A Colored Version of the λ-Calculus

Dieter Hutter[1]* and Michael Kohlhase[2]**

[1] German Research Center for Artificial Intelligence, Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany, hutter@dfki.uni-sb.de

[2] University of Saarland, Fachbereich Informatik, D-66041 Saarbrücken, Germany, kohlhase@cs.uni-sb.de

Abstract. Rippling is a technique developed for inductive theorem proving which uses syntactic differences of terms to guide the proof search. Annotations (like colors) to terms are used to maintain this information. This technique has several advantages, e.g. it is highly goal oriented and involves little search. In this paper we give a general formalisation of coloring terms in a higher-order setting. We introduce a simply-typed lambda calculus with color annotations and present algorithms for unification, pre-unification and pattern unification. Our work is a formal basis to the implementation of rippling in a higher-order setting which is required e.g. in case of middle-out reasoning.
Another application is in the construction of natural language semantics, where the color annotations rule out linguistically invalid readings that are possible using standard higher-order unification.

1 Introduction

In the field of inductive theorem proving syntactical differences between the induction hypothesis and induction conclusion are used to guide the proof search (cf. [Bun88,BSvH$^+$93], or [Hut90]). This method to guide induction proofs is called rippling / coloring terms. Annotations or colors to each occurrence of a symbol are used to mark the syntactical differences between induction hypothesis and induction conclusion. Specific colors denote the *skeleton*, the common parts of both formulas, while the other parts belong to the *wave-fronts*. Analogously, syntactical differences between both sides of equations or implications given in the database are colored. These formulas are classified depending on the locations of the wave-fronts inside the skeleton (e.g. wave-fronts on both sides, wave-fronts only on the right-hand side, or wave-fronts only on the left-hand side). Using these annotated (or colored) equations we are able to move, insert, or delete wave-fronts within the conclusion. This *rippling* of wave-fronts allows to reduce the differences between conclusion and hypothesis in a goal directed way.

This paper extends the coloring method to higher-order logic and presents unification, pre-unification, and pattern unification algorithms. Thus our work provides a formal basis to the implementation of rippling in a higher-order setting which is required e.g. in case of middle-out reasoning [Hes91] or generalization of

* This work was supported by the German BMBF in project PADS

** This work was supported by the DFG in project HOTEL

theorems using proof critics [IB96]. In the latter the unknown generalized version of a formula is described by a pattern containing parts of the original formula and higher-order variables denoting the unknown syntactical extensions of it. Simulating the induction proof the higher-order variables will be instantiated step by step by the unification with appropriate wave-rules resulting in a possible (hopefully provable) generalization of the original formula.

But the set of possible applications of our method is not limited to automated deduction. From an abstract point of view, the coloring technique allows adding annotations to symbol occurrences in λ-terms. Thus in contrast to other semantic annotation techniques like sorts, it is possible to encode syntactic information and use that to guide inferencing processes. In [GK96,Gar] applications of our technique in computational linguistics and natural language semantics are described, where the use of higher-order unification is used to construct the values of elliptical references in the context of Montegovian semantics.

An extended version of this paper containing all formal definitions and proofs is published as SEKI-report [HK95].

2 Colored λ-Terms

Since the formal development of the theory involves quite a lot of technical machinery which we cannot present fully here, concentrate on the intuitions and motivations and refer the reader to the supporting documentation for details and proofs.

The colored λ-calculus is a variant of the simply typed λ-calculus [Chu40], where symbol occurrences can be annotated with so-called *colors* (**color constants** $\mathcal{C} = \{\mathtt{a}, \mathtt{b}, \ldots\}$ and **color variables** $\mathcal{X} = \{\mathtt{A}, \mathtt{B}, \ldots\}$; whenever colors are irrelevant, we simply omit them; colors are indicated by subscripts labeling symbol occurrences).

The set wff_α of **well-formed formulae of type**[3] α consists of

– (colored) constants $c^\alpha_{\mathtt{b}}, f^\alpha_{\mathtt{a}}, f^\alpha_{\mathtt{A}}, \ldots$,
– (colored or uncolored) variables $X^\alpha_{\mathtt{b}}, G^\alpha_{\mathtt{a}}, F^\alpha \ldots$ (of which we assume an infinite supply for each type and color) of type α,
– (function) **applications** of the form $\mathbf{M}^{\beta\rightarrow\alpha}\mathbf{N}^\beta$ and
– **λ-abstractions** of the form $\lambda X^\beta.\mathbf{M}^\gamma$, if $\alpha = (\beta \rightarrow \gamma)$. Note that variables occur only without annotated colors in wff_α iff they are bound variables, thus only variables without colors can be abstracted over.

We call a formula $\mathbf{M}$ **$\mathtt{c}$-monochrome**, if all symbols (except bound variables) in $\mathbf{M}$ are annotated by a single color $\mathtt{c}$.

Clearly the colored λ-calculus is a generalization of the simply typed λ-calculus, since we can always restrict the supply of colors to a single color constant. Therefore we will use various elementary concepts of the λ-calculus, such as **free** and **bound** occurrences of variables or substitutions without defining

[3] Since for the purposes of this informal introduction types only play a theoretical role (they for instance make $\beta\eta$-reduction terminating and therefore $\beta\eta$-equality decidable), they are often omitted from the examples.

them explicitly here. We will denote the substitution of a term $\mathbf{N}$ for all free occurrences of X in $\mathbf{M}$ with $[\mathbf{N}/X]\mathbf{M}$.

It is crucial for our system that colors annotate symbol occurrences (i.e. colors are not sorts!), in particular, it is intended that different occurrences of symbols carry different colors (e.g. $f(X_{\mathtt{a}}, X_{\mathtt{b}})$) and that symbols that carry different colors are treated differently. This observation leads to the notion of colored substitutions, a notion of substitution that takes the color information of formulae into account. In contrast to traditional (uncolored) substitutions, a colored substitution σ is a pair $\langle\sigma^t, \sigma^c\rangle$, where the **term substitution** σ^t maps colored variables (i.e. the pair $X_{\mathtt{c}}$ of a variable X and the color c) to formulae of appropriate types and the **color substitution** σ^c maps color variables to colors. In order to be a legal **$\mathcal{C}$-substitution** such a mapping σ must obey the following constraints:

– If $\mathtt{A}$ and $\mathtt{B}$ are different colors, then $|\sigma(X_{\mathtt{A}})| = |\sigma(X_{\mathtt{B}})|$, where $|\mathbf{M}|$ is the **color erasure** of $\mathbf{M}$, i.e. the formula obtained from $\mathbf{M}$ by erasing all color annotations in $\mathbf{M}$.
– If $\mathtt{c} \in \mathcal{C}$ is a color constant, then $\sigma(X_{\mathtt{c}})$ is $\mathtt{c}$-monochrome.

The first condition ensures that the color erasure of a $\mathcal{C}$-substitution is a classical substitution of the simply typed λ-calculus. The second condition formalizes the fact that free variables with constant colors stand for monochrome subformulae, whereas variable colors do not constrain the substitutions.

