Radical Underspecification[1]

Manfred Pinkal
Universität des Saarlandes

1 Three Levels of Semantic Underspecification

Natural language can be conceived of as a system of rules or principles that serve the purpose to assign semantic representations to utterances on the basis of a given amount of linguistic information, which can be read off the utterance, plus situational or contextual information in the widest sense. An utterance is vague if the semantic representation assigned to it does not completely determine its denotation. It is semantically underspecified, if the given linguistic and contextual input information is not sufficient to uniquely determine a semantic representation. Formalisms for underspecified semantic representation offer techniques for the compact description of the possible range of representations, as an alternative to the mere disjunction of readings. Their usefulness is evident, if there are methods available for direct further processing of underspecified representations. A typical case may be semantic transfer rules in Machine Translation, which in certain cases bring about appropriate translations by associating the underspecified source language representation with an equally underspecified target language representation. Another possible use for underspecified representation is in modelling the process of utterance interpretation: an utterance may, e.g., be processed by first interpreting it in complete absence of contextual information, which usually leads to an enormous degree of underspecification, and then reducing the range of possible readings by adding non-linguistic information of different kinds.

Different kinds or levels of underspecification can be distinguished according to their structural properties, which pose different problems for the design of an underspecification formalism. On a first level, we have cases of local underspecification in an otherwise uniquely given global semantic structure. Typical examples are lexical ambiguities (like *pen* and *light*), referential ambiguities, as they occur with anaphoric or deictic uses of pronouns and with definite descriptions, and cases of relational vagueness, for which the genitive attribute (like *John's book*) is a typical case. These cases, which we will refer to as *Level-1 Underspecification* in this paper, have been the subject of extensive discussion, particularly with respect to their compositionality properties on the level of denotations (cf. Pinkal 1985, Pinkal 1995).

Scope ambiguities, on the other hand, as the one given in Example (1), are the classical representatives of a second level of semantic underspecification: In these cases, the global semantic structure of the representation is underspecified. Other phenomena falling into this category are focus-ground distinctions, and the distinction between collective and distributive readings. Examples are (2) and (3), respectively.

(1) *Every student did not pay attention*
(2) *Only intelligent students are successful*
(3) *Three boys lifted four pianos*

These *Level-2 phenomena* have been intensively discussed in the last years. No full and adequate denotational semantics has been developed for them. The lack of a

1. The work reported here arose partly from experience with dialogue translation in the Verbmobil project (funded by BMBF, German Ministery for Education and Research) and partly from many discussions in the FraCaS project (funded by the EC in the LRE program). It was also inspired by work of Reinhard Muskens and Wojciech Skut. I am especially indepted to Michael Kohlhase for hints and helpful comments concerning technical aspects.

unique underlying structure of representations is a hard problem for a compositional denotational interpretation. However, there is a variety of similar attempts available, which treat underspecification on the representational level (cf. QLF, Alshawi and Crouch 1992, UDRT, Reyle 1993, LUD, Bos 1996, MRS Egg and Lebeth 1995, Poesio 1994, Muskens 1995, van Eijck 1996). The common idea in most of these approaches is to provide a partial description of semantic representations in terms of constraints using meta-variables. As an example, we give a Reyle-style scope-underspecified representation of sentence (1) in (4), which covers the two specific scope readings of (5). The representation uses so-called labels l_i as meta-variables, and an additional relation of semantic subordination as a relation between labels.

(4) $\{l_1 : \forall x(student(x) \rightarrow l_2), l_3 : \neg l_4, l_0 : pay\_attention(x),\ l_1 \leq l_T,\ l_2 \leq l_T,$
$l_0 \leq l_2,\ l_0 \leq l_4\}$

(5) $\{\forall x(student(x) \rightarrow \neg pay\_attention(x)),$
$\neg\forall x(student(x) \rightarrow pay\_attention(x))\}$

On the level of representation, Level-1 cases can be treated as trivial subcases of Level 2. The pair of readings in (8) for Sentence (6) is described by (7), where the first constraint gives a complete structural description containing a meta-variable to represent the lexical ambiguity.

(6) *Bill owns a pen*

(7) $\exists x(X(x) \wedge own(b, x)), X \in \{pen_1, pen_2\}$

(8) $\exists x(pen_1(x) \wedge own(b, x)), \exists x(pen_2(x) \wedge own(b, x))$

There are additional underspecification phenomena, however, which cannot be subsumed under one of the two classes described above: Semantic underspecification induced by ambiguous, incoherent, or incomplete grammatical (i.e., extra-semantic) linguistic information. We give examples for the different subcases of these *Level-3 phenomena*, in the following, and will discuss their specific properties and problems afterwards.

Ambiguous syntactic information:

(9) *John saw the man with the telescope*

(10) *Die Linguistin zitiert das Buch*
$[_{NPnom \vee acc}$ *Die Linguistin*$]$*zitiert*$[_{NPnom \vee acc}$ *das Buch*$]$
The linguist cites the book / The book cites the linguist

(9) exemplifies the well-known attachment ambiguity. In (10), the grammatical function of the NP complements are underspecified, since subject and object can take the sentence-initial position in German, and the morphological information attached to the NPs *die Linguistin* and *das Buch* is not sufficient to identify case. This grammatical underspecification induces semantic underspecification with respect to the association of terms with argument positions.

Incoherent linguistic information:

(11) * *Den Logiker zitiert den Aufsatz*

(11) is a typical example in spoken-language processing: Speech recognition has mistakenly identified both articles as accusative forms, which leaves us in an information state that precisely corresponds to the information in example (10): We know that it is a transitive construction with a subject and a direct object, but we do not know which is which.

Incomplete linguistic information:

(12) _ *Logiker zitiert* _ *Aufsatz*

(13) ... *Logiker* ... *zitiert* ... *Aufsatz* ...

(12) indicates the case where the article positions could not be identified (with the same effect, again, as in (10) and (11)). (13) is an example for the much more

radical case where some single words have been recognized, but there is no explicit information at all available about the structure of the utterance, and about the role which the words play in that structure, which typically happens if an utterance is made in a noisy environment, or if keyword spotting techniques are used in an artificial speech recognition system.

It is evident that these *radical underspecification* phenomena are important. In particular, a treatment of cases like (11)-(13) is a necessary prerequisite for semantic processing of spoken input in any "real-world" application. Unfortunately, in sharp contrast to the variety of frameworks available for Level-1 and Level-2, Level-3 phenomena rarely have been taken notice of[2]. Also, there is no straightforward way of extending formalisms for scope underspecification to Level-3 cases. Unlike scope phenomena, these cases immediately require expressing semantic information of arbitrary degrees of underspecification: in the extreme case (13) we just know that the semantics of an identified subexpression contributes in some way or the other to the semantics of the full utterance, but we do not know for sure in which way it contributes. Also, Level-3 phenomena raise the question what the notion of compositional analysis might mean at all—even at the level of representation—given that there is no uniquely identifiable underlying syntactic structure for the utterance to be analyzed.

Our proposal is an attempt to solve the task of the representation of arbitrary underspecified semantic information. We will employ meta-variables that range over semantic operations, in addition to the meta-variables for object-language expressions used in the scope underspecification formalisms. The use of operator variables will also provide a solution to the task of semantic composition. Scope underspecification (as Level-2 phenomena in general) will turn out to be a special case of a much more general class of ambiguity cases.

In Section 2, we give a first illustration of how our higher-order constraint account works, looking at a simple example of syntactic ambiguity. In Section 3, we discuss which kinds of restrictions on our description language may be necessary or useful. Section 4 specifies an operational semantics for the constraint language in terms of Linear Unification, a modification of Huet's Higher-Order Unification Algorithm. In Section 5, we demonstrate the application of the formalism to syntactic and scope ambiguity examples. Section 6 concludes the paper with a discussion of further possible applications and required extensions of the formalism.

2 Underspecification with Higher-Order Constraints

Let us start with the simplest case. Take $X = walk'@j^*$ to be the fully specified semantic description of the object language representation $walk'(j^*)$ for the sentence *John walks*. The operation of functional application is explicitly represented by the (left-associative) operator @, taking object language functor and argument as its left and right argument, respectively. By means of meta-variables X and Y, we can form the underspecified descriptions $X = walk'@Y$ and $X = Z@j^*$, which are intended to specify the sets of all expressions of the form $walk'(\alpha)$, and $\beta(j^*)$, respectively, where α and β are some object-language expressions of appropriate type. Note that Y and Z are supposed to take expressions of different OL types as values, but both are first-order variables of the description language. Adding second-order variables C, C' to the description language also allows to underspecify the semantic operation, as in $X = C(walk', j^*)$ and $X = C'(walk')$. By substituting the operator variables C and C' with $\lambda XY.X@Y$ and $\lambda X.X@j^*$, respectively, we obtain $X = walk'@j^*$. However, $X = C'(walk')$ is supposed to cover not only the

2. In fact it has been an observation by Wojciech Skut which led my attention to the higher order character of Level-3 phenomena the first time. Cf. Skut 1995

$walk'(\alpha)$ cases, but in addition those cases where $walk'$ takes the argument position of a functional application, as in $every'(man')(walk')$, or appears arbitrarily deeply embedded either in the functor part of a function application, or in its argument part, or in an argument of any other semantic operation.

