Combining Semantics by Unification

Henk Zeevat
Computational Linguistics
University of Amsterdam

1 The Problem

This paper is an exploration of the relation between the compositionality principle and U(nification) C(ategorial) G(rammar)[1], a grammar framework that combines ideas from categorial grammar and unification grammar. Like other unification grammars and implementations of other frameworks in unification grammar, UCG constructs the semantic representations of complex expressions not by applying functions to arguments —the way this is done in Montague grammar— but by unifying semantic representations. The normal interpretation of this procedure is as an operation that constructs a new feature structure out of two given ones: it is an operation over the syntax of the semantic representation language. In this paper we seek an answer to the question whether we can reinterpret this syntactic manipulation as an operation over the meanings themselves and construct a compositional interpretation of the syntax-semantics relationship in UCG that is faithful to the syntactic procedure of feature structure unification.

There is a different program with roughly the same aim of using unification whilst maintaining compositionality. This program is to use a traditional compositional approach (e.g. PTQ or some other type theory) and prove that unification is a correct implementation of that theory. This is a fine program, but it appears it cannot be carried out in the desired way. Both Moore1989 (a discussion of this problem for the Core Language Engine, another unification formalism) and Reyle1987 (discussing a mechanism for the semantic interpretation of LFG) have to bring in restrictions and other mechanisms to maintain correctness. In the case of Moore, this is a post-processor that reduces λ-terms, in the case of Reyle, this is abstraction over complex lambda-terms and a copying mechanism for terms.

More positively, it can be claimed that unification offers possibilities that are both useful and not obtainable within more traditional λ-based approaches. These allow information concerning the argument semantics to derive from specification in the functor. An example is the combination of a VP marked only for non-stativity (*walk over the beach*) and a modifier that requires it

[1]CKMZ1986, CKZ1988, ZKC1987, Zeevat1988 are introductions to UCG. The considerations in this paper should apply to related frameworks such as CUG (Uszkoreit1987, Bouma1988) or HPSG (Pollard1985, Pollard&Sag1988)

argument to be an achievement (*in 5 minutes*). The result of the combination is more than fill in a slot in the functor semantics: the filler of the slot is also turned into an achievement. A similar effect is not directly encodable in Montague Grammar (though there are other ways of achieving the same effect). A similar effect is constraining the interpretation of a noun like *butter* to countable entities by applying the determiner *one* to it. There are mechanisms in the lambda-calculus (multiple reduction as in the semantics of the quantifier *every man*: $\lambda P \forall x(man(x) \rightarrow P(x)))$ that cannot be rendered directly in a unification framework. To what extent these are needed is not clear.

The view of this paper is that there is nothing wrong with the lambda calculus as a theory of semantic combination, but that at the same time it is useful to see whether unification can be defended as a serious alternative. If it can be, there are three reasons for preferring a unification approach. First, unification does not commit the user to a belief in abstract entities: no variables over the higher types get bound, as in the lambda calculus. The theory of combination has an interpretation on the meta-language. This allows the use of more restricted languages for defining the resulting meanings. Second, unification in UCG and other unification grammars is the fundamental operation in the syntax -or even in some versions the only operation. This allows a uniform account of the semantics of the grammar formalism[2]. Third, the intuitions behind unification seem to constitute a different and interesting account of semantic combination, an account which is based on epistemic rather than ontological principles. Thus, in UCG, one can say of many expressions that they provide a partially known semantics of an entity that can result from a syntactic combination. The combinations are then supplying new information to make the original semantics better known. The process of interpretation can be seen as the interpreter becoming more and more aware of the semantic object.

The intuition behind the lambda calculus theory of combination is to think of meanings as objects that are "unsaturated", objects that need other objects to become saturated. The corresponding epistemic notion is that of a concept of meaning which still varies with the circumstances of evaluation: different circumstances will make it denote different actual meanings. In this way, an incomplete expression is like an incompletely perceived object: in both cases there is a number of ways in which our knowledge concerning its nature can grow as new perceptions or further linguistic material comes in.