Note that since the bound variables do not carry color information, $\beta\eta$-reduction in the colored λ-calculus is just the classical notion. Thus we can lift all the known theoretical results to the colored calculus

Higher-order unification computes substitutions σ such that $\sigma(\mathbf{M}) =_{\beta\eta} \sigma(\mathbf{N})$ for a given equation $\mathbf{M} = \mathbf{N}$. In the colored λ-calculus the space of (semantic) solutions is further constrained by requiring the solutions to be $\mathcal{C}$-substitutions. Such a substitution is called a $\mathcal{C}$-unifier of $\mathbf{M}$ and $\mathbf{N}$. Even with the color restriction, the set of $\mathcal{C}$-unifiers of a given equation is enormous. Furthermore, most of these solutions introduce un-necessary instantiations; thus one is not interested in the set of *all* $\mathcal{C}$-unifiers, but rather in a subset that generates this set by instantiation. A substitution σ is called **more general** than τ, iff there is a substitution ρ, such that $\tau =_{\beta\eta} \rho \circ \sigma$, i.e. τ can be reconstructed from σ by instantiation with ρ.

3 Applications of a Colored Lambda Calculus

In this section we will present different kinds of application of higher-order colored unification. These consist in rippling in a higher-order setting which is required e.g. in case of middle-out reasoning [Hes91,IB96] and also in a logical basis for an interface for linguistic extra-semantical information in the construction of natural-language semantics. Since this paper is mainly concerned with automated deduction, we will only concentrate on the rippling aspects and refer the interested reader to [GK96,Gar].

3.1 Inductive Proofs

Rippling was developed for proving theorems by induction and has been applied to a large number of practical examples from this domain [Bun88,BSvH$^+$93,Hut90]. It is based on an observation that one can iteratively unfold recursive functions in the induction conclusion, preserving the structure of the induction hypothesis while unfolding. We use colors in order to indicate the structure of the hypothesis within the conclusion. Symbols belonging to this joined structure are annotated with the color "white" while differences between both formulas are colored "grey". Also left- and right-hand sides of given equations are difference unified: the common structure of both terms of a given equation is annotated by color variables while differences are colored grey. The grey parts are called *wave-fronts* while the non-grey parts denote the *skeleton*. Rippling restricts the search space in a way that only deduction steps are allowed which preserve the skeleton, i.e. do not change the non-grey parts of the formula.

For sake of simplicity we use a shading for symbols which are annotated by the color grey while non-shaded areas are annotated either by white or color variables. But in case the distinction between color variables and white is necessary we shall annotate the colors explicitly.

Rippling then applies just the annotated equations which move the difference out of the way leaving behind the skeleton. In their simplest form, these equations to be used are of the form $\alpha(\boxed{\beta(\gamma)}) = \boxed{\rho(\alpha(\gamma))}$. By design, the skeleton $\alpha(\gamma)$ remains unaltered by their application. If rippling succeeds then the induction conclusion $P(\boxed{s(n)})$ is rewritten using wave-rules into some function of the induction hypothesis, $P(n)$; that is, into $\boxed{f(P(n))}$ (f may be the identity). At this point we can call upon the induction hypothesis to simplify the result.

To illustrate rippling and motivate our work on colored higher order unification let us consider the following simple theorem that can be proven by inductive theorem provers using rippling/colouring techniques.

$$\sum_{i=1}^{n} f(i) + \sum_{i=1}^{n} g(i) = \sum_{i=1}^{n} [f+g](i)$$

where f, g are functions from natural numbers to naturals and we have overloaded the function $+$ also to act on such functions. This example illustrates the properties of rippling and introduces also some higher-order colored unification problems.

We formalise summation by a binary function *sum* that takes a function (that is summed over) and a upper bound as arguments. Furthermore, we will the following definition of *sum* (let f, g, H be of type $nat \rightarrow nat$ and N, n be of type nat)[4]:

$$\forall H : sum(H, 0) = 0 \qquad (1)$$

$$\forall H, N : sum(H, s(N)) = sum(H, N) + H(s(N)) \qquad (2)$$

Then our theorem takes the form

$$\forall f, g, n : sum(f, n) + sum(g, n) = sum(\lambda x.f(x) + g(x), n)$$

[4] We employ the Prolog convention of using capital letters to indicate metavariables.

In order to prove this, simple heuristics employed by most inductive provers suggest induction on n which results in the following step case[5]:

$$\begin{array}{l} \mathit{sum}(f,n) + \mathit{sum}(g,n) = \mathit{sum}(\lambda x. f(x) + g(x), n) \\ \rightarrow \mathit{sum}(f, s(n)) + \mathit{sum}(g, s(n)) = \mathit{sum}(\lambda x. f(x) + g(x), s(n)) \end{array}$$

To simplify the step case using rippling, the differences between the induction conclusion and the induction hypothesis are shaded as follows:

$$\begin{array}{l} \mathit{sum}(f,n) + \mathit{sum}(g,n) = \mathit{sum}(\lambda x. f(x) + g(x), n) \\ \rightarrow \mathit{sum}(f, s(n)) + \mathit{sum}(g, s(n)) = \mathit{sum}(\lambda x. f(x) + g(x), s(n)) \end{array} \tag{3}$$

If we can move the shaded areas, so-called wave-fronts, out of the way, then we will be able to simplify the induction conclusion by appealing to the induction hypothesis.

Rippling moves wave-fronts using annotated equations based on axioms, recursive definitions and previously proven lemmas that preserve the skeleton of the term being rewritten. Corresponding to the recursive definitions for *sum* we have the following annotated equation of (2)

$$\mathit{sum}(H, s(N)) = \mathit{sum}(H, N) + H(s(N)) \tag{4}$$

When rippling, the annotations on the left-hand side of the wave-rule must match those in the term being rewritten. As a consequence, there is very little search during rewriting. In order to simplify the conclusion of (3) by rippling we apply (4) on both sides yielding the modified conclusion:

$$\begin{array}{l} (\mathit{sum}(f,n) + f(s(n))) + (\mathit{sum}(g,n) + g(s(n))) \\ = \mathit{sum}(\lambda x. f(x) + g(x), n) + (f(s(n)) + g(s(n))) \end{array}$$

Applying associativity and commutativity law of + results in

$$\begin{array}{l} ((\mathit{sum}(f,n) + \mathit{sum}(g,n)) + f(s(n)) + g(s(n))) \\ = (\mathit{sum}(\lambda x. f(x) + g(x), n) + (f(s(n)) + g(s(n)))) \end{array}$$

which allows weak fertilization[6] on either side which completes the proof.

3.2 Lemma Speculation

The rippling process — as illustrated in the example above — relies on the existence of appropriate annotated equations in order to ripple out (or ripple inside) the occurring wave-fronts. In cases appropriate equations are missing, Ireland & Bundy [IB96] presented a technique to speculate lemmata which push the rippling process further and which are treated as subtasks to be proven separately. Their approach is based on some kind of higher order rippling.

