Higher–Order Colored Unification: A Linguistic Application

Claire Gardent[†] **Michael Kohlhase**[‡] **Karsten Konrad**[‡]

[†]*Department of Computational Linguistics*
[‡]*Department of Computer Science*

Universität des Saarlandes
D–66041 Saarbrücken, Germany

RÉSUMÉ. *Au cours de la dernière décennie, l'Unification d'Ordre Supérieur (*UOS*) est devenue un outil privilégié pour construire la représentation sémantique des expressions de la langue naturelle. Pourtant, elle présente un problème de taille: elle sur-génère, c'est-à-dire qu'elle produit des solutions qui, quoique valides au plan mathématique, sont incorrectes d'un point de vue linguistique car elles ne représentent pas une interprétation possible de l'expression analysée. Dans cet article, nous montrons que l'unification colorée d'ordre supérieur (*UCOS*) permet de remédier à ce problème et nous présentons les aspects linguistiques, logiques et informatiques d'un système de construction sémantique pour la langue naturelle basé sur l'*UCOS.

ABSTRACT. *During the last decade, Higher-Order unification (*HOU*) has become a popular tool for constructing the semantic representation of natural language expressions. But there is a well-known problem with this approach: it over-generates that is, it produces solutions which although they are mathematically valid, are linguistically incorrect because they do not represent possible meanings of the expression being analysed. In this paper, we argue that Higher-Order Colored Unification (*HOCU*) can help prevent over-generation and we describe the linguistic, logical and computational aspects of an* HOCU*–based approach to semantic construction.*

MOTS-CLÉS*: Unification d'ordre supérieur, Structures de traits, Sémantique de la langue naturelle*

KEY WORDS*: Higher-Order Unification, Feature Trees, Natural Language Semantics*

1 Introduction

In the field of inductive theorem proving, there is a technique for guiding induction proof search which is known as term coloring [HUT 97b, HUT 99] and uses a modified version of higher-order unification (HOU), namely Higher-Order Colored Unification (HOCU). Crucially, coloring allows one to add arbitrary information to term *occurrences*, and to maintain and exploit this information during the inference process (this differentiates coloring from such semantic annotation techniques as sorts which maintain attributes of symbols but not attributes of symbol occurrences). In this paper,

we argue that this property makes the HOCU framework a more appropriate basis for natural language semantics than simple Higher-Order Unification (HOU).

HOU has become increasingly popular in computational linguistics as a tool for constructing the semantic representation of constructs such as ellipsis (the ommission of semantically redundant material), focus (the salient accentuation of part of an utterance) and deaccenting (the prosodic reduction of semantically redundant material). This is because HOU provides a well-defined means of producing functions (the solutions to an HOU problem are substitutions which assign λ-terms and typically *functions* and *predicates* to free variables) and because such functions are necessary for interpreting the above mentioned constructs. In sentence (1) for instance, the meaning of the elliptical verb phrase (VP) *does* is the property of "liking mary", a meaning which in the long standing tradition of Montague Grammar (the theory which underlies the contemporary semantic theories of natural language) can be modeled as a function from individuals to truth values or equivalently as the λ-abstraction $\lambda X.like(X, m)$.

(1) *Jon likes Mary and Peter does too*

However, this meaning is not given by the VP-ellipsis (the omission of a semantically redundant VP, henceforth VPE) *does* itself. Rather it must be recovered from the present context in this case, the preceding sentence *Jon likes Mary*. The underlying motivation for using HOU to model the interpretation of such cases is the following insight: given the appropriate semantic representation and the adequate equations, HOU will yield the "missing" meaning of the VPE – namely the function $\lambda Z.like(Z, m)$ – and will do so on the basis of a well-defined computational procedure. Hence the interest for computational linguists.

Unfortunately the fit between HOU and linguistic analysis is not a perfect one and given a linguistic problem, HOU will systematically over-generate. That is, it produces solutions which although they are mathematically valid, are linguistically incorrect. The reason for this is that HOU operates on semantic representations whereas in general, the semantic construction process (i.e., the process of constructing semantic representations for natural language expressions) is subject to constraints stemming from sources of linguistic knowledge other than semantics[1]. Semantic construction can for instance be constrained by syntactic, by prosodic or by pragmatic information. More specifically, there are two unsolved problems with using HOU as a tool for doing semantic construction. First, distinct occurrences of the same term might have to be treated differently. Failure to do so leads to over-generation. Second, linguistic constraints on semantic construction might stem from very distinct (but interacting) modules. This in turn suggests a modular treatment of the interaction between the different linguistic components. The goal of this paper is to show that HOCU solves the over-generation problem encountered by HOU analyses of natural language semantics in that it enables a correct modeling of the interacting constraints placed by the various linguistic modules on semantic construction.

[1]Linguistic theory usually distinguishes between several sources of linguistic knowledge or modules e.g., the syntactic module which encompasses knowledge about the structure of linguistic constituents (usually in the guise of rewrite rules), the prosodic module which contains knowledge about intonation and accentuation or the semantic module in which knowledge about meaning is stored. The pragmatic module pertains to almost everything else e.g., user modeling, world knowledge, the effect of discourse structure on interpretation and the import of communication principles on meaning.

The paper is structured as follows. Section 2 starts by showing how the typed λ-calculus and Higher-Order Unification are used in natural language theory. It then discusses in more detail why HOU approaches to natural language semantics overgenerate. Section 3 provides a partial answer to this problem and shows how (atomic) colors permit the necessary distinction between different occurrences of the same term. Section 4 generalizes the approach to capture the interaction of independent constraints and their effect on semantic construction. The proposal is to have colors as feature trees rather than atoms, the underlying intuition being that feature structures provide a natural way of dynamically creating complex constraints out of simple ones. In section 5, we present the HOCU algorithm first from a logical and second, from a computational perspective. Section 6 presents an implemented system which uses both HOU and HOCU to do semantic construction for natural language and describes the coverage of the approach. Section 7 concludes with links to other usages of HOCU and identifies topics for further research.

2 Natural language semantics

Language communicates meaning and at the very least, a theory of natural language semantics should provide a means to systematically associate meaning not only with simple but also with complex expressions of natural language. Another way to put it is that a semantic theory should tell us how the meaning of a complex expression can be derived from the meaning of its parts. This is the question Montague [MON 74] addressed and for which he proposed to use the simply typed λ-calculus. The idea is that each natural language expression is associated with some model theoretic object which can be represented by a λ-term. The rules of natural language syntax are then coupled with semantic rules which say how the semantics of the sub-constituents described by this rule combine to yield the semantics of its resulting constituent. A short example will illustrate this. Suppose the sentence whose meaning we want to determine is:

(2) *Jon loves Mary*

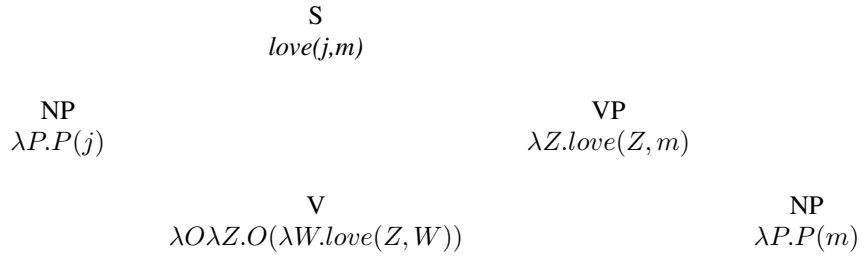
The lexicon of a Montague-type grammar will associate with each word in this sentence the following meanings and syntactic categories:

Jon	$\lambda P.P(j)$	NP
love	$\lambda O.\lambda Z.O(\lambda W.love(Z,W))$	V
Mary	$\lambda P.P(m)$	NP

The syntactic and semantic rules will be:

rule 1	S $\rightarrow$ NP, VP
	S'= NP'(VP')
rule 2	VP $\rightarrow$ V, NP
	VP'= V'(NP')

where S, NP, VP and V stand for the syntactic categories: sentence, noun phrase, verb phrase and verb respectively and the prime indicates meanings (so for instance S' is the meaning of S). Under this approach, the syntactic and semantic analysis of the sentence in (2) is represented by the following tree:



In short, Montague's approach gives us a systematic tool for assigning meanings to natural language expressions, a tool which is based on the use of the simply typed λ-calculus as a semantic representation language.

More recently, another tool has become increasingly standard in natural language semantics namely, Higher-Order Unification (HOU).

[DAL 91] shows that it allows a treatment of VP–Ellipsis (the omission of a verb phrase) which successfully captures the interaction of VP-ellipsis with quantification (the use of quantifying noun phrases e.g., *every man*, *no woman*) and nominal anaphora (phonologically non-empty elements such as *he* whose meaning is given by the linguistic or situational context). For instance, they show that HOU permits reconstructing the meaning of the missing VP in the second clause of (3a); that it correctly predicts the ambiguity of (3b) where due to the presence of a pronoun (*his*) in the first clause and assuming that *his* refers to *Jon*, the second clause can be interpreted either as *Peter likes Jon's cat* or as *Peter likes Peter's cat*; and that it captures the fact that although the first clause of (3c) does contain a pronoun, the presence of the quantifier *every boy* results in (3c) having one rather than the expected two readings.

(3) a. *Jon likes mary and Peter does too.*
b. *Jon likes his cat and Peter does too.*
c. *Every boy likes his cat. Peter does for instance.*

[PUL 97, GAR 96a] use HOU to model the interpretation of focus (i.e., prosodic prominence) and its interaction with various linguistic constructs. Consider for instance the examples in (4) where upper letters indicate prosodic prominence.

