in eventual instantiations for the free variable. This restriction means that all forms of β-reduction encountered during unification are actually (α-equivalent) to the rule $(\lambda x.B)x = B$ (a conversion rule called β_0 in [27]). Notice that in this setting, β-reduction reduces the size of terms: hence, unification takes on a much simpler nature. The unification problems that result retain all three properties we listed for first-order unification [27]. As a result, the integration of pattern unification into a prover is usually much simpler than incorporating all the search behavior implied by Huet's (pre-)unification procedure.

A somewhat surprising fact about pattern unification is that many computational logic systems need only this subset of higher-order unification to be "practically complete": that is, restricting unification to just this subset did not stop the bulk of specifications from being properly executed. For example, while both early implementations of λProlog and LF [32,37] implemented full higher-order unification, the most recent versions of those languages implement only pattern unification [1,39]. A design feature of both of those systems is to treat any unification problem that is not in the pattern fragment as a suspended constraint: usually, subsequent substitutions will cause such delayed problems to convert into the pattern fragment. Processing of constraints may also be possible: the application of *pruning* to flexible-flexible constraints in [29] is such an example. Also, since pattern unification does not require typing information, it has been possible to describe variants of such unification in settings where types can play a role during unification: see, for example, generalizations of pattern unification for dependent and polymorphic types [36], product types [15,16], and sum types [2].

Since pattern unification is a weak fragment of higher-order unification, it is natural to ask if it can be extended and still keep the same high-level properties. There have been a few extensions of pattern unification considered in the literature. The generalization by Fettig and Löchner [16] and Duggan [15] allows for constructors denoting projections to be admitted in the scope of free functional variables. These projections are specific unary functions that are closed under several properties, such as associativity. When attempting to encode the meta-theory of sequent calculus proofs for quantificational logic, McDowell and Miller [26] encoded the abstraction represented by eigenvariables using a single bound variable to encode a list of eigenvariables. Thus, to encode the sequent judgment $x_0, \ldots, x_n \vdash Cx_0 \ldots x_n$ (for $n \geq 0$ and all variables being of the same primitive type), they instead used the simply typed term $\lambda l.C(\mathrm{fst}\ l) \ldots (\mathrm{fst}(\mathrm{snd}^n l))$, where the environment abstraction l has type, say, evs, and fst and snd are constructors of type evs $\rightarrow i$ and evs $\rightarrow$ evs, respectively. Tiu showed how to lift pattern unification to this setting [43].

The problems of extending an algorithm from first-order to higher-orders exist not only in unification but also in matching. Yokoyama et al. have shown how an extension to patterns can be used in higher-order matching and have presented a linear higher-order matching algorithm [47].

In proof assistants, such as Coq, where functions can be defined using both lambda abstraction and iteration, the role of unification (see, for example, [5]) could be increased if it could be used with occurrences of such functions and lambda abstractions. In Coq, however, allowing unification in such more general settings (for example, with the `bigop` library of SSReflect) produces non-pattern unification problems [18]. Consider, for example, the problem of unifying $\lambda x.Y(gx)$ with $\lambda x.f(gx)$, where Y is a free variable (a question mark variable in Coq) of type $i \rightarrow i$ and f and g are functions of type $i \rightarrow i$. If f and g are defined functions, then find instances of Y that make this equality true can involve rather deep mathematical reasoning: for example, if the function g always returns fixed points of the function f, then Y could be the identity function. We shall consider such equations to be addressed by unification, only when all *functions are considered to be constructors.* When this example equation is considered as a unification problem, then this problem has the most general unifier $Y \mapsto \lambda z.fz$. We note, however, that this problem is not a pattern unification problem. Fortunately, the restriction of *functions-as-constructors* is still a sound restriction since any solution for the restricted setting is a solution for the richer, mathematical interpretation of functions.

Let us return to the definition of pattern unification problems. The restriction on occurrences of the free variable, say, M is that (1) it can be applied only to variables that cannot appear free in terms that are used to instantiate M and (2) that those arguments are distinct. Condition (1) essentially says that the arguments of M form a primitive pattern that allows one to form an abstraction to solve a unification problem. Thus, $M\ x\ y$ can equal, say, $(s\ x) + y$ simply by forming the abstraction $\lambda x \lambda y.(s\ x) + y$. Condition (2) implies that such abstractions are unique.

The examples of needing richer unification problems above illustrate that it is also natural to consider arguments built using variables *and* term constructors: that is, we should consider generalizing condition (1) above so that the open term $\lambda l(M\ (\mathrm{fst}\ l)\ (\mathrm{fst}\ (\mathrm{snd}\ l)))$ is permitted. If this term is required to unify with a term of the form $\lambda l.t$ then all occurrence of l in t must occur in subterms of the form $(\mathrm{fst}\ l)$ or $(\mathrm{fst}\ (\mathrm{snd}\ l))$. In that case, this unification problem can be solved by substituting for M the result of abstracting out of t the expressions (fst l) and (fst (snd l)) with separate bound variables. To guarantee uniqueness of such solutions, we shall also generalize condition (2) so that the arguments of M cannot be subterms of each other. This additional constraint is required here (but not in the papers by Duggan [15] and Tiu [43]) since we wish to handle richer signatures than just those with monadic constructors.

Many of the examples leading to this generalization of pattern unification arise in situations where operators (such as fst and snd) are really functions and *not* constructors: the intended meaning of those two operators are as functions that map lists to either their first element or to their tail. When they arise in unification problems, however, we can only expect to treat them as constructors. Thus, we shall name this extended pattern unification as *function-as-constructor* (pattern) unification, or just FCU for short.

The rest of this paper is structured as follows. We cover the basic concepts related to higher-order unification in Section 2. The class of unification problems addressed in this paper, the *functions-as-constructor* class, is defined in Section 3 as is a unification algorithm for that class. We prove the correctness of that algorithm in Section 4 and consider extensions in the FCU class in Section 5. We conclude in Section 6.

This paper extends the conference paper [25] by containing revisions to all sections and by including a new section that discusses the relationships of our results to other works and contains an extension of the algorithm in the second-order case.

2 Preliminaries

2.1 The Lambda-Calculus

In this section we will present the logical language that will be used throughout the paper. The language is a version of Church's simple theory of types [13] following presentations given in [8] and [41]. Unless stated otherwise, all terms are treated up to α-conversion and are implicitly converted to long $\beta\eta$-normal form, i.e., β-normal and η-expanded form.

Let $\mathfrak{T}_\mathfrak{o}$ be a set of basic types, then the set of types $\mathfrak{T}$ is generated by $\mathfrak{T} := \mathfrak{T}_\mathfrak{o} \mid \mathfrak{T} \to \mathfrak{T}$. Let $\mathfrak{C}$ be a signature of function symbols and let $\mathfrak{V}$ be a countably infinite set of variable symbols. *Variables* are normally denoted by the letters $l, x, y, w, z, X, Y, W, Z$ and *function symbols* by the letters a, f, g, h, k or fixed width font names such as `cons`. We sometimes use subscripts and superscripts as well. Occasionally, we use a superscript to indicate the type of a symbol. The set $\texttt{Term}^\alpha$ of terms of type α is generated by the grammar $\texttt{Term}^\alpha := f^\alpha \mid x^\alpha \mid (\lambda x^\beta.\texttt{Term}^\gamma) \mid (\texttt{Term}^{\beta\to\alpha}\texttt{Term}^\beta)$ where $f \in \mathfrak{C}, x \in \mathfrak{V}$ and $\alpha \in \mathfrak{T}$ (in the abstraction case, $\alpha = \beta \to \gamma$). Applications associated are to the left. We will sometimes omit brackets when the meaning is clear. We will also normally omit typing information when it is not crucial for the correctness of the results. The set `Term` denotes the set of all terms. *Subterms* and *positions* within terms are defined as usual. We denote the fact that t is a (strict) subterm of s using the infix binary symbol ($\sqsubset$) $\sqsubseteq$. The *size of a position* denotes the length of the path to the position. We denote the subterm of t at position p by $t|_p$. *Bound* and *free variables* are defined as usual. We will use the convention of denoting bound and universally quantified variables by lower letters while existentially quantified variables will be denoted by capital letters. Given a term t, we denote by $\mathtt{hd}(t)$ its *head symbol* and distinguish between *flexible* terms, whose head is a free variable and *rigid* terms, whose head is a function symbol or a bound variable.

Substitutions are defined as usual and their *composition* ($\circ$) is given as $(\sigma \circ \theta)X = \theta(\sigma X)$. The trivial substitution that maps each variable to itself is denoted by $\mathtt{id}$. We denote by $\sigma|_W$ the substitution obtained from substitution σ by restricting its domain to variables in W. We denote by $\sigma[X \mapsto t]$ the substitution obtained from σ by composing it with the substitution $X \mapsto t$. We extend the application of substitutions to terms in the usual way and denote it by postfix notation. Bound variables are changed as necessary in order to avoid variable capture during substitution. A substitution σ is *more general* than a substitution θ, denoted $\sigma \leq \theta$, if there is a substitution δ such that $\sigma \circ \delta = \theta$. The *domain* of a substitution σ is denoted by $\mathtt{dom}(\sigma)$.

We introduce also a vector notation $\overline{t_n}$ for the sequence of terms $t_1, \ldots, t_n$. This notation also holds for nesting of sequences. For example, the term $f\ (X_1\ z_1\ z_2)\ (X_2\ z_1\ z_2)\ (X_3\ z_1\ z_2)$ will be denoted by $f\overline{X_3\overline{z_2}}$. The meaning of the notation $\lambda\overline{z_n}$ is $\lambda z_1 \ldots \lambda z_n$. When the order of the sequence is not important, we will use this notation also for multisets. We will use both *set union* ($\cup$) and *disjoint set union* ($\uplus$) in the text.

2.2 Higher-order Pre-unification

In this section we follow the presentation in [41] of Huet's pre-unification procedure [23]. The procedure will be shown, in Section 3.2, to be deterministic for the class of FCU problems. This result, together with the completeness of the procedure, implies the existence of most-general unifiers for unifiable problems of this class.

For the sake of our presentation of unification, we use the following non-standard normal form: all terms, including existential variables but excluding the arguments of these variables, are considered to be in η-expanded form. The arguments of these variables are expected to be in η-normal forms. In a similar manner to the one in [41], one can prove that all substitutions used in this paper preserve this normal form.

Definition 1 (Unification Problem) An equation is a formula $t \doteq s$ where t and s are $\beta\eta$-normalized (see remark above) terms. A unification problem is a formula of the form $\exists\overline{X_m}[e_1 \wedge \ldots \wedge e_n]$ where e_i for $0 < i \leq n$ is an equation. Let $\mathtt{bvars}(e_i) = \overline{z_n}$ for $e_i = (\lambda\overline{z_n}.t_i \doteq \lambda\overline{z_n}.s_i)$.