In order to illustrate this application of our calculus, consider the following example involving list manipulations

$$\forall x, y : \mathit{list} \quad \mathit{rev}(\mathit{app}(\mathit{rev}(x), y)) = \mathit{app}(\mathit{rev}(y), x)$$

Here *rev* and *app* stand for the operations of reversing and concatenating lists. Using induction on x we obtain the following formula as an induction conclusion:

$$\mathit{rev}(\mathit{app}(\mathit{rev}(\mathit{cons}(h, x)), y)) = \mathit{app}(\mathit{rev}(y), \mathit{cons}(h, x))$$

[5] The proof of the base case is directly obtained by applying (1) and is omitted here.

[6] This standard technique from inductive theorem proving allows to use the inductive hypothesis to rewrite the inductive conclusion

The rippling process gets blocked[7] after unfolding the definition of *rev* on the left-hand side:

$$rev(app(app(rev(x), cons(h, nil)), y)) = app(rev(y), cons(h, x))$$

In order to push the rippling process further, Ireland & Bundy speculate appropriate lemmata which are then considered as subtasks of the proof. In this example they calculate a schematic form of an appropriate annotated equation

$$app(X, cons(Y, Z)) = app(F_1(X, Y, Z), Z) \tag{5}$$

which can be used to move the blocked wave-front on the right-hand side towards the sink y. While the left-hand side of the speculated lemma is just a generalization of the subterm to be modified, the higher-order variable F_1 represents the unknown wave-front on the right-hand side which has still to be constrained by the further rippling process. Applying this equation on the right-hand side yields:

$$rev(app(app(rev(x), cons(h, nil)), y)) = app(F_1(rev(y), h, x), x) \tag{6}$$

In order to enable the use of the induction hypothesis in this example the wave-front has to be moved in front of the universally quantified variable y which operates as a so-called *sink*. Thus, we use the annotated equation

$$app(rev(Y), cons(X, nil)) = rev(cons(X, Y)) \tag{7}$$

in order to ripple the wave-front on the right-hand side towards y. In order for (7) to be applicable to (6), we unify[8] $F_1(rev(y), h, x)$ and $app(rev(Y), cons(X, nil))$. Higher-order colored unification (see example at the end of section 5 for a trace of the computation) results in a solution $[\lambda U, V, W.app(U, cons(V, nil))/F_1, y/Y]$ Applying the instance of (7) to the right-hand side of (6) the wave-front is moved towards the sink y:

$$rev(app(app(rev(x), cons(h, nil)), y)) = app(rev(cons(h, y)), x) \tag{8}$$

The unifier used to perform this step now also refines the scheme of the speculated annotated equation (5) we used previously to unblock the rippling process, to

$$app(X, cons(Y, Z)) = app(app(X, cons(Y, nil)), Z) \tag{9}$$

Using this speculated equation (9) also on the left-hand side finally yields:

$$rev(app(rev(x), cons(h, y))) = app(rev(cons(h, y)), x)$$

which enables the use of the induction hypothesis and completes this particular proof. Proving also the speculated lemma (9) by induction finishes the overall proof.

---

[7] There are no applicable annotated equations in the data base.

[8] To ease readability we have slightly simplified the method of [IB96]. In practice the overall method is a bit more elaborate: In order to allow the speculation of more complex wave-fronts the occurrence of the meta-variable $F_1(rev(y), h, x)$ is replaced by a nested term $F_2(F_1(rev(y), h, x), h, x)$. Thus, in general F_2 allows one to create additional wave-fronts in the later rippling process but in this example it is of no use and will only be instantiated to the projection $\lambda X, Y, Z.X$.

4 Skeleton

So far we used the informal definition of a skeleton in section 3.1 as being the non-grey parts of a colored term. In this section we will give a precise definition of a skeleton, which turns out to be much more involved that in the first-order case (compare also [SG96] and section 7). For this, we split up the set of color constants $\mathcal{C}$ into a subset $\mathcal{D}$ of color constants which contribute to the skeleton, like for instance "white", and other colors $\mathcal{C} \setminus \mathcal{D}$, like "grey", which indicate wave-fronts.

In order to compute the skeleton $\Omega_{\mathcal{D}}(t)$ of a colored term t we like to extract all parts of t which are either annotated by some color constant of $\mathcal{D}$ or by a color variable and "glue" them together. Besides the information which symbols annotated with specific colors occur within t, the skeleton keeps also track of the subterm-relation between these different symbols. Thus the skeleton of a colored term describes which symbols of specific colors occur within a colored term and how these occurrences are related to each other wrt. the subterm-relation.

In section 3.1 we used the skeleton to restrict the number of possible deductions. Only deduction steps which preserve the skeleton are admissible. In order to ensure this property in advance we would like to trace back the property of an deduction step being skeleton-invariant to some property of the used equation. Thus, we require that a skeleton has to be stable with respect to subterm-replacement: if $\Omega_{\mathcal{D}}(\mathbf{A}|_{\pi}) = \Omega_{\mathcal{D}}(\mathbf{B})$ then $\Omega_{\mathcal{D}}([\mathbf{B}/\pi]\mathbf{A}) = \Omega_{\mathcal{D}}(\mathbf{A})$. I.e. once both sides of an equation $\mathbf{A}|\pi = \mathbf{B}$ share the same skeleton, applying this equation to an annotated term does not change its skeleton. In order to obtain stability wrt. subterm-replacement we define the skeleton $\Omega_{\mathcal{D}}$ of an annotated term t along the term-constructors of the λ-calculus:

$$\Omega_{\mathcal{D}}(\mathbf{AB}) = \{(\mathbf{A}'\mathbf{B}')|\mathbf{A}' \in \Omega_{\mathcal{D}}(\mathbf{A}) \text{ and } \mathbf{B}' \in \Omega_{\mathcal{D}}(\mathbf{B})\}$$

In contrast to the first-order case, the simple-typed λ-calculus imposes an implicit equality relation of terms by α, β, and η-reduction. Since the skeleton guides the proof search we demand that the skeletons of terms being equal wrt. α, β, η-reduction are also equal wrt. α, β, η-reduction. Otherwise, the proof search would depend on the representation of a term. Thus - and also to obtain stability wrt. subterm-replacement - we define

$$\Omega_{\mathcal{D}}(\lambda X.\mathbf{A}) = \{\lambda X.\mathbf{A}'|\mathbf{A}' \in \Omega_{\mathcal{D}}(\mathbf{A})\}.$$