(4) a. *Jon only introduced MARY to Sue*
b. *Jon did not introduce MARY to Sue*
c. *Jon always introduces MARY to Sue*

In each case, moving the focus from *Mary* to *Sue* induces a change in meaning. For instance, (4a) means that the only x such that Jon introduced x to Sue is Mary. By contrast, if the focus were on *Sue*, (4a) would mean that the only x such that Jon introduced Mary to x is Sue. To account for this effect of focus on meaning, linguists have developed theories which have in common that they require a function to be built which denotes that part of the utterance which is not in focus (e.g., in (4a) above, the function of Jon introducing x to Sue). [PUL 97, GAR 96a] show that HOU provides an

appropriate tool for computing that function and hence a good basis for a computational treatment of focus.

[GAR 96c, GAR 97, **?**] show that [DAL 91]'s treatment of ellipsis can be generalised to apply to deaccenting and correctly predicts a number of phenomena resulting from the interaction of parallelism (the structural similarity of two semantic representations), anaphora and deaccenting.

Finally, [PIN 96] uses HOU to resolve (i.e. fully specify) under-specified semantic representations. The idea is to assign ambiguous sentences an under-specified semantic representation containing free variables. The value of these free variables is then determined by solving equations using HOU. Typically, HOU will yield several solutions thus capturing the fact that the under-specified semantic representation actually stands for several fully specified ones.

Although HOU has been put to many different uses, the idea underlying the HOU–approach to semantic construction is invariant and can be summarised as follows.

First, a semantic representation is built which contains one or more free variable(s). For instance, a VP-ellipsis will be represented by a free variable of type $e \rightarrow t$ where e is the type associated with individuals and t is the type of truth-values (in the simply typed λ-calculus, predicates are usually represented by functions of type $e \rightarrow t$). Given (5a), this induces a semantic representation as in (5b) where R is the free variable representing the VP-ellipsis *does*.

(5) a. *Jon likes Mary and Peter does too.*

b. *like(j,m)* $\wedge$ *R(p)*

Second, equations are systematically set up which define the value of these free variables. For instance in [GAR 96c, GAR 97, **?**], the proposal is that the interpretation of elliptical sentences such as (5) is subject to a semantic constraint (the *parallelism constraint*) which in essence requires that elliptical and deaccented clauses share a common semantics with the preceding clause. It is assumed that contrastive elements are given (contrastive elements are structurally parallel elements which are semantically conflicting. For instance, in (5) above, *Jon* is contrastive to *Peter*).

Parallelism constraint

Let $\langle S, T \rangle$ be a pair of semantic representations such that S is the semantic representation of the source (i.e., first) utterance and T that of an elliptical or deaccentuated utterance, the target (i.e., second) utterance. Let $S_1, \ldots, S_n, T_1, \ldots, T_n$ be the semantic representations of the source and target contrastive elements respectively; and let C be a free variable of the appropriate type. Then S and T must obey the following constraint:

$$C(S_1, \ldots, S_n) = S$$
$$C(T_1, \ldots, T_n) = T$$

Under this proposal, the equations which determine the value of R in (5b) are:

(6) $\qquad C(j) = l(j, m) \qquad\qquad C(p) = R(p)$

Third, HOU is used to solve the available equations and thereby determine the value of the free variables occurring in the unresolved (i.e., under-specified) semantics. Given the equations in (6) for instance, HOU will return the solution (7a). Applying this substitution to (5b) yields the resolved (i.e., fully specified) meaning (7b), a proposition which correctly captures the meaning of (5a).

(7) a. $\sigma = \{R \leftarrow \lambda X.like(X, m), C \leftarrow \lambda X.like(X, m)\}$

b. *σ(like(j,m) $\wedge$ R(p))= like(j,m) $\wedge$ like(p,m)*

But there is a well-known problem with using HOU for doing linguistics: HOU systematically over-generates. For instance in the treatment of (5a) above, we saw that HOU permits reconstructing the correct meaning for the elliptical verb phrase namely $\lambda Z.like(Z, m)$. Unfortunately, it also yields two additional solutions neither of which is linguistically valid:

Sol. 2 $\{C \leftarrow \lambda Z.l(Z, m), R \leftarrow \lambda Z.l(p, m)\}$
Sol. 3 $\{C \leftarrow \lambda Z.l(j, m), R \leftarrow \lambda Z.l(j, m)\}$

Sol. 2 is linguistically incorrect because it assigns the VP-ellipsis the meaning $\lambda Z.l(p, m)$, a constant function which assigns to all individuals the proposition of Peter liking Mary. *Sol. 3* is incorrect because it resolves the elliptical clause to *Jon likes Mary* while (5) means that *Jon likes Mary and Peter likes Mary*, not: *Jon likes Mary and Jon likes Mary*.

Indeed, the problem is quite general and affects all HOU-based treatments of NL-semantics. As [GAR 96b] shows, it affects [PUL 97, GAR 96a]'s treatment of focus, [DAL 91]'s analysis of ellipsis and [**?**]'s account of deaccenting. For all of these analyses, HOU will over-generate. Intuitively, the reason for this is that HOU allows solutions of the form $\{X \leftarrow \lambda Y.\Phi\}$ where Y does *not* occur in Φ. That is, it allows substitutions which assign constant functions (functions which return the same object whatever their input). Linguistically, this is most unfortunate as the motivation for using HOU in semantics is the need to construct "real" (i.e., non constant) functions. On the face of this, it might seem that a simple way out would be to exclude from the set of linguistically valid solutions these solutions which involve constant functions. But as the following example illustrates, this proposal fails to generalize.

(8) *Jon loves his cat and Peter does too.*

Assuming that *his* refers to Jon, (8) is ambiguous between the following two readings:

(9) *a. Jon loves Jon's cat and Peter loves Jon's cat.*

b. Jon loves Jon's cat and Peter loves Peter's cat.

Traditionally [ROS 67], the first reading is referred to as *strict* (the binding of the pronoun *his* to *John* is preserved when reconstructing the meaning of the second clause) whereas the second is referred to as *sloppy* (the binding is not preserved). Under [GAR 96c, GAR 97, **?**]'s approach, the ambiguity of (9) is captured as follows. First, (8) is assigned the following semantic representation:

$$l(j, c(j)) \wedge R(p)$$

Second, the equations resulting from the parallelism constraint are:

(10) $C(j) = l(j, c(j))$ $C(p) = R(p)$

Given these equations, HOU yields nine solutions, only three of which are linguistically correct: *Sol. 1* yields the strict reading while *Sol. 3* and *6* yield the sloppy reading.

$Sol.1$ $\{C \leftarrow \lambda Z.l(Z, c(j)), R \leftarrow \lambda Z.l(Z, c(j))\}$
$Sol.2$ $\{C \leftarrow \lambda Z.l(Z, c(j)), R \leftarrow \lambda Z.l(p, c(j))\}$
$Sol.3$ $\{C \leftarrow \lambda Z.l(Z, c(Z)), R \leftarrow \lambda Z.l(Z, c(Z))\}$
$Sol.4$ $\{C \leftarrow \lambda Z.l(Z, c(Z)), R \leftarrow \lambda Z.l(p, c(Z))\}$
$Sol.5$ $\{C \leftarrow \lambda Z.l(Z, c(Z)), R \leftarrow \lambda Z.l(p, c(p))\}$
$Sol.6$ $\{C \leftarrow \lambda Z.l(Z, c(Z)), R \leftarrow \lambda Z.l(Z, c(p))\}$
$Sol.7$ $\{C \leftarrow \lambda Z.l(j, c(Z)), R \leftarrow \lambda Z.l(j, c(p))\}$
$Sol.8$ $\{C \leftarrow \lambda Z.l(j, c(Z)), R \leftarrow \lambda Z.l(j, c(Z))\}$
$Sol.9$ $\{C \leftarrow \lambda Z.l(Z, c(j)), R \leftarrow \lambda Z.l(j, c(j))\}$

It is clear that in this case, ruling out solutions involving constant functions does not help.*Sol. 4* and *Sol. 8* do not involve constant functions. Nevertheless, they are linguistically incorrect. *Sol. 4* is incorrect because it assigns to the VP-ellipsis the function $\lambda Z.like(p, c(Z))$ whereas the intended meaning is either $\lambda Z.like(Z, c(j))$ or $\lambda Z.like(Z, c(Z))$. *Sol. 8* is incorrect because it assigns to the VP-ellipsis the function $\lambda Z.like(j, c(Z))$.

The key observation is that different occurrences of the same term may be subject to different constraints. Thus a linguistically plausible constraint that would adequately deal with such examples as (8) is the following:

> **Constraint 1:** The term occurrences representing contrastive elements[2] may occur neither in the common semantics nor in that of the VP-ellipsis.

Applied to example (8), this constraint requires that the term occurrences representing the meaning of the contrastive elements *Jon* and *Peter* be abstracted over in the solution. In other words, both p and the first occurrence of j may not occur in the solution (by contrast, the second may). Under this constraint then, solutions 4 and 8 are ruled out as *Sol. 4* contains p and *Sol. 8* contains the first occurrence of j.

More generally, what this example shows is that an HOU-based approach to semantic construction ought to be able to treat distinct *occurrences* of the same term differently. As we shall see in section 3, this is precisely what Higher-Order Colored Unification (HOCU) gives us: colors are syntactic objects enabling a differentiated treatment of various term occurrences. But this is not quite enough. To see this, consider the following example:

(11) *Jon said that Mary kissed Peter. No, Tim said she did.*

Assuming that *she* in the second sentence refers to *Mary*, the semantic representation and equations associated with this discourse are as follows:

$$s(j, k(m, p)) \wedge s(t, R(m))$$
$$C(j) = s(j, k(m, p)) \qquad C(t) = s(t, R(m))$$

[2]Recall that contrastive elements are those elements in the target utterance which have an overt, semantically conflicting parallel counterpart in the source utterance (the utterance from which it derives its interpretation).