As object language, we will use the language of extensional type theory throughout this paper, for the sake of simplicity. Note, however, that our approach is not bound to any specific semantic representation language. We could have used any compositional semantic formalism instead, e.g., CDRT (Muskens 1993), λ-DRT (Kuschert 1996), or a compositional version of Situation Semantics (Cooper 1993).

We will call our description language "USDL" (Underspecified Semantic Description Language). Its inventory is given in (14).

(14) Symbols of USDL are:

(i) Logical constants: λ, $=$, and generalized conjunction, which is used to form finite conjunctive constraint sets (the latter being written in the usual set notation, without an explicit operator symbol).

(ii) A set of non-logical individual constants CON_0 denoting constants and variables of the object language, and sets of n-place function constants CON_n, denoting object-language operations (e.g., $@ \in CON_2$ for function application, $L_v \in CON_1$ for every OL variable v, denoting abstraction over v).

(iii) A set of first-order variables VAR_0 (written as X, Y, Z,...) ranging over object language expressions, as well as sets of n-place second-order variables VAR_n for every $n \in N$ (written as C_1, C_2,...).

If we take USDL to be a reduced version of the language of typed λ-calculus, the notion of a well-formed expression follows immediately from (14). We designed the language as lean as possible - no Boolean operations except conjunction, no quantifiers - to keep the formalism simple at first. Although it is rather certain that we will need extensions of USDL for more complex examples (see Section 6 below), we will stick to the inventory given in (14) in this paper.

We illustrate the way we employ the description language with (15), a simple example of syntactically induced underspecification. (15) has the two possible syntactic analyses in (16), which lead to the type-theoretic representations given in (17).

(15) *former professor in Amsterdam*

(16) (a) $[_{N'}\ [_A\ former]\ [_{N'}\ [_N\ professor\ ]\ [_{PP}\ in\ Amsterdam]]]$

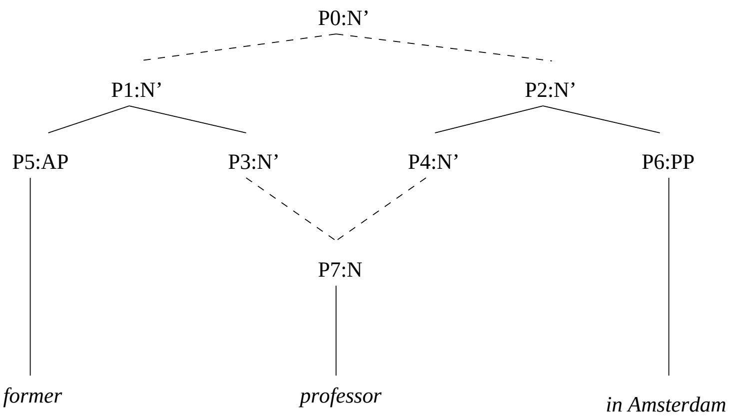
(b) $[_{N'}\ [_{N'}\ [_A\ former]\ [_N\ professor\ ]]\ [_{PP}\ in\ Amsterdam]]$

(17) (a) $former'@(in\_Amsterdam'@professor')$

(b) $in\_Amsterdam'@(former'@professor')$

In the following, we will demonstrate how underspecified semantic descriptions are assigned to underspecified syntactic representations. As a prerequisite, we need a method for compact representation of syntactic ambiguity. We use the general dominance relation $\triangleleft^*$ (reflexive and transitive closure of immediate dominance; see Backofen et al. 1995), which allows us to describe the syntactic information common to (16a) and (16b) in (18), graphically represented as a "quasi tree" in (19):

(18) $\{N'(P_0), N'(P_1), N'(P_2), N'(P_3), N'(P_4), AP(P_5), N(P_7), PP(P_6),$
$P_0 \triangleleft^* P_1, P_0 \triangleleft^* P_2, P_1 \triangleleft P_5, P_1 \triangleleft P_3, P_2 \triangleleft P_4, P_2 \triangleleft P_6, P_3 \triangleleft^* P_7, P_4 \triangleleft^* P_7\}$

(19)


Our description language allows for a straightforward "quasi-compositional" interpretation of the quasi tree, by forming a constraint set according to the rules given in (20). We take X_i to be the semantic representation for node P_i. For the time being, we use object language constants and variables as meta-language type-0 constants (denoting themselves).

(20) (i) For every lexical node P_k, add $X_k = \alpha$, where α is the name for the object language constant representing P_k's lexical semantic value.

(ii) For each binary branching local tree, add a constraint that identifies the semantics of the mother node with the result of the function application between its daughters.

(iii) For each pair of nodes P_i and P_j for which the dominance relation is stated, we add the constraint $X_i = C(X_j)$

The interesting clause of (20) is (iii), introducing constraints which express that the semantic value of P_i is the result of the application of some (otherwise unspecified) semantic operation to the semantic value of P_j.

Application of (20) to (18)/(19) yields the constraint set (21), which can be transformed to (22) by simple substitutions.

(21) $\{X_5 = former', X_6 = in\_Amsterdam', X_7 = professor',$
$X_1 = X_5@X_3, X_2 = X_6@X_4, X_0 = C_1(X_1), X_0 = C_2(X_2), X_3 = C_3(X_7), X_4 = C_4(X_7)\}$

(22) $\{X_0 = C_1(former'@C_3(professor')),$
$X_0 = C_2(in\_Amsterdam'@C_4(professor'))\}$

(22) gives a compact underspecified description of the semantically ambiguous expression (15), saying that the semantic value of X_0 is the result of applying some (otherwise unspecified) semantic operation to $former'(\alpha)$, where α results from application of some other semantic operation to $professor'$, and at the same time the result of applying some semantic operation to $in\_Amsterdam'(\beta)$, where β results from application of some other semantic operation to $professor'$. By appropriate substitution of the function variables, it can be verified that the two readings (17a) and (17b) are in fact covered by the description. Application of (23a) to (22) leads to (17a) as value for X_0, whereas by applying (23b) we obtain (17b).

(23) (a) $\{C_1 \leftarrow \lambda X.X, C_2 \leftarrow \lambda X.former'@X,\ C_3 \leftarrow \lambda X.in\_Amsterdam'@X,$
$C_4 \leftarrow \lambda X.X\}$

(b) $\{C_1 \leftarrow \lambda X.in\_Amsterdam'@X,\ C_2 \leftarrow \lambda X.X,\ C_3 \leftarrow \lambda X.X,$
$C_4 \leftarrow \lambda X.former'@X\}$

3 Additional Restrictions on the Formalism

Straightforward and intuitively appealing as the treatment of the example is, it is far from guaranteeing adequate results. The constraint set (22) covers the two wanted solutions of our example. However, as brief consideration shows, (22) covers a wide variety of undesirable readings at the same time. There are several different sources for inadequate results.

(24) $\{C_1 \leftarrow \lambda X.it\_rains', C_2 \leftarrow \lambda X.it\_rains'\}$

First, the substitution (24) renders $X_0 = it\_rains'$ as a solution for (22). Given our formalism, as it has been defined so far, the solution goes through, although it is completely unrelated to the semantic information which is encoded in the constraint set. This means that we have to take care that operator variables be instantiated only in a way that preserves the information contained in the arguments. We will come back to the requirement in a minute, and see that it has wide-reaching consequences. In advance, we would like to briefly discuss two other problems that arise for our approach.

One of the problems is that we have to take object-language type compatibility into account as an additional source for restrictions, because unconstrained substitution may lead to solutions that describe ill-formed object-language expressions.

(25) $\{X_0 = X@Y, X_0 = C_1(walk'), X_0 = C_2(j^*)\}$

(25) should contain sufficient information to determine $walk'@j^*$ as the unique meaning representation, given closure information. However, we get the ill-formed $j^*@walk'$ as well. We will not provide an explicit treatment for OL type information, here, but rather take it that OL type information can be integrated as sortal information in the description language. We assume that OL type constraints will be reflected in terms of sort descriptions in USDL in roughly the following way: Type-0 constants are assigned their OL type; type-0 variables are in general assigned Top, but type information contained in NL syntactic categories may be reflected in sort restrictions for the associated USDL variables; a USDL function constant like the two-place @, e.g., is assigned $(<\sigma, \tau>, \sigma) \rightarrow \tau$ as sort description, where σ, τ are variables ranging over sorts. OL type clashes do not affect syntactic well-formedness of DL expressions, but lead to the assignment of a designated semantic value (of sort Bottom). We hope that computation of the kind of sort information we need can be integrated into our constraint solving mechanism using standard first-order unification. We have not looked into the topic in greater detail, however.