Definition 2 (Unification System) A unification system over a signature $\mathfrak{C}$ is the following quadruple $\langle Q_\exists, Q_\forall, S, \sigma\rangle$ where $Q_\exists$ and $Q_\forall$ are disjoint sets of variables, S is a set of equations and σ a substitution. Given a unification problem $\exists\overline{X_m}[e_1 \wedge \ldots \wedge e_n]$ we consider the unification system over signature $\mathfrak{C}$ by setting $Q_\exists = \overline{X_m}$, $Q_\forall = \{\}$, $S = \{e_1, \ldots, e_n\}$ and $\sigma = \mathtt{id}$.

Given a unification system $\langle Q_\exists, Q_\forall, S, \mathtt{id}\rangle$ over $\mathfrak{C}$, a unification algorithm attempts to find a substitution σ such that for each equation $t \doteq s \in S$, $t\sigma = s\sigma$. *Matching problems* and *matching systems* are variants of their unification counterparts in which we try to find a substitution σ such that for each matching problem $t \doteq s \ \in S$, $t\sigma = s$.

Before presenting Huet's procedure for pre-unification, we will repeat the definition of partial bindings as given in [41]. Note that this definition is inline with our non-standard η-normal forms.

Definition 3 (Partial bindings) A partial binding of type $\alpha_1 \to \ldots \to \alpha_n \to \beta$ where $\beta \in \mathfrak{T}_\mathfrak{o}$ is a term of the form

$$\lambda\overline{y_n}.a(\lambda\overline{z^1_{p_1}}.X_1(\overline{y_n}, \overline{z^1_{p_1}}), \ldots, \lambda\overline{z^m_{p_m}}.X_m(\overline{y_n}, \overline{z^m_{p_m}}))$$

for some constant or variable a where

– y_i has type α_i for $0 < i \leq n$.
– a has type $\gamma_1 \to \ldots \to \gamma_m \to \beta$ where $\gamma_i = \delta^i_1 \to \ldots \to \delta^i_{p_i} \to \gamma'_i$ for $0 < i \leq m$.
– $\gamma'_1, \ldots, \gamma'_m \in \mathfrak{T}_\mathfrak{o}$.
– z^i_j has type δ^i_j for $0 < i \leq m$ and $0 < j \leq p_i$.
– $X_1, \ldots, X_m$ is a list of distinct free variables and X_i has type $\alpha_1 \to \ldots \to \alpha_n \to \delta^i_1 \to \ldots \to \delta^i_{p_i} \to \gamma'_i$ for $0 < i \leq m$.

A partial binding is either an *imitation binding*, denoted by $\mathtt{PB}(a, \alpha)$, if a is either a constant or a free variable of type α, or it is a *projection binding* denoted by $\mathtt{PB}(i, \alpha)$, where a is the bound variable y_i of type α for some $0 < i \leq n$. Since partial bindings are uniquely determined by a bound variable index, a type and an atom (up to renaming of the free variables $\overline{X_m}$), this defines a particular term.

Definition 4 (Huet's Pre-unification Procedure) Huet's pre-unification procedure is given by the non-deterministic rewriting system that is given in Table 1. Note that the sets $Q_\exists$ and $Q_\forall$ are fixed during the execution and are mentioned explicitly just for compatibility with the algorithms given later in the paper.

A unification problem is in a *reduced form* if it cannot be reduced further by the algorithm. A unification problem in a reduced form is *pre-solved* if it contains only the so-called *flexible-flexible pairs*, i.e., equations between two flexible terms. Otherwise, the problem is unsolvable.

Note that it is possible to have an infinite series of valid steps in this pre-unification procedure: such is the case when we start with an equation, such as

`(Delete)`

$$\langle Q_\exists, Q_\forall, S \uplus \{t \doteq t\}, \sigma\rangle \;\rightarrow\; \langle Q_\exists, Q_\forall, S, \sigma\rangle$$

`(Simpl)`

$$\langle Q_\exists, Q_\forall, S \uplus \{\lambda\overline{z_k}.f\overline{t_n} \doteq \lambda\overline{z_k}.f\overline{s_n}\}, \sigma\rangle \;\rightarrow\; \langle Q_\exists, Q_\forall, S \uplus \{\lambda\overline{z_k}.t_1 \doteq \lambda\overline{z_k}.s_1, \ldots, \lambda\overline{z_k}.t_n \doteq \lambda\overline{z_k}.s_n\}, \sigma\rangle$$

`(Bind)`

$$\langle Q_\exists, Q_\forall, S \uplus \{\lambda\overline{z_k}.X\overline{z_k} \doteq \lambda\overline{z_k}.t\}, \sigma\rangle \;\rightarrow\; \langle Q_\exists, Q_\forall, S\theta \uplus \{X \doteq \lambda\overline{z_k}.t\}, \sigma \circ \theta\rangle$$

where $X \notin \texttt{fvars}(t)$ and $\theta = [X \mapsto \lambda\overline{z_k}.t]$

`(Imitate)`

$$\langle Q_\exists, Q_\forall, S \uplus \{\lambda\overline{z_k}.X^\alpha(\overline{s_n}) \doteq \lambda\overline{z_k}.f(\overline{t_m})\}, \sigma\rangle \;\rightarrow\; \langle Q_\exists, Q_\forall, S \uplus \{X \doteq u, \lambda\overline{z_k}.X^\alpha(\overline{s_n}) \doteq \lambda\overline{z_k}.f(\overline{t_m})\}, \sigma\rangle$$

where $u = \texttt{PB}(f, \alpha)$ and $f \in \mathfrak{C}$

`(Project)`

$$\langle Q_\exists, Q_\forall, S \uplus \{\lambda\overline{z_k}.X^\alpha(\overline{s_n}) \doteq \lambda\overline{z_k}.a(\overline{t_m})\}, \sigma\rangle \;\rightarrow\; \langle Q_\exists, Q_\forall, S \uplus \{X \doteq u, \lambda\overline{z_k}.X^\alpha(\overline{s_n}) \doteq \lambda\overline{z_k}.a(\overline{t_m})\}, \sigma\rangle$$

where $0 < i \le k, u = \texttt{PB}(i, \alpha)$ and either $a \in \mathfrak{C}$ or $a = z_i$ for some $0 < i \le k$

Table 1 Huet's pre-unification procedure

$X \doteq f(X)$ where f is a constant. Note that a first-order unification algorithm will fail on a first-order version of this equation due to an occur check. These checks cannot always determine non-unifiability in the higher-order case and are therefore not used by the pre-unification algorithm to eliminate such cases.

The next theorem states the completeness of this procedure (the proof can be found in [23,41]).

Theorem 1 *Consider a system $\langle Q_\exists, Q_\forall, S, \textit{id}\rangle$ and assume it is unifiable by σ. Then there is a sequence of rule applications in Def. 4 resulting in $\langle Q_\exists, Q_\forall, S', \theta\rangle$ such that $\theta \le \sigma$ and S' is in pre-solved form.*

The following corollary of this theorem will be used in Section 4.

Corollary 1 *Given a system $\langle Q_\exists, Q_\forall, S, \theta\rangle$ and assuming it is unifiable by $\theta \circ \sigma$, then the following hold:*

- *An application of `(Bind)` or `(Simpl)` results in a system $\langle Q_\exists, Q_\forall, S', \theta'\rangle$ which is unifiable by $\theta' \circ \sigma$.*
- *If `(Imitate)` or `(Project)` are applicable to the system, then there is a particular application of `(Imitate)` or `(Project)`, which results in a system $\langle Q_\exists, Q_\forall, S', \theta'\rangle$ such that it is unifiable by $\theta' \circ \sigma$.*

2.3 Pattern Unification

In this section we describe the higher-order pattern unification algorithm given in [27] although we use notation more similar to Nipkow's notation in [34]. This algorithm forms the basis for our algorithm.

Definition 5 (Pattern Systems) A system $\langle Q_\exists, Q_\forall, S, \sigma\rangle$ is called a pattern system if for all equations $e_i \in S$ and for all subterms $X\overline{z_n}$ in these equations such that $X \in Q_\exists$, it is the case that $z_1, \ldots, z_n$ is a list of distinct variables from the set $Q_\forall \cup \texttt{bvars}(e_i)$.

The following simplification will be called during the execution of the algorithm given in Def. 7.

(0)	$\langle Q_\exists, Q_\forall, S \uplus \{t \doteq t\}, \sigma\rangle \to \langle Q_\exists, Q_\forall, S\}, \sigma\rangle$
(1)	$\langle Q_\exists, Q_\forall, S \uplus \{\lambda x.s \doteq \lambda x.t\}, \sigma\rangle \to \langle Q_\exists, Q_\forall \uplus \{x\}, S \uplus \{s \doteq t\}, \sigma\rangle$
(2)	$\langle Q_\exists, Q_\forall, S \uplus \{f\overline{t_n} \doteq f\overline{s_n}\}, \sigma\rangle \to \langle Q_\exists, Q_\forall, S \uplus \{t_1 \doteq s_1, \ldots, t_n \doteq s_n\}, \sigma\rangle$ where $f \in \mathfrak{C} \cup Q_\forall$
(3)	$\langle Q_\exists \uplus \{X\}, Q_\forall, S \uplus \{X\overline{z_n} \doteq f\overline{s_m}\}, \sigma\rangle \to \langle Q_\exists, Q_\forall, S\theta, \sigma \circ \theta\rangle$ where $f \in \mathfrak{C}$, $X \notin \mathtt{fvars}(f\overline{s_m})$ and $\theta = [X \mapsto \lambda\overline{z_n}.f\overline{s_m}]$
(4)	$\langle Q_\exists \uplus \{X\}, Q_\forall, S \uplus \{X\overline{z_n} \doteq X\overline{y_n}\}, \sigma\rangle \to \langle Q_\exists \uplus \{W\}, Q_\forall, S\theta, \sigma \circ \theta\rangle$ $\theta = [X \mapsto \lambda\overline{z_n}.W\overline{w_k}]$ and $\overline{w_k} = \{z_i \mid z_i = y_i\}$
(5)	$\langle Q_\exists \uplus \{Y\}, Q_\forall, S \uplus \{X\overline{z_n} \doteq Y\overline{y_m}\}, \sigma\rangle \to \langle Q_\exists, Q_\forall, S\theta, \sigma \circ \theta\rangle$ where $X \neq Y$, $\theta = [Y \mapsto \lambda\overline{y_m}.X\overline{y_{\phi(m)}}]$ and ϕ is a permutation such that $\phi(j) = i$ if $z_i = y_j$ for $0 < i \leq n$ and $0 < j \leq m$

Table 2 Pattern Unification Algorithm

Definition 6 (Pruning) Let $\langle Q_\exists, Q_\forall, S, \sigma\rangle$ be a pattern system that contains the equation $X\overline{z_n} \doteq r \in S$. If r contains an occurrence of a variable $y \in Q_\forall$ and that y is not a member of $\overline{z_n}$, then:

- if there is a subterm $W\overline{w_m}$ of r such that $y = w_i$ for some $0 < i \leq m$, then return $\langle Q_\exists \uplus \{W'\} \setminus \{W\}, Q_\forall, S\theta, \sigma \circ \theta\rangle$ where W' is a fresh variable with respect to S, $\theta = [W \mapsto \lambda\overline{w_m}.W'\overline{u_{m-1}}]$ and $\overline{u_{m-1}} = \overline{w_m} \setminus \{w_i\}$.
- otherwise, the algorithm fails

Example 1 Consider a pattern system that contains the equation

$$\lambda x\lambda y\lambda z.Xyx \doteq \lambda x\lambda y\lambda z.Xxz$$

and where X is an existential variable. If we pick the subterm r and the variable y from the above definition to be Xxz and z, respectively, then the pruning substitution $[X \mapsto \lambda x\lambda z.W'x]$ is produced. The resulting equation is $\lambda x\lambda y\lambda z.W'y \doteq \lambda x\lambda y\lambda z.W'x$. This pruning process can be performed also on this resulting equation, yielding the substitution $[W' \mapsto \lambda x.W'']$. The resulting equation is now trivial: $\lambda x\lambda y\lambda z.W'' \doteq \lambda x\lambda y\lambda z.W''$. Thus, the original pattern system has the unifier $[X \mapsto \lambda x\lambda y\lambda z.W'']$.