This paper focuses on UCG, but the basic notions should be applicable to similar unification theories. Given the number of different theories that are possible in a unification framework, it does not appear feasible to give a single general formulation of their theory of combination.

[2]The approach in this paper would suggest that there is a series of different unification operations involved for the different levels of linguistic description.

1.1 UCG

As a grammar theory UCG consists of two kinds of statements. It describes by means of lexical entries to which other expressions a word will apply and which expressions will apply to it (the categorial information). In addition, it gives a partial description of the phonology, the categorial information and the semantics of the expression that will result from a fu7nctional application. In case the lexical element has a functor category, there may and often will be a dependency in this description on the phonological, categorial and semantic values of the argument expressions, a dependency that is expressed by variable sharing, as in the following example. (Capital letters indicate variables, expressions are written *Phonology* : *Category* : *Semantics*).

(1) A likes B:
(sent/ A:np:X/ B:np:Y:
$[s]like(X,Y)$

Here, the expression needs to be combined with a nominative and an accusative NP to form a finite sentence. The phonology of the sentence will be formed by filling in the phonological values of the arguments for the variables in the phonological slot, and the semantics by filling in the semantic values in the semantic slot. Of the second kind of statement, there is only a single one[3]: the rule of functional application. This rule says that a complex expression may be formed from a functor expression and another expression E, in case the functor's information about its argument is coherent with the information associated with E. The complex expression consists of the functor without its argument slot, together with the information that its argument "was" the expression E.

(*Def.* 1) **Functional Application**
If B and D are unifiable and θ is their most general unifier (mgu), then $\theta(P : C : S)$ can be derived from $P : C/B : S$ and D

Together with the lexical entries, functional application states what are the well-formed expressions of the UCG-fragment. Viewed as a statement on the semantics it says that the result expression has the meaning of the functor expression and that all the semantic values associated with B are the same as the corresponding semantic values in D.

Corresponding is here spelled out by the structure of B and D. Minimally, the category of B and D is primitive, in which case their semantics should be the same. But the shared category can be complex, in which case the semantics of the signs involved in their categories should also be the complex.

[3]This is a simplification: there are actually two functional applications (bidirectionality) and a number of unary rules, but both functional applications have the same semantic effect and the unary rules just copy the semantics of their argument.

2 Semantic Unification

If two expressions can vary in meaning, it is possible to consider the situation that they might have the same meaning. Thus if we have two intensions in PTQ (e.g. propositions), we can consider the proposition that the two intensions have the same extension. In this case the variability resides in the possible worlds the intensions range over. Similarly we can deal with variables, or with expressions containing variables: we can consider the class of assignments under which the two variables or expressions have the same meaning. If we are in a partial theory, we could form a unification operation in the language. For the intensions, unifying α and β would be the partial function from worlds into intensions that is defined precisely over those worlds in which α and β have the same denotation and that takes exactly the same values as α or β in those worlds. For the variables, it would be that function defined on the assignments under which α and β have the same meaning and that takes the meaning of α or β.

The notion we will be developing is related to the second example.

Why are these unification operations? It is customary to define term unification by interpreting the terms as the class of their ground instances (the terms obtained by filling in a constant or ground term for each of the variables occurring in the term) and the unification as the intersection of these sets. Thus the unification of the two terms is that term (if the intersection is non-empty) that has the ground instances that the terms share. Similarly, feature structure descriptions are interpreted as the set of feature structures that satisfy the description and the unification is again the intersection of the sets corresponding to the description. In this case, we interpret intensions as sets of world extension pairs, and open objects as sets of pairs each consisting of an assignment and the value of the object under that assignment.

If we have a strictly monotonic partial logic, any composite containing the result of a unification, will only have a denotation or meaning if we are considering an index or assignment with respect to which α and β are the same. Each of these can be thought of as a ground unifier: one that turns α and β into the same ground object: its extension at the index or the meaning of the expression with constants for the variables.

The unification collects all shared ground unifications, and can be seen as the disjunctive combination of these. Under further unifications we may exclude various of the still open possibilities. This makes the operation considered here a greater lower bound in the lattice of partial functions from the relevant variation dimension into the relevant type of meanings. The empty function is here the inappropriate result, the $\perp$ element of the lattice.