There is a further problem which, however, is in fact a minor one. Instead of the substitutions in (23), we could have chosen substitutions that introduce function variables into any position in the expression. The most general readings indeed seem to be (27a) and (27b), which are generated by (26a) and (26b), respectively.

(26) (a) $\{C_2 \leftarrow \lambda X.C_1(former'@C(X)),$
$C_3 \leftarrow \lambda X.C(in\_Amsterdam'@C_4(X)\}$

(b) $\{C_1 \leftarrow \lambda X.C_2(in\_Amsterdam'@C(X),$
$C_4 \leftarrow \lambda X.C(former'@C_3(X))\}$

(27) (a) $X_0 = C_1(former'@C(in\_Amsterdam'@C_4(professor')))$

(b) $X_0 = C_2(in\_Amsterdam'@C(former'@C_3(professor')))$

This inadequacy is not due to the formalism itself, but results from the way in which the input information has been presented. The syntactic constraint set (18), which served as the basis for the semantic constraints in (21) and (22), respectively, contains positive statements about syntactic relations only. Thus, it does not exclude the existence of additional intervening linguistic material. Accordingly, the semantic constraints tell us only about the relative scope ordering of the OL constants *former'*, *in_Amsterdam'*, and *professor'*, and do not exclude additional semantic objects to appear at any level of the utterance representation. If we assume that the linguistic material is completely covered by the syntactic constraints, we are entitled to perform a closure operation, i.e. instantiate each function variable in the solutions with $\lambda X.X$, which, in the case of our example, takes us to the wanted variable-free representations in (17).

We come back to the first problem, which was illustrated in (24). The moral of the example is that no vacuous abstraction, as in $\lambda X.it\_rains$, should be allowed: NL semantic operations can do a variety of different things to their arguments, but they do not delete them or overwrite them with different material.

It also seems to be desirable to exclude uncontrolled proliferation of semantic objects due to copying. Take for example constraint set (28), which corresponds to an information state where the word forms *John* and *likes* have been recognized, and like has been identified as main verb of the sentence.

(28) $\{X_0 = like'@Y@X, X_0 = C(j^*)\}$

$X_0 = like'@C(j^*)@X$ and $X_0 = like'@Y@C(j^*)$ are fine, as well as the special cases $X_0 = like'@j^*@X$ and $X_0 = like'@Y@j^*$, however not $X_0 = like'@j^*@j^*$, a solution gained by substituting C with $\lambda X.like@X@X$. This seems to indicate a general need to exclude multiple occurrences of bound variables in function terms, and a justification for a general linearity constraint. However, the situation is somewhat less clear than in the deletion case, for some kind of copying seems to be required in NL semantics, to model, e.g., coordination and ellipsis constructions: The VP *sing and dance* should have a representation like $\lambda x.sing'(x) \wedge dance'(x)$.

Our policy of distinguishing between the levels of object language and description language suggests the following view: OL abstraction and USDL abstraction are completely different notions. In fact, the USDL representation of $\lambda x.sing'(x) \wedge dance'(x)$ should be something like $L_x(Conj(sing'@x, dance'@x))$, where L_x and $Conj$ are USDL function constants referring to the OL operations of abstraction with variable x and conjunction, respectively. Object-language λ-abstraction can take any form that may be required by NL meaning structure. On the other hand, USDL λ-terms are required to be linear throughout: Each variable abstracted over occurs exactly once in the body of the λ-expression. It is an interesting question how much of OL λ-conversion can be integrated into the meta-language equality concept. This question is, however, beyond the scope of this paper.

Linearity makes the number of occurrences of semantic objects a significant feature of the constraint set, since it guarantees that description language β-reduction never involves a change in the number of occurrences of a certain expression. Therefore, the difference between multiple reference to the same occurrence and reference to multiple occurrences of an OL expression becomes significant. To make the difference expressible, we take type-0 meta-language constants to be names for occurrences of OL symbols, rather than for OL symbols themselves. Thus, we will in principle refer to different occurrences of the OL (constant or variable) symbol a by $\underline{a}_i$ and $\underline{a}_j$,where $i, j \in N$ and $i \neq j$. Correspondingly, we discriminate different applications of semantic operations, and reinterpret first-order variables as ranging over occurrences.

Thus, (29a) and (29b) are significantly different constraint sets, which specify different sets of OL expressions.

(29) (a) $\{X = \underline{like'}_1 @_1 Y @_2 X, X_0 = C_1(\underline{j^*}_1), X_0 = C_2(\underline{j^*}_1)\}$

(b) $\{X_0 = \underline{like'}_1 @_1 Y @_2 X, X_0 = C_1(\underline{j^*}_1), X_0 = C_2(\underline{j^*}_2)\}$

The OL formula $like'(j^*)(b^*)$ matches the constraints in (29a). It is incompatible with those in (29b), which denote the singleton set $\{like'(j^*)(j^*)\}$, given closure information. Note that $like'(j^*)(j^*)$ is also compatible with (29a). However, it should not be by virtue of the solution $X_0 = like'@j_1^*@j_1^*$ for (29a) (if possible, we omit discriminating subscripts and underlines), but by free instantiation of either X and Y with some j_i^*, $i \neq 1$. For if we take USDL constants to be names of OL occurrences, $like'@j_1^*@j_1^*$ does not have a consistent interpretation any more: literally understood, the term denotes an OL expression where both argument places of $like'$ are filled by the same occurrence of the term j^*. Thus it seems to be a reasonable general constraint to allow only single occurrences of (individual or function) constants in USDL terms. The same holds for free first-order variables, since any possible instantiation would lead to multiple occurrences of constants. For second-order USDL variables, the situation is slightly different, since they are consistently interpretable as projections like $\lambda X.X$. Thus, we assume that in USDL a *Single-Occurrence Constraint* holds for terms with respect to any description-language constants and first-order variables (we will become more specific about the constraint later).

4 Linear Unification

Given the Linearity Constraint, the Single-Occurrence Constraint, and OL type sensitivity, our approach has a much better chance to meet the intuitive requirements. However, nothing definite can be said as long as we do not have a general method to find out what our USDL expressions precisely denote. The standard way to provide such a method is the specification of a denotational semantics. Unfortunately, this standard technique is only of limited use in our case. A real semantics that maps underspecified representations directly into some appropriate denotation based on truth-conditions for OL expressions is not only out of the range of this paper, but out of the range of current research in general, as far as structural underspecification of Levels 2 and 3 is concerned. A more modest semantics using term algebras looks good at first sight, since it assigns USDL expressions sets of OL terms as denotations, and moreover could serve as the first step of an indirect, two-step process of assigning real denotations. The problem with this approach is that the denotational interpretation does not take us really closer to the actual determination of the class of described OL terms. It merely transforms the problem of identifying appropriate substitutions to the structurally similar problem of identifying appropriate assignments. We will not pursue the question of a denotational semantics in this paper, but instead try to specify a constraint-solving mechanism, which generates the possible solutions for constraint sets in a systematic way, and thus serves as an operational semantics for USDL.

There is a constraint-solving algorithm for higher-order constraints, i.e., the higher-order unification (HOU) algorithm of Huet 1975. However, this does not mean that our task has already been solved. On the one hand, unconstrained HOU is an extremely unhandy tool to work with: not only is it semi-decidable, but also has a strong tendency towards combinatorial explosion. On the other hand, we have to make sure that the algorithm works under the linearity constraint for USDL. In this section, we will first give a short outline of the basic mechanism of Huet's algorithm, and then try to adapt it to the needs of our description language. The resulting formalism of "Linear Unification" will also exhibit a much better computational behaviour.

4.1 Huet's Algorithm

The general problem of HOU corresponds to the standard FO unification problem: The global goal is to solve a set of equations. The basic subtask is to find substitutions which render both sides of an equation $\alpha = \beta$ identical. The general form for the equations is given by $\alpha = \lambda X_1 ... X_k.\Psi(\alpha_1, ..., \alpha_n)$ and $\beta = \lambda X_1 ... X_k.\Phi(\beta_1, ..., \beta_m)$. The important difference to first-order unification is that the head symbols of Ψ and Φ need not be constants, but can also be bound or free variables. Bound variables as head symbols behave like constants; terms with constant or bound-variable head symbol are called *rigid*. *Flexible* terms with free variables as head symbols are the really novel feature of HOU. For the sake of simplicity, we consider only cases of terms without λ-prefix (and therefore without the possibility of bound variables), as in (30). The extension to the general case is straightforward.