The solutions given by HOU are:

$$\begin{array}{ll} \textit{Sol. 1} & \{C \leftarrow \lambda Z.s(Z, k(m,p)), R \leftarrow \lambda Z.k(Z,p))\} \\ \textit{Sol. 2} & \{C \leftarrow \lambda Z.s(Z, k(m,p)), R \leftarrow \lambda Z.k(m,p))\} \\ \textit{Sol. 3} & \{C \leftarrow \lambda Z.s(j, k(m,p)), R \leftarrow \lambda Z.k(Z,p))\} \\ \textit{Sol. 4} & \{C \leftarrow \lambda Z.s(j, k(m,p)), R \leftarrow \lambda Z.k(m,p))\} \end{array}$$

Linguistically, only *Sol. 1* is correct. Assuming Constraint 1 above would rule out *Sol. 3* and *Sol. 4* because they contain the representation of the contrastive element *Jon*. But *Sol. 2* would still be generated. To rule out this unwanted solution, the following additional constraint needs to be stated:

> **Constraint 2:** The term occurrences representing the subject of a VP-ellipsis and its parallel counterpart may not occur in the representation of that ellipsis.

Importantly, this constraint is very different in nature from Constraint 1. Constraint 1 refers to a pragmatic notion (that of contrastive elements) and therefore should be seen as stemming from the pragmatic component of language theory. In contrast, Constraint 2 evidently originates in the syntax in that it makes reference to such syntactic notions as subject and VP-ellipsis. Moreover note that the two constraints can interact. For instance, in example (8), *Peter* is both the subject of the VP-ellipsis and a contrastive element.

Moreover, as we saw at the beginning of this section, HOU is used for several distinct semantic phenomena (e.g., focus, ellipsis, deaccenting). Now these phenomena can interact. For instance, in (12) below, there is both a focus on *Sarah* in the first clause and a VP-ellipsis *does* in the second clause.

(12) *Jon only likes SARAH. So does Peter.*

To rule out all linguistically invalid readings, constraints pertaining both to focus and to VP-ellipsis needs to be stated and combined within a single HOU framework. It would take us to far afield to discuss the details of such an analysis (but see [GAR 96b] for such details). The important point is that the interaction of linguistic phenomena forces an interaction of constraints which may stem from very distinct linguistic modules. More generally, it appears that semantic construction is subject to multiple, interacting constraints which can stem from distinct linguistic modules. Feature trees are used in computational linguistics to model just these kind of multi-dimensional interactions. Indeed this is the key insight unification-based grammars such as Head-Driven Phrase Structure Grammar [POL 94] or Lexical Functional Grammar [KAP 82] capitalize on. In these grammars, constituent categories are feature trees grouping together various types of information (i.e., syntactic, semantic, phonologic) about the constituent. Rewriting rules can then operate simultaneously on the various levels and capture possible interactions. Therefore, it seems natural to use feature trees to represent the interacting constraints governing semantic construction. In what follows, we show that a version of HOCU which allows colors as feature trees helps us deal with the above data and more generally, with the over-generation problem faced by HOU-based approaches to semantic construction.

3 Colors as atoms

In this section, we introduce a simple version of HOCU in which colors are atoms. We then show how it permits differentiating between term occurrences and thereby provides an appropriate basis for modeling linguistic constraints on semantic construction.

Since the formal presentation of the HOCU-theory involves a lot of technical machinery which we cannot present fully here, we concentrate instead on presenting its underlying intuitions and motivations. For details and proofs, we refer the reader to [HUT 97b, HUT 99].

The colored λ-calculus is a variant of the simply typed λ-calculus [CHU 40], where symbol occurrences can be annotated with so-called *colors* (**color constants** $\mathcal{C} = \{\mathsf{a}, \mathsf{b}, \ldots\}$ and **color variables** $\mathcal{X} = \{\mathsf{A}, \mathsf{B}, \ldots\}$). Colors are indicated by subscripts labeling symbol occurrences. Whenever they are irrelevant, we simply omit them.

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

- **colored constants** $c^\alpha_\mathsf{b}, f^\alpha_\mathsf{a}, f^\alpha_\mathsf{A}, \ldots$ of type α i.e., triples consisting of a constant, a color and a type,
- **colored variables** $X^\alpha_\mathsf{b}, G^\alpha_\mathsf{a}, F^\alpha \ldots$i.e., triples consisting of a variable, a color and a type (these should not be confused with "color variables"). We assume an infinite supply of colored variables for each type and color,
- (uncolored) **placeholders** Z^α, W^α of type α (of which we also assume an infinite supply for each type),
- (function) **applications** of the form $\mathbf{M}^{\beta\to\alpha}\mathbf{N}^\beta$ and
- **λ-abstractions** of the form $\lambda Z^\beta.\mathbf{M}^\gamma$, where Z is a placeholder and the type of $\lambda Z^\beta.\mathbf{M}^\gamma$ is $\alpha = (\beta \to \gamma)$.

We chose the somewhat non-standard notion of bound variables as "placeholders" to emphasize the difference between free and bound variables with respect to colors. This difference is only terminological and does not change the logic itself.

We call a formula $\mathbf{M}$ **well-formed** iff it does not contain unbound placeholders and we call it **c-monochrome** if all constants and variables in $\mathbf{M}$ are annotated by a single color $\mathsf{c} \in \mathcal{X} \cup \mathcal{C}$.

Since we can restrict the supply of colors to a single color constant, the colored λ-calculus is clearly a generalization of the simply typed λ-calculus (see [HIN 86, BAR 84] for an introduction). Therefore we will use various elementary concepts of the λ-calculus, such as **free** and **bound** occurrences of variables or substitutions without defining them explicitly. We will denote the substitution of a term $\mathbf{N}$ for all free occurrences of X in $\mathbf{M}$ with $\{X \leftarrow \mathbf{N}\}\mathbf{M}$.

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 can carry different colors (e.g., $f(X_\mathsf{a}, X_\mathsf{b})$) and that symbols that carry different colors are

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

treated differently. This observation leads to the notion of $\mathcal{C}$**-substitution** (well-colored substitutions), a notion of substitutions 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_{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:

Erasure condition: If A and B are different colors, then $|\sigma(X_{\mathsf{A}})| = |\sigma(X_{\mathsf{B}})|$, where $|\mathbf{M}|$ is the **color erasure** of $\mathbf{M}$ i.e., the formula obtained from $\mathbf{M}$ by erasing all color annotations in $\mathbf{M}$.

Monochromicity condition: If $\mathsf{c} \in \mathcal{C}$ is a color constant, then $\sigma(X_{\mathsf{c}})$ is 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 subterms. In contrast, variable colors do not constrain the substitutions.

Note that since the bound variables (placeholders) do not carry color information, $\beta\eta$-reduction in the colored λ-calculus is just the classical notion:

$$(\lambda X.\mathbf{A})\mathbf{B} \rightarrow_\beta \{X \leftarrow \mathbf{B}\}\mathbf{A} \qquad\qquad \lambda X.\mathbf{C}X \rightarrow_\eta \mathbf{C}$$

where the variable X is not free in $\mathbf{C}$. Note that in particular the substitution $\{X \leftarrow \mathbf{B}\}$ is *uncolored* and there is no applicability condition on either of the reduction relations. Thus we can lift all the known theoretical results from the simply typed λ-calculus to the colored case. In particular, $\beta\eta$-reduction is terminating and confluent in the presence of α-conversion (alphabetic renaming of bound variables, which we consider as built into the system) and we can decide $\beta\eta$-equality by reducing to $\beta\eta$-normal form.

Higher-order unification computes substitutions σ such that $\sigma(\mathbf{M}) =_{\beta\eta} \sigma(\mathbf{N})$ for a given equation $\mathbf{M} = \mathbf{N}$. However, since most of these solutions (called higher-order unifiers) introduce un-necessary instantiations, one is not interested in the set of *all* higher-order 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. In general, the higher-order unification problem is undecidable and there need not be most general solutions to solvable equations. We will discuss a variant of Huet's semi-decision algorithm for HOU and its theoretical and practical implications in section 5.

In the colored λ-calculus the space of solutions is further constrained by requiring the solutions to be $\mathcal{C}$-substitutions: A $\mathcal{C}$-substitution is called a $\mathcal{C}$-unifier of colored formulae $\mathbf{M}$ and $\mathbf{N}$. Note that the instantiation ordering for higher-order unifiers must also be adapted to the colored case by requiring that the substitution ρ be a $\mathcal{C}$-substitution. Thus the higher-order colored unification problem is not just a refinement of the HOU problem. In particular, it is possible, that a colored equation can be unsolvable, even if its color erasure is: Consider the equation $X_{\mathsf{c}}a_{\mathsf{d}} = b_{\mathsf{d}}$, $Xa = b$ has the solution $\{X \leftarrow \lambda Z.b\}$, but the colored variant $\{X_{\mathsf{c}} \leftarrow \lambda Z.b_{\mathsf{d}}\}$ violates the monochromicity condition. Furthermore, it is possible that an equation has a single most general uncolored unifier, but more than one most general colored unifier: Consider for instance the

equation

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

where $\mathsf{a}, \mathsf{b} \in \mathcal{C}$. Obviously, the colored equation has two most general solutions

$$\begin{array}{ll} \textit{Sol. 1} & \{F_{\mathsf{a}} \leftarrow \lambda XYZW.Z, F_{\mathsf{b}} \leftarrow \lambda XYZW.Z\} \\ \textit{Sol. 2} & \{F_{\mathsf{a}} \leftarrow \lambda XYZW.W, F_{\mathsf{b}} \leftarrow \lambda XYZW.W\} \end{array}$$

In constrast, its color erasure has only one most general unifier namely,

$$\{F \leftarrow \lambda XYZW.HZW\}$$

where H is a new variable of appropriate type.