Of particular interest are partial or total functions from worlds or assignments that are *constant* in the sense that for each element in their domain they have the same value. Objects that do not vary along the relevant dimension can be considered as constant objects (compare rigid designators). In the approximation process, constant outcomes are what we are aiming for.

In the case of UCG, the process of constructing a semantic representation of a complete expression is a continuous refinement of the concept of a semantic representation associated with the main functor and is only successful if it leads to a constant function. The constant function determines the actual meaning.

2.1 A General Definition

Let L be some language with a compositional semantics. Let L^+ be an extension of this language that includes meta-variables over some or all of the compositional types. Then $L^=$ may be defined to be an extension of L^+ including a unification operation in the following way.

(*Def.* 2) $\alpha \sqcap \beta$ is an expression of type τ if α and β are expressions of type τ.

In order to interpret our new operation $\sqcap$ we have to deal first of all with the compositional interpretation of L^+. The solution for that is standard. We consider the set MA of functions that map the meta-variables into meanings of the appropriate type, and we define $\| \alpha \|$ in terms of the L-interpretation function $[\alpha]$ as follows.

(*Def.* 3)

1. $\| \alpha \| = \{< h, h(\alpha) >: h \in MA\}$ if α is a meta-variable.
2. $\| \alpha \| = \{< h, [\alpha] >: h \in MA\}$ if α is a primitive L expression.
3. $\| O(\alpha_1, \ldots, \alpha_n \| = \{< h, [O(\alpha_1\|(h), \ldots, \|\alpha_n\|(h) >: h \in MA, \alpha_1\|(h)$ is defined, $\ldots \|\alpha_n\|(h)$ is defined $\}$ if $O(\alpha_1, \ldots, \alpha_n)$ is a composite $L^=$-expression and O is an L operation.

Unification can be incorporated by definition (*def*4).

(*Def.* 4) $\|\alpha \sqcap \beta\| = \{< h, \|\alpha\|(h) >: \|\alpha\|(h)) = \|\beta\|(h), h \in MA\}$

In combination with compositionality, we have the proposition (*prop*1).

(*Prop.*1) for $h \in dom\|\alpha \sqcap \beta\|, \|\gamma(\alpha)\|(h) = \|\gamma(\beta)\|(h)$
or both $\|\gamma(\alpha)\|(h)$ and $\|\gamma(\beta)\|(h)$ are undefined.

From this it follows that we can compare the term unification $t(\alpha, \beta)$ with semantic unification. What we would like to have is a principle like (2).

(2) for $< u, v > \in mgu(\alpha, \beta), h \in dom||\alpha \sqcap \beta||, ||u||(h) = ||v||(h)$

This would allow us to apply the mgu to any environment of the unified terms. But this is not forthcoming if we look at (3). The term unification of

(3) $\varphi \wedge X = X$ and
$\varphi \wedge j = j$

would give X the value of j, even though the meaning identity of the two expressions does not imply that the values of X and j are the same. What we get is the weaker (*prop*2).

(*Prop.*2) If $||\gamma(\alpha \sqcap \beta)||(h)$ is defined, $||\gamma(\alpha)||(h) = ||\gamma(t(\alpha, \beta)||(h) = ||\gamma(\alpha \sqcap \beta)||(h)$

This allows replacements: if we know that h is correct for the derivation of some expression γ and h unifies α and β, then we can replace α or β with the term unification of α and β in γ.

The counterexample establishes that abuse of term unification is possible[4]. Term unification will always unify the corresponding subterms, but semantic unification does not have this property. The unification of subterms is only guaranteed, if the identity of the meaning of the subterms follows from the identity of the semantics of the whole term. For some kinds of semantics, this is fairly unrestricted (e.g. property theory or situation semantics), for others, this only holds in special cases. The unification of the indices of formulas in InL[5], when formulas are unified, requires the more structured semantics introduced below.