Definition 7 (The Pattern Unification Algorithm) The pattern unification algorithm is the application of the rules from Table 2 in the following order. First, apply rules (0), (1), and (2) until they are no longer applicable. Second, apply the pruning rewriting rules until exhaustion. Finally, select an equation and apply either rule (3), if that equation (or its converse) is flexible-rigid, and either rule (4) or (5) if that equation is flexible-flexible. Repeat these steps until no steps can be taken.

A proof that the rewriting in Definition 7 is terminating, sound, and complete can be found in [27].

2.4 Deterministic Second-order Matching

As was mentioned in the introduction, Miller's pattern fragment is too weak for some applications. Yokoyama, Hu, and Takeichi have presented an extension [46] that proved useful for matching problems that arose from some program transformation applications. Their fragment, called *deterministic second-order patterns (DSP)*, relaxes the requirements on the arguments of second-order variables.

We discuss their fragment in detail in Sec. 5. Nevertheless, we find it informative to present this fragment before proceeding with ours, due to the similarity of some of their restrictions to the ones used in the algorithm we present in the next section.

Definition 8 (DSP restricted terms[46]) Given $\mathfrak{C}$ and $Q_\forall$, a DSP restricted term t is a term such that:

- t is a first-order term using only the constants and variables in $\mathfrak{C}$ and $Q_\forall$, and
- t has a free occurrence of a variable in $Q_\forall$.

The next several examples will make use of fixing $\mathfrak{C} = \{\texttt{cons}, \texttt{fst}, \texttt{snd}, \texttt{nil}\}$ and $Q_\forall = \{l, z\}$.

Example 2 The terms l, $(\texttt{cons}\ z\ l)$, $(\texttt{cons}\ (\texttt{fst}\ l)\ l)$, $(\texttt{snd}\ (\texttt{cons}\ z\ l))$ and $(\texttt{cons}\ z\ \texttt{nil})$ are DSP restricted terms over the above $\mathfrak{C}$ and $Q_\forall$, while $\texttt{nil}$ is not.

Definition 9 (DSP Systems[46]) A system $\langle Q_\exists, Q_\forall, S, \sigma\rangle$ is a DSP system if the following two conditions are satisfied:

- DSP-`argument restriction`: For all occurrences of $X\overline{t_n}$ in S where $X \in Q_\exists$, the term t_i, for all $0 < i \leq n$, is a DSP restricted term.
- DSP-`local restriction`: For all occurrences $X\overline{t_n}$ in S where $X \in Q_\exists$ and for each t_i and t_j such that $0 < i, j \leq n$ and $i \neq j$, $t_i \not\sqsubseteq t_j$.

In [46], Yokoyama, Hu, and Takeichi present an efficient second-order matching algorithm and prove its correctness.

3 A Unification Algorithm for FC Higher-order Unification Problems

3.1 FC Higher-order Unification (FCU) Problems

The main difference between pattern and FCU problems is in the form of arguments of existentially quantified variables. In pattern unification problems, the arguments to an existentially quantified variable must be a list of distinct universally quantified variables whose binders are in the scope of the binder for the existentially quantified variable. In FCU problems, these restrictions on the structure of the arguments are relaxed. We discuss the relationship between our restrictions and those of Deterministic Second-order terms in Sec. 5.

The following definition generalizes the pattern unification arguments of higher-order variables. These arguments are extended from bound variables to more complex terms. These terms allow function symbols, as long as bound variables occupy all leaf positions. There are no existential variables or abstractions allowed though.

Definition 10 (Restricted Terms) Given $\mathfrak{C}$, $Q_\forall$ and an equation e, a restricted term in e is defined inductively as follows:

- $a \in Q_\forall \cup \texttt{bvars}(e)$ is a restricted term.
- $f\overline{t_n}$ is a restricted term if $n > 0$, $f \in \mathfrak{C} \cup Q_\forall \cup \texttt{bvars}(e)$ and t_i is a restricted term for all $0 < i \leq n$.

When e is clear from the context, we will refer to these terms just as restricted terms. Note that restricted terms do not contain λ-abstractions.

Example 3 The terms l, $(\texttt{cons}\ z\ l)$, $(\texttt{cons}\ (\texttt{fst}\ l)\ l)$ and $(\texttt{snd}\ (\texttt{cons}\ z\ l))$ are restricted terms over the above $\mathfrak{C}$ and $Q_\forall$, while `nil` and $(\texttt{cons}\ z\ \texttt{nil})$ are not.

Definition 11 (FCU Systems) A system $\langle Q_\exists, Q_\forall, S, \sigma\rangle$ is an FCU system if the following three conditions are satisfied.

- `argument restriction`: For every occurrence of $X\overline{t_n}$ in S where $X \in Q_\exists$, it is the case that t_i is a restricted term for all $0 < i \leq n$.
- `local restriction`: Whenever S contains an occurrence of $X\overline{t_n}$ where $X \in Q_\exists$, then the arguments $\overline{t_n}$ are constrained so that for every distinct pair of members of $\{1, \ldots, n\}$, say i and j, it is the case that $t_i \not\sqsubseteq t_j$.
- `global restriction`: For each two different occurrences $X\overline{t_n}$ and $Y\overline{s_m}$ in S where $X, Y \in Q_\exists$ and for every $0 < i \leq n$ and $0 < j \leq m$, it is the case that $t_i \not\sqsubset s_j$.

Example 4 Pattern unification equations, such as $\{X\ l\ z \doteq \texttt{fst}\ (\texttt{snd}\ l)\}$, are also FCU systems. The equation

$$\{\texttt{cons}\ (X\ (\texttt{fst}\ l))\ (\texttt{snd}\ l) \doteq \texttt{snd}\ (Y\ (\texttt{fst}\ l)\ (\texttt{fst}\ (\texttt{snd}\ l)))\}$$

yields an FCU system. Examples of non-FCU problems are the following:

- $\{X\ (\texttt{cons}\ z\ \texttt{nil}) \doteq \texttt{snd}\ l\}$, which violates the `argument restriction`, since (`cons` z `nil`) is not a restricted term;
- $\{X\ (\texttt{fst}\ l)\ l \doteq \texttt{cons}\ z\ l\}$, which violates the `local restriction` since X is applied to an argument that is a subterm of another argument; and
- $\{X\ (\texttt{fst}\ l) \doteq \texttt{snd}\ (Y\ (\texttt{cons}\ (\texttt{fst}\ l)\ (\texttt{snd}\ l)))\}$, which violates (only) the `global restriction`, since the argument of X is a strict subterm of an argument of Y.

The next proposition is easy to verify.

Proposition 2 *Pattern systems are FCU systems.*

Before going on to show some properties of these problems, we provide some motivation behind the restrictions above. The three restrictions are used in the next section to prove the determinacy of Huet's pre-unification procedure over FCU problems. Nevertheless, we do not prove that this result does not hold when weakening the above restrictions. The `local restriction` and `global restriction` can easily be shown to be required even for very simple examples. This is not the case for the `argument restriction`. One alternative is to weaken the restricted term definition from above to require only one subterm in the second condition to be restricted: that is, allow terms such as ($\texttt{cons}\ z\ \texttt{nil}$) as arguments of existential variables. In the following, we will give a counter-example to this weaker restriction. Still, it should be noted that the counterexample depends on allowing inductive definitions containing more than one base case (in particular, we allow for different empty list constructors $\texttt{nil}_1$ and $\texttt{nil}_2$). When such definitions are not allowed, it may be possible to prove the results given in this paper for a stronger class of problems.

Example 5 The unification system

$$\langle\{X,Y\},\{z,z_2\},\{X\ (\texttt{cons}\ z\ \texttt{nil}_1)\ (\texttt{cons}\ z\ \texttt{nil}_2) \doteq \texttt{cons}\ z\ (Y\ z_2)\},\texttt{id}\rangle$$

is unifiable by the following two incompatible substitutions:

1. $[Y \mapsto \lambda z_1.\texttt{nil}_1, X \mapsto \lambda z_1,z_2.z_1]$.
2. $[Y \mapsto \lambda z_1.\texttt{nil}_2, X \mapsto \lambda z_1,z_2.z_2]$.

3.2 The Existence of Most-general Unifiers

From this section on, an FCU problem will be referred to simply as a *system*, unless indicated otherwise.

It is pointed out in [41] that the only "don't-know" non-determinism in the general higher-order procedure stems from the choice between the different applications of `(Imitate)` and `(Project)`. We prove that fulfilling the three restrictions in Def. 11 makes these choices deterministic.

We first prove a couple of lemmas.

Lemma 1 *If r is a restricted term and $X\overline{t_n} \doteq r$ is a unifiable equation e, then there is $0 < i \leq n$ such that t_i is a subterm of r.*

Proof By induction on the structure of the restricted term r. In the base case, $r \in Q_\forall \cup \texttt{bvars}(e)$. We first note that the substitution cannot be an imitation as r contains only bound variables (those in $Q_\forall$ contain variables which are bound as well). The only way a substitution applied to X can unify with r is in the case it projects one of the bound variables. In that case, one of the t_i arguments will appear at the head of the result substitution instance. Since the latter must equal r, then r contains that t_i as a subterm. In the inductive case, r has the form $f\overline{r_m}$ where $m > 0$, $f \in \mathfrak{C} \cup Q_\forall \cup \texttt{bvars}(e)$ and r_i is a restricted term for all $0 < i \leq m$. The unifier must be covered by a projection or imitation term. In the former case, the argument above proves the conclusion. In the latter case, $[X \mapsto \lambda\overline{w}.f(X_1\overline{w})\cdots(X_m\overline{w})]$ for $m > 0$ and $X_1, \ldots, X_m$ new free variables. We thus have that $(X_1\overline{t_n}) \doteq r_1$ is unifiable and, by inductive assumption, some t_i is a subterm of r_1, implying that t_i is a subterm of r.