(30) (a) $\alpha = \Psi(\alpha_1, ..., \alpha_n)$

(b) $\beta = \Phi(\beta_1, ..., \beta_m)$

We have to distinguish three sub-cases:

(i) α and β are rigid (Ψ , Φ constants)

(ii) α is flexible, β is rigid (Ψ variable, Φ constant)

(iii) α and β are flexible (Ψ , Φ variables)

Case (i) corresponds to conventional first-order unification with constant functor symbols k_1, k_2. If $k_1 \neq k_2$, unification fails. If $k_1 = k_2$, the unification task is reduced to the set of smaller tasks $\alpha_i = \beta_i (1 \leq i \leq m = n)$ (*Decomposition*). E.g., $\{walk'@X = Y@j^*\}$ is decomposed to $\{walk' = Y, X = j^*\}$. As in the first-order case, decomposition is deterministic.

Case (ii): The general idea for processing flexible-rigid combinations is to reduce them to the decomposition case by appropriate instantiation of the head variable . The task is to bring the head constant k of the rigid term to the head position of the flexible term α, which can be done in essentially two ways: first, by *Projection*, i.e., by instantiating the head variable C of the flexible term to a function that projects one of its arguments; second, by *Imitation*, i.e.,by instantiating C to a λ-term with head symbol k. Take the equation $C(walk') = walk'$ as an example. A projection solution is $C = \lambda X.X$, an imitation solution $C = \lambda X.walk'$.

This simple example already shows that some non-determinism comes into play. In certain cases the Projection rule and the Imitation rule can be alternatively applied. Moreover, if the head variable is many-place and argument head symbols are variables, there are different ways to apply the projection rule. The most important source for non-determinism, however, is that the rules of projection and imitation are specified in a very general way (and must be so specified in order to make the algorithm complete): In the imitation case, e.g., variable C of our example cannot be instantiated just to the constant k. It may well be that C and k are of different type (which means in our second-order language that they are of different arity, as in $C(walk') = walk'@j^*$), and the arguments of C may contribute to the arguments of k in any grouping and order. To cover all possible cases, the imitation rule requires C to be instantiated with the *most general τ-term with head symbol k* (where τ is the type of C). If C is of type $< \sigma_1, ..., \sigma_n, \sigma >$, k of type $< \tau_1, ..., \tau_m, \sigma >$, this is $\lambda\overrightarrow{X}.k(C_1(\overrightarrow{X}), ..., C_m(\overrightarrow{X}))$, where $\overrightarrow{X} = X_1, ..., X_n$, the X_i are variables of type σ_i, and the C_j are new variables of type $< \sigma_1, ..., \sigma_n, \tau_j >$.

In (31), a simple example is given to illustrate the working of the Imitation rule.

$$
(31)\quad
\begin{array}{ll}
C(walk', j^*) = walk'@j^* & \\
\Rightarrow C = \lambda XY.C_1(X,Y)@C_2(X,Y) & \text{(Imitation)} \\
\Rightarrow C_1(walk', j^*)@C_2(walk', j^*) = walk'@j^* & \text{(Binding)} \\
\Rightarrow C_1(walk', j^*) = walk', C_2(walk', j^*) = j^* & \text{(Decomposition)} \\
\Rightarrow C_1 = \lambda XY.X, C_2 = \lambda XY.Y & \text{(Projection)} \\
\Rightarrow C = \lambda XY.X@Y & \text{(Binding)}
\end{array}
$$

The imitation step itself is deterministic. However, it gives rise to quite a bit of non-determinism already in this trivial example: In the first step, the arguments of C are copied m-times (where m is the arity of k), in the second step arbitrarily many of the occurrences can be removed by an appropriate specification of the C_i functions. This also leads to many solutions which are unwanted in the view of our intended application.

Case (iii): The idea for processing the equation of two flexible terms is to reduce it to a case (ii) problem by substituting the left head variable with the most general τ-term with head symbol k, for some arbitrarily chosen constant k of the language (*Explosion*). This amounts to a recursive Generate-and-test procedure. The effect on the computational properties needs no comment.

In the following sub-sections, we will discuss the necessary changes that the HOU algorithm must undergo, in order to meet the requirements of a constraint solving algorithm for USDL. We will present proposals for a "Linear Imitation" rule, as well as a rule of "Flexible Decomposition" which is supposed to replace Explosion.

The Projection rule does not require a special treatment. Since USDL is second-order, only projection terms with first-order head variables are possible, i.e., terms of the form $\lambda X_1 ... X_n.X_i$. By the linearity constraint, the possible projection terms further reduce to the trivial case $\lambda X.X$.

4.2 Linear Imitation

The linearity constraint on function terms requires a modification of the imitation rule, for every more than one-place head constant on the right-hand side of the equation induces λ-terms with multiple occurrences of bound variables. Thus adding the linearity constraint would reduce the application of Imitation to one-place cases. We call our modified Imitation rule "Linear Imitation"[3]. The head variable C of the flexible term is not instantiated with the most general τ-term with head symbol k of the rigid term, but rather with a member of the set of most general *linear* τ-terms with head symbol k, which is for $\alpha = C(\alpha_1, ..., \alpha_n), \beta = k(\beta_1, ..., \beta_m)$ some $\lambda\overrightarrow{X}.k(C_1(\overrightarrow{X}_1), ..., C_m(\overrightarrow{X}_m))$, such that $\overrightarrow{X_1}...\overrightarrow{X_m} \in perm(\overrightarrow{X})$. We give the Linear Imitation ("LI") version of Example (31) in (32) below.

$$
(32)\quad
\begin{array}{ll}
C(walk', j^*) = walk'@j^* & \\
\Rightarrow C = \lambda XY.C_1(X)@C_2(Y) & \text{(LI)} \\
\Rightarrow C_1(walk')@C_2(j^*) = walk'@j^* & \text{(Binding)} \\
\Rightarrow C_1(walk) = walk', C_2(j^*) = j^* & \text{(Decomposition)} \\
\Rightarrow C_1 = \lambda X.X, C_2 = \lambda X.X & \text{(Projection)} \\
\Rightarrow C = \lambda XY.X@Y & \text{(Binding)}
\end{array}
$$

Linear Imitation is clearly non-deterministic, since in general there is no unique most general linear τ-term. In our example, there are several possible ways to apply the LI rule, according to the fact that there is no unique most general term, but rather four of them, i.e., $\lambda XY.C_1(X)@C_2(Y)$, $\lambda XY.C_1(Y)@C_2(X)$, $\lambda XY.C_1(X, Y)@Z$, and $\lambda XY.Z@C_2(X, Y)$. Globally, however, non-determinism is reduced, since all subsequent choices of alternative projections are removed. In fact, it is the linear imitation rule that renders non-trivial projection dispensible, since it anticipates the different projection alternatives that would have been required by general imitation, and coordinates them by sorting out those cases which are incompatible with the linearity constraint.

4.3 Flexible Decomposition

The part of the general HOU algorithm which has the best chance to lead to a computational desaster is the Explosion rule. If in $C_1(\alpha_1, ..., \alpha_n) = C_2(\beta_1, ..., \beta_m)$ both C_1 and C_2 are variables, the algorithm recursively produces all kinds of compatible instantiations of C_1 and C_2, respectively. It must do so in order to systematically generate those instantiations which satisfy all the constraints. The linearity constraint requires us to modify the Explosion rule in a way that only linear substitutions for C_1 are guessed. This modification guarantees correctness with respect to the linearity constraint, but it does not lead to an essential improvement of computational behaviour.

3. The idea is due to Michael Kohlhase (personal communication).

However, linearity allows a more radical modification of the Explosion rule. Look at the simple case $X_0 = C_1(\alpha) = C_2(\beta)$, which is also the most typical one in our intended applications. Given linearity of the function terms, there are precisely three ways in which α and β can be related.

First, α and β may form disjoint parts of the resulting term. In this case, the solution should be a two-place constraint $X_0 = H(\alpha, \beta)$, which just collects the semantic information contained in $C_1(\alpha)$ and $C_2(\beta)$ under a common variable, and in addition expresses the information that α and β form disjoint parts of the described semantic structure.

Second, α and β may share material, or "overlap". Then, either α is completely contained in β, or β is completely contained in α. In the former case, β is the result of applying some operation to $\alpha : \beta = C'(\alpha)$, and C_1 can be analysed as composition of C_2 and some new function variable C' ($C_1 = \lambda X.C_2(C'(X))$). The latter case works correspondingly: We obtain $\alpha = C'(\beta)$ and $C_2 = \lambda X.C_1(C'(X))$.

This solution cannot be straightforwardly generalized to flexible many-place terms, for the three cases, disjointness and inclusion in either direction, may simultaneously apply. Also, the two head variables do not necessarily stand in the simple relation that one completely includes the other. In $C_1(\alpha, \beta) = C_2(\gamma, \delta)$, it may well be the case that α includes γ, where β is included by δ ($\alpha = C'(\gamma), \delta = C''(\beta)$). This requires C_1 and C_2 to be properly overlapping functions with a common head part. In (33), we give a general formulation of what we call the rule of Flexible Decomposition (FD), which is able to alternatively generate the full range of disjoint, overlapping, and mixed cases, in one step.