2.2 Examples

Take propositional logic with the following semantics:

(*Def.* 5)

1. $[p] = F(p) \in 2$
2. $[\varphi \wedge \psi] = [\varphi].[\psi]$
3. $[\neg\varphi] = 1 - [\varphi]$

[4]This can be compared to proposing operations in Montague grammar, that can only be defined on IL but lack a corresponding semantic interpretation.

[5]InL, the semantic representation language used in UCG, assigns a special variable (the index) to each formula. Formula unification therefore entails unification of the indices of both formulas

The language $L^=$ is interpreted by considering meta-variables A, mapped by meta-assignments h into 2, the interpretation of the only type in propositional logic. (. and − return undefined if one of their arguments is undefined.)

(*Def.* 6)

1. $||p||(h) = F(p)$
2. $||A||(h) = h(A)$
3. $||\varphi \wedge \psi||(h) = ||\varphi||(h).||\psi||(h)$
4. $||\neg\varphi||(h) = 1 - ||\varphi||(h)$
5. $||\varphi \sqcap \psi|| = ||\varphi|| \cap ||\psi||$

The triviality of the semantics in this case is reflected in the unification operation. Looking at the new meanings as some kind of intensions, unification is an intensional operator that restricts the interpretation of its arguments to those meta-assignments under which they denote the same truth-value. Unification between constants is either the identity operator (to either argument) or the one that delivers the trivial meaning (the empty function from meta-assignments to truth-values).

As a second example, consider modal first order logic. Here we have operators $\wedge$, $\neg$, $\forall$ and $\Box$. The latter two are operators over propositional functions and propositions respectively. So now the meaning of the formula meta-variables must similarly have both a propositional and a propositional function character, as can be seen from the compositional semantics ($def7$), assuming a model $< U, W, R, F, w, g >$, with U and W non empty, $F(c) \in U$, $w \in W$, $g \in VAR \rightarrow U$.

(*Def.* 7)

1. $[P](g)(w) = F(P)(w)$
2. $[x](g)(w) = g(x)$
3. $[c](g)(w) = F(c)$
4. $[Pt_1 \dots t_n](g)(w) = [P](g)(w)(< [t_1](g)(w) \dots [t_n](g)(w) >)$
5. $[\varphi \wedge \psi](g)(w) = [\varphi](g)(w).[\psi](g)(w)$
6. $[\neg\varphi](g)(w) = 1 - [\varphi](g)(w)$
7. $[\forall x\varphi](g)(w) = \prod_{\{g':g'=_x g\}}[\varphi](g')(w)$
8. $[\Box\varphi](g)(w) = \prod_{\{w':wRw'\}}[\varphi](g)(w')$

We can add both meta-variables for objects and for formulas to this language and define $L^{=}$. We obtain the following semantics:

(*Def.* 8)

1. $\|P\|(h)(g)(w) = F(P)(w)$
2. $\|A\|(h)(g)(w) = h(A)(g)(w)$
3. $\|a\|(h)(g)(w) = h(a)(g)$
4. $\|x\|(h)(g)(w) = g(x)$
5. $\|c\|(h)(g)(w) = F(c)$
6. $\|Pt_1 \ldots t_n\|(h)(g)(w) =$
   $\|P\|(h)(g)(w)(< \|t_1\|(h)(g)(w) \ldots \|t_n\|(h)(g)(w) >)$
7. $\|\varphi \wedge \psi\|(h)(g)(w) = \|\varphi\|(h)(g)(w).\|\psi\|(h)(g)(w)$
8. $\|\neg\varphi\|(h)(g)(w) = 1 - \|\varphi\|(h)(g)(w)$
9. $\|\forall x\varphi\|(h)(g)(w) = \prod_{\{g':g'=_x g\}} \|\varphi\|(h)(g')(w)$
10. $\|\Box\varphi\|(h)(g)(w) = \prod_{\{w':wRw'\}} \|\varphi\|(h)(g)(w')$
11. $\|\varphi \sqcap \psi\| = \|\varphi\| \cap \|\psi\|$
12. $\|t_1 \sqcap t_2\| = \|t_1\| \cap \|t_2\|$

If we regard all three parameters as ranging over a kind of possible worlds, L-sentences —where all variables are bound— correspond to propositions (functions of type *st*), L-formulas —that may contain free variables, but no meta-variables— to concepts of propositions (type *sst*) and the meanings on which the unification operation operates to concepts of concepts of propositions (type *ssst*).