Lemma 2 *Consider the equation* $t\,\overline{t_k} \doteq f\,\overline{s_n}$*, where* t *is a restricted term and* $f \in \mathfrak{C} \cup Q_\forall$*. If this equation is unifiable then* $t = f\,\overline{v_{n-k}}$ *for restricted terms* $\overline{v_{n-k}}$*.*

Proof Since t is restricted, it does not contain abstractions and existential variables and since the equation is unifiable, t can be written as $f\,\overline{v_{n-k}}$. Since t is restricted, all its subterms are restricted as well.

The next two lemmas prove the determinism claim on applications of `(Project)` and `(Imitate)`. The first of these lemmas states that applying different projection substitutions to an equation in an FCU system yield two systems that cannot both lead to a successful unification.

Lemma 3 *Consider the equation* $X\,\overline{t_n} \doteq f\,\overline{s_m}$ *where* X *does not occur in* $f\,\overline{s_m}$*. Let* i *and* j *be such that* $0 < i < j \leq n$ *and let* $\sigma_0 = [X \mapsto \lambda\overline{z_n}.\,z_i\overline{X_l\overline{z_n}}]$ *and* $\theta_0 = [X \mapsto \lambda\overline{z_n}.\,z_j\overline{Y_k\overline{z_n}}]$ *be two projection substitutions. Applying these substitutions yield, respectively, the following two equations:*

$$t_i\,\overline{X_l\,\overline{t_n}} \doteq f\,\overline{s_m} \tag{1}$$

$$t_j\,\overline{Y_k\,\overline{t_n}} \doteq f\,\overline{s_m} \tag{2}$$

There are no substitutions σ *and* θ *such that* σ *unifies equation 1 and* θ *unifies equation 2.*

Proof We assume the existence of two such unifiers and then obtain a contradiction. According to Lemma 2, we can rewrite the two equations as

$$f\,\overline{v_{m-l}}\,\overline{X_l\,\overline{t_n}} \doteq f\,\overline{s_m} \quad \text{and} \quad f\,\overline{u_{m-k}}\,\overline{Y_k\,\overline{t_n}} \doteq f\,\overline{s_m}$$

for restricted terms $\overline{v_{m-l}}$ and $\overline{u_{m-k}}$. Without loss of generality, we assume that $l \geq k$. Note also, that since $t_i \neq t_j$ and t_i, t_j have f as a head symbol, $m \geq m - k > 0$. We consider two cases:

– All $s_1, \ldots, s_{m-k}$ are ground terms. In this case and since both equations are unifiable, we get the equation

$$f\,\overline{v_{m-l}}\,(\overline{X_{l-k}\,\overline{t_n}})\sigma = f\,\overline{s_{m-k}} = f\,\overline{u_{m-k}} \tag{3}$$

Clearly, $k \neq l$ since otherwise $t_i = t_j$ which violates the `local restriction` from Def. 11. We can now conclude that

$$(X_1\,\overline{t_n})\sigma = u_{m-l+1} \tag{4}$$

Since u_{m-l+1} is a restricted term then, according to Lemma 1, there is $0 < k_1 \leq n$ such that t_{k_1} is a subterm of u_{m-l+1}. Since u_{m-l+1} is a subterm of t_j, we get that t_{k_1} is a subterm of t_j which contradicts the `local restriction`.

– There is s_{k_1} for $0 < k_1 \leq m-k$ which contains an occurrence of $Z\,\overline{r_{k_2}}$. This must occur as a subterm of s_{k_1} as otherwise the subterm $Z\,\overline{r_{k_2}}r'$ where r' is not a restricted term, violates the `argument restriction`. Since u_{k_1} is a restricted term, any subterm of it is restricted as well. Therefore and as $s_{k_1}\theta = u_{k_1}$, we have that there exists a restricted term u' such that $Z\,\overline{r_{k_2}}\theta = u'$. Using Lemma 1, we can conclude that there is $0 < k_3 \leq k_2$ such that r_{k_3} is a subterm of u', which is a subterm of u_{k_1} which is a strict subterm of t_j, which violates the `global restriction`.

The following lemma states that applying an imitation and a projection substitution to an equation in an FCU system yields two system that cannot both lead to a successful unification.

Lemma 4 *Consider the equation $X\,\overline{t_n} \doteq f\,\overline{s_m}$ where X does not occur in $f\,\overline{s_m}$. Let $\sigma_0 = [X \mapsto \lambda\overline{z_n}.f\,\overline{X_m\,\overline{z_n}}]$ and $\theta_0 = [X \mapsto \lambda\overline{z_n}.z_j\,\overline{Y_k\,\overline{z_n}}]$ for some $0 < j \leq n$. Applying these substitutions yield, respectively, the following two equations:*

$$f\,\overline{X_m\,\overline{t_n}} \doteq f\,\overline{s_m} \tag{5}$$

$$t_j\,\overline{Y_k\,\overline{t_n}} \doteq f\,\overline{s_m} \tag{6}$$

There are no substitutions σ and θ such that σ unifies equation 5 and θ unifies equation 6.

Proof We assume the existence of the two such unifiers and obtain a contradiction. Using Lemma 2, we can rewrite Eq. 6 as:

$$f\,\overline{v_{m-k}}\,\overline{Y_k\,\overline{t_n}} \doteq f\,\overline{s_m} \tag{7}$$

where $v_1, \ldots, v_{m-k}$ are restricted terms and strict subterms of t_j. Since f is imitated, it is not a restricted term and $f \neq t_j$ which implies that $m - k > 0$. Eq. 7 tells us that $v_1 = s_1\theta$ which implies that $s_1\theta$ is a strict subterm of t_j and a restricted term. On the other hand, we have that $X_1\,\overline{t_n} = s_1\sigma$ from Eq. 5. We consider two cases:

– s_1 is ground. In this case we can use Lemma 1 and the fact that s_1 is a restricted term to conclude that there is $0 < k_1 \leq n$ such that t_{k_1} is a subterm of s_1. On the other hand, we know that s_1 is a strict subterm of t_j and therefore we get that t_{k_1} is a strict subterm of t_j, which contradicts the `local restriction`..
– If s_1 is not ground, it must contain an occurrence $Z\,\overline{r_l}$. This occurrence cannot occur as the subterm $Z\,\overline{r_l}r'$ where r' is not a restricted term as it violates the `argument restriction`. Therefore, $Z\,\overline{r_l}$ is a subterm of s_1. Since v_1 is a restricted term, any subterm of it is restricted as well. Therefore and as $s_1\theta = v_1$, we have that there is a restricted term v' such that $Z\,\overline{r_l} = v'$. Now we use Lemma 1 and get that there is $0 < k_1 \leq l$ such that r_{k_1} is a subterm of v', which is a strict subterm of t_j. We get again a contradiction to the `global restriction`.

3.3 The Unification Algorithm

For defining the FC unification algorithm, we need to slightly extend the definition of pruning.

Definition 12 (Covers) A cover for $X\,\overline{t_n}$ and a restricted term q is a substitution σ such that $(X\,\overline{t_n})\sigma = q$.

Note that uniqueness of covers follows from Theorem 5

Example 6 The substitution $[X \mapsto \lambda z_1 \lambda z_2.\texttt{cons}\ (\texttt{fst}\ z_1)\ z_2]$ is a cover for $(X\ l\ z)$ and $(\texttt{cons}\ (\texttt{fst}\ l)\ z)$.

Definition 13 (Pruning) Given an FCU system $\langle Q_\exists, Q_\forall, S, \sigma\rangle$ such that $(X\,\overline{t_n} \doteq r) \in S$ and r contains an occurrence of a restricted term q such that $q \notin \overline{t_n}$. The result of *pruning* this system is one of the following three systems.

- If there is a subterm $W\overline{s_m}$ of r such that $q = s_i$ for some $0 < i \leq m$, then return $\langle Q_\exists \uplus \{W'\} \setminus \{W\}, Q_\forall, S\theta, \sigma \circ \theta\rangle$ where $\theta = [W \mapsto \lambda\overline{z_m}.W'\overline{z'_{m-1}}]$ and $\overline{z'_{m-1}} = \overline{z_m} \setminus \{z_i\}$.
- If there is no cover ρ for $X\overline{t_n}$ and q, then the algorithm fail.
- Otherwise, return the original system.

Example 7 Given the system

$$\begin{aligned}\langle \{X, Y, W, Z\}, \{l, w, z\}, \{&X\ (\texttt{snd}\ l)\ z \doteq Y\ z\ (\texttt{fst}\ l),\\ &W\ (\texttt{fst}\ l)\ z \doteq \texttt{snd}\ (Z\ w\ (\texttt{fst}\ l))\}, \texttt{id}\rangle,\end{aligned}$$

we can apply the following three prunings, $\sigma_1 = [Z \mapsto \lambda z_1, z_2.Z'z_2]$, $\sigma_2 = [Y \mapsto \lambda z_1, z_2.Y'z_1]$ and $\sigma_3 = [X \mapsto \lambda z_1, z_2.X'z_2]$ and obtain the system $\langle\{X', Y', W, Z'\}, \{l, w, z\}, \{X'\ z \doteq Y'\ z, W\ (\texttt{fst}\ l)\ z \doteq \texttt{snd}\ (Z'\ (\texttt{fst}\ l))\}, \sigma_1 \circ \sigma_2 \circ \sigma_3\rangle$.

For the next definition, we will use the following replacement operator $r\ |^{\overline{t_n}}_{\overline{z_n}}$ to denote the simultaneous replacement of each occurrence t_i in r with z_i for $0 < i \leq n$. We note that this replacement operator will be used only when none of the term in the list $\overline{t_n}$ are subterms of another term in that list: as a result, this replacement operation will be well-defined. Also, replacement respects the rules of α, β, and η-conversion: in particular, the replacement of $\lambda w.w$ with the variable u in the expression $(f\ (\lambda x.x)\ (\lambda y.y))$ yields $(f\ u\ u)$, where f is a constructor.