Having introduced the formal framework, let us now go back to the problem discussed in section 2, the problem of unwanted readings. Consider example (8) again repeated here as (13):

(13) *Jon loves his cat and Peter does too.*

In the preceding section, we proposed using the following constraint to eliminate overgeneration:

> **Constraint 1:** The term occurrences representing contrastive elements may occur neither in the common semantics nor in that of the VP-ellipsis.

Within the HOCU framework, this constraint finds a natural encoding. We start by assuming two colors: c (for "contrastive") and n (for "non-contrastive"). We then color the semantic representation of contrastive elements c whereas the variables C and R representing the common semantics and the VP-ellipsis respectively, are colored n. All other symbols are colored with color variables (which are ommitted in the equations). Given the monochromicity condition mentioned above, this guarantees that no contrastive element (i.e., c-colored symbols) occur in the terms assigned to C and R or in other words that these terms abstract over the contrastive elements. In specific, example (8) is analyzed as follows. The (colored) equations triggered by the parallelism constraint are

$$C_{\mathsf{n}}(j_{\mathsf{c}}) = l(j_{\mathsf{c}}, c(j)) \qquad\qquad C_{\mathsf{n}}(p_{\mathsf{c}}) = R_{\mathsf{n}}(p_{\mathsf{c}})$$

for which there are only two legal $\mathcal{C}$-substitutions namely:

$$\begin{array}{ll} \textit{Sol. 1} & \{C_{\mathsf{n}} \leftarrow \lambda Z.l_{\mathsf{n}}(Z, c_{\mathsf{n}}(j_{\mathsf{n}})), R_{\mathsf{n}} \leftarrow \lambda Z.l_{\mathsf{n}}(Z, c_{\mathsf{n}}(j_{\mathsf{n}}))\} \\ \textit{Sol. 3} & \{C_{\mathsf{n}} \leftarrow \lambda Z.l_{\mathsf{n}}(Z, c_{\mathsf{n}}(Z)), R_{\mathsf{n}} \leftarrow \lambda Z.l_{\mathsf{n}}(Z, c_{\mathsf{n}}(Z))\} \end{array}$$

As before (cf. section 2), solution 1 yields the strict reading *Peter loves Jon's cat* whereas solution 3 gives us the sloppy reading *Peter loves Peter's cat*. However, those substitutions which in the uncolored framework yield linguistically incorrect solutions are not legal $\mathcal{C}$-substitutions because they all violate the monochromicity condition on

free variables:

$$\begin{array}{ll}
\textit{Sol. 2} & \{C_{\mathsf{n}} \leftarrow \lambda Z.l(Z, c(j_{\mathsf{n}})), R_{\mathsf{n}} \leftarrow \lambda Z.l(p_{\mathsf{c}}, c(j_{\mathsf{n}}))\} \\
\textit{Sol. 4} & \{C_{\mathsf{n}} \leftarrow \lambda Z.l(Z, c(Z)), R_{\mathsf{n}} \leftarrow \lambda Z.l(p_{\mathsf{c}}, c(Z))\} \\
\textit{Sol. 5} & \{C_{\mathsf{n}} \leftarrow \lambda Z.l(Z, c(Z)), R_{\mathsf{n}} \leftarrow \lambda Z.l(p_{\mathsf{c}}, c(p_{\mathsf{c}}))\} \\
\textit{Sol. 6} & \{C_{\mathsf{n}} \leftarrow \lambda Z.l(Z, c(Z)), R_{\mathsf{n}} \leftarrow \lambda Z.l(Z, c(p_{\mathsf{c}}))\} \\
\textit{Sol. 7} & \{C_{\mathsf{n}} \leftarrow \lambda Z.l(j_{\mathsf{c}}, c(Z)), R_{\mathsf{n}} \leftarrow \lambda Z.l(j_{\mathsf{c}}, c(Z))\} \\
\textit{Sol. 8} & \{C_{\mathsf{n}} \leftarrow \lambda Z.l(j_{\mathsf{c}}, c(Z)), R_{\mathsf{n}} \leftarrow \lambda Z.l(j_{\mathsf{c}}, c(p_{\mathsf{c}}))\} \\
\textit{Sol. 9} & \{C_{\mathsf{n}} \leftarrow \lambda Z.l(j_{\mathsf{c}}, c(j_{\mathsf{c}})), R_{\mathsf{n}} \leftarrow \lambda Z.l(j_{\mathsf{c}}, c(j_{\mathsf{c}}))\}
\end{array}$$

In short, HOCU enables the differentiation of distinct term occurrences and thereby a straightforward encoding of the required linguistic constraints. But as was noted in section 2, this is not sufficient. One must also be able to capture multiple, interacting constraints. This is the object of the next section.

4 Colors as feature trees

We now extend the HOCU specification given in the previous section as follows: instead of having purely *atomic* colors, we allow colors to be feature trees.

Feature trees are trees whose branches are labeled by symbols called **attributes** or **features** and whose leaves are labeled by symbolic **labels**. Branches in feature trees are identified by their feature rather than by their order as in conventional trees.

Feature logics are logical systems for describing feature trees and doing inference with them. There is a variety of feature logics available, developped mostly for applications in linguistics and logic programming languages (see for instance [CAR 92, SMO 92, AÏT 94, MÜL 97]). Feature logics vary for instance in the general expressivity of their languages (e.g., whether and which forms of negation can be expressed), the semantic model of the underlying trees and of course, in the inference methods that are defined for them.

The feature system that we chose for our colors is based on a very simple one namely, Smolka and Treinen's *records for logic programming* [SMO 94]. These records are rational feature trees i.e., trees that may contain cycles but have only finitely many nodes. The only inference method for them that we are interested in is unification which is efficiently decidable (see below).

There are two kinds of labels in the feature trees used by HOCU: constant labels (called **color values** in our setting) and variables (the **color variables**). Features are used to distinguish different symbolic colors or to group colors that are related to each other. For instance, a feature tree might carry an attribute *syntax* whose value is again a feature tree with attributes that are related to linguistic information about syntax.

We will use $\mathsf{a}, \mathsf{b}, \mathsf{c}, \ldots$ as meta-variables for feature trees (colors) and $\mathsf{A}, \mathsf{B}, \mathsf{C}, \ldots$ for variables within feature trees (color variables). Further, we will represent feature trees by their attribute-value matrices. The following are example attribute-value matrices for two equivalent well-formed feature trees:

$$[p +, e -, agr\ [num\ \mathrm{p}, case\ \mathsf{A}]]$$
$$[agr\ [case\ \mathsf{A}, num\ \mathrm{p}], e -, p +]$$

A feature tree can alternatively be described by a set of constraints i.e., a set of formulas that defines which nodes carry which features and which values are associated with them. For instance, the tree $[p +, e\ [case\ \mathsf{A}]]$ can be described by the set

$$\{r_0\langle p\rangle +, r_0\langle e\rangle r_1, r_1\langle case\rangle \mathsf{A}\}$$

where $r\langle p\rangle +$ means that *node r carries value + at feature p*. Here, we use an identifier r_0 for the root node of the tree $[p +, e\ [case\ \mathsf{A}]]$ while r_1 denotes the root node of the subtree $[case\ \mathsf{A}]$.

Feature tree unification is a process that unifies two feature tree under the same root node and decides whether the result denotes a well-formed feature tree. For the records in [SMO 94], this process is a variant of first-order unification which simply decomposes trees and recursively unifies sub-trees until finally comparing/unifying atomic information. Variables in feature trees behave like logical variables in standard first-order unification. For instance, unifying $[p\ \mathsf{A}, e\ \mathsf{A}]$ and $[p +, e -]$ is not possible because A can not simultaneously be bound to $+$ and $-$. As [SMO 94] shows, record unification can be decided in quasi-linear time.

Two feature trees a and b are **compatible** iff they can be unified. The notion of feature trees as constraints allows us to define the concept of a **constraint store**, a set of feature constraints that may refer to several feature trees with different root nodes $r_1, r_2, \ldots, r_n$. A constraint store $\mathcal{CS}$ is **satisfiable** if each set R_i of feature constraints with the same root r_i describes a well-formed feature tree. Again, this constraint solving problem can be efficiently decided.

Naturally, the monochromicity condition, we have imposed on $\mathcal{C}$-substitutions now has to be generalized to feature term compatibility:

> **Generalized monochromicity condition:** A substitution $\{X_{\mathsf{c}} \leftarrow \mathbf{M}\}$ is only well-colored iff every color annotation d in $\mathbf{M}$ is compatible with c.

With colors as feature trees, we can now return to the more complex example introduced in section 2:

(14) *Jon said that Mary kissed Peter. No, Tim said she did.*

To deal with such examples, we saw that two interacting constraints are necessary:

> **Constraint 1:** The term occurrences representing contrastive elements may occur neither in the common semantics nor in that of the VP-ellipsis.
> **Constraint 2:** The term occurrences representing the subject of a VP-ellipsis and its parallel counterpart may not occur in the representation of that ellipsis.