In Stokhof & Groenendijk's theory of questions, questions are concepts of propositions. Applying such a concept to the actual world gives the true, complete and rigid answer to the question. If we regard the actual world as the world where, among other things, the assignments to the meta-variables are as the speaker intends with his utterance, applying the actual world to the concept of concepts of propositions will give the propositional function that the speaker intends with this use of the expression (strictly speaking this is only so modulo renaming of variables). A normal unification is like a partial answer: a partial answer eliminates a set of possible answers: here the ones corresponding that do not unify the variables as the unifcation indicates. Or maybe better, to finding out that two questions have the same answer, as a special case of a partial answer.

(4) Question: Who talks?
Answer: Who walks.
Result: the question who talks restricted to the worlds where the walkers are the talkers.

A last example could be the representation language assumed in UCG, a DRT-like language called InL assumed for UCG, but I omit this here for reasons of space.

3 Extending Unification to Functions

For UCG we need more than just unification of L-objects (InL-meanings). If in a functional application we unify two functor signs, we do not only unify the semantics of the functor signs, but at the same time the semantics of the signs that they are subcategorised for. The combination of the semantics with the semantic representations of the subcategorised signs is no longer an L-object. So the semantic role of a complex expression is only incompletely described by the semantics. One of the roles of the functional extension defined below is to allow for these more complex unifications.

That we need the functional extension follows from a closer look at functional application (*def*9).

(*Def.* 9) **Functional Application**
If B and D are unifiable and θ is their m.g.u.
$\theta(\text{P:C:}S) \rightarrow \text{P:C/B:}S\text{, D}$

Here the semantics of the result expression is not the result of a unification, but the result of applying the most general unifier of the expressions D and B, to the semantics S and to the semantics of the expressions still contained in C.

The semantics of the result expression S' is thereby a function of both its old version S and of semantics the expressions B and D, but not the result of a unification of S with something else[6].

So functions are needed to make sense of the application operation: we will represent the semantic contribution of functors as functions, and define the application of a function to an argument in terms of unification. (*def*10) contains the definition of a function forming operator, defined over L$^=$-meanings, now called L$^=$-objects.

(*Def.* 10) 1. x is an LF-object if x is an L$^=$-object

2. $a.b$ is an LF-object if a and b are LF-objects.

[6]It is possible to describe the application as (the reduction of) the unification of $P : C/B : S$ with $P : C/D : S$, but again this does not make it clear in what way the new semantics S' depends on the semantics of D.

In order to arrive at the notion of application we need the notion of a restriction. This is given in (def11). The problem here is that the new functions are not themselves functions from MA into other values, so that we cannot restrict them in the way in which we restricted meanings before. I will write the restriction of an object x to a set F as $x \upharpoonright F$.

(*Def.* 11) 1. $x \upharpoonright F = \{< h, xh >: h \in F\}$ if x is an L$^=$-object.

2. $(a.b) \upharpoonright F = a.(b \upharpoonright F)$ if $b \upharpoonright F \neq \emptyset$ else $\emptyset$

We write $x \upharpoonright h$ for $x \upharpoonright \{h\}$.

We define the set of unifying meta-assignments of a and b, $u(a,b)$ in (def12).

(*Def.* 12) $u(a,b) = \{h : a \upharpoonright h = b \upharpoonright h \neq \emptyset\}$

In (def13) we can now define $a.b$ as that function f such that

(*Def.* 13) $fx = b \upharpoonright u(a,x)$ if $b \upharpoonright u(a,x) \neq \emptyset$, else fx is undefined.

In other words, application comes out as (def14).

(*Def.* 14) $app(a.b, c) = b \upharpoonright u(a,c)$, defined for c such that $b \upharpoonright u(a,c) \neq \emptyset$.