Definition 14 (Algorithm for FCU Systems) The rules of an algorithm for the unification of FCU systems is given in Table 3 where before the application of rules (3) and (5), we apply exhaustively first rule (1) and then pruning.

$$\begin{array}{ll}
(0) & \langle Q_\exists, Q_\forall, S \uplus \{t \doteq t\}, \sigma\rangle \to \langle Q_\exists, Q_\forall, S, \sigma\rangle \\
(1) & \langle Q_\exists, Q_\forall, S \uplus \{\lambda x.s \doteq \lambda x.t\}, \sigma\rangle \to \langle Q_\exists, Q_\forall \uplus \{x\}, S \uplus \{s \doteq t\}, \sigma\rangle \\
(2) & \langle Q_\exists, Q_\forall, S \uplus \{f\overline{t_n} \doteq f\overline{s_n}\}, \sigma\rangle \to \langle Q_\exists, Q_\forall, S \uplus \{t_1 \doteq s_1, \ldots, t_n \doteq s_n\}, \sigma\rangle \\
 & \text{where } f \in \mathfrak{C} \uplus Q_\forall \\
(3) & \langle Q_\exists \uplus \{X\}, Q_\forall, S \uplus \{X\overline{t_n} \doteq f\overline{s_m}\}, \sigma\rangle \to \langle Q_\exists, Q_\forall, S\theta, \sigma \circ \theta\rangle \\
 & \text{where } f \in \mathfrak{C},\ X \notin \mathtt{fvars}(f\overline{s_m}) \text{ and } \theta = [X \mapsto \lambda\overline{z_n}.f\overline{s_m}\,|^{\overline{t_n}}_{\overline{z_n}}] \\
(4) & \langle Q_\exists \uplus \{X\}, Q_\forall, S \uplus \{X\overline{t_n} \doteq X\overline{s_n}\}, \sigma\rangle \to \langle Q_\exists \uplus \{W\}, Q_\forall, S\theta, \sigma \circ \theta\rangle \\
 & \text{where } W \notin Q_\exists,\ \overline{t_n} \neq \overline{s_n},\ \theta = [X \mapsto \lambda\overline{z_n}.W\overline{z_{r_k}}] \text{ and} \\
 & \overline{r_k} = \{i \mid 0 < i \leq n \wedge t_i = s_i\} \\
(5) & \langle Q_\exists \uplus \{Y\}, Q_\forall, S \uplus \{X\overline{t_n} \doteq Y\overline{s_m}\}, \sigma\rangle \to \langle Q_\exists, Q_\forall, S\theta, \sigma \circ \theta\rangle \\
 & \text{where } X \neq Y,\ \theta = [Y \mapsto \lambda\overline{z_m}.X\overline{z_{\phi(m)}}] \text{ and } \phi \text{ is a permutation (see} \\
 & \text{Lemma 11) such that } \phi(j) = i \text{ if } t_i = s_j \text{ for } 0 < i \leq n \text{ and } 0 < j \leq m
\end{array}$$

Table 3 An algorithm for FCU problems

$$\begin{array}{ll}
 & \langle \{X, Y\}, \emptyset, \{\lambda l_1 \lambda l_2.X\ (\mathtt{fst}\ l_1)\ (\mathtt{fst}\ (\mathtt{snd}\ (l_1))) \doteq \lambda l_1 \lambda l_2.\mathtt{snd}\ (Y\ (\mathtt{fst}\ l_2)\ (\mathtt{fst}\ l_1))\}, \mathtt{id}\rangle \\
\to^{(1)\times 2} & \langle \{X, Y\}, \{l_1, l_2\}, \{X\ (\mathtt{fst}\ l_1)\ (\mathtt{fst}\ (\mathtt{snd}\ (l_1))) \doteq \mathtt{snd}\ (Y\ (\mathtt{fst}\ l_2)\ (\mathtt{fst}\ l_1))\}, \mathtt{id}\rangle \\
\to^{\mathtt{prun}} & \langle \{X, Y'\}, \{l_1, l_2\}, \{X\ (\mathtt{fst}\ l_1)\ (\mathtt{fst}\ (\mathtt{snd}\ (l_1))) \doteq \mathtt{snd}\ (Y'\ (\mathtt{fst}\ l_1))\}, [Y \mapsto \lambda z_1 \lambda z_2.Y' z_2]\rangle \\
\to^{(3)} & \langle \{Y'\}, \{l_1, l_2\}, \{\mathtt{snd}\ (Y'\ (\mathtt{fst}\ l_1)) \doteq \mathtt{snd}\ (Y'(\mathtt{fst}\ l_1))\}, \\
 & \qquad\qquad [Y \mapsto \lambda z_1 \lambda z_2.Y' z_2, X \mapsto \lambda z_1 \lambda z_2.\mathtt{snd}\ (Y' z_1)]\rangle \\
\to^{(0)} & \langle \{Y'\}, \{l_1, l_2\}, \emptyset, [Y \mapsto \lambda z_1 \lambda z_2.Y' z_2, X \mapsto \lambda z_1 \lambda z_2.\mathtt{snd}\ (Y' z_1)]\rangle
\end{array}$$

Table 4 An example of a reduction on an FCU

$$\begin{array}{ll}
 & \langle \{F\}, \{sum, n\}, \{(\lambda j.\ add\ (project\ j)\ (project\ j)) \doteq (\lambda j.\ F\ (project\ j))\}, \mathtt{id}\rangle \\
\to^{(1)} & \langle \{F\}, \{sum, n, j\}, \{(add\ (project\ j)\ (project\ j)) \doteq (F\ (project\ j))\}, \mathtt{id}\rangle \\
\to^{(3)} & \langle \{\}, \{sum, n, j\}, \emptyset, \{F \mapsto \lambda z.(add\ z\ z)\}\rangle
\end{array}$$

Table 5 An example of a reduction on an FCU, adapted from [18]

Example 8 The quantified equation

$$\exists X \exists Y \lambda l_1 \lambda l_2.X\ (\mathtt{fst}\ l_1)\ (\mathtt{fst}\ (\mathtt{snd}\ l_1)) \doteq \lambda l_1 \lambda l_2.\mathtt{snd}\ (Y\ (\mathtt{fst}\ l_2)\ (\mathtt{fst}\ l_1))$$

is proved using the rewriting for FCU systems, as illustrated in Table 4.

One can also find examples of FCU problems in the libraries of proof assistants. The `bigop` library of the Coq proof assistant [10] contains various occurrences of FCU instances. Since the proof assistant, so far, has not implemented FCU, it cannot resolve such lemmas and constructors are sometimes being stripped away manually[1] in order for pattern unification to be applicable [42].

The following example from Coq and Elpi is adapted from [18].

Example 9 The unification problem from [18]

$$\exists F.(sum\ n\ (\lambda j.\ add\ (project\ j)\ (project\ j)) \doteq (sum\ n\ (\lambda j.\ F\ (project\ j)))$$

is proved using the rewriting for FCU systems, as illustrated in Table 5, where the first rewriting step (2) is elided.

[1] See, for example, (`rewrite` - (`big_mkord` _ (`fun` _ `=>` _) `G`)) in the bigop library.

Example 10 To illustrate the FCU algorithm on flexible-flexible unification problems, we first introduce the f be a constructor of two arguments. The equation

$$\lambda u \lambda v.\ X\ (f\ u\ v)\ (f\ v\ u) = \lambda u \lambda v.\ X\ (f\ v\ u)\ (f\ u\ v)$$

is solved by using rule (4) to produce the substitution $X \mapsto \lambda w_1 \lambda w_2.\ W$. The equation

$$\lambda u \lambda v.\ X\ (f\ u\ v)\ (f\ v\ u) = \lambda u \lambda v.\ Y\ (f\ v\ u)\ (f\ u\ v)$$

is solved by using rule (5) to produce the substitution $Y \mapsto \lambda w_1 \lambda w_2.\ X\ w_2\ w_1$.

By dropping the requirement that terms are in η-long form, this algorithm can also work with terms that are untyped: the algorithm only requires applying η-expansion enough so that terms in an equation have bindings of the same length (see [27, Section 9.3]). It is the presence or absence of constructors and bound variables that matters in this algorithm and not the types of those constructors and variables. Rich typing can, of course, be imposed to disallow unifiers that might be over-generated when one considers the untyped case.

4 Correctness of the Algorithm

The unification algorithm transforms systems by the application of substitutions and by the elimination of equations. We prove next that the application of rules of the algorithm in Def. 14 on FCU problems results in FCU problems as well.

Lemma 5 *Given an FCU problem, then the application of rules from Def. 14 results in an FCU problem.*

Proof Removing equations from the system clearly preserves the restrictions of FCU problems. This result is also immediate when applying substitutions as the only change to the arguments of the variables in the problem is to eliminate them.

The following lemma states that projected arguments of variables on one side of the equation must always match arguments on the other side.

Lemma 6 *Let $X\overline{t_n} \doteq r$ be an equation such that r contains an occurrence of $Y\overline{s_m}$ where $r \neq Y\overline{s_m}$ and let σ be a unifier of this equation such that $\sigma Y = \lambda \overline{z_m}.s$. Then, for each bound variable z_i with a free occurrence in s for $0 < i \leq m$, there is $0 < j \leq n$ such that $s_i = t_j$.*

Proof We prove by induction on the number of bound variables with free occurrences. If s does not contain such a bound variable, then the lemma holds vacuously. Assume s contains a free occurrence of z_i for $0 < i \leq m$ and that there is no $0 < j \leq n$ such that $s_i = t_j$. In case there is more than one such occurrence in s, choose this occurrence to be in a minimal such subterm, i.e. z_i occurs in a subterm $z_i\overline{v_k}$ such that all occurrences of $z \in \overline{z_m}$ in $\overline{v_k}$ fulfill the requirement that there is $t_j = z$ for some $0 < j \leq n$. Let $(\lambda\overline{z_m}.z_i\overline{v_k})(\overline{s_m}) = s_i\overline{v'_k}$. Since $r \neq Y\overline{s_m}$ and the `argument restriction`, we have that $Y\overline{s_m} \sqsubset r$. Since $(X\overline{t_n})\sigma = r\sigma$, we get that $s_i\overline{v'_k} \sqsubset (X\overline{t_n})\sigma$. Since s_i is a restricted term, we get that there is $0 < j \leq n$ such that either

– $s_i\overline{v'_k} \sqsubseteq t_j$. By the minimality assumption, if $\overline{v'_k}$ contains a restricted term, then it must be equal to some t_l $(0 < l \leq n)$ and therefore, that $t_l \sqsubset t_j$, which contradicts the `local restriction`. Therefore, since t_j is a restricted term, $k = 0$. We obtain that $s_i \sqsubseteq t_j$ and since $s_i \neq t_j$ by assumption, we get, again, a contradiction to the `global restriction`.
– $t_j \sqsubseteq s_i$. Again, since $s_i \neq t_j$, we get a contraction to the `global restriction`.

We now prove, for each rule in the FCU algorithm, a relative completeness result. We start by the non-unifiability of problems with a positive occur check.