Using feature-tree colors, these constraints can be encoded using the following colors: $[c +]$ (for "contrastive"), $[c -]$ (for "non-contrastive"), $[e +]$ (for "VPE subject or parallel counterpart") and $[e -]$ (for "neither VPE-subject nor parallel counterpart"). We then color the semantic representation of contrastive elements $[c +]$ whereas the variable C representing the common semantics and the variable R representing the VP-ellipsis are colored $[c -]$. Similarly, the terms representing the VPE subject and its parallel counterpart are $[e +]$ colored whereas the VPE variable is $[e -]$ colored. Now example (14) is analyzed as follows. The equations triggered by the parallelism constraint become:

(15)
$$C_{[c\ -]}(j_{[c\ +]}) = s(j_{[c\ +]}, k(m_{[e\ +]}, p))$$
$$C_{[c\ -]}(t_{[c\ +]}) = s(t_{[c\ +]}, R_{[e\ -,c\ -]}(m_{[e\ +]}))$$

And the only $\mathcal{C}$-substitution[4] satisfying these equations is:

(16) *Sol. 1* $\{C_{[c\ -]} \leftarrow \lambda Z.s(Z, k(m_{[e\ +]}, p)), R_{[e\ -,c\ -]} \leftarrow \lambda Z.k(Z, p))\}$

In contrast, the following substitutions are ill-formed as they violate the generalized monochromicity condition defined above.

Sol. 2 $\{C_{[c\ -]} \leftarrow \lambda Z.s(Z, k(m_{[e\ +]}, p)), R_{[e\ -,c\ -]} \leftarrow \lambda Z.k(m_{[e\ +]}, p))\}$
Sol. 3 $\{C_{[c\ -]} \leftarrow \lambda Z.s(j_{[c\ +]}, k(m_{[e\ +]}, p)), R_{[e\ -,c\ -]} \leftarrow \lambda Z.k(Z, p))\}$
Sol. 4 $\{C_{[c\ -]} \leftarrow \lambda Z.s(j_{[c\ +]}, k(m_{[e\ +]}, p)), R_{[e\ -,c\ -]} \leftarrow \lambda Z.k(m_{[e\ +]}, p))\}$

Sol. 2 is ill-formed because it assigns the $[e\ -, c\ -]$-colored variable R a term containing the color annotation $[e\ +]$. Since $[e\ -]$ and $[e\ +]$ fail to unify (the feature e is assigned the two incompatible values + and -), there is a color annotation in the value assigned to R which is not compatible with R's color. Hence the generalised monochromicity condition is violated. Hence the substitution is not a legal $\mathcal{C}$-substitution. Similarly, *Sol. 3* is ruled out as it assigns to $C_{[c\ -]}$ a term containing the color annotation $[c\ +]$ whereby $[c\ -]$ and $[c\ +]$ cannot unify. *Sol. 4* combines the ill-formedness of both *Sol. 2* and *Sol. 3*.

So far, the motivation we gave for having colors as feature trees is that the linguistic constraints which regulate semantic construction can stem from very different linguistic modules. For instance, in the above example, the $[c\ +]$ constraint indicates that an element is a contrastive element. This information (i.e., information about parallelism) stems from the discourse component, that module in the linguistic theory which describes relations between utterances. In contrast, the $[e\ +]$ color signals the subject of a VP-ellipsis and is thus given by the syntactic component i.e., that component which describes the syntax of natural language. Similarly, the $[c\ -]$-colored C variable is introduced at the discourse level whereas the $[e\ -]$-colored R variable representing the ellipsis results from sentence level semantic construction and is thus introduced by the semantic component.

Of course the fact that linguistic constraints may stem from different linguistic modules is no argument in and of itself, for having colors as feature trees. However, the fact that they sometimes overlap and jointly constrain the same piece of semantic information is. So for instance, the ellipsis subject *Peter* in example (13) is both a contrastive element (hence $[c\ +]$-colored) and the subject of the ellipsis (hence $[e\ +]$-colored). If colors are feature trees, this double-labeling simply results from the unification ($[e\ +, c\ +]$) of two feature trees attached to the same piece of semantic information by different modules. In the atomic HOCU framework introduced in section 3, such information would require the use of an additional atomic color say $[ec\ +]$. Although this is certainly technically feasible, it is linguistically rather unsatisfactory as it requires a tight interrelation between two linguistic modules (the syntax and the discourse module) which are normally fairly independent from another: the discourse module when

[4]To improve readability those colors that cannot lead to unification failure have been ignored. For instance, *Sol. 1* really is:
$\{C_{[c\ -]} \leftarrow \lambda Z.s_{[e\ -,c\ -]}(Z, k_{[e\ -,c\ -]}(m_{[e\ +]}, p_{[e\ -,c\ -]})), R_{[e\ -,c\ -]} \leftarrow \lambda Z.k_{[e\ -,c\ -]}(Z, p_{[e\ -,c\ -]}))\}$.

marking contrastive elements must check whether these elements are or not colored $[e\,+]$ by the syntactic component and in the positive case, change this color to $[ec\,+]$. The scenario is all the more unappealing in that it is non-monotonic and forces the over-writing of information.

There are at least two further reasons for having colors as feature trees. First, note that given the above two colors $[c\,+]$ and $[e\,+]$ all the combinations of these colors can be realized namely $[e\,+, c\,+]$,$[e\,-, c\,-]$,$[e\,+, c\,-]$ and $[e\,-, c\,+]$. $[e\,-, c\,-]$ is the color associated with a VP-ellipsis representation, $[e\,-, c\,+]$ that of a contrastive parallel element that is not structurally parallel to the subject of a VP-ellipsis, $[e\,+, c\,-]$ marks e.g., the subject of a VPE that is not a contrastive element and finally, $[e\,+, c\,+]$ is the color associated with an element that is both contrastive and either the subject of an ellipsis or its parallel counterpart.

But such combination paradigms are precisely the sort of information feature tree unification was designed for: instead of postulating a potentially very large set of atomic colors, a small set of features is used which are then combined using unification to yield the much larger set resulting from their combinatorics.

An additional reason for having colors as feature trees is the efficient and straightforward account of under-specification it permits. Consider an example such as (17) where the antecedent clause (the first clause in the text) of the ellipsis contains an element in focus (i.e., a prosodically prominent element) namely, MARY.

(17) *Jon likes MARY. No, TIM does.*

Assuming an HOU-based treatment of focus as advocated in [PUL 97, GAR 96a], the semantic representation of the element in focus will have to be marked by a specific color, say $[f\,+]$. As a result, the equations resulting from the parallelism constraint are:

$$C_{[c\,-]}(j_{[c\,+]}) = l(j_{[c\,+]}, m_{[f\,+]}) \qquad C_{[c\,-]}(t_{[e\,+,c\,+]}) = R_{[e\,-]}(t_{[e\,+,c\,+]}))$$

for which the only $\mathcal{C}$-substitution is:

$$\{C_{[c\,-]} \leftarrow \lambda Z.l(Z, m_{[f\,+]}), R_{[e\,-]} \leftarrow \lambda Z.l(Z, m_{[f\,+]})\}$$

The important point is that the substitution is well-colored even though it assigns to a $[c\,-]$-colored variable a term containing an $[f\,+]$ subterm. Intuitively, this is because feature trees allow us to underspecify constraints with respect to all irrelevant features and to do so in an efficient way. Within an atomic setting, this type of underspecification would lead us to introduce boolean operators. For instance (see [GAR 96b] for details) the constraint that C be $[c\,-]$-colored would be replaced by a negative constraint say $\neg p$ stating that the subterms of the term assigned to C may be labeled with any color except p. In general however, we don't need the full expressivity of propositional logic (which in general has an NP-complete satisfiability problem instead of the quadratic one for feature tree constraints). Therefore, it seems preferable to stay in the more constrained setting of feature tree unification[5].

[5]Another possibility – which we do not explore here – would be to identify a more tractable fragment of propositional logic that is sufficiently expressive for our purpose

5 The HOCU algorithm

5.1 Logical aspects of the algorithm

Since HOCU is the principal computational device of the analysis in this paper, we will now try to give an intuition for the HOCU problem. In this section we will only consider the logical side of the algorithm and leave implementational questions to section 5.3.

It is well-known that for first-order terms there is always a unique 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. Here, a solvable equation can have more than one most general solution and we use this to model ambiguity in the linguistic application.

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 so-called *solved forms*, which have unique and obvious most general $\mathcal{C}$-unifiers that also solve the initial set of equations.

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 $\mathsf{c} =^c \mathsf{d}$ for color equations).

A set $\mathcal{E}$ of equations is in $\mathcal{C}$**-solved** form iff all color equations are of the form $\mathsf{A} =^c \mathsf{c}$, where A occurs exactly once in $\mathcal{E}$ and all term equations are of the form $X_\mathsf{c} = \mathbf{M}^\mathsf{c}$ such that

1. the colored variable X_c occurs exactly once in $\mathcal{E}$,
2. $\mathbf{M}^\mathsf{c}$ is well-formed (i.e., does not contain unbound placeholders) and
3. c-compatible and finally, for all equations $X_\mathsf{d} =^t \mathbf{M}^\mathsf{d}$ in $\mathcal{E}$, the color erasures of $\mathbf{M}^\mathsf{c}$ and $\mathbf{M}^\mathsf{d}$ are equal (i.e., $\{X_\mathsf{c} \leftarrow \mathbf{M}^\mathsf{c}\}, \ldots, \{X_\mathsf{d} \leftarrow \mathbf{M}^\mathsf{d}\}$ is a $\mathcal{C}$-substitution).