It is a consequence of these definitions, that

(*Prop.*3) $app(a.b \upharpoonright F, c) = b \upharpoonright F \upharpoonright u(a,c) = b \upharpoonright u(a,c) \upharpoonright F = app(a.b, c) \upharpoonright F$ if $b \upharpoonright u(a,c) \neq \emptyset$ and $b \upharpoonright F \neq \emptyset$

In the rest of this section, it will be checked that this definition gets us what we want. The lemma establishes the correctness of our definition of the unifying meta-assignments.

(5) **Lemma**
$\forall h\ f \upharpoonright h = g \upharpoonright h$ iff $f = g$

Proof

$\Rightarrow$ (Induction over the complexity of the functions.)

First let f and g be L$^=$-objects. Then $f = \bigcup_{h \in H} f \upharpoonright h = \bigcup_{h \in H} g \upharpoonright h = g$.

Assume now that $f = a.b$ and $g = c.d$ are functions. Let x be arbitrary.

$app(f \upharpoonright h, x) = b \upharpoonright h$ iff $x \upharpoonright h = a \upharpoonright h$.

By assumption and definition: for $h \in u(a, x)$, $app(f \uparrow h, x) = app(g \uparrow h, x) = app(f, x) \uparrow h = app(g, x) \uparrow h$.

Similar for $h \in u(c, x)$.

So for $h \in u(a, x) \cup u(c, x), app(f, x) \uparrow h = app(g, x) \uparrow h$.

For $h \notin u(a, x) \cup u(c, x)$, $app(f \uparrow h, x)$ is undefined and $app(g \uparrow h, x)$ is undefined.

So for all $h \in H$, $app(f, x) \uparrow h = app(g, x) \uparrow h$, and by the induction hypothesis, $app(f, x) = app(g, x)$.

But x was arbitrary, so $\forall x \; app(f, x) = app(g, x)$ and $f = g$.

$\Leftarrow$ Trivial.

(*prop4*) makes the link with term unification. We write our functions as a term $O(a, b)$, with O always the operation $x.y$, so the term unification of two functions is just the combination of the unification of first arguments with that of the second arguments. (*prop4*) establishes the coincidence of term unification for our function with the semantic unification proposed here.

(*Prop.4*) **Proposition**
$u(a.b, c.d) = u(a, c) \cap u(b.d)$

Proof

$\Rightarrow$

Let $h \in u(a.b, c.d)$. By definition (*def* 13) and (*def* 11) $a.b \uparrow h = \{< x, b \uparrow h >: a \uparrow h = x \uparrow h\} = \{< x, d \uparrow h >: c \uparrow h = x \uparrow h\} = c.d \uparrow h$.

It follows immediately that $b \uparrow h = d \uparrow h$ so $h \in u(b, d)$.

Moreover, $app((a.b) \uparrow h, a) = b \uparrow h$ ($a \uparrow h = a \uparrow h$ and by the definition (*def* 12) of $u(x, y)$, $b \uparrow h$ must be defined) so $app((c.d) \uparrow h, a) = b \uparrow h$ so $c \uparrow h = a \uparrow h$.

So $h \in u(a, c)$.

$\Leftarrow$

Let $h \in u(a, c) \cap u(b, d)$.

Let x be arbitrary.

$app((a.b) \uparrow h, x) = b \uparrow h$ if $x \uparrow h = a \uparrow h \Leftrightarrow$
$app((a.b) \uparrow h, x) = d \uparrow h$ if $x \uparrow h = c \uparrow h \Rightarrow$
$app((a.b) \uparrow h, x) = app((c.d) \uparrow h, x)$ if $x \uparrow h = a \uparrow h$.

So for all x either $app((a.b) \uparrow h, x)$ is undefined and $app((c.d) \uparrow h, x)$ is undefined or $app((a.b) \uparrow h, x) = app((c.d) \uparrow h, x)$ and so $(a.b) \uparrow h = (c.d) \uparrow h$.

So $h \in u(a.b, c.d)$, by definition (*def* 12) .

(*prop*5) establishes our definition of application to lead to a a proper notion of unification between functions: the unification unifies the two functions.