We are left with the case t is an atomic expression of the form $h_{\mathtt{d}}$. In case $\mathtt{d}$ is a skeleton color or a color variable then $h_{\mathtt{d}}$ belongs to the skeleton and $\Omega_{\mathcal{D}}(h_{\mathtt{d}}) = \{h_{\mathtt{d}}\}$. In case $\mathtt{d}$ is a wave-front color then $h_{\mathtt{d}}$ does not belong to the skeleton. Thus, for h being of a base type we define $\Omega_{\mathcal{D}}(h_{\mathtt{d}}) = \emptyset$. In case h is a function or function variable, i.e. $h_d \in \mathit{wff}_{\overline{\alpha_n} \to \beta}$, we have to ensure $\Omega_{\mathcal{D}}(h_{\mathtt{d}} t_1 ... t_n) = \Omega_{\mathcal{D}}(t_1) \cup ... \cup \Omega_{\mathcal{D}}(t_n)$, i.e. the skeleton of the application of $h_{\mathtt{d}}$ is equal to the union of the skeletons of its arguments. Thus we define the skeleton of $h_{\mathtt{d}}$ to be all the possible projections to its arguments. Since we want to conserve well-typedness during the process of skelettification, we need for any pair α_i, β of types the existence of syntactical type-conversion functions $f_{\omega}^{\alpha_i \beta}$ of type $\alpha_i \to \beta$. Then, we define $\Omega_{\mathcal{D}}(h_{\mathtt{d}}) = \{\lambda \overline{X_n}.f_{\omega}^{\alpha_1 \to \beta} X_1, \ldots, \lambda \overline{X_n}.f_{\omega}^{\alpha_n \to \beta} X_n\}$. In case t is a bound variable Z, t has no attached color information and we define $\Omega_{\mathcal{D}}(Z) = \{Z\}$.

In order to obtain typed terms as members of the skeleton we had to invent type-conversion functions. In some terms, occurrences of such constants are redundant, so we use the following ω-reduction rules to remove redundancies in the members of a skeleton

$$(f_\omega^{\beta\to\gamma}(f_\omega^{\alpha\to\beta}X_\alpha)) \longrightarrow_\omega (f_\omega^{\alpha\to\gamma}X_\alpha), (f_\omega^{\alpha\to\alpha}X_\alpha) \longrightarrow_\omega X_\alpha, \lambda x\boldsymbol{.}x \longrightarrow_\omega f_\omega^{\alpha\to\alpha}$$

As shown in [HK95] these ω-reduction rules commute with $\beta\eta$-equality and there exists a $\beta\eta\omega$-normal-form which can be used to compare the skeletons of different terms. Just as in the first-order case, the skeleton is stable with respect to subterm replacement.

$$\text{for all } \mathbf{A},\mathbf{B} \in \mathit{wff}_\alpha(\Sigma;\Gamma_{\mathcal{Z}})\ \Omega_{\mathcal{D}}(\mathbf{A}) = \Omega_{\mathcal{D}}(\mathbf{B}|_\pi) \text{ implies } \Omega_{\mathcal{D}}(\mathbf{B}) = \Omega_{\mathcal{D}}([\mathbf{A}/\pi]\mathbf{B})$$

and also, the skeleton is invariant wrt. $\beta\eta$-reductions

$$\text{for all } \mathbf{A} \in \mathit{wff}_\alpha(\Sigma;\Gamma_{\mathcal{Z}}):\ \mathbf{A} \longrightarrow_{\beta,\eta} \mathbf{B} \text{ implies } \Omega_{\mathcal{D}}(\mathbf{A}) \to_\alpha \Omega_{\mathcal{D}}(\mathbf{B})$$

But the skeleton does not have all nice properties it has in the first-order logic: In particular, it is not stable with respect to $\mathcal{C}$-substitutions, i.e. if $\Omega_{\mathcal{D}}(\mathbf{A}) = \Omega_{\mathcal{D}}(\mathbf{B})$ then $\Omega_{\mathcal{D}}(\sigma(\mathbf{A})) = \Omega_{\mathcal{D}}(\sigma(\mathbf{B}))$ does not hold for all $\mathbf{A},\mathbf{B}$ and substitutions σ.

Note that this is not a problem of our particular definition: In the lambda calculus any meaningful definition of a skeleton will violate this restriction. Consider, for example, the terms $F_{\mathsf{d}}a_{\mathsf{c}}$ and $f_{\mathsf{d}}a_{\mathsf{c}}$. Let $\mathsf{c} \in \mathcal{D}$ while $\mathsf{d} \notin \mathcal{D}$ such that both terms coincide in their skeletons a_{c}. Instantiating F_{d} by $\lambda X\boldsymbol{.}b_{\mathsf{d}}$ will (after β-reduction) result in terms b_{d} and $f_{\mathsf{d}}a_{\mathsf{c}}$ which do obviously not coincide in their skeleton. The reason is that the instantiation enables the use of the β-rule which then, deletes parts of the skeleton. Two possibilities to get rid of this problem immediately suggest themselves: adding function variables (regardless of their annotations) always to the skeleton or restricting admissible substitutions in order to avoid these substitutions. However both will be too restrictive for practical reasons. Thus, the skeleton is in general not substitution-stable if some variable of a non-base type is affected by the substitution. This is no major drawback in using a $\mathcal{C}$-calculus for deduction. For instance in case of instantiating a $\mathcal{C}$-equation we have only to test whether the skeletons of both instantiated sides of the equation still coincides.

5 Calculating Colored Unifiers

It is well-known, that in first-order logic there is always a most general unifier for any equation that is solvable at all. This is not the case for higher-order (colored) unification, where variables can range over functions, instead of only individuals. In fact there can even be solvable equations that have infinite chains of unifiers that become more and more general. In other words most general unifiers need not exist in general.

Just as in the case of unification for first-order terms, the higher-order unification algorithm is a process of recursive decomposition and variable elimination that transform sets of equations into solved forms. Since $\mathcal{C}$-substitutions have two parts, a term– and a color part, we need two kinds of equations ($\mathbf{M} =^t \mathbf{N}$ for term equations and $c =^c d$ for color equations). Sets $\mathcal{E}$ of equations in solved form (i.e. where all equations are of the form $X = \mathbf{M}$ such that the variable X

does not occur anywhere else in $\mathbf{M}_c$ or $\mathcal{E}$) have a unique most general $\mathcal{C}$-unifier $\sigma_{\mathcal{E}}$ that also $\mathcal{C}$-unifies the initial equation.

There are several rules that decompose the syntactic structure of formulae, we will only present two of them. The rule for abstractions transforms equations of the form $\lambda X.A =^t \lambda Y.B$ to $[Z/X]A =^t [Z/Y]B$, where Z is a new constant, while the rule for applications decomposes $h_{\mathtt{a}}(s^1, \ldots, s^n) =^t h_{\mathtt{b}}(t^1, \ldots, t^n)$ to the set $\{a =^c b, s^1 =^t t^1, \ldots, s^n =^t t^n\}$, provided that h is a constant. Furthermore equations are kept in $\beta\eta$-normal form. Note that this decomposition process also eliminates trivial equations, where both sides are $\beta\eta$-equal.