Lemma 7 *Let $\langle Q_\exists, Q_\forall, S \cup \{X\overline{t_n} \doteq f\overline{s_m}\}, \sigma\rangle$ be a system such that X occurs in $\overline{s_m}$, then the system is not unifiable.*

Proof Assume it is unifiable by θ and $\theta X = \lambda\overline{z_n}.s$. Consider two cases:

– s does not contain any occurrence of a variable z_i for $0 < i \leq n$. Let $\#t$ be the number of occurrences of symbols from $\mathfrak{C}$ in t. Then, $\#((X\overline{t_n})\theta) = \#(\theta X) \leq \#(\overline{s_m}\theta) < \#((f\overline{s_m})\theta)$ and we get a contradiction to $(X\overline{t_n})\theta = (f\overline{s_m})\theta$.
– In case s contains such an occurrence and let $X\overline{q_n}$ be the occurrence in $\overline{s_m}$. According to Lemma 6, we know that for all occurrences of z_i in s for $0 < i \leq n$, there is an index $0 < j \leq n$ such that $t_j = q_i$. Let ρ be the mapping between indices defined as above such that $\rho(i) = j$. Let $\overline{r_k}$ be the set of indices $0 < i \leq n$ which occur in s for some $k \leq n$. Let p' be the non-trivial position of $X\overline{q_n}$ in $f\overline{s_m}$ and let p be the maximal position of a z_i in s for $i \in \overline{r_k}$. This means we have q_i at position $p' \circ p$ in $(f\overline{s_m})\theta$ and since $t_{\rho(i)} = q_i$ and $(X\overline{t_n})\theta = (f\overline{s_m})\theta$, we get that $t_{\rho(i)}$ occurs at position $p' \circ p$ in $(X\overline{t_n})\theta$. Since $t_{\rho(i)}$ is a restricted term, there is an occurrence of $z_{\rho(i)}$ in s at position $p' \circ p$, in contradiction to the maximality of p.

Lemma 8 *Given a system $\langle Q_\exists, Q_\forall, S, \rho\}$ where $X\overline{t_n} \doteq r \in S$ and assuming we apply pruning in order to obtain system $\langle Q_\exists, Q_\forall, S', \rho'\}$ as defined in Def. 13, then, if S is unifiable by substitution σ, then there is a substitution σ', such that $\sigma = \theta \circ \sigma'$, for some substitution θ. If S is not unifiable, then S' is not unifiable.*

Proof Assume we apply pruning as in Def. 13. Let q be the restricted term appearing in the equation according to the definition. The rule is applicable only if there is such an occurrence q. Otherwise, $S = S'$ and $\theta =$ `id`. We consider the two cases in the lemma:

- there is a subterm $Y\overline{s_m}$ of r such that $q = s_i$ for $0 < i \leq m$. If S is not unifiable, then assume S' is unifiable by σ' and since $S' = S\theta$, we get that S is unifiable by $\theta \circ \sigma'$, a contradiction. Assume the system is unifiable and let $\sigma Y = \lambda\overline{z_m}.s$. Then, according to Lemma 6, either there is $0 < j \leq n$, such that $t_j = s_i$, which contradicts the assumption, or s does not contain an occurrence of s_i. In the second case, by taking $\sigma' = \sigma|_{\mathfrak{V}(S)\setminus\{Y\}}[W \mapsto \lambda\overline{z_{r_{m-1}}}.Y\overline{z_m}\sigma]$ where $\overline{z_m} \setminus \overline{z'_{m-1}} \not\subseteq Q_\forall$, we get that $Y\overline{s_m}\sigma = W\overline{s_{r_{m-1}}}\sigma' = Y\overline{s_m}\theta \circ \sigma'$.
- If there is no such cover, then there is no substitution which unifies this equation.

Lemma 9 *Given a system $\langle Q_\exists, Q_\forall, S, \rho\rangle$ where $X\overline{t_n} \doteq s \in S$ and X does not occur in s and assuming we apply the substitution θ as defined in rule (3) in Table 3. Then, if σ is a unifier of S, then there is σ' such that $\sigma = \theta \circ \sigma'$.*

Proof We prove by induction on the structure of s. Note that two base cases are also defined in the last two cases below for $m = 0$.

- $s = Y\overline{s_m}$ for $0 \leq m$. In this case, mutual pruning will ensure that we have $X\overline{t_k} = Y\overline{t_{\phi(k)}}$ for ϕ defined as in rule (5) in Table 3 and $k \leq n, m$. The rest of the proof is similar to the proof of Lemma 11.
- $s = t_i\overline{s_m}$ for some $0 < i \leq n$ and $0 \leq m$ and therefore $\theta X = \lambda\overline{z_n}.z_i\overline{s'_m}$ for some $\overline{s'_m}$. Assume applying `(Project)` with the substitution $\theta' = \lambda\overline{z_n}.z_i\overline{X_m\overline{z_n}}$ as defined in Def. 4. After applying `(Bind)` and possibly also several `(Simpl)`, we get the problem, since X cannot occur in s,

$$S'\theta' \cup \{X_1\overline{t_n} \doteq s_1, \ldots, X_m\overline{t_n} \doteq s_m\} \tag{8}$$

  Since, by assumption, $X\overline{t_n} \doteq t_i\overline{s_m}$ is unifiable and t_i is a restricted term, it follows, using an argument similar to the one in the proof of Lemma 3, that also each of the $X_j\overline{t_n} \doteq s_j$ is unifiable by some σ^o_j for $0 < j \leq m$. By following lemmas 3 and 4, we have that applying any other projection or imitation to $X\overline{t_n} \doteq t_i\overline{s_m}$ will render it non-unifiable. By using Corollary 1, we have that σ^o_j can be extended into a unifier σ_j of $S'\theta' \cup \{X_j\overline{t_n} \doteq s_j\}$ for all $0 < j \leq m$ and $\sigma = \theta' \circ \sigma_1 \circ \ldots \circ \sigma_m$. By applying the induction hypothesis, we get that there are substitutions σ'_j unifying $S'\theta' \cup \{X_j\overline{t_n} \doteq s_j\}$ for all $0 < j \leq m$ such that $\sigma_j = \theta_j \circ \sigma'_j$ for $\theta_j X_j = \lambda\overline{z_n}.s'_j$. I.e. that $\sigma = \theta' \circ \theta_1 \circ \sigma'_1 \circ \ldots \circ \theta_m \circ \sigma'_m$. Since each σ_j is a unifier of the above equations and σ is a unifier of S, we get that for each $m \geq j' > j$, $\sigma'_{j'} \leq \sigma'_j|_{\mathtt{dom}(\sigma'_{j'})}$. From this, together with the fact that the domain and range of each σ'_j do not contain variables from the domain of each $\theta_{j'}$ for $0 < j < j' \leq m$, we get that $\sigma = \theta' \circ \theta_1 \circ \ldots \circ \theta_m \circ \sigma'_1 \circ \ldots \circ \sigma'_m$. On the other hand, by applying θ, we get the problem

$$S'\theta \cup \{s'_1 \doteq s_1, \ldots, s'_m \doteq s_m\} \tag{9}$$

  But, since $\theta = \theta' \circ \theta_1 \circ \ldots \circ \theta_m$, we just need to choose $\sigma' = \sigma'_1 \circ \ldots \circ \sigma'_m$ and we have $\sigma = \theta \circ \sigma'$.

– $s = f\overline{s_m}$ for $0 \leq m$ and therefore $\theta X = \lambda\overline{z_n}.f\overline{s'_m}$. Assume applying `(Imitate)` with the substitution $\theta' = \lambda\overline{z_n}.f\overline{X_m\overline{z_n}}$. After applying `(Bind)` and a `(Simpl)`, we get the problem, since x cannot occur in s,

$$S'\theta' \cup \{X_1\overline{t_n} \doteq s_1, \ldots, X_m\overline{t_n} \doteq s_m\} \tag{10}$$

From here we follow as before and use again Corollary 1 and Lemma 4.

Lemma 10 *Given a system $\langle Q_\exists, Q_\forall, S' \cup \{X\overline{t_n} \doteq X\overline{s_n}\}, \rho\rangle$ and assuming we apply the substitution θ as defined in rule (4) in Table 3. Then, if the system is unifiable by a substitution σ, then there is σ' such that $\sigma = \theta \circ \sigma'$.*

Proof Assume that $\sigma X = \lambda\overline{z_n}.s$, we first prove that there is no occurrence z_i in s such that there is $0 < j \leq k$ where $i = r_j$ and $t_i \neq s_i$. Assume on the contrary, then $(X\overline{t_n})\sigma = (X\overline{s_n})\sigma$ which implies that $t_i = s_i$. Now we can define $\sigma' = \sigma|_{\mathfrak{V}(S)\setminus\{X\}}[W \mapsto \lambda\overline{z_{r_k}}.\sigma X\overline{z_n}]$ where $\overline{z_n} \setminus \overline{z_{r_k}} \not\subseteq Q_\forall$ are new variables.

Lemma 11 *Given a system $\langle Q_\exists, Q_\forall, S' \cup \{X\overline{t_n} \doteq Y\overline{s_m}\}, \rho\rangle$ where $X \neq Y$ and assuming we apply the substitution θ as defined in rule (5) in Table 3. Then, if the system is unifiable by a substitution σ, then there is σ' such that $\sigma = \theta \circ \sigma'$.*

Proof First note that since we apply pruning beforehand (in a symmetric way), $n = m$ and ϕ is indeed a permutation. Assume, without loss of generality, that $\sigma X = \lambda\overline{z_n}.s$. We know that for each occurrence z_i in s for $0 < i \leq n$, $s_i = t_{\phi(i)}$. By choosing $\sigma' = \sigma|_{\mathfrak{V}(S)\setminus\{Y\}}$, we get that $(Y\overline{s_m})\sigma = (X\overline{s_{\phi(m)}})\sigma = (X\overline{s_{\phi(m)}})\sigma' = (Y\overline{s_m})(\theta \circ \sigma')$. Therefore, $\sigma = \theta \circ \sigma'$.

Theorem 3 (Termination) *Given a system $\langle Q_\exists, Q_\forall, S, \sigma\rangle$, the algorithm in Def. 14 always terminates.*

Proof Let the tuple $m = \langle m_1, m_2\rangle$ where m_1 is the size of the set $Q_\exists$ and m_2 is the number of all symbols except $\doteq$ in S. Consider its lexicographical order, it is clear that m is well founded. We show that it decreases with every rule application of the algorithm:

– rules (0), (1) and (2) decrease m_2 and do no increase m_1.
– rule (3) decreases m_1.
– rule (4) decreases m_2 and does not increase m_1.
– rule (5) decreases m_1.
– pruning decreases m_2 and does not increase m_1.

Theorem 4 (Completeness) *Let $\langle Q_\exists, Q_\forall, S, \boldsymbol{id}\rangle$ be a system with unifier σ. There is a sequence of rule applications in Def. 14 resulting in $\langle Q'_\exists, Q'_\forall, \emptyset, \theta\rangle$ such that $\theta \leq \sigma$.*

Proof Since the algorithm terminates and since it has a rule application for each unifiable equation, we obtain at the end the above system. Lemmas 7, 8, 9, 10 and 11 then give us that for any unifier σ of S, there is a substitution σ' such that $\sigma = \theta \circ \sigma'$. Therefore, $\theta \leq \sigma$.