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 $\lambda Z.\mathbf{A} =^t \lambda W.\mathbf{B}$ to $\{Z \leftarrow Z'\}\mathbf{A} =^t \{W \leftarrow Z'\}\mathbf{B}$, where Z' is a new place holder of appropriate type, while the rule for applications decomposes $h_\mathsf{a}(\mathbf{A}^1, \ldots, \mathbf{A}^n) =^t h_\mathsf{b}(\mathbf{B}^1, \ldots, \mathbf{B}^n)$ to the set $\{\mathsf{a} =^c \mathsf{b}, \mathbf{A}^1 =^t \mathbf{B}^1, \ldots, \mathbf{A}^n =^t \mathbf{B}^n\}$, provided that h is a constant or a placeholder. The color equations are solved by first-order feature tree unification. Furthermore equations are kept in $\beta\eta$-normal form.

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

Due to the presence of function variables, systematic application of these rules can terminate with equations of the form $X_\mathsf{c}(\mathbf{A}^1, \ldots, \mathbf{A}^n) =^t h_\mathsf{d}(\mathbf{B}^1, \ldots, \mathbf{B}^m)$. Such

equations can neither be further decomposed, since this would lose unifiers (if G and F are variables, then $Ga = Fb$ has a solution $\lambda Z.c$ for F and G, but $\{F = G, a = b\}$ is unsolvable), nor can the right hand side be substituted for X as in a variable elimination rule, since the types would clash. To see what a possible solution would be, let us consider a concrete example, the uncolored equation $X(a) =^t a$ which has the solutions $(\lambda Z.a)$ and $(\lambda Z.Z)$ for X.

The standard solution [HUE 75] 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. For finding all $\mathcal{C}$-unifiers it is sufficient to bind X to the most general terms of the same type as X (otherwise the unifier would be ill-typed) 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 placeholder that enables the head of one of the arguments of X to become the head; the so-called **projection** binding ($\lambda Z.Z$).

If X has type[6] $\alpha = \overline{\beta_n} \to \delta$ and h_{d} has type $\overline{\gamma_m} \to \delta$, then these so-called **general bindings** have the following form:

$$\mathcal{G}^h_\alpha = \lambda Z^{\beta_1} \ldots Z^{\beta_n}.h(H^1(\overline{Z}), \ldots, H^m(\overline{Z}))$$

where the H^i are new variables of type $\overline{\beta_n} \to \gamma_i$. If h is one of the placeholders Z^{α_i}, then $\mathcal{G}^h_\beta$ is called a **projection binding** and else, (h is a constant or a free variable) an **imitation binding**. It turns out (for details and proofs see [SNY 91]) 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_{c} and decomposition of the changed equations (the elimination of x changes $x_{\mathsf{c}}(s^1, \ldots, s^n)$ to $\mathcal{G}^h_{\mathsf{c}}(s^1, \ldots, s^n)$ which has head h).

The general rule for flex/rigid equations in HOCU is just the one for the uncolored case; it simply adds the uncolored equation $X^\alpha = \mathcal{G}^h_\alpha$ to a set containing a flex/rigid equation of the form $\{X_{\mathsf{c}}(\mathbf{A}^1, \ldots, \mathbf{A}^n) = h_{\mathsf{d}}(\mathbf{B}^1, \ldots, \mathbf{B}^m)\}$. Intuitively, this fixes the uncolored information about a particular binding for the head variable x_{c}.

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

Finally, in the only remaining case, the heads of both sides of the equation are free variables; this is 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. Moreover,

[6]For the construction of general bindings, types do play a crucial role, since they e.g., determine the length of the binder.

in contrast to the uncolored case, flex/flex-equations may be unsolvable under a set of color constraints (see [HUT 97b, HUT 99] for a discussion).

Fortunately, for our application, we do not need higher-order unification in full generality, so we will not go into the discussion of flex/flex situations here. In fact, if we look at our examples (e.g., 6), we see that even though they do contain flex/flex equations (in the second equation), the first equation only contains variables on the left-hand side. So here, higher-order unification reduces to the so-called **higher-order matching** problem, which has been proven to be decidable for the subclass of fourth-order formulae [PAD 96] and is conjectured to be for the general case[7]. The examples in this paper actually only require second-order equations, so we know that there is a finite set of most general unifiers for each (higher-order matching problem [HUE 78]). Thus if we start by solving the first equation (obtaining a set of solutions for C), then we can eliminate C from the second equation and reduce this to a matching problem too.

In general though, the methods presented in this paper are not restricted to matching and there are similar linguistic applications (see e.g., [GAR 96c]), where higher-order colored unification is needed.

5.2 An HOCU Example

Before we turn to the implementation, let us work through example (14) and solve the higher-order unification problem given in (15). We start with the first equation:

$$C_{[c\ -]}(j_{[c\ +]}) = s(j_{[c\ +]}, k(m_{[e\ +]}, p))$$

is a flex/rigid situation, so we bind[8] the variable C to the general uncolored imitation binding $\lambda Z.s(H^1(Z), H^2(Z))$. Variable elimination with the induced substitution $\{C_{[c\ -]} \leftarrow \lambda Z.s(H^1_{[c\ -]}(Z), H^2_{[c\ -]}(Z))\}$ yields

$$s(H^1_{[c\ -]}(j_{[c\ +]}), H^2_{[c\ -]}(j_{[c\ +]})) = s(j_{[c\ +]}, k(m_{[e\ +]}, p))$$

and subsequent decomposition the flex/rigid equations

$$H^1_{[c\ -]}(j_{[c\ +]}) = j_{[c\ +]} \qquad H^2_{[c\ -]}(j_{[c\ +]}) = k(m_{[e\ +]}, p)$$

Now let us turn to the first equation: we have two possibilities

imitation with the instantiation binding $\{H^1_{[c\ -]} \leftarrow \lambda Z.j_{[c\ -]}\}$ yields the unsolvable equation $j_{[c\ -]} =^t j_{[c\ +]}$ and thus to failure

projection binding $H^1_{[c\ -]}$ to $\lambda Z.Z$ yields the trivial equation $j_{[c\ +]} = j_{[c\ +]}$ and thus to success.

[7] In general, decidability and complexity results for restricted versions of HOCU can be established with the same methods as those for HOU, since the rules for flex/rigid equations are identical. The only exception is in the flex/flex situations (which are irrelevant e.g., for higher-order matching), where a special treatment is needed. An example is that of higher-order pattern unification [MIL 92], where the unique most general unifier property is lost (there may be finitely many most general unifiers).

[8] We will not pursue choices of bindings that would lead to immediate failure, like the projection binding in this example.

In the second equation, we can only imitate three times and then project twice, eventually binding $H^2_{[c\ -]}$ to $\lambda Z.k(m_{[e\ +]}, p)$, which is a $\mathcal{C}$-substitution, since $[e\ +]$ is compatible with $[c\ -]$. If we collect the successful bindings, we obtain the partial solution

$$\{C_{[c\ -]} \leftarrow \lambda Z.s(Z, k(m_{[e\ +]}, p))\}$$

which we can apply to the second equation in (15) to obtain the equation

$$s(t_{[c\ +]}, k(m_{[e\ +]}, p)) = s(t_{[c\ +]}, R_{[e\ -,c\ -]}(m_{[e\ +]}))$$

which can be decomposed to

$$k(m_{[e\ +]}, p) = R_{[e\ -,c\ -]}(m_{[e\ +]})$$

Imitation for $R_{[e\ -,c\ -]}$ (binding it to $\lambda Z.k(H^5_{[e\ -,c\ -]}(Z), H^6_{[e\ -,c\ -]}(Z))$) gives the equations

$$H^5_{[e\ -,c\ -]}(m_{[e\ +]}) = m_{[e\ +]} \qquad H^6_{[e\ -,c\ -]}(m_{[e\ +]}) = p$$

Again we do not have a choice in this situation, since we are forced by the colors to project for H^5 and by the term part to imitate for H^6. Collecting the bindings, we obtain the unique solution in (16).

5.3 Implementation

In this section we outline an implementation of the HOCU algorithm sketched in section 5. This implementation is used as the main inference module of the natural-language processing system CHOLI [**?**] with which we extensively tested the approach described in sections 3 and 4 and which is described in more detail in section 6.

5.3.1 HOCU as an Algorithm

HOCU as an algorithm is a recursive process of *simplification*, flexible head *elimination* and *color constraint solving*. These sub-processes can be implemented as two separate procedures: a deterministic part, called **Simpl** in Huet's original specification of HOU and **GB** (for **G**eneral **B**indings), that implements the nondeterministic choice of general bindings for flex/rigid equations (called **Match** by Huet).

Simpl mainly decomposes formulas, removes trivial equations of the form $a = a$ and detects "clashes" of the form $a = b$ where a and b are different constants. **Simpl** also treats the trivial case of variable elimination triggered by equations of the form $X = t$ where X does not occur free in term t.

Solving color constraints is deterministic as well: eliminating a variable X by substitution from the set of equations initiates a constraint propagation process that implements the monochromicity condition that we demand for well-colored substitutions (see section 4). As discussed above, color compatibility is defined over feature unification. We store all constraints that define which feature trees must be compatible (unifiable) into a global constraint store $\mathcal{CS}$ and test its satisfiability each time we make a change that could affect $\mathcal{CS}$'s satisfiability.

The nondeterministic part **GB** selects an arbitrary flex/rigid equation from the current set of equations and computes its general bindings (see section 5.1). We do not consider any color information in **GB** (these are handled within **Simpl**) but use the standard process of computing general bindings in HOU.

Like Huet's original algorithm [HUE 75], the HOCU search branches over the finite set of approximated bindings. Hence, HOCU is a search in an OR-tree (see figure 1) where each node is associated with a set of equations, a binding environment for free variables and a color constraint store. The arcs of the search tree are applications of either **Simpl** or **GB**. Disjunctive choices occur when **GB** computes more than one general binding i.e., an imitation I and one or more of n projections $P_1, \ldots P_n$.