The variable elimination process for color variables is very simple, it allows to transform a set $\mathcal{E} \cup \{\mathtt{A} =^c \mathtt{d}\}$ of equations to $[\mathtt{d}/\mathtt{A}]\mathcal{E} \cup \{\mathtt{A} =^c \mathtt{d}\}$, making the equation $\{\mathtt{A} =^c \mathtt{d}\}$ solved in the result. In case of formula equations, elimination is not that simple, since we have to ensure that $|\sigma(X_{\mathtt{A}})| = |\sigma(X_{\mathtt{B}})|$ to obtain a $\mathcal{C}$-substitution σ. Thus we cannot simply transform a set $\mathcal{E} \cup \{X_{\mathtt{d}} =^t \mathbf{M}\}$ into $[\mathbf{M}/X_{\mathtt{d}}]\mathcal{E} \cup \{X_{\mathtt{d}} =^t \mathbf{M}\}$, since this would (incorrectly) solve the equations $\{X_{\mathtt{c}} = f_{\mathtt{c}}, X_{\mathtt{d}} = g_{\mathtt{d}}\}$. The correct variable elimination rule transforms $\mathcal{E} \cup \{X_{\mathtt{d}} =^t \mathbf{M}\}$ into $\sigma(\mathcal{E}) \cup \{X_{\mathtt{d}} =^t \mathbf{M}, X_{\mathtt{c}_1} = \mathbf{M}^1, \ldots, X_{\mathtt{c}_n} =^t \mathbf{M}^n\}$, where $\mathtt{c}_i$ are all colors of the variable X occurring in $\mathbf{M}$ and $\mathcal{E}$, the $\mathbf{M}^i$ are appropriately colored variants (same color erasure) of $\mathbf{M}$, and σ is the $\mathcal{C}$-substitution that eliminates all occurrences of X from $\mathcal{E}$.

It would be convenient, if the transformations described so far, were sufficient for transforming all unifiable sets of equations into solved form and thus finding all unifiers. But, due to the presence of function variables, systematic application can terminate with equations of the form $X_{\mathtt{c}}(s^1, \ldots, s^n) =^t h_{\mathtt{d}}(t^1, \ldots, t^m)$. Such equations can neither be further decomposed by the rules above, since this would loose unifiers, nor can the right hand side be substituted for X as in a variable elimination rule, since the types would clash. Let us consider the uncolored equation $X(a) =^t a$ which has the solutions $(\lambda Z.a)$ and $(\lambda Z.Z)$ for X.

The standard solution for finding a complete set of solutions in this so-called **flex/rigid** situation is to substitute a term for X that will enable decomposition to be applicable afterwards. It turns out that for finding all $\mathcal{C}$-unifiers it is sufficient to bind X to terms of the same type as X (otherwise the unifier would be ill-typed) and compatible color (otherwise the unifier would not be a $\mathcal{C}$-substitution) that either have the same head as the right hand side; the so-called **imitation** solution ($\lambda Z.a$ in our example) or where the head is a bound variable that enables the head of one of the arguments of X to become head; the so-called **projection** binding ($\lambda Z.Z$).

In order to get a better understanding of the situation let us reconsider our example using colors. $X(a_{\mathtt{c}}) = a_{\mathtt{d}}$. For the imitation solution ($\lambda Z.a_{\mathtt{d}}$) we "imitate" the right hand side, so the color on a must be $\mathtt{d}$. For the projection solution we instantiate $(\lambda Z.Z)$ for X and obtain $(\lambda Z.Z)a_{\mathtt{c}}$, which β-reduces to $a_{\mathtt{c}}$. We see that this "lifts" the constant $a_{\mathtt{c}}$ from the argument position to the top. Incidentally, the projection is only a $\mathcal{C}$-unifier of our colored example, if $\mathtt{c}$ and $\mathtt{d}$ are identical.

Fortunately, the choice of instantiations can be further restricted to the most general terms in the categories above. If $X_{\mathtt{c}}$ has type $\overline{\beta_n} \to \alpha$ and $h_{\mathtt{d}}$ has type

$\overline{\gamma_m} \to \alpha$, then these so-called **general bindings** have the following form:

$$\mathcal{G}^h_{\mathtt{d}} = \lambda Z^{\alpha_1} \ldots Z^{\alpha_n}.h_{\mathtt{d}}(H^1_{\mathtt{e}_1}\overline{Z}) \ldots (H^m_{\mathtt{e}_m}\overline{Z})$$

where the H^i are new variables of type $\overline{\beta_n} \to \gamma_i$ and the $\mathtt{e}_i$ are either distinct color variables (if $\mathtt{c} \in \mathcal{X}$) or $\mathtt{e}_i = \mathtt{d} = \mathtt{c}$ (if $\mathtt{c} \in \mathcal{C}$). If h is one of the bound variables Z^{α_i}, then the annotation $\mathtt{d}$ at h is omitted and $\mathcal{G}^h_{\mathtt{d}}$ is called a **projection binding**, and else, (h is a constant or a free variable), $\mathcal{G}^h_{\mathtt{d}}$ is called an **imitation binding**. Note that while imitation bindings are unique up to the names of the new free variables H^i, there can be up to n projection bindings, depending on the types involved.

The general rule for flex/rigid equations transforms equations of the form

$$\mathcal{E} \wedge X_{\mathtt{c}}(s^1, \ldots, s^n) =^t h_{\mathtt{d}}(t^1, \ldots, t^m)$$

into
$$\mathcal{E} \wedge X_{\mathtt{c}}(s^1, \ldots, s^n) =^t h_{\mathtt{d}}(t^1, \ldots, t^m) \wedge X_{\mathtt{c}} = \mathcal{G}^h_{\mathtt{c}}$$

which in essence only fixes a particular binding for the head variable $X_{\mathtt{c}}$. It turns out that these general bindings suffice to solve all flex/rigid situations, possibly at the cost of creating new flex/rigid situations after elimination of the variable $X_{\mathtt{c}}$ and decomposition of the changed equations (the elimination of X changes $X_{\mathtt{c}}(s^1, \ldots, s^n)$ to $\mathcal{G}^h_{\mathtt{c}}(s^1, \ldots, s^n)$ which has head h).

Finally the only remaining case, where the heads of both sides of the equation are free variables the so-called **flex/flex** case. The solution of this case is either to project as in the flex/rigid case or to "guess" (computationally: to search for) the right head for the equation and bind the head variable to the appropriate imitation binding. Clearly this need for guessing the right head leads to a serious explosion of the search space, which makes higher-order colored unification computationally infeasible. Fortunately, most applications do not need full higher-order unification, we will discuss two restrictions in section 6.