(*Prop*.5) **Proposition**
$f \uparrow u(f,g) = g \uparrow u(f,g)$

Proof

Let $f = a.b$, $g = c.d$. Let $app(f \uparrow u(f,g), x)$ be defined then $\exists h \in u(f,g)\ x \uparrow h = a \uparrow h = c \uparrow h$ so $app(g \uparrow u(f,g), x)$ is defined. So the domains of the two functions coincide.

Moreover assume $app(f \uparrow u(f,g), x) \uparrow h$ is defined. Then:

$app(f \uparrow u(f,g), x) \uparrow h = b \uparrow h = d \uparrow h = app(g \uparrow u(f,g), x) \uparrow h$.

If $app(f \uparrow u(f,g), x) \uparrow h$ is undefined, then $h \notin u(f,g) \cap u(x,a)$ or $b \uparrow u(f,g) \uparrow h$ is undefined. But $a \uparrow u(f,g) = c \uparrow u(f,g)$ and $b \uparrow u(f,g) = d \uparrow u(f,g)$ so $app(g \uparrow u(f,g), x) \uparrow h$ is undefined.

But then by lemma (5) $app(f \uparrow u(f,g), x) = app(g \uparrow u(f,g), x)$. So the two functions coincide.

So what we end up with is a notion of unification for the functions that is the same as the one for the L$^=$-objects: a set of meta-assignments. Moreover, using term unification as the implementation of function unification does not add any extra distortion.

4 Polymorphicity

With the material in the last section we can describe the semantic contribution of most UCG expressions. The expressions with basic categories could be handled already by L-objects, the last section has added the functor categories. Unfortunately UCG also has a third class of categories: the polymorphic ones. These occur for example in NPs.

(6) W :
C/(W:C/every boy:np:x):B
$[E][boy(x) \Rightarrow B]$

Superficially it would seem that we could use (7)as a characterization of its semantic contribution.

(7) $B.[E][boy(x) \Rightarrow B]$

But this goes wrong in case C gets matched to a functor category like the one in (8)

(8) X loves Y:
sent/X:np:x/Y:np:y:
$[s]love(x, y)$

or (9).

(9) X gives Z Y:
sent/X:np:x/Y:np:y:/Z:np:z:
$[e]give(x, z, y)$

In these cases, B would be instantiated to only $[s]love(x, y)$ or $[e]give(x, z, y)$ and would fail to take in the arguments of these expressions, giving (10) as the results.

(10) $[E][boy(y) \Rightarrow [s]love(x, y)]$
$[E][boy(z) \Rightarrow [e]give(x, z, y)]$

The correct outcomes must be

(11) $x.[E][boy(y) \Rightarrow [s]love(x, y)]$
$y.x.[E][boy(z) \Rightarrow [e]give(x, z, y)]$

So UCG needs more complex functions than the ones we have sketched so far, because of categorial polymorphism. Fortunately, this is not very hard to integrate with the picture sofar. Categorial polymorphism allows us to refer to functor categories. In semantic terms this is abstraction over a list of objects: the semantics associated with the list of arguments. Since we want to allow the list to contain objects which are themselves lists, we have to give a full recursive function of LL, the language L with lists and functions, in (*def*15).

(*Def.* 15)

1. If x is a function and y is a sequence $x.y$ is a sequence.
2. If x and y are functions, $x.y$ is a function.
3. If x and y are lists, $x * y$ is a sequence.
4. If x is a list and y is a function, $x * y$ is a function.
5. **nil** is a list.

A reconstruction of e.g. the semantic contribution of the sign (12),

(12) W:
C/(W:C/(a boy:np:x):[a]A):
$[a][boy(x), [a]A]$

can now be given in (13), where Λ is a meta-variable of the new list type.

(13) $(x.\Lambda * [a]A).\Lambda * [a][boy(x), [a]A]$

a function that would be defined for each of the elements of (14),

(14) $a.b.[s]love(b,a)$,
$a.b.c.[e]give(c,b,a)$,
$v.(p.\Lambda * [a]A).\Lambda * [a][[p]in(v), [a]A]$

the semantic contribution of the signs in (15).