Figure 1: A HOCU search tree

The pseudo-code shown in figure 2 sketches the HOCU procedure. HOCU applies **Simpl** to its input set of equations while considering a binding environment E for free variables and a color constraint store $\mathcal{CS}$. If simplification is successful i.e., if no clash between constant heads or color constraints occurs, then HOCU selects an arbitrary flex-rigid equation. **GB** then computes possible approximative bindings for this equation and chooses one with which HOCU continues computation. The actual selection of the approximated binding is done by the overall search strategy.

```
proc HOCU ({s_1 ... s_n}: equations, E: environment, CS: constraints );
    Φ ← Simpl({s_1 ... s_n}, E, CS)
    case Φ = false
        then return fail
        else
            case Φ = ⟨S', E', CS'⟩
                then g ← FlexRigid(Φ)
                case g = false
                    then return Φ
                else
                    choose e_i ∈ GB(g) do
                            HOCU({e_i} ∪ S', E', CS')
                    end
                end
            end
    end
```

Figure 2: The HOCU main procedure in pseudo-code.

5.3.2 An OZ implementation

OZ [Pro 98] is a constraint logic programming language that combines several paradigms

such as higher-order functional programming, constraint logic programming and concurrent objects with multiple inheritance.

The OZ implementation of HOCU [KON 97] makes extensive use of OZ's special capabilities. For instance, the **logical variables** of OZ give us an elegant binding mechanism for free variables in the λ-calculus. OZ's **encapsulated search**, the abstraction of algorithms from their search strategies, helped us to experiment with different search strategies such as depth-first, iterative deepening or even visual search where search trees can manually be explored and compared in a graphical browser [SCH 97].

The implementation is concurrent. Each equation in the set of HOU constraints is treated by a separate process, called a thread in OZ terminology, that tries to solve its own equation. Threads share common data structures such as the constraint store which among other things contains variable bindings. Formula decomposition leads to new threads that again try to simplify and solve sub-formulas. This implementation technique is especially useful for the treatment of flex/flex-equations. If a thread encounters a flex/flex-equation, it simply suspends computation and waits until another thread instantiates one of the flexible heads. In this way, flex/flex-equations can basically be ignored within HOU until they become flex/rigid-equations by instantiation[9]. By using a concurrent implementation, no central control structure is necessary any more for dealing with sets of equations.

For HOCU's comparably simple color feature trees, we use OZ's built-in record constraints. In our implementation, each occurrence of a colored variable and constant carries a data slot which is initially bound to an unspecified OZ record that can be seen as a variable referring to the root node of a feature tree. A binding of a logical variable X to a term $\mathbf{A}$ initializes all colored variables X_{a} with a fresh copy of $\mathbf{A}$ that is colored with a. Color information is applied to terms while decomposition takes place: An equation $c_{\mathsf{a}} = c_{\mathsf{b}}$ is eliminated by unifying the colors a and b.

OZ's open records implement rational (i.e., possibly cyclic) feature trees [SMO 94]. OZ provides a fast unification for records that generalizes the feature unification that we need for HOCU and which propagates information about unified information into a global constraint store.

The global constraint store and the satisfiability check that we need for color constraints are built-in features of the OZ language. While OZ can handle cyclic trees, we have not found any use for this feature in our application—cyclic color information could only come from faulty grammar specifications. Since cyclic records are easy to detect in the output, we have left out an occur check for cycles.

Although efficiency was not our primary concern while implementing HOCU, we reused the code almost unchanged for automated higher-order theorem proving HOT [KON 98]. Higher-order theorem proving is a demanding application for HOU since even small proof problems can create thousands of unification problems that must be solved in a reasonable amount of time. Note that our implementation has been kept simple and does not use complex techniques such as explicit substitutions [DOW].

[9] In particular, this automatically gives the ordering that reduces the HOCU problem to a higher-order colored matching problem discussed at the end of section 5.1.

6 Natural Language Processing with HOCU

Although HOU is a popular tool in the computational linguistic literature on semantic construction, there is to the best of our knowledge no implemented system with which to test and evaluate existing HOU-based analyses. In what follows, we describe the natural language processing system CHOLI [**?**], a system which was developed to fill this gap.

6.1 The CHOLI system

CHOLI integrates standard techniques and tools from computational linguistics and extends them with an HOU/HOCU inference engine. More specifically, the CHOLI system consists of three main components: a dedicated *graphical user interface* which realizes a menu-driven interaction and control of both the linguistic parsing component and the HOCU inference module; a *parsing component* which given a string of words returns a (possibly under-specified) semantic representation for that string; and a *resolution component*, which given an under-specified semantic representation and some equations returns a fully specified semantic representation.

Figure 3: Parsing and resolving in the CHOLI system

The parsing component consists of a lexicon, a Head-Driven Phrase Structure grammar (HPSG) for English and of a chart-parser for that grammar. The lexicon contains around three hundred inflected words and their associated categories while the HPSG grammar covers all the basic constructions of English including: simple declarative sentences, simple sentence coordination, relative clauses, topicalisation and adjuncts. As is traditional in HPSG, the linguistic categories used in the grammar are feature trees. The idea is that each feature in these feature trees represents a different type of linguistic information. In particular, syntactic, semantic and pragmatic information are grouped as values of the three features *synt, sem* and *prag*. Contrary to the HPSG tradition, the semantic representation which is the value of the *sem* feature is not a term (combined with others through unification) but a λ-term (combined with other using β-reduction). In this way, a Montague-like semantic construction process (cf. section 2) can be modeled. As output, the system displays a graphical chart showing the parse tree(s) of the input. The user can then use several menu- and clicking options to see either the full category of a given constituent in the chart or its semantic representation.

The resolution component allows the user to select either HOU or HOCU as the inference procedure and to choose contrastive elements (cf. section 2) from the graphical representation of the parse tree (see figure 3). As a result, the system automatically sets up the corresponding equations and passes these equations to the HOU/HOCU engine which resolves them. Once the equations are resolved, the system returns a window giving: the under-specified semantic representation input by the parsing component, the equations set up with the user's help, the substitution(s) returned by the unifier and the resolved semantic representation (i.e., the under-specified semantic representation

after application of the result substitutions).

6.2 The coverage of CHOLI

CHOLI implements the HOU/HOCU based analyses discussed in [**?**, GAR 96c, GAR 97, GAR 96a, GAR 96b]. More specifically, it encompasses the treatment of VP-ellipsis presented in [DAL 91], the treatment of parallel structures presented in [**?**, GAR 96c], the treatment of focus described in [GAR 96a] and the use of Higher-Order Colored Unification to prevent over-generation.

The test-suite for these analyses is of around one hundred distinct cases and covers most of the challenge examples to be found in the linguistic literature on ellipsis, focus and deaccenting. It is structured as a disjoint set of phenomena that can serve as a scalable test set for semantic analysis systems. Each test-suite example can be tested either with the simple higher-order unifier or with the colored one. In what follows, we briefly describe the functioning of each analysis and the constraining effect of colors.

VP-Ellipsis. [DAL 91] presents an HOU-based treatment of VP-ellipsis which can be summarized as follows. Given a source clause followed by a target clause containing a VP-ellipsis, the semantic value of this ellipsis is determined by the equation $SSem = R(SP_1, \ldots, SP_n)$ where $SSem$ is the semantic representation of the source clause, R that of the target VP-ellipsis and $SP_1, \ldots, SP_n$ the representations of the source contrastive elements. The test-suite for this analysis covers the interaction of VP-ellipsis with anaphora, proper nouns (e.g., *Mary, Paul*) and control verbs (i.e., verbs such as *try* whose subject "control" i.e., is co-referential with some other element in the verb complement).

Deaccenting. As mentioned in section 2, [**?**] generalizes [DAL 91]'s analysis to deaccenting structures. The test-suite for this analysis covers the interaction of deaccenting with: anaphora, VP-ellipsis, context and sloppy/strict ambiguity.

Focus. [GAR 96a] argues that an HOU-based construction of the Ground/Focus partitioning has both computational and theoretical advantages. The basic idea is that given a sentence S with semantics *Sem* and focus *F*, the ground *Gd* of this sentence is determined by the equation *Sem = Gd(F)*. This ground is then used to determine the meaning of S. For instance, given the sentence *John only likes MARY*, the following representations and equations are computed:

1. $like(john)(mary) \wedge \forall x(G(x) \rightarrow x = mary)$
2. $like(john)(mary) = G(mary)$
3. $\{G \leftarrow \lambda x.like(john)(x)\}$
4. $\{G \leftarrow \lambda x.like(john)(mary)\}$
5. $like(john)(mary) \wedge \forall x(like(john)(x) \rightarrow x = mary)$
6. $like(john)(mary) \wedge \forall x(like(john)(mary) \rightarrow x = mary)$

The first line gives an under-specified meaning representation of *Jon only likes Mary*. On the second line, the equation is given which helps determine the underspecified element namely G (the ground). The third and fourth line give the substitutions

returned by HOU and the fifth and sixth line the corresponding fully-specified meaning representations of the input sentence. When HOCU is used instead of HOU only one substitution is returned (namely the third) and subsequently a single resolved semantics is (correctly) output namely the semantics given on line 5.

The test suite for this phenomena includes sentences with varying and ambiguous foci. It is currently being extended to sentences with multiple foci and the interaction with deaccenting.

Colors. As already mentioned all of the examples in the test suite can be tested with or without colors. Three colors are used: c (for (non)-contrastive), e (for (non)-VP subject), f (for (non)-focus). In all cases, colors filter out unwanted readings, the number of which varies depending on the analysis and on the sentence being analyzed. Here are some examples.