First, however, we will fortify our intuition on calculating higher-order colored unifiers by reconsider the Lemma speculation example. There the key step was to solve the equation

$$F_{\mathtt{g}}(rev_{\mathtt{W}}(u_{\mathtt{W}}), h_{\mathtt{g}}, v_{\mathtt{g}}) =^t app_{\mathtt{g}}(rev_{\mathtt{B}}(Y_{\mathtt{A}}), (cons_{\mathtt{g}}(X_{\mathtt{g}}, nil_{\mathtt{g}})))$$

Since F is a function variable, we are in a flex/rigid situation, and have the possibilities of projection and imitation. There are three possible projections, $\lambda U, V, W.U$, $\lambda U, V, W.V$, $\lambda U, V, W.W$, which all lead to immediate failure, since they project up the rigid subterms $rev_{\mathtt{W}}(u_{\mathtt{W}})$, $h_{\mathtt{g}}$ or $v_{\mathtt{g}}$, which would clash with the head $app_{\mathtt{g}}$ of the right hand side. So we only have the imitation binding $\lambda U, V, W.app_{\mathtt{g}}(HUVW)(KUVW))$ for $\mathbf{F}_{\mathtt{g}}$. If we bind $F_{\mathtt{g}}$ to that and decompose (we can directly eliminate $F_{\mathtt{g}}$ since there are no variants of it around), then we are left with the equations

$$H_{\mathtt{g}}(rev_{\mathtt{W}}(u_{\mathtt{W}}), h_{\mathtt{g}}, v_{\mathtt{g}}) =^t rev_{\mathtt{B}}(Y_{\mathtt{A}}) \wedge K_{\mathtt{g}}(rev_{\mathtt{W}}(u_{\mathtt{W}}), h_{\mathtt{g}}, v_{\mathtt{g}}) =^t cons_{\mathtt{g}}(X_{\mathtt{g}}, nil_{\mathtt{g}})$$

Here, we choose[9] the imitation $\lambda UVW\boldsymbol{.}cons_{\mathtt{g}}(M_{\mathtt{g}}UVW)(N_{\mathtt{g}}UVW)$ for $K_{\mathtt{g}}$ and the 1-projection binding $(\lambda UVW\boldsymbol{.}U)$ for $H_{\mathtt{g}}$ and arrive at

$$rev_{\mathtt{W}}(u_{\mathtt{W}}) =^t rev_{\mathtt{B}}(Y_{\mathtt{A}}) \wedge$$
$$cons_{\mathtt{g}}(M_{\mathtt{g}}(rev_{\mathtt{W}}(u_{\mathtt{W}}), h_{\mathtt{g}}, v_{\mathtt{g}}), N_{\mathtt{g}}(rev_{\mathtt{W}}(u_{\mathtt{W}}), h_{\mathtt{g}}, v_{\mathtt{g}})) =^t cons_{\mathtt{g}}(X_{\mathtt{g}}, nil_{\mathtt{g}})$$

finally we can decompose again

$$\mathtt{A} =^c \mathtt{W} \wedge u_{\mathtt{W}} =^t Y_{\mathtt{A}} \wedge X_{\mathtt{g}} =^t M_{\mathtt{g}}(rev_{\mathtt{W}}(u_{\mathtt{W}}), h_{\mathtt{g}}, v_{\mathtt{g}}) \wedge N_{\mathtt{g}}(rev_{\mathtt{W}}(u_{\mathtt{W}}), h_{\mathtt{g}}, v_{\mathtt{g}}) =^t nil_{\mathtt{g}}$$

the first two equations can directly be solved by eliminating $\mathtt{A}$ for $\mathtt{W}$ and $Y_{\mathtt{A}}$ (which is actually $Y_{\mathtt{W}}$ after the previous elimination) for $u_{\mathtt{W}}$. The third equation cannot be solved this way, since $M_{\mathtt{g}}(rev_{\mathtt{W}}u_{\mathtt{W}})h_{\mathtt{g}}v_{\mathtt{g}})$ is not $\mathtt{g}$-monochrome, so we choose the 2-projection binding[10] $\lambda U, V, W\boldsymbol{.}V$ for $X_{\mathcal{C}}$ and solve the fourth equation with the imitation binding $\lambda UVW\boldsymbol{.}nil_{\mathtt{g}}$ and for $N_{\mathtt{g}}$. Eliminating these bindings allows us to simplify the equations to the trivial set $h_{\mathtt{g}} =^t h_{\mathtt{g}}$ and $nil_{\mathtt{g}} =^t nil_{\mathtt{g}}$.

Thus one final solution of the unification problem is

$$[\lambda UVW\boldsymbol{.}app_{\mathtt{g}}(U, cons_{\mathtt{g}}(V, nil_{\mathtt{g}}))/F_{\mathtt{g}}], [u_{\mathtt{W}}/Y_{\mathtt{A}}], [h_{\mathtt{g}}/X_{\mathtt{g}}]$$

We have indicated the choice points for the other solutions in the footnotes.

6 Tractable Fragments

Most applications do not need full higher-order unification, as we have discussed it above. First, for theorem proving purposes it is only important to know about the existence of any unifier. In the case of classical higher-order unification it is therefore sufficient to consider flex/flex pairs as solved, since they are guaranteed to have unifiers. Second, there are applications of rippling in program synthesis [Hes91,Kra94] which rely on higher-order patterns [Mil92]. This syntactic fragment has a unitary unification problem which is decidable in linear time for the uncolored case. Third, in the linguistic applications [GK96,Gar] formulae belong to very restricted syntactic subclasses, for which much better results are known (for classical higher-order unification). In particular, the fact that free variables only occur on the left hand side of our equations reduces the problem of finding solutions to higher-order matching, of which decidability has been proven for the subclass of third-order formulae [Dow92]. This class, (intuitively allowing only nesting functions as arguments up to depth three) covers all examples studied so far. In this section we will discuss two of the fragments: pre-unification and higher-order patterns.

6.1 Pre-$\mathcal{C}$-Unification

For the pre-unification problem flex-flex pairs are considered already solved, since they can always be trivially solved by binding the head variables to special constant functions that identify the formulae by absorbing their arguments. In

[9] The 2-projection binding is impossible for type reasons and the 3-projection binding leads to immediate subsequent clash. The imitation binding leads to a solution $\lambda UVW\boldsymbol{.}app_{\mathtt{g}}(rev_{\mathtt{g}}(L_{\mathtt{g}}UVW))(cons_{\mathtt{g}}Y\,nil)$ for $F_{\mathtt{g}}$ that is not wanted in our motivating example, so we will not pursue it here.

[10] The 3-projection $(\lambda UVW\boldsymbol{.}W)$ or the imitation binding $(\lambda UVW\boldsymbol{.}Q_{\mathtt{g}})$ for some new variable Q would also have worked.

case of the colored lambda-calculus a flex-flex pair may have no solution if the top-level variables of both terms are annotated by different colors. For instance $F_{\mathtt{d}}a_{\mathtt{d}} =^t G_{\mathtt{c}}a_{\mathtt{c}}$ has no unifier. On the other hand $F_{\mathtt{d}}a_{\mathtt{c}} =^t G_{\mathtt{c}}a_{\mathtt{c}}$ has a unifier $[\lambda X.X/F_{\mathtt{d}}], [\lambda X.X/G_{\mathtt{c}}]$. The reason for this is the fact that projections, i.e. terms of the form $\lambda\overline{X^k}.X^i$, carry no color information but are valid instances of colored variables like $F_{\mathtt{d}}$ or $G_{\mathtt{c}}$. Hence, in order to solve such flex-flex pairs we have to map one of the top-level variables to a projection formula. The relevant notion, when this has to happen is that of a **flexible chain**, i.e. a set $\mathcal{E}' = \mathbf{A}^1 =^t \mathbf{B}^1 \wedge \ldots \wedge \mathbf{A}^n =^t \mathbf{B}^n$ of flex/flex pairs, such that $\mathbf{head}(\mathbf{A}^i) = \mathbf{head}(\mathbf{B}^{i-1}) \in \mathcal{V}_{\mathcal{X}}$ for $2 \leq i \leq n$. We call $\mathbf{head}(\mathbf{A}^1) = F_{\mathtt{c}}$ and $\mathbf{head}(\mathbf{B}^n) = G_{\mathtt{d}}$ the left and right ends of $\mathcal{E}'$. Such a chain is called **reducible**, if $\mathtt{c}, \mathtt{d} \in \mathcal{C}$ and $\mathtt{c} \neq \mathtt{d}$ then we call $\mathcal{E}'$ a **reducible chain**, otherwise **safe**.