(15) X loves Y:
sent/X:np:b/Y:np:a:
$[s]love(b,a)$,

X gives Y Z:
sent/X:np:c/Z:np:b/Y:np:a:
$[e]give(c,b,a)$,

W:
C/W:(C/in X:pp:p):$[a]A$/X:np:v:
$[a][[p]in(v), [a]A]$

We need list denotations to make sense of the list variables, but it is not necessary to interpret the concatenation operation. Instead we will eliminate the $*$ operator in the definition of denotation by the rules in (*def*16), a definition of possible list denotations.

(*Def.* 16)

1. <> is a list denotation
2. $a.c$ is a list denotation if a is a function denotation and c is a list denotation.
3. a is a function denotation if a is an L-meaning
4. $a.b$ is a function denotation if a is a function denotation and b is a function denotation

We now define the new meta-assignments to assign the list meta-variables a value among the list denotations. We can then extend our notion of denotation to LL in (*def*17).

(*Def.* 17)

1. $[L]^h = h(L)$
2. $[\mathbf{nil}]^h = <>$
3. $[a.b]^h = \{< [a]^h, [b]^h >\}$
4. $[app(a.b, c)]^h = [b]^h$ if $[a]^h = [c]^h$ else undefined
5. $[\mathbf{nil}{*}b]^h = [b]^h$
6. $[(a.x) * b]^h = [a.(x * b)]^h$.

Conclusions

The reconstruction of the semantic combination theory has provided a space of functions over partial meanings for the meta-theory of the semantic representation language. This space is rich enough to reconstruct other methods of assigning meaning in unification grammars, notably the meanings of rules in a system with many phrase structure rules. It seems as respectable from the point of view of determining a compositional semantics as the standard method.

Unlike the lambda calculus, there is no dependency on the actual representation language and this is an advantage. The language itself can be kept as limited as one requires for natural language. What we have to characterize in assigning meaning to complex expressions is where the parts of an underdetermined semantic representation are going to come from. This is pure linguistics and does not involve complex reduction strategies.

Like in the case of Montague grammar, looking for a precise answer to the question what constitutes the semantics of the combination of meanings, gives useful methodological restrictions on possible definitions of meanings and syntactic rules.

References

Bouma, G. Modifiers and Specifiers in Categorial Unification Grammar. *Linguistics* **26**, 1988, p.21-46.

Calder, J., E. Klein, M. Moens and H. Zeevat. Problems of Dialogue Parsing. Edinburgh Research Papers in Cognitive Science 1, 1987.

Jonathan Calder, Ewan Klein and Henk Zeevat. 1988. Unification Categorial Grammar: A Concise, Extendable Grammar for Natural Language Processing. In: *COLING 88*, Budapest. p. 83-86.

Janssen, T.M.V. *Foundations and Applications of Montague Grammar.* PhDThesis. Mathematisch Centrum. Amsterdam 1983.

Moore, R. Unification-Based Semantic Interpretation. In: *ACL1989*, p.33-41.(Moore89).

Pollard, C.J. Lectures on HPSG. 1985 Unpublished lecture notes, CSLI, Stanford University.

Pollard, C.J. and I.A. Sag. *An Information-Based Approach to Syntax and Semantics: Volume1 Fundamentals* CSLI Lecture Notes no. 13, Chicago University Press, Chicago 1988.

Reyle U. Compositional Semantics for LFG. In U. Reyle &C. Rohrer (eds.). *Natural Language Parsing and Linguistic Theories.* Reidel, Dordrecht 1987. (Reyle87)

Uszkoreit, H. Categorial Unification Grammars. In: COLING 11. Bonn 1986. p. 187-194.

Zeevat, H. Combining Categorial Grammar and Unification. In: Reyle, U. and C. Rohrer (eds.) Natural Language Parsing and Linguistic Theories. Dordrecht 1988.

Zeevat, H., J.Calder and E.Klein. Unification Categorial Grammar. In: Haddock, N, E.Klein and G.Morrill (eds.). Categorial Grammar, Unification Grammar and Parsing. Centre for Cognitive Science, Edinburgh 1987.