(33) Flexible Decomposition:

Replace $C_1(\alpha_1, ..., \alpha_n) = C_2(\beta_1, ..., \beta_m)$ by

(i) equations A_i for every i: $1 \leq i \leq k$ (for some $k \leq m + n$), which are either of the form $C(\beta_{j1}, ..., \beta_{jl}) = \alpha_j$, for some $l \geq 0$, or of the form $C(\alpha_{j1}, ..., \alpha_{jl}) = \beta_j$,for some $l \geq 0$, such that the C are pairwise different new variables of arity ≥ 0, all argument terms of C_1 and C_2 occur exactly once in one of the A_i, and

(ii) the equations $C_1 = \lambda X_1...X_n.H(\gamma_1, ..., \gamma_k)$ and $C_2 = \lambda Y_1...Y_m.H(\delta_1, ..., \delta_k)$, where H is a new k-place function variable, $\sigma(\gamma_i)$ is the α side, and $\sigma(\delta_i)$ is the β side of A_i, for $1 \leq i \leq k$, $\sigma = \{\alpha_l \leftarrow X_l | 1 \leq l \leq n\} \cup \{\beta_l \leftarrow Y_l | 1 \leq l \leq m\}$

Note that the choice of the A_i completely determines the equations for C_1 and C_2.

First, let us look at an example which illustrates the working of the rule. For equation (35), we give two possible solutions, where (35) is the maximally disjoint one, and (36) is an example for a solution that realizes all possible relations between the arguments of C_1 and C_2. The (a) clauses of (35) and (36) show the immediate output of the FD rule. The (b) clauses show the intermediate result obtained for X_0 by Binding.

(34) $X_0 = C_1(\alpha_1, \alpha_2, \alpha_3, \alpha_4) = C_2(\beta_1, \beta_2, \beta_3)$

(35) (a) $Z_1 = \alpha_1, Z_2 = \alpha_2, Z_3 = \alpha_3, Z_4 = \alpha_4, U_1 = \beta_1, U_2 = \beta_2, U_3 = \beta_3,$
$C_1 = \lambda X_1 X_2 X_3 X_4.H(X_1, X_2, X_3, X_4, U_1, U_2, U_3)$
$C = \lambda Y_1 Y_2 Y_3.H(Z_1, Z_2, Z_3, Z_4, Y_1, Y_2, Y_3)$

(b) $X_0 = H(\alpha_1, \alpha_2, \alpha_3, \alpha_4, \beta_1, \beta_2, \beta_3)$

(36) (a) $Z = \alpha_1, C'(\alpha_2, \alpha_3) = \beta_1, C''(\beta_2) = \alpha_4, U = \beta_3,$
$C_1 = \lambda X_1 X_2 X_3 X_4.H(X_1, C'(X_2, X_3), X_4, U)$
$C_2 = \lambda Y_1 Y_2 Y_3.H(Z, Y_1, C''(Y_2), Y_3)$

(b1) $X_0 = H(\alpha_1, C'(\alpha_2, \alpha_3), \alpha_4, \beta_3)$

(b2) $X_0 = H(\alpha_1, \beta_1, C''(\beta_2), \beta_3)$

In (37a) to (37c), we show the three alternative solutions obtained by application of Rule (33) to the $X_0 = C_1(\alpha) = C_2(\beta)$ case, which we discussed above.

(37) (a) $Z = \alpha, Y = \beta, C_1 = \lambda X.H(X,Y), C_2 = \lambda Y.H(X,Y)$
(b) $C'(\alpha) = \beta, C_1 = \lambda X.H(C'(X)), C_2 = \lambda X.H(X)$
(c) $C'(\beta) = \alpha, C_1 = \lambda X.H(X), C_2 = \lambda X.H(C'(X))$

We can eliminate the use of the new variable H in the overlap clauses (37b) and (37c) because of $C_2 = \lambda X.H(X)$ and $C_1 = \lambda X.H(X)$, respectively. We thus arrive at $\{C_1 = \lambda X.C_2(C'(X)),\ C'(\alpha) = \beta\}$ and $\{C_2 = \lambda X.C_1(C'(X)),\ C'(\beta) = \alpha\}$, respectively, and will take these as the direct output alternatives of the FD rule for the special case of two one-place terms, in order to keep the notation simple.

4.4 Computational Properties of Linear Unification

The modified rules presented in the last two sections allow us to accommodate HOU to the needs of USDL. Accommodation is possible in a more cautious and a more radical way. The cautious variant consists in just replacing Imitation by Linear Imitation and Explosion by its linearized variant. The resulting system is complete. It is also correct, if we disregard the Single-Occurrence Constraint. The latter can be taken care of by a filter on the output, which sorts out all solutions containing terms with multiple occurrences of constants and first-order variables.

This seems to be an appropriate place to give the Single-Occurrence Constraint a precise definition: A USDL term must not contain more than one occurrence of any constant or variable, except one-place function variables. Multiple occurrences of one-place function variables do not necessarily lead to the duplication of occurrences, for they can be substituted by $\lambda X.X$. Since this is also the only way to make sense of them, we are entitled to add a rule that carries out the substitution as soon as a function variable occurs twice, and thus eliminates both occurrences.

The problem with the cautious variant of the algorithm is, however, that it does not terminate, and moreover exhibits a similar tendency to combinatorial explosion, as the original unconstrained version does. From a computational point of view, the radical variant is much more attractive. It uses Linear Imitation, and Flexible Decomposition in the place of Explosion. Moreover, the Single Occurrence Constraint can be used as an on-line filter, which immediately blocks a path in the search tree, as soon as a term is generated which violates the constraint. We expect this version of the algorithm to terminate, although the corresponding proof has not yet been carried out.

We also have not proved completeness under the additional restrictions, but are confident about it. At least we can say that FD plus Single-Occurrence Constraint lead to the elimination of several notorious sources of undesirable computational behaviour. In the beginning of the last section, we have already mentioned the possibly infinite recursive generation of possible instantiations as a motivation for the FD rule. Linear Imitation may also lead to infinite recursion in the case of independent multiple occurrences of a function variable in the constraint set. Look,e.g., at $F(g(a)) = g(F(a))$. Here immediate projection ($F = \lambda X.X$), which unifies the two occurrences of g, is not the only solution. LI renders $F = \lambda X.g(C(X))$, which leads to $g(C(g(a))) = g(g(C(a)))$, and further to $g(g(a))$, as a solution, as well as to $g^n(a)$ for any natural number n, by iteration of the LI rule. The Single-Occurrence Constraint filter excludes all but the first alternative, in accordance with the basic intuitions discussed in Section 3, and at the same time blocks the recursion process in its initial step.

Finally, we would like to point to the interaction between FD and Single-Occurrence Constraint. FD reduces the substitution of the head variable in a pair of flexible terms to a finite number of alternatives - a number, however, that drastically increases with the arity of the head variables. But many-place function variables are only introduced to cover the disjointness option for the arguments. Thus, the constraint automatically blocks the computationally expensive introduction of function variables with higher arity, if the arguments of the flexible terms overlap—and many of the typical examples we want to treat, syntactic ambiguities as well as scope underspecification, are such overlap cases.

5 Examples

In this section, we will demonstrate the application of the linear unification formalism to our syntactic underspecification example, as well as to cases of scope underspecification.

5.1 Syntactic Underspecification

We start with the *former professor in Amsterdam* example. We will be as explicit as possible, for this first example, indicating not only the changes certain formulae in the constraint set undergo, but also keeping track of the constraint set as a whole. For notational convenience, we introduce a numbering for the different occurrences of formulae (written in square brackets).

We start with constraint set (38), where we assume that the two description language occurrences of $prof'$ refer to the same occurrence of the OL constant $prof'$, and that the two occurrences of @ denote different instances of functional application, without indexing them explicitly. (38) is transformed to (39) by binding ([2′] being the result of the substitution of X_0 in [2] by $C_1(former'@C_3(prof'))$, the right-hand side of equation [1].

(38) [1] $X_0 = C_1(former'@C_3(prof'))$, [2] $X_0 = C_2(in\_A'@C_4(prof'))$

(39) [1], [2′] $C_1(former'@C_3(prof')) = C_2(in\_A'@C_4(prof'))$

Application of the Flexible Decomposition rule (special variant) to [2′] in (39) yields (40) with the two new constraints [3] and [4], and binding plus β-reduction transforms [1] to [1′] of (41).

(40) [1], [3] $C'(former'@C_3(prof')) = in\_A'@C_4(prof')$,
[4] $C_1 = \lambda X.C_2(C'(X))$

(41) [1′] $X_0 = C_2(C'(former'@C_3(prof')))$, [3], [4]

By choosing the other overlap variant of FD, we would have obtained (42) and (43) instead, which lead to the second solution for our constraint set.

(42) [1], [3] $C'(in\_A'@C_4(prof')) = former'@C_3(prof')$,
[4] $C_2 = \lambda X.C_1(C'(X))$

(43) [1′] $X_0 = C_1(C'(in\_A'@C_4(prof')))$, [3], [4]

Since the computation of the second alternative is completely analogous to that of the first reading, we do not present it explicitly here. Application of the third FD option to (39) is blocked, since right side and left side of [2′] overlap (because of shared occurrence of the constant $prof'$).