It turns out that safe chains always have solutions, whereas a reducible chain in a system $\mathcal{E}$ indicates a clash of different color annotations to the top-level variables. Thus the notion of solved form for colored pre-unification allows flex/flex pairs, as long as there are no reducible chains.

This definition is tailored to guarantee that pre-$\mathcal{C}$-unifiers can always be extended to $\mathcal{C}$-unifiers by finding trivial unifiers for the flexible pairs and that equational problems in pre-$\mathcal{C}$-solved form always have most general unifiers. Therefore an equational system $\mathcal{E}$ is pre-$\mathcal{C}$-unifiable, iff it is $\mathcal{C}$-unifiable. The trivial unifiers of the safe flex/flex pairs $F_{\mathtt{a}}\overline{\mathbf{U}} =^t G_{\mathtt{b}}\overline{\mathbf{V}}$ are constructed as constant functions

$$[\lambda X^1_{\alpha_1} \ldots X^n_{\alpha_n}.H^{\beta}_{cr(F_{\mathtt{a}},\mathcal{E})}/F_{\mathtt{a}}], [\lambda X^1_{\gamma_1} \ldots X^m_{\gamma_m}.H^{\beta}_{cr(G_{\mathtt{b}},\mathcal{E})}/G_{\mathtt{b}}]$$

where the colors of the new head variables H are so-called **color restrictions** $cr(X_{\mathtt{a}}, \mathcal{E})$. These are defined in terms of the flexible chains in the unification problem $\mathcal{E}$:

$cr(X_{\mathtt{a}}, \mathcal{E}) = \mathtt{d}$ if $\mathtt{a} \in \mathcal{X}$ and there is flexible chain $\mathcal{E}'$ in $\mathcal{E}$ with left head $X_{\mathtt{a}}$ and right head $Y_{\mathtt{d}}$ for some $\mathtt{d} \in \mathcal{C}$ and $cr(X_{\mathtt{a}}, \mathcal{E}) = \mathtt{a}$ otherwise.

Given a safe system $\mathcal{E}$ the notion of color restriction is well-defined: two subsets of $\mathcal{E}$ satisfying the condition of the definition above which result in different color restrictions $\mathtt{c}, \mathtt{c}' \in \mathcal{C}$ could always be merged into a reducible chain, contradicting our assumption that $\mathcal{E}$ is safe.

For the actual pre-unification transformations, we exchange the flex/flex rule by a rule that reduces the number of reducible chains step by step by binding some head variable to suitable projection binding. While — unlike in the uncolored case — we cannot drop the flex/flex rule altogether, the restrictions arising from reducible chains are severe enough to make pre-$\mathcal{C}$-unification tractable. In particular the restriction alleviates the need for unspecified imitations the flex/flex case, which makes full unification infinitely branching.

In our example above, the third equation in the last unification problem is a flex/flex equation, which can be reduced by pre-$\mathcal{C}$-unification. In particular, the imitation mentioned in footnote 10 is unnecessary.

6.2 Higher-Order Patterns

For the colored λ-calculus, the definition of higher-order patterns is exactly as in the uncolored case (i.e. free variables may only occur at the leaves of formulae or applied to distinct bound variables). However, in the colored case, the problem of pattern unification is slightly more complex, and we will profit from the understanding of colored flex/flex pairs that we have achieved in the last section. In particular, colored pattern unification cannot be unitary, since conflicting colors on flex/flex pairs can force the instantiations to be (uncolored) projections. As we have seen above, conflicting colors can entail that flex/flex pairs are unsolvable, on the other hand, for pattern unification, they can also lead multiple solutions (the erasure of which can be represented by a more general uncolored higher-order pattern[11]). Consider for instance the pair

$$\lambda XYZW.F_{\mathtt{a}}XYZW =^t \lambda XYZW.F_{\mathtt{b}}YXZW$$

where $\mathtt{a}, \mathtt{b} \in \mathcal{C}$. Obviously, there are two most general solutions

$$[\lambda XYZW.Z/F_{\mathtt{a}}], [\lambda XYZW.Z/F_{\mathtt{b}}] \text{ and } [\lambda XYZW.W/F_{\mathtt{a}}], [\lambda XYZW.W/F_{\mathtt{b}}]$$

The tractable nature of pattern unification hinges on the observation that the solving of flex/rigid pairs is deterministic, that is, all but the imitation or one projection immediately lead to failure. Thus for pattern unification we only can directly inherit the decomposition and flex/rigid rules from general colored higher-order unification and only have to concern ourselves with the flex/flex situations. Clearly, all the discussion about flexible chains also applies also to higher-order patterns, so we keep the flex/flex rule for reducible flexible chains. This leaves us with the case of safe flex/flex pairs, where (as we have seen above) color clashes are not a problem. Therefore, we can directly adapt the well-known rules for higher-order pattern unification: If we have a pair $F_{\mathtt{a}}\overline{X^n} ==^t F_{\mathtt{b}}\overline{Y^n}$, then $F_{\mathtt{a}}$ is bound to to $\lambda\overline{Z^n}.H_{\mathtt{a}}\overline{W_k}$ and $F_{\mathtt{b}}$ to $\lambda\overline{Z^n}.H_{\mathtt{b}}\overline{W_k}$, where the $\overline{W^k}$ are those bound variables, where $X^i = Y^i$[12]. Unlike in the uncolored case, an application of this rule does not immediately solve the pair, (the colors $\mathtt{a}$ and $\mathtt{b}$ need not be identical), but it transforms it into a form, in which decomposition can do the rest (this will always succeed, iff the pair the rule acts upon is safe).

For the remaining case of safe flex/flex pairs with differing head variables, we use a similar argumentation (directly modeled after the uncolored case) and rule. From this argumentation (the flex/rigid and the safe flex/flex cases are deterministic and the reducible flex/flex cases only involve projections), we can directly derive that colored pattern unification is decidable[13] and finitary i.e. pattern unification problems have at most finitely many most general unifiers.

[11] This observation shows that a generate-and-test procedure for colored pattern unification is infeasible, since this would have generated the uncolored solution and rejected it, erroneously predicting the absence of colored solutions.