(18) a. *John runs and Peter does too (***2** → **1**,**1** → **1***)*

b. *John saw his mother and Peter did too (***4** → **2**,**6** → **2***)*

c. *John tried to run and Peter did too. (***4** → **1**,**8** → **1***)*

d. *John tried to like his mother and Peter did too. (***8** → **2**,**12** → **2***)*

e. *John's mother says that John runs. Peter's mother does too. (***2** → **1***)*

The annotation **N** → **M** at the end of each example indicates the constraining effect of colors on the number of generated readings: **N** is the number of readings generated by the HOU-approach, **M** the number of readings generated by the HOCU-approach. The first **N** → **M** pair relates to the Dalrymple, Shieber & Pereira analysis, the second to the deaccenting account. In each case, **M** is the number of linguistically correct readings. In other words, colors cuts down the number of generated readings to exactly these readings which are linguistically acceptable.

7 Conclusion

We have advocated the use of HOCU for doing semantic analysis in contrast to HOU as was so far used in computational linguistics. So far, the motivation we gave for using HOCU is that it enables rstricting over-generation. But there are several additional reasons for using HOCU rather than some other means.

First, HOCU was not specifically developed for linguistics. Rather it was developed for guiding the proof search of inductive theorem provers and is as such an off-the shelf mechanism with well-studied mathematical and computational properties. Importantly, this mechanism has both a declarative and a procedural interpretation. This enables linguists to both state linguistic problems declaratively (in terms of equations) and test these analyses on a computational implementation which in essence builds on Huet classic HOU's algorithm ([HUE 75]).

Second, as shown in section 3, the HOCU–framework is a fairly conservative extension of the HOU one. In particular, the colored λ-calculus is a simple generalization of the simply typed λ–calculus. This is important as it means that linguistic results which build on the use of the λ–calculus and of HOU can be preserved. By contrast the use of some other logic or mechanism would require 'starting from scratch' again: all the

insights embodied in the various HOU-based analyses of natural language semantics would need to be recast within this particular logic or with respect to this particular mechanism.

Third, the HOU-approach is (despite the worst-case complexity predicted by the theoretical results) tractable in practice. In all of the examples treated in CHOLI, the computation of *all* higher-order unifiers is a sub-second task on a medium-sized PC. By contrast, if some other logic such as intuitionistic or linear logic were used for semantic construction, automated theorem proving techniques for these logics would have to be resorted to, which are currently far less efficient, if they exist at all.

Finally, we want to emphasize that logically, the notion of adding colors to symbol *occurrences* in a logical system constitutes a fundamentally new logical concept (it was introduced by Dieter Hutter for first-order logic; see [HUT 97a] for details and references) that adds the necessary expressivity to encode an interface to extra-semantical information into the logical system. In this respect it is similar to the idea of labeled deduction systems (LDS [GAB 96]) with which it shares basic intuitions. Both use annotations to restrict the applicability of inference rules and provide a mechanism for maintaining the annotations during the inference. However, while LDS attach labels to formulae, HOCU annotates symbol occurrences with colors. It seems plausible that colored logics can be embedded into suitable LDS if we assume that the labels have the same algebraic structure as the formulae they are attached to. Moreover, any LDS that deals with equality will probably need to maintain labels in such a term-structured form, since equality operates on subterm occurrences which have to be represented in some way. Interestingly, the colored λ-calculus allows one to deal with labels and formulae in a uniform and efficient way taking advantage of the common structure of both. We leave a formal analysis of the relation between LDS and colored logics to further research.

Acknowledgments

The results reported here owe much to clarifying discussions with Martin Müller and Dieter Hutter. We would also like to thank the anonymous referees for their comments. Finally, we thank Stephan Thater, Ralf Debusman and Anouk Perquin for their contribution to the implementation of CHOLI. For financial support, we gratefully acknowledged the Deutsche Forschungsgemeinschaft (DFG) in Sonderforschungsbereich SFB–378, Project C2 (LISA). Claire Gardent would like to thank Aravind Joshi and the staff of the Institute for Research in Cognitive Science, University of Pennsylvania, for their hospitality while the final version of this paper was being prepared. Michael Kohlhase is similarly grateful to Jan van Eijck and the Centrum for Wiskunde and Informatica (CWI).

Bibliographie

[AÏT 94] Aït-Kaci H., Podelski A.Smolka G., A Feature Constraint System for Logic Programming with Entailment . *Theoretical Computer Science*, 122, 263–283, 1994.

[BAR 84] Barendregt H. P., *The Lambda Calculus*. North Holland, 1984.

[CAR 92] Carpenter B., *The Logic of typed feature structures*. Cambridge University Press, 1992.

[CHU 40] Church A., A Formulation of the Simple Theory of Types . *Journal of Symbolic Logic*, 5, 56–68, 1940.

[DAL 91] Dalrymple M., Shieber S.Pereira F., Ellipsis and Higher-Order Unification . *Linguistics & Philosophy*, 14, 399–452, 1991.

[DOW] Dowek G., Hardin T.Kirchner C., Higher-Order Unification via Explicit Substitutions (Extended Abstract) . 366–374.

[GAB 96] Gabbay D., *Labelled Deductive Systems*. 33 Oxford Logic Guides. Oxford University Press, 1996.

[GAR 96a] Gardent C.Kohlhase M., Focus and Higher–Order Unification . *Proceedings of the 16th International Conference on Computational Linguistics*, Copenhagen, 1996.

[GAR 96b] Gardent C.Kohlhase M., Higher–Order Coloured Unification and Natural Language Semantics . *Proceedings of the 34th Annual Meeting of the ACL*, Santa Cruz, 1996.

[GAR 96c] Gardent C., Kohlhase M.van Leusen N., Corrections and Higher-Order Unification . *Proceedings of KONVENS'96*, 268–279, Bielefeld, Germany, 1996. De Gruyter.

[GAR 97] Gardent C., Sloppy Identity. Retoré C., , *Logical Aspects of Computational Linguistics*, 188–207. Springer, 1997.

[HIN 86] Hindley J.Seldin J., *Introduction to Combinators and Lambda Calculus*. Cambridge University Press, 1986.

[HUE 75] Huet G. P., A Unification Algorithm for the Typed λ-Calculus . *Theoretical Computer Science*, 1, 27–57, 1975.

[HUE 78] Huet G. P.Lang B., Proving and applying Program Transformations expressed with Second Order Logic . *Acta Informatica*, 11, 31–55, 1978.

[HUT 97a] Hutter D., Colouring Terms to Control Equational Reasoning . *Journal of Automated Reasoning*, 18, 399–442, 1997.

[HUT 97b] Hutter D.Kohlhase M., A Coloured Version of the λ-Calculus . *Proceedings of CADE'97*, 291–305, 1997.

[HUT 99] Hutter D.Kohlhase M., Managing Structural Information by Higher-Order Colored Unification . *Journal of Automated Reasoning*, 1999. forthcoming.

[KAP 82] Kaplan R.Bresnan J., Lexical-Functional Grammar: A formal system for Grammatical Representation. *The Mental Representation of Grammatical Relations*, 173–280. MIT Press, 1982.

[KON 97] Konrad K., 1997. ColorLambda: An implementation of the simply typed λ-calculus in Oz. `http://www.ags.uni-sb.de/~konrad/soft.html`.

[KON 98] Konrad K., HOT: An Automated Theorem Prover based on Higher-Order Tableaux . Seki Report SR-98-03, Fachbereich Informatik, Universität Saarbrücken, 1998. To appear in: TPHOLs'98: The 11th International Conference on Theorem Proving in Higher Order Logics. Springer Verlag LNCS, 1998.

[MIL 92] Miller D., Unification under a mixed Prefix . *Journal of Symbolic Computation*, 14, 321–358, 1992.

[MON 74] Montague R., The Proper Treatment of Quantification in Ordinary English. Thomason R., , *Formal Philosophy. Selected Papers*. Yale University Press, New Haven, 1974.

[MÜL 97] Müller M., Niehren J.Podelski A., Ordering Constraints over Feature Trees . Smolka G., , *Proceedings of the Third International Conference on Principles and Practice of Constraint Programming*, 1330 *Lecture Notes in Computer Science*, 297–311, Schloss Hagenberg, Linz, Austria, 1997. Springer-Verlag.

[PAD 96] Padovani V., *Filtrage d'order supérieur* . Thése de doctorat, Université Paris VII, 1996.

[PIN 96] Pinkal M., Radical underspecification . Dekker P.Stokhof M., , *Proceedings of the 10th Amsterdam Colloquium*, 587–606, Amsterdam, 1996. ILLC.

[POL 94] Pollard C.Sag I., *Head-driven Phrase Structure Grammar*. CSLI and University of Chicago Press, 1994.

[Pro 98] Programming Systems Lab Saarbrücken, 1998. Oz Webpage: `http://www.ps.uni-sb.de/oz/`.

[PUL 97] Pulman S., Higher Order Unification and the Interpretation of Focus . *Linguistics & Philosophy*, 20, 73–115, 1997.

[ROS 67] Ross J., *Constraints on Variables in syntax* . PhD thesis, MIT, 1967.

[SCH 97] Schulte C., Oz Explorer: A Visual Constraint Programming Tool . Naish L., , *Proceedings of the Fourteenth International Conference on Logic Programming*, 286–300, Leuven Belgium, July 1997. MIT Press.

[SMO 92] Smolka G., Feature Constraint Logics for Unification Grammars . *Journal of Logic Programming*, 12, 51–87, 1992.

[SMO 94] Smolka G.Treinen R., Records for Logic Programming . *Journal of Logic Programming*, 18, 3, 229–258, 1994.

[SNY 91] Snyder W., *A Proof Theory for General Unification*. Progress in Computer Science and Applied Logic. Birkhäuser, 1991.