The next theorem is an easy corollary of the completeness theorem.

Theorem 5 (Most-general unifier) *Given a system $\langle Q_\exists, Q_\forall, S, \boldsymbol{id}\rangle$, if the algorithm defined in Def. 14 terminates with system $\langle Q'_\exists, Q'_\forall, \emptyset, \sigma\rangle$, then σ is an mgu of S.*

Proof Since the algorithm in Def. 14 is deterministic, then we can use Theorem 4 in order to prove that σ is an mgu.

The next theorem is proved by simulating the algorithm in Def. 14 using the procedure in Def. 4.

Theorem 6 (Soundness) *Given a system $\langle Q_\exists, Q_\forall, S, \boldsymbol{id}\rangle$ and assuming there is a sequence of rule applications in Def. 14 resulting in $\langle Q'_\exists, Q'_\forall, \emptyset, \theta\rangle$, then θ is a unifier of S.*

Proof It is obvious we can simulate each of the rules (0), (1), (2) and (3) using the procedure in Def. 4. We get the required result by using Theorem 5. For rules (4), (5) and the first case of pruning, assume there is another substitution ρ such that ρ unifies the problem and $\rho \not\leq \theta$. This can only happen if $\rho X = \lambda\overline{z_n}.W'\overline{r_{k'}}$ such that $\overline{r_k} \subseteq \overline{r'_{k'}}$. Lemma 10 states that there is no unifier ω and a substitution γ such that $\omega = \rho \circ \gamma$. Since the second case of pruning results in failure, we are done.

5 Discussion and Extensions of FC Unification

Hamana [19] has remarked on the strong similarity between FCU and DSP problems, when restricted to the second-order case. In this section we discuss this similarity and compare the two fragments. This comparison allows us to discuss possible extensions to FCU which are strictly stronger than the one defined in Sec. 3. We will also discuss the developments with regard to FCU which have taken place in the last 4 years (2016-2020).

5.1 A Comparison of FCU and DSP

FCU and DSP differ on the following points:

1. FCU problems are unification problems while DSP are matching problems.
2. FCU problems are higher-order while DSP are restricted to second-order.
3. The set of DSP restricted terms strictly contains the set of second-order (FCU) restricted terms.
4. DSP matching problems are deterministic without the `global restriction` defined for FCU systems, while FCU problems are not.

When considering these 4 points, it seems appropriate to compare the two fragments over the following 2 properties:

1. Matching vs. unification

2. Second-order vs. higher-order terms

In fact, the first point was already discussed by Hamana in [19]. According to which, Yokoyama, Hu and Takeichi have given an example of a unification problem over DSP terms which admits two incompatible unifiers [48][2]

Example 11 Let $\mathfrak{C} = \{\texttt{fst}\}$ and $Q_\forall = \{l_1, l_2\}$. The DSP problem

$$\{X\ (\texttt{fst}\ l_1)\ (\texttt{fst}\ l_2) \doteq \texttt{fst}\ (Y\ l_2\ l_1)\}$$

admits the two incompatible unifiers $[Y \mapsto \lambda z_1, z_2.z_1, X \mapsto \lambda z_1, z_2.z_2]$ and $[Y \mapsto \lambda z_1, z_2.z_2, X \mapsto \lambda z_1, z_2.z_1]$.

This example is not an FCU problem since the arguments of Y are strict subterms of those of X, which violates the `global restriction`. We will further discuss an extension of FCU to the second-order case in the next section.

We will come back to the question whether this condition is the only one required when moving from matching problems to unification ones. We can now shift our attention to the second point of comparison – second-order vs. higher-order terms – and note our strictly more restrictive definition of terms (see Def. 10) than DSP terms (see Def. 8).

Proposition 7 *A second-order (FCU) restricted term is a DSP restricted term but not vice versa.*

Proof Let t be a second-order (FCU) restricted term. According to the inductive definition in Def. 10, there is at least one position p such that $t|_p \in Q_\forall$, which is exactly the requirement of DSP terms. Conversely, the following is a DSP term but not a second-order (FCU) term $(\texttt{cons}\ z\ \texttt{nil})$.

Can DSP terms be extended to higher-order? Yokoyama et at. have extended their fragment to higher-order matching [47]. Their fragment has the same restrictions as in the second-order case and depends on the standard higher-order definition of the subterm relation. [3]

Example 12 Let $\mathfrak{C} = \{\texttt{nil}\}$ and $Q_\forall = \{\texttt{fun}\}$. The third-order problem

$$\{X\ (\texttt{fun}\ \texttt{nil})\ \texttt{fun} \quad \doteq (\texttt{fun}\ \texttt{nil})\},$$

admits the incompatible matchers $[X \mapsto \lambda z_1, z_2.z_1]$ and $[X \mapsto \lambda z_1, z_2.(z_2\ \texttt{nil})]$. This problem, however, is not in the higher-order DSP fragment, since the second argument of X is a subterm of the first one.

[2] We could not find the example in the English version of the paper and refer to the one mentioned by Hamana in [19].

[3] Hamana, in private discussions, has described another class discussed by Yokoyama et al. [48], which is restricted to linear second-order terms but is more permissive with regard to the arguments of second-order variables.

In the introduction to this paper, we have noted three properties which are admitted by the pattern unification algorithm and which are desirable in extensions of it: decidability, determinism and type-freeness. In [46], only the first two requirements were desired. This is, of course, appropriate since type-freeness plays no role in second-order terms. Clearly, since our restricted terms do not admit abstractions in the arguments of higher-order variables, type-freeness makes sense only if we do not restrict those arguments to be in η-extended form. In our presentation, we require those arguments to be, in fact, in η-normal forms (as are the arguments in [47]). In a similar way to [47], the above example is also not an FCU problem, since we have the same subterm restriction.

This example might hint that DSP problems can be used for unification, if the additional `global restriction` is added. The following example shows that this is not the case, already in second-order.

Example 13 Let $\mathfrak{C} = \{\mathtt{nil}_0, \mathtt{nil}_1, \mathtt{cons}\}$ and $Q_\forall = \{w\}$. The second-order problem

$$\{X\ (\mathtt{cons}\ w\ \ \mathtt{nil}_0)\ (\mathtt{cons}\ w\ \ \mathtt{nil}_1))\ \doteq (\mathtt{cons}\ w\ \ Y))\},$$

admits the two incompatible unifiers $[X \mapsto \lambda z_1, z_2.z_1, Y \mapsto \mathtt{nil}_0]$ and $[X \mapsto \lambda z_1, z_2.z_2, Y \mapsto \mathtt{nil}_1]$.

We now discuss an extension to FCU which is based on the observations in this section.

5.2 Second-order FCU

Previously we have discussed Hamana's second-order fragment [19], which restricts FCU to the second-order case. One might conjecture that we can extend second-order FCU to DSP restricted terms. The unification problem in Example 13 involves DSP terms which are not FCU, since it contains constant symbols in leaf positions. Since this problem is not deterministic, it serves as a counterexample to this conjecture.

This example shows that as long as restricted arguments allow for subterms not containing bound variables, such as $\mathtt{nil}_1$ and $\mathtt{nil}_2$ in the above example, these arguments can be projected and determinism might be lost. One can now ask if the FCU restricted terms can be relaxed in the second-order setting in another way. In fact, the above example is prevented if we add the following further restriction

Definition 15 (local and global restrictions) Let the abstract projection operator (*apo*) be defined as following. Given a restricted term t, $apo(t)$ replaces all subterms not containing a bound variable (from $Q_\forall \cup \mathtt{bvars}$) with the new constant ϵ. Then, `local restriction` and `global restriction` are defined not over the restricted terms but over their abstract projections. We define problems of this variant of FCU, restricted to the second-order case, as FC^2 problems.

Example 14 Example 13 is not an FC^2 unification problem since the abstract projections of the arguments of X, $(\texttt{cons}\ w\ \texttt{nil}_0)$ and $(\texttt{cons}\ w\ \texttt{nil}_1)$, are both $(\texttt{cons}\ w\ \epsilon)$ and violate the `local restriction`.

Does this fragment strictly contain the FCU one?

Proposition 1 *A second-order (FCU) restricted terms is a FC^2 restricted term but not vice versa.*

Proof Consider the following problem, which is FC^2 but not FCU. Let $\mathfrak{C} = \{\texttt{nil}_0, \texttt{cons}\}$ and $Q_\forall = \{w\}$, The second-order problem $\{X\ (\texttt{cons}\ w\ \texttt{nil}_0)) \doteq (\texttt{cons}\ w\ Y))\}$ is an FC^2 problem but not FCU, since there is a constant at a leaf position.

We can now move forward and prove that the unification problem of FC^2 terms is deterministic. The key to this argument is the observation as to why restricted terms play a role in ensuring the determinism of unification, which can be found in the proof of lemmas 3, 4 and the auxiliary lemmas.

Proof (Adaptation of Lem. 3 to FC^2 terms) In the second-order case, the proof is much simpler. Each of the projections projects first order terms, so equations 1 and 2 become

$$t_i \doteq f\overline{s_m} \tag{11}$$

$$t_j \doteq f\overline{s_m} \tag{12}$$

where t_i and t_j are not subterms of each other, do not contain free variables and each contain at least one variable from $Q_\forall$. We proceed by finding a contradiction. Let p be the first position in the abstract projections of t_i and t_j such that the symbol at this position differs between the two terms or equal to ϵ. Clearly, this means that $f\overline{s_m}|_p$ must be a variable $Y\overline{v_k}$ where $k \geq 0$, We prove by considering different cases

- $t_i|_p$ (or $t_j|_p$) contains a variable from $Q_\forall \cup \texttt{bvars}$. In this case, we get a violation to `global restriction` in a similar way to the one in in the proof of Lem. 3.
- Otherwise, both positions must be equal to ϵ and we get a violation of `local restriction`.

Proof (Adaptation of Lem. 4 to FC^2 terms) This proof is also simpler in the second-order case, since the equations 5 and 6 become

$$f\overline{X_m\overline{t_n}} \doteq f\overline{s_m} \tag{13}$$

$$t_j \doteq f\overline{s_m} \tag{14}$$

Since t_j does not contain free variables, it can be written as

$$f\overline{v_m} \doteq f\overline{s_m} \tag{15}$$

Since, t_j is a restricted term, there is $0 < l \leq m$ such that v_l contains a bound variable. We again proceed by considering different cases.

Fragment	SO matching	HO matching	SO unification	HO unification
DSP	✓	✓	X	X
FC^2	✓	✓	✓	?
FC	✓	✓	✓	✓

Table 6 A summary of the properties of the three fragments discussed in this section

- s_l is ground. We obtain that there is $0 < k \leq n$ such that t_k is a subterm of t_j, in a similar way to the proof of Lem. 4, which violates the `local restriction`.
- Otherwise, s_l contains a variable and we get again a violation to `global restriction` in a similar way to the proof of Lem. 4.