There are two alternative ways to apply the Linear Imitation rule to [3] of (40)/(41), since a one-place predicate is to be matched with a two-place constant. The first one results in adding constraint [5] $C' = \lambda X.Z@C''(X)$ to (41). The resulting constraint set is transformed to (44), (45), and (46), by the deterministic steps of Binding with [5] and β-reduction, (Standard) Decomposition of [3′], and another step of Binding with [6], respectively.

(44) [1″] $X_0 = C_2(Z@C''(former'@C_3(prof')))$,
[3′] $Z@C''(former'@C_3(prof')) = in\_A'@C_4(prof')$,
[4′] $C_1 = \lambda X.C_2(Z@C''(X))$, [5] $C' = \lambda X.Z@C''(X)$

(45) [1″], [4′], [5], [6] $Z = in\_A'$, [7]$C''(former'@C_3(prof')) = C_4(prof')$

(46) [1‴] $X_0 = C_2(in\_A'@C''(former'@C_3(prof')))$,
[4″] $C_1 = \lambda X.C_2(in\_A'@C''(X))$, [5′] $C' = \lambda X.in\_A'@C''(X)$, [6], [7]

The other LI alternative would have been $C' = \lambda X.C''(X)@Z$, instead of [5]. As you can easily check, processing along this branch stops after another two steps, since decomposition renders equation $C''(former'@C_3(prof')) = in\_A'$, which has no linear solution. Special FD, which transforms [7] to [8] and [9], makes (46) into (47).

(47) $[1'''], [4''], [5'], [6]$,
$[8]\ C'''(prof') = former'@C_3(prof'), [9] C_4 = \lambda X.C''(C'''(X))$

Note that the other possible result of applying special FD to [7] would lead to the unsolvable equation $C'''(former'@C_3(prof')) = prof'$. - Next, we apply LI to [8], which adds [10] $C''' = \lambda X.U@C''''(X)$, bind with [10], which transforms [8] to [8'] $U@C''''(prof') = former'@C_3(prof')$, decompose [8'] into [11] and [12], and bind with [11], which renders (48).

(48) $[1'''], [4''], [5'], [6], [9']\ C_4 = \lambda X.C''(former'@C''''(X))$,
$[10'] C''' = \lambda X.former'@C''''(X)$, $[11]\ U = former'$, $[12]\ C'''''(prof') = C_3(prof')$

Application of special FD to [12] yields [13] $C'''''(prof') = prof'$ and [14] $C_3 = \lambda X.C''''(C'''''(X))$ (note that in this case both FD alternatives render identical results); application of Projection to [13] adds [15] $C''''' = \lambda X.X$, and by binding with [14] and [15] we obtain (49).

(49) $[1'''']\ X_0 = C_2(in\_A'@C''(former'@C''''(prof')))$, $[4''], [5'], [6], [9']$, $[10']$,
$[11]$, $[13']\ prof' = prof', [14], [15]$

$[13']$ as a tautology can be removed from the constraint set, which completes the derivation. The other branch originating from (42) and (43) would have led to a solution containing (50).

(50) $X_0 = C_1(former'@C''(in\_A'@C''''(prof')))$

By the closure operation argued for in Section 5—substituting all function variables with $\lambda X.X$, we obtain the readings of former professor in Amsterdam, which are repeated here as (51).

(51) (a) $former'@(in\_Amsterdam'@professor')$

(b) $in\_Amsterdam'@(former'@professor')$

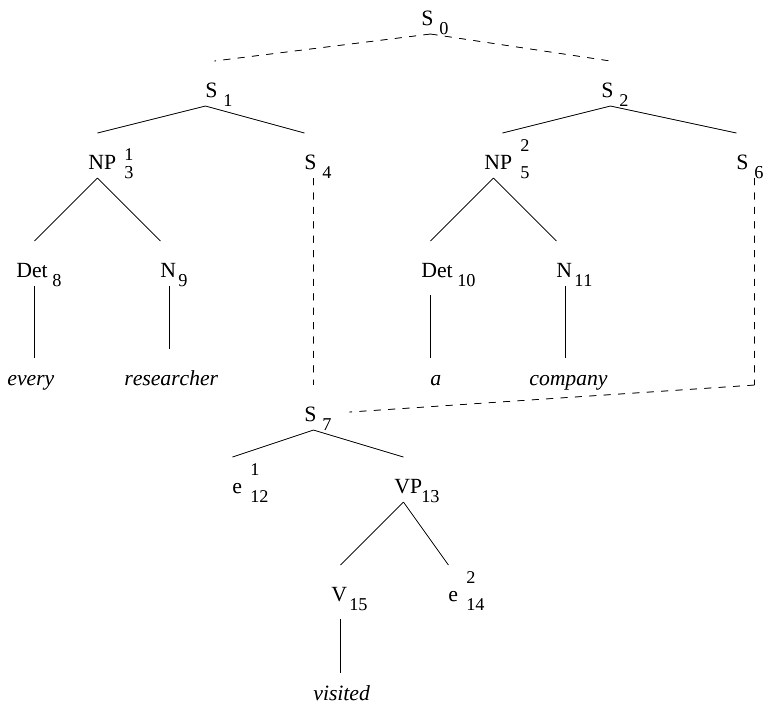
As you may check, these are in fact the only readings which can be obtained from the original constraint set.

5.2 Scope Underspecification

The example of the last sub-section in fact was a case of scope underspecification, already. Two alternative scope orderings of the two modifiers correspond to the different syntactic analyses in terms of constituent structure, which can be assigned to the N' phrase in question. Insofar, as we analyse scope ambiguities of quantifiers and other sentential operators as being based on syntactic ambiguities, which is the line of Montague 1973 and May 1985, we can just carry over our analysis. In (53), we give the quasi-syntactic tree representing the two scope readings of Sentence (52)[4].

(52) *Every researcher visited a company*

4. We do not care about linear precedence information here. —The syntactic representation is taken over from Muskens 1995.



(53)

A slight extension is necessary to provide the standard quantifying-in treatment. We add the two clauses of (54) to the interpretation principles given in (20). The symbols x_i and x_j are USDL constants that stand for the corresponding OL first-order variables x_i and x_j, respectively, not for two different occurrences of x. L_{xj} is taken as a one-place USDL function constant denoting the OL abstraction operation with x_j.

(54) (i) For each terminal node P_k which is occupied by an indexed empty element e^i add the constraint $X_k = x_i$.

(ii) For each binary branching node P_k with daughter nodes P_l and P_m, where P_l is an NP with index j, add the constraint $X_k = X_l @ L_{xj}(X_m)$

To avoid notational complexity, we take $every\_researcher'$ to be short for the USDL term which uniquely specifies the OL expression $\lambda F[\forall x(researcher'(x) \rightarrow F(x))]$. Likewise, we use $a\_company'$ and $visit'$ as shortcut names for $\lambda G[\exists y(company'(y) \rightarrow G(y))]$ and $\lambda u \lambda v[visit'(v)(u)]$.

We assume that the computation of the internal semantics of the noun phrases and the constituent S_7 has been carried out already, and thus start with a subset of the syntactic constraint set, which is graphically represented in (55):

(55) $\{S_0 \triangleleft^* S_1, S_0 \triangleleft^* S_2, S_4 \triangleleft^* S_7, S_6 \triangleleft^* S_7, S_1 \triangleleft S_3, S_1 \triangleleft S_4, S_2 \triangleleft S_5, S_2 \triangleleft S_6\}$

The semantic constraint set corresponding to (55) is the union of the sets specified in (a), (b), and (c) of (56), where (56a) contains the pre-computed parts of the description, (56b) the constraints gained by application of (54)(ii), and (56c) the constraints induced by the $\triangleleft^*$ clauses of (55) according to (20)(iii).

(56) (a) $X_3 = every\_researcher', X_5 = a\_company', X_7 = visit' @ x_1 @ x_2$

(b) $X_1 = X_3 @ L_{x1}(X_4), X_2 = X_5 @ L_{x2}(X_6)$

(c) $X_0 = C_1(X_1), X_0 = C_2(X_2), X_4 = C_3(X_7), X_6 = C_4(X_7)$

From these constraints we easily arrive at (57).

(57) (a) $X_0 = C_1(every\_researcher' @ L_{x1}(C_3(visit' @ x_1 @ x_2)))$

(b) $X_0 = C_2(a\_company' @ L_{x2}(C_4(visit' @ x_1 @ x_2)))$

At this stage, the constraint set has essentially the same structure as the corresponding set (38) for the example discussed in Section 5.1. Accordingly, the algorithm works in precisely the same way, resulting in the pair of solutions given in (58), which are transformed by closure to (59).