[12] For any unifier σ we have $\sigma(F_{\mathtt{a}}) = \lambda\overline{Z^n}.\mathbf{A}$. $\mathbf{A}$ can only have occurrences of Z^i, such that $X^i = Y^i$: If we assume that $\mathbf{A}$ contains an occurrence of Z_i (say at position p) with $X^i \neq Y^i$, then $\sigma(F)\overline{X^n} =_{\beta\eta} \overline{[X^n/Z^n]}\mathbf{A}$ and $\sigma(F)\overline{Y^n} =_{\beta\eta} \overline{[Y^n/Z^n]}\mathbf{A}$, so these differ at position p, which contradicts the assumption that σ is a unifier of $\mathcal{E}$.

[13] The termination and confluence arguments can be directly modeled after the standard case.

7 Related Work

Recently, Smaill and Green [SG96] developed the notion of *higher-order embeddings*. An embedding $\hookrightarrow$ is a relation on terms and $s \hookrightarrow t$ - speaking $\mathbf{A}$ *is embedded* in $\mathbf{B}$ - denotes intuitively that $\mathbf{A}$ is a skeleton of $\mathbf{B}$. As a base case each atomic expression $\mathbf{B}$ is embedded into itself: $t \hookrightarrow t$. Also a term $\mathbf{A}$ is embedded into an application $(\mathbf{B}_1\mathbf{B}_2)$ if it is embedded into one of its arguments or $\mathbf{A}$ is itself an application $(\mathbf{A}_1\mathbf{A}_2)$ and each $\mathbf{A}_i$ is embedded in $\mathbf{B}_i$. $\mathbf{A}$ is embedded into an abstraction $(\lambda X.\mathbf{B})$ if it is embedded into all instantiations $(\lambda X.\mathbf{B})\mathbf{C}$ for all $\mathbf{C}$ or $\mathbf{A}$ is itself an abstraction $(\lambda X.\mathbf{A}')$ and $(\lambda X.\mathbf{A}')\mathbf{C}$ is embedded into $(\lambda X.\mathbf{B})\mathbf{C}$ for all $\mathbf{C}$.

Comparing this notion of embedding and our definition of skeletons one observe several conceptual differences. In case of applications the definition of embeddings does not preserve the intended subterm relation on (first-order) terms. For example, consider a first order term $(g(fac)b)$ then $(fab) \hookrightarrow (g(fac)b)$ holds. This confusion of arguments of f and g may cause severe problems when defining termination orderings on rippling with the help of embeddings. In our setting the skeleton of $\boxed{(g(f a c)b)}$ is the empty set and the use of the syntactic type-conversion functions prevents the mix-up of arguments. Hence, the intended subterm-relation is preserved.

The notion of embeddings is a step towards a generate-and-test procedure[14] based on standard higher order matching/unification which performs an (arbitrary) deduction steps and tests whether a specific term is embedded into the result of this step. Our approach to attach additional information at each symbol allows one to maintain the information about embedding during the deduction process since skeletons are stable wrt. subterm-replacement. This information is also necessary to restrict the number of possible solutions (e.g during higher-order unification) as soon as possible.

In contrast to our approach, higher-order embeddings are restricted to the λI-calculus which hampers their use for example in case of lemma speculation (cf. section 3.2). In this application instantiations of higher-order variables have to be speculated which typically use only some (but not necessarily all) of its arguments and thus, are outside the scope of λI-calculus.

8 Conclusion and Further Work

We have extended the first-order rippling/coloring method to higher-order logic and present unification, pre-unification and pattern unification algorithms that we prove correct and complete. Thus we have provided a formal basis to the implementation of rippling in a higher-order setting which is required e.g. in case of middle-out reasoning [Hes91,IB96] and also a logical basis for an interface for linguistic extra-semantical information in the construction of natural-language semantics [GK96].

Furthermore, the work presented in this paper provides a starting point for the mechanization of higher-order reasoning with equality along the lines

[14] For principal difficulties of this approach cf. footnote 11.

of [WNB92,CH94,Hut96] which develop heuristics that guide the difference reduction process in first-order equality calculi such as [Mor69,Dig81]. These difference reducing approaches seem to be more promising for higher-order logic, since they can reduce the search spaces (comparing to those induced by encoding equality via the Leibniz formula) without needing reduction orderings, which become very weak in the presence of higher-order (function)-variables.

References

[BSvH$^+$93] A. Bundy, A. Stevens, F. van Harmelen, A. Ireland, and A. Smaill. Rippling: a heuristic for guiding inductive proofs. *AI*, 62:185–253, 1993.

[Bun88] A. Bundy. The use of explicit plans to guide inductive proofs. *9th CADE*, Springer, LNCS 310, Argonne, Illinois, USA, 1988.

[CH94] J. Cleve and D. Hutter. A methodology for equational reasoning. *Hawaii International Conference on System Sciences 27 Volume III*, IEEE Computer Society Press, 1994.

[Chu40] Alonzo Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.

[Dig81] Vincent J. Digricoli. The efficacy of RUE resolution, experimental results and heuristic theory. In Ann Drinan, editor, *7th ICJAI*,Vancouver, Canada, Morgan Kaufmann, 1981.

[Dow92] Gilles Dowek. Third order matching is decidable. In *Proceedings of the 7th Annual IEEE Symposium on Logic in Computer Science (LICS-7)*, pages 2–10. IEEE Computer Society Press, 1992.

[GK96] Claire Gardent and Michael Kohlhase. Higher-order coloured unification and natural language semantics. Proceedings ACL'96, Santa Cruz 1996.

[Gar] Claire Gardent. Sloppy identity. Proceedings LACL'96, Springer Verlag forthcoming.

[Hes91] Jane Hesketh. *Using Middle-Out Reasoning to Guide Induction.* PhD thesis, University of Edinburgh, 1991.

[HK95] Dieter Hutter and Michael Kohlhase. A colored version of the lambda calculus. Seki-95-05, University of Saarland, Saarbruecken, Germany, 1995.

[Hut90] Dieter Hutter. Guiding induction proofs. *10th CADE*, Springer, LNCS 449, Kaiserslautern, Germany, 1990.

[Hut96] Dieter Hutter. Using rippling for equational reasoning. *20. Jahrestagung Künstliche Intelligenz*, Springer LNAI 1137, Dresden, Germany, 1996.

[IB96] Andrew Ireland and Alan Bundy. Productive use of failure in inductive proof. *Special Issue of the Journal of Automated Reasoning*, 16(1/2), 1996.

[Kra94] Ina Kraan. *Proof Planning for Logic Program Synthesis.* PhD thesis, University of Edinburgh, 1994.

[Mil92] Dale Miller. Unification under a mixed prefix. *Journal of Symbolic Computation*, 14:321–358, 1992.

[Mon74] R. Montague. The proper treatment of quantification in ordinary english. In R. Montague, editor, *Formal Philosophy. Selected Papers.* Yale University Press, New Haven, 1974.

[Mor69] James B. Morris. E-resolution: Extension of resolution to include the equality relation. *1st IJCAI*, 1969

[SG96] A. Smaill and I. Green. Higher-order annotated terms for proof search. *TPHOLs'96*, 1996

[WNB92] Toby Walsh, A. Nunes, and Alan Bundy. The use of proof plans to sum series. *11th CADE*, Saratoga Spings, USA, Springer LNCS 607, 1992.