We leave proving this extension to HO FCU as a future work.

We have already proved, in this section, that the DSP fragment is strictly bigger than the FCU^2 one, which is strictly bigger than the second-order FCU one. Table 6 summarizes the determinism results of this section, as well as of [46] and [48]. DSP corresponds to the fragment defined by Yokoyama et al., FC is the functions as constructors fragment discussed in this paper while FC^2 is the extension of the restricted terms discussed in this section. The ✓ symbol denotes that the specific matching/unification problem is deterministic and terminating over the specific fragment, while 'X' denotes the existence of a counter example. If the answer is unknown, the corresponding cell is denoted by '?'. 'HO' and 'SO' denote higher-order and second-order, respectively. In order to clarify further, we stress that the FC^2 restriction is more general than that of FCU and we have therefore managed to show its termination and determinism for the second-order case only.

6 Conclusion

We have described an extension of pattern unification called *function-as-constructor* unification. Such unification problems typically show up in situations where functions are applied to bound variables and where such functions are treated as term constructors (at least during the process of unification). We have shown that the properties that make first-order and pattern unification desirable for implementation—decidability and the existence of mgus for unifiable pairs—also holds for this class of unification problems.

The current trend in higher-order theorem proving is to apply deterministic unification algorithms as often as possible to help reduce the search space. To the best of our knowledge Hamana's system SOL [21] is the only system that has implemented the algorithm described in this paper [20].

The need to check for the subterm relation between different arguments of variables makes this algorithm less efficient than pattern unification. Nevertheless, the possibility to preserve completeness while using a deterministic algorithm is advantageous. One place where such an approach is essential is when one is looking for a complete set of higher-order unifiers, instead of pre-unifiers. One of the systems which have taken this approach is the Zipperposition theorem prover [45], where FCU is mentioned as a desirable "oracle" for solving fragments.

Another possible extension of the work is to improve its complexity. The current algorithm, like the one in [27] and the first-order unification algorithms which are used in practice, is of exponential complexity. We want to follow the work of Qian [44] and prove that FCU is of linear complexity.

An interesting question is how the algorithm presented in this paper compares to other higher-order unification algorithms, such as the one in [45], when operating on the FC fragment. Clearly, such a comparison is unfair to the more general algorithms, which, as we have seen above, actually use dedicated unification algorithms to tackle decidable fragments. One thing which stands out, though, is the application of pruning, which is deterministic in our algorithm and is non-deterministic in, for example, [45].

Acknowledgments. We thank Enrico Tassi, Tetsuo Yokoyama and Makoto Hamana for discussions related to this paper. We benefited greatly from the many comments of the anonymous reviewers of an earlier draft of this paper. This work has been funded by the ERC Advanced Grant ProofCert.

References

1. The Twelf project, 2016. http://twelf.org/.
2. Andreas Abel and Brigitte Pientka. Higher-order dynamic pattern unification for dependent types and records. In *Typed Lambda Calculi and Applications*, pages 10–26. Springer, 2011.
3. P. B. Andrews, F. Pfenning, S. Issar, and C. P. Klapper. The TPS theorem proving system. In Jörg H. Siekmann, editor, CADE 8, LNCS 230, pages 663–664. Springer, July 1986.
4. Andrea Asperti, Claudio Sacerdoti Coen, Enrico Tassi, and Stefano Zacchiroli. Crafting a proof assistant. In *Types for Proofs and Programs*, pages 18–32. Springer, 2006.
5. Andrea Asperti, Wilmer Ricciotti, Claudio Sacerdoti Coen, and Enrico Tassi. Hints in unification. In *International Conference on Theorem Proving in Higher Order Logics*, pages 84-98. Springer, Berlin, Heidelberg, 2009.
6. D. Baelde, K. Chaudhuri, A. Gacek, D. Miller, G. Nadathur, A. Tiu, and Y. Wang. Abella: A system for reasoning about relational specifications. *J. of Formalized Reasoning*, 2014.
7. Bhayat, Ahmed, and Giles Reger. Restricted combinatory unification. International Conference on Automated Deduction. Springer, Cham, 2019.
8. Henk Barendregt. *The Lambda Calculus: Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. Elsevier, New York, revised edition, 1984.
9. Christoph Benzmüller and Michael Kohlhase. LEO – a higher order theorem prover. In *15th Conf. on Automated Deduction (CADE)*, pages 139–144, 1998.
10. Yves Bertot, Georges Gonthier, Sidi Ould Biha, Ioana Pasca. Canonical Big Operators. In Theorem Proving in Higher Order Logics, Aug 2008, Montreal, Canada.
11. Ana Bove, Peter Dybjer, and Ulf Norell. A brief overview of Agda - A functional language with dependent types. In *TPHOLs*, volume 5674, pages 73–78. Springer, 2009.
12. Chad E Brown. Satallax: An automatic higher-order prover. In *Automated Reasoning*, pages 111–117. Springer, 2012.
13. A. Church. A formulation of the Simple Theory of Types. *J. of Symbolic Logic*, 1940.
14. Mary Dalrymple, Stuart M. Shieber, and Fernando C. N. Pereira. Ellipsis and higher-order unification. *Linguistics and Philosophy*, 14:399–452, 1991.

15. Dominic Duggan. Unification with extended patterns. *Theoretical Computer Science*, 206(1):1–50, 1998.
16. R. Fettig and B. Löchner. Unification of higher-order patterns in a simply typed lambda-calculus with finite products and terminal type. In RTA 1996, LNCS 1103, pages 347–361.
17. Warren Goldfarb. The undecidability of the second-order unification problem. *Theoretical Computer Science*, 13:225–230, 1981.
18. Guidi, Ferruccio, Claudio Sacerdoti Coen, and Enrico Tassi. Implementing type theory in higher order constraint logic programming. Mathematical Structures in Computer Science 29.8 (2019): 1125-1150.
19. Hamana, Makoto. How to prove your calculus is decidable: practical applications of second-order algebraic theories and computation. Proceedings of the *ACM on Programming Languages* 1.ICFP (2017): 1-28.
20. Hamana, Makoto. A functional implementation of function-as-constructor higher-order unification. Proc. 31st International Workshop on Unification (UNIF'17). 2017.
21. Hamana, M., Abe, T., Murase, Y., and Sakaguchi, K. The System SOL: Second-Order Laboratory. In 6th International Workshop on Confluence
22. Gérard Huet. The undecidability of unification in third order logic. *Information and Control*, 22:257–267, 1973.
23. Gérard Huet. A unification algorithm for typed λ-calculus. *Theoretical Computer Science*, 1:27–57, 1975.
24. Gérard Huet and Bernard Lang. Proving and applying program transformations expressed with second-order patterns. *Acta Informatica*, 11:31–55, 1978.
25. Tomer Libal and Dale Miller. Functions-as-constructors higher-order unification. In *Proceedings of the 1st International Conference on Formal Structures for Computation and Deduction*, 2016.
26. Raymond McDowell and Dale Miller. Reasoning with higher-order abstract syntax in a logical framework. *ACM Trans. on Computational Logic*, 3(1):80–136, 2002.
27. Dale Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *J. of Logic and Computation*, 1(4):497–536, 1991.
28. Dale Miller and Gopalan Nadathur. Some uses of higher-order logic in computational linguistics. In *Proceedings of the 24th Annual Meeting of the Association for Computational Linguistics*, pages 247–255, 1986.
29. Dale Miller. Unification under a mixed prefix. *Journal of Symbolic Computation*, 14(4):321–358, 1992.
30. Dale Miller and Gopalan Nadathur. *Programming with Higher-Order Logic*. Cambridge University Press, June 2012.
31. Dale Miller. *Proof checking and logic programming. Formal Aspects of Computing*, 29(3):383–399, 2017.
32. Gopalan Nadathur and Dustin J. Mitchell. System description: Teyjus — A compiler and abstract machine based implementation of λProlog. CADE 16, LNAI 1632, pp. 287–291, 1999.
33. Nagele, Julian. Mechanizing Confluence. Diss. PhD thesis, University of Innsbruck, 2017.
34. Tobias Nipkow. Functional unification of higher-order patterns. In M. Vardi, editor, *8th Symp. on Logic in Computer Science*, pages 64–74. IEEE, June 1993.
35. Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*. Number 2283 in LNCS. Springer, 2002.
36. Frank Pfenning. Unification and anti-unification in the Calculus of Constructions. In G. Kahn, editor, *6th Symp. on Logic in Computer Science*, pages 74–85. IEEE, July 1991.
37. Frank Pfenning and Carsten Schürmann. System description: Twelf — A meta-logical framework for deductive systems. CADE 16, LNAI 1632, pages 202–206, Trento, 1999.
38. Brigitte Pientka and Joshua Dunfield. Beluga: A framework for programming and reasoning with deductive systems (system description). In J. Giesl and R. Hähnle, editors, IJCAR, LNCS 6173, pages 15–21, 2010.
39. Xiaochu Qi, Andrew Gacek, Steven Holte, Gopalan Nadathur, and Zach Snow. The Teyjus system – version 2, 2015. http://teyjus.cs.umn.edu/.
40. Helmut Schwichtenberg. Minlog. In Freek Wiedijk, editor, *The Seventeen Provers of the World*, volume 3600 of *LNCS*, pages 151–157. Springer, 2006.

41. Wayne Snyder and Jean H. Gallier. Higher order unification revisited: Complete sets of transformations. *Journal of Symbolic Computation*, 8(1-2):101–140, 1989.
42. Enrico Tassi. Private communication. 2016-2021.
43. Alwen F. Tiu. An extension of L-lambda unification. http://www.ntu.edu.sg/home/atiu/llambdaext.pdf, September 2002.
44. Zhenyu Qian. Linear Unification of Higher-Order Patterns. Proc. Coll. Trees in Algebra and Programming, 1993.
45. Vukmirović, Petar, Alexander Bentkamp, and Visa Nummelin. Efficient full higher-order unification. 5th International Conference on Formal Structures for Computation and Deduction (FSCD 2020). Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.
46. Yokoyama, Tetsuo, Zhenjiang Hu, and Masato Takeichi. "Deterministic second-order patterns." *Information Processing Letters* 89.6 (2004): 309-314.
47. Yokoyama, Tetsuo, Zhenjiang Hu, and Masato Takeichi. "Deterministic Higher-Order Patterns for Program Transformation" *International Symposium on Logic-Based Program Synthesis and Transformation.* Springer, Berlin, Heidelberg, 2004.
48. Yokoyama, Tetsuo, Zhenjiang Hu, and Masato Takeichi. "Deterministic second-order patterns for program transformation." *Computer Software*, 21(5):71–76, 2004. In Japanese.
49. Beta Ziliani and Matthieu Sozeau. A unification algorithm for Coq featuring universe polymorphism and overloading. ICFP 2015, pp. 179–191.