(58) $X_0 =$
$C_1(every\_researcher'@L_{x1}(C'(a\_company'@L_{x2}(C''(visit'@x_1@x_2)))))$
$X_0 =$
$C_2(a\_company'@L_{x2}(C'(every\_researcher'@L_{x1}(C''(visit'@x_1@x_2)))))$

(59) $X_0 = every\_researcher'@L_{x1}(a\_company'@L_{x2}(visit'@x_1@x_2))$
$X_0 = a\_company'@L_{x2}(every\_researcher'@L_{x1}(visit'@x_1@x_2))$

The pair of solutions in (59) describes the two possible scope readings of sentence (52).

There are well-known arguments to the end that scope ambiguity should not be regarded as an effect of an underlying ambiguity at some abstract syntactic level, but rather as a genuinely semantic ambiguity that results from the analysis of an unambiguous close-to-surface syntactic representation. The standard technique to realize this view is Cooper storage. The QLF formalism offers a technique to generate underspecified representations from a specific syntactic basis . Muskens 1995 proposes another treatment, which makes use of a subordination relation between labels taken over from Reyle 1993. In the following, we want to show that a direct modelling of scope underspecification can be taken to be a special application of the higher-order constraint approach of USDL.

We start by adding two new interpretation rules, one for quantifiers, and the other one for one-place sentential operators like negation. As far as treatment of scope is concerned, they replace (54). However, the latter may be necessary to interpret other uses of syntactic co-indexing (e.g., in connection with long-distance phenomena). Also, we associate two semantic meta-variables X_k and X'_k with each non-lexical syntactic node P_k (the "upper" and the "lower" semantic value of P_k), and modify the functional application rule of (20)(ii) in the way that the lower value of the mother is equated with the functional application of the upper values of the daughters. We will comment on the purpose of this change below. The two new rules are given in (60); the functions **I** and **B** will be explained below.

(60) (i) For each binary branching node P_k with daughter nodes P_l and P_m, P_l an NP, add the constraints $X'_k = X_m@x_i$ and $X_{\mathbf{I}(k)} = C(X_l@L_{xi}(C'(X'_k)))$.
(ii) For each binary branching node P_k with daughter nodes P_l and P_m, P_l a one-place sentential operator, add the two constraints $X'_k = X_m$ and $X_{\mathbf{B}(k)} = C(X_l@C'(X'_k))$.

As in the Cooper Storage treatment of quantifiers, the immediate contribution of an NP to the composition process is just an individual variable; the real quantifier denotation comes into play in a different way. In contrast to Cooper storage, however, the quantifier is not stored for later use, but immediately associated with a certain syntactic node higher up in the tree, by the second constraint. (60ii) works accordingly, except for the fact that no variable need to be introduced and abstracted over. $\mathbf{I}(k)$ is meant to render the node ID corresponding to the widest scope position which a NP daughter of P_k can take. $\mathbf{B}(k)$ provides the corresponding information for operators (which may be sensitive to different syntactic constraints). In the simple examples we are discussing in this paper, it always holds that $\mathbf{I}(k)$ and $\mathbf{B}(k)$ identify the root node, so we will not make explicit use of the notation in the following.

The rules in (60) in some sense correspond to the operation of quantifier storage. (61), on the other hand, is a kind of counterpart to the retrieval mechanism in Cooper Storage and related treatments of quantifiers. As mentioned above, we provide two semantic meta-variables for each syntactic constituent P_k. Clause (i) of (61) allows semantic operations to intervene between the upper value X_k and the lower value X'_k at any S node. Clause (ii) excludes this possibility for all other categories (lexical nodes are only assigned one meta-variable).

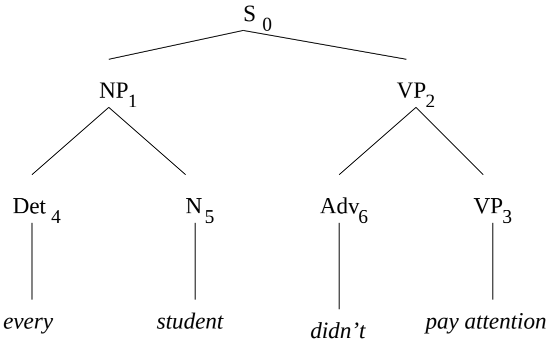
(61) (i) For every S node P_k, add $X_k = C(X'_k)$

(ii) For every other non-lexical node P_k, add $X_k = X'_k$

Note that (60) and (61) do not interact in the way that the operator must be applied at the top-most S node $P_{\mathbf{I}(k)}$ itself. S nodes lying on the path in between P_k and $P_{\mathbf{I}(k)}$ are possible candidates, as well.

As an example, we take Sentence (1) from Section 1, repeated here as (62), and assume (63) as its syntactic analysis.

(62) *Every student didn't pay attention*



(63)

(64) (a) $X'_1 = every\_student', X'_3 = pay\_attention', X_6 = not',$

(b) $X'_2 = X_3, X_0 = C_2(X_6@C_4(X'_2)))$

(c) $X'_0 = X_2@x, X_0 = C_1(X_1@L_x(C_3(X'_0)))$

(d) $X_1 = X'_1, X_2 = X'_2, X_3 = X'_3, X_0 = C_5(X'_0)$

(64) shows the constraints which we obtain from lexical insertion or pre-computation (64a), processing of the operator-containing VP (64b) and the NP-containing S (64c), as well as from the rules relating upper and lower values (64d). By simple substitutions, we obtain the three constraints in (65), and in turn by linear unification and closure the desired readings given in (66).

(65) $X_0 = C_1(every\_student'@L_x(C_3(pay\_attention'@x)))$

$X_0 = C_2(not'@C_4(pay\_attention'))$

$X_0 = C_5(pay\_attention'@x)$

(66) $X_0 = every\_student'@L_x(not'@(pay\_attention'@x))$

$X_0 = not'@(every\_student'@L_x(pay\_attention'@x))$

6 Concluding Remarks

In the last section, we have demonstrated some applications of the USDL treatment of semantic underspecification, based on the Linear Unification concept presented in Section 4. The algorithm we have employed is terminating, as we hope, and the examples show that it allows the treatment of different cases with a quite reasonable amount of computation. Also, the general situation seems to be promising in view of the fact that the maximum arity of semantic operator constants needed in NL analysis is low, maybe only 2, that the constraints we start with employ only one-place function variables, that many-place variables come into play only when disjointness is a possible option, and that the problem classes of syntactic ambiguity and scope ambiguity are definite overlap cases.

We would like to conclude the paper by sketching a few other application cases, and commenting on some open problems that become apparent in connection with the examples. First, look at the German example (67), which is a combined case of syntactic and scope underspecification.

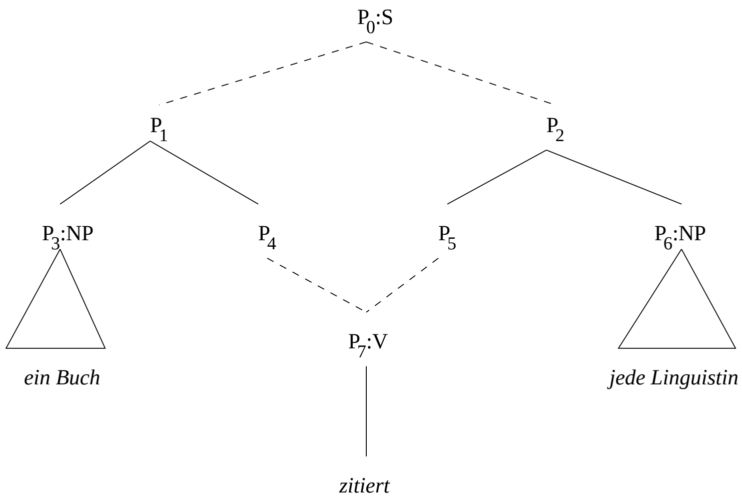
(67) *Ein Buch zitiert jede Linguistin*

(67) has four plausible readings, namely the two pairs of scope readings that correspond to the English sentences in (68). As to the syntactic ambiguity, which

typically arises in connection with flexible German word-order, see our comments on Example (10) in Section 1.

(68) (a) *A book cites every linguist*
(b) *Every linguist cites a book*

(69)



Given (69) as underspecified syntactic representation, our interpretation rules produce a constraint set which can be shown to specify exactly the set of four plausible readings given in (70), by Linear Unification and Closure.

(70) (a) $\forall x(linguistin'(x) \rightarrow \exists y(buch'(y) \wedge zitier'(x,y)))$
(b) $\exists y(buch'(y) \wedge \forall x(linguistin'(x) \rightarrow zitier'(x,y)))$
(c) $\forall x(linguistin'(x) \rightarrow \exists y(buch'(y) \wedge zitier'(y,x)))$
(d) $\exists y(buch'(y) \wedge \forall x(linguistin'(x) \rightarrow zitier'(y,x)))$

Thus our formalism is able to treat this more complex case, too. A closer look at (69), however, shows that the treatment is not completely unproblematic. First, there is no unique assignment of syntactic categories to some node symbols of (69). If we want to exploit the type constraints on USDL variables which are induced by the syntactic category information (we do not need it in this particular example), we have to use disjunction. Second, (69) is based on the assumption of highly simplified syntactic structures, i.e. those in (70), rather than syntactically more plausible ones, like those of (72).

(71) (a) $[_S[_{NP} \textit{ein Buch}][_{VP} \textit{zitiert}[_{NP} \textit{jede Linguistin}]]]$
(b) $[_S[_{VP}[_{NP} \textit{ein Buch}]\textit{zitiert}][_{NP} \textit{jede Linguistin}]]$

(72) (a) $[_{S'}[_{NP} \textit{ein Buch}]_i[_s t_i[_{VP}[_{NP} \textit{jede Linguistin}]\textit{zitiert}]]]$
(b) $[_{S'}[_{NP} \textit{ein Buch}]_i[_S[_{NP} \textit{jede Linguistin}][_{VP} t_i \textit{zitiert}]]]$

However, it seems that a compact representation of the alternatives of (72) can only be done by making essential use of disjunction (which also holds for many other non-trivial cases of syntactic ambiguity). Accordingly, the semantic information induced by this underspecified syntactic structure can only be expressed by introducing a disjunction operator into USDL.

Finally, let us have a short look at the incompleteness type of underspecification, which is typical for spoken input. Let us consider the English variant of (13) in Section 1 as an example, which we repeat here as (73).

(73) *... logician ... cites ... paper ...*

The USDL treatment is straightforward, but does not give us a lot of insight. Given that the three fragments belong to one syntactic unit, we can establish the syntactic constraint set (74), and infer the semantic constraint set (75).

(74) $P_0 \triangleleft^* P_1, P_0 \triangleleft^* P_2, P_0 \triangleleft^* P_3$

(75) $X_0 = C_1(logician'), X_0 = C_2(cite'), X_0 = C_3(paper')$

(76) $X_0 = H(logician', cite', paper')$

By double application of the disjointness variant of FD, we obtain (76). (76) is the only possible solution because the other FD alternatives require overlap of the arguments, which is impossible for distinct constants. The solution is intuitively correct insofar as it encodes the amount of certain semantic knowledge we can derive on the basis of (73). However, in a case where we have a fragmentary understanding of a spoken utterance, we usually conclude considerably more about the intended semantics of the utterance, using heuristic devices like the following: "In the case of lacking counter-evidence take the recognized verb to be the main verb of the sentence, and take the recognized common nouns to be the heads of the complements of the verb". This kind of heuristics in fact has been used in AI speech understanding systems, and is known under the name of "head-driven island parsing". We can reconstruct the heuristically based inference in the following way: In absence of certain syntactic information, we guess (77) as an option for the underspecified syntactic description of (73).

(77) $[_S[_{NP}[_{Det}...][_{N'}...logician...][_{VP}cite[_{NP}[_{Det}...][_{N'}...paper...]]]]]$

The general dominance relation is indicated here by the dots surrounding *logician* and *paper*. (77) leads to the semantic description (78), as one possible scope reading, where $\mathbf{Q_1}$ and $\mathbf{Q_2}$ are Type-0 USDL variables ranging over OL determiner representations.

(78) $X_0 =$
$C_1(\mathbf{Q_1}@C_2(logician')@L_{x1}(\mathbf{Q_2}@C_3(paper')@L_{x2}(C_4(cite'@x_2@x_1))))$

By application of further heuristic rules, e.g.: "Assume an indefinite determiner in the absence of an appropriate sufficiently salient antecedent", and "Don't assume modifiers without positive evidence" (which provides the basis for default application of the closure operation), we arrive at (79) as a hypothesized reading:

(79) $X_0 = a'@logician'@L_{x1}(a'@paper'@L_{x2}(cite'@x_2@x_1))$

Two messages can be read off this example. The first one is that the higher-order approach opens an interesting perspective for a non-trivial combination of "deep" conventional and "flat" heuristic techniques in semantic analysis. It has often been stated that both kinds of processing are required to bring about robust and reliable NL understanding systems. In particular, this holds for systems which process spoken input. However, the few systems under development which combine deep and flat processing techniques at all (like the dialogue translation system Verbmobil), run them as separate modules, using different data-structures for semantic representation: If the deep analysis fails at some stage, flat analysis starts from the input string or word lattice again and tries to come up with an approximate result, disregarding any information obtained from the deep analysis. The reason is that there has been no possibility so far to provide a common framework, since logic-based, compositional interpretation seems to require full grammatical information. The approach presented here opens the way to a sensible integration of flat and deep techniques, because their operation can be modelled in a common formal framework. This allows heuristic interpretation techniques to make use of any portion of information that can be obtained from deep analysis, and vice versa; and it allows to compare the results of alternative analysis techniques in order to keep them consistent. Thus, the proposed approach may also help to bridge the gap between logically based semantics and cognitively adequate processing techniqes.

The second message conveyed by our example is somewhat less encouraging. By presenting (78) as the USDL representation of the result of using the head-driven island heuristics for parsing our example, we have cheated a little. In fact, by applying the heuristics, we arrive at a slightly more specific information state than the one expressed by (78). The assumption that *logician* and *paper* are the syntactic heads of the subject and direct object NP, respectively, implies more than just that the predicates *logician'* and *paper'* occur somewhere in the restriction of the respective quantifiers. The NP heads may occur arbitrarily deeply embedded in function terms of the form $\alpha_1@(...@(\alpha_n@\beta)...)$, but (disregarding the possibility of relational nouns with internal arguments) only as the rightmost leaf β, since every possible intervening modifier takes them as arguments. To express this constraint, we would need a partial description of sequences of operations of arbitrary length, which is reminiscent of the "Functional Uncertainty" concept which has been developed in the context of LFG (see Kaplan and Zaenen 1989). The formalism does not allow this kind of specification so far.

We are just at the beginning of our work towards a tool for modelling radical underspecification. However, we hope that the first results are convincing enough to raise interest in higher-order constraint-based analysis of natural-language semantic information.

References

Alshawi, H. and Crouch, R.: 1992, Monotonic semantic interpretation, in *Proceedings 30th Annual Meeting of the Association for Computational Linguistics*, pp 32–38

Backofen, R., Rogers, J., and Vijay-Shanker, K.: 1995, A first-order axiomation of the theory of finite trees, *Journal of Logic, Language and Information*

Bos, J.: 1996, Predicate logic unplugged, *Proceedings of the 10th Amsterdam Colloquium*, forthcoming

Cooper, R.: 1993, Towards a general semantic framework, in R. Cooper (ed.), *DYANA-2 Deliverable R2.1.A*, pp 49–97, The DYANA-2 Project Administrator, ILLC, University of Amsterdam

Egg, M. and Lebeth, K.: 1995, Semantic underspecification and modifier, in J. Kilbury and R. Wiese (eds.), *Integrative Ansaetze in der Computerlinguistik (DGfS/CL '95), Düsseldorf, Seminar fuer Allgemeine Sprachwissenschaft*, pp 19–24, DGfS/CL

Huet, G.: 1975, An unification algoritm for typed lambda calculus, *Theoretical Computer Science*

Kaplan, R. M. and Zaenen, A.: 1989, Long-distance dependencies, constituent structure, and functional uncertainty, *Alternative Conceptions of Phrase Structure*

Kuschert, S.: 1996, A type-theoretic semantics for λ-drt, *Proceedings of the 10th Amsterdam Colloquium*, forthcoming

May, R.: 1985, *Logical Form*, MIT Press, Cambridge Mass

Montague, R.: 1973, The proper treatment of quantification in ordinary english, in J. H. e.a. (ed.), *Approaches to Natural Language*, pp 221–242, Reidel

Muskens, R.: 1993, A compositional discourse representation theory, in P. Dekker and M. Stokhof (eds.), *Proceedings of the 9th Amsterdam Colloquium*, pp 467–486, ILLC, University of Amsterdam

Muskens, R.: 1995, Order-independence and underspecification, *Dyana-2 Deliverable R2.2.C "Ellipsis, Underspecification, Events and More in Dynamic Semantics"*

Pinkal, M.: 1985, *Logik und Lexikon: Die Semantik des Unbestimmten*, de Gruyter, Berlin

Pinkal, M.: 1995, *Logic and Lexicon*, Kluwer Academic Publishers

Poesio, M.: 1994, *Semantic Ambiguity and Perceived Ambiguity*, unpublished manuscript

Reyle, U.: 1993, Vagueness and ambiguity in *drt*, *Journal of Semantics*

Skut, W.: 1995, *Underspecification-Based Representation of Noun Phrase Semantics, Diplomarbeit*, Universität des Saarlandes

van Eijck, J.: 1996, *Reasoning with Ambiguities*, Technical report, CWI Amsterdam, Ms.