

Type Theory with Records and Unification-based Grammar

Robin Cooper

Abstract

We suggest a way of bringing together type theory and unification-based grammar formalisms by using records in type theory. The work is part of a broader project whose aim is to present a coherent unified approach to natural language dialogue semantics using tools from type theory.

1 Introduction

Uwe Mönnich has worked both on the use of type theory in semantics and on formal aspects of grammar formalisms. This paper suggests a way of bringing together type theory and unification as found in unification-based grammar formalisms like HPSG by using records in type theory which provide us with feature structure like objects. It represents a small offering to Uwe to thank him for many kindnesses over the years sprinkled with insights and rigorous comments.

This work is part of a broader project whose aim is to present a coherent unified approach to natural language dialogue semantics using tools from type theory. We are seeking to do this by bringing together Head Driven Phrase Structure Grammar (HPSG) (Sag et al. 2003), Montague semantics (Montague 1974), Discourse Representation Theory (DRT) (Kamp and Reyle 1993; van Eijck and Kamp 1997, and much other literature), situation semantics (Barwise and Perry 1983) and issue-based dialogue management (Larsson 2002) into a single type-theoretic formalism. A survey of our approach to the semantic theories (i.e., Montague semantics, DRT and situation semantics) and HPSG can be found in Cooper (2005b). Other work in progress can be found on <http://www.ling.gu.se/~cooper/records>. We give a brief summary here: Record types can be used as discourse representation structures (DRSs). Truth of a DRS corresponds to there being an object of the appropriate record type and this gives us the effect of simultaneous binding of discourse referents (corresponding to labels in records) familiar from the

semantics of DRSs in Kamp and Reyle (1993). Dependent function types provide us with the classical treatment of donkey anaphora from DRT in a way corresponding to the type theoretic treatment proposed by Mönnich (1985), Sundholm (1986) and Ranta (1994). At the same time record types can be used as feature structures of the kind found in HPSG since they have recursive structure and induce a kind of subtyping which can be used to mimic unification. Because we are using a general type theory which includes records we have functions available and a version of the λ -calculus. This means that we can use Montague's λ -calculus based techniques for compositional interpretation. From the HPSG perspective this gives us the advantage of being able to use "real" variable binding which can only be approximately simulated in pure unification based systems. From the DRT perspective this use of compositional techniques gives us an approach similar to that of Muskens (1996) and work on λ -DRT (Kohlhase et al. 1996).

In this paper we will look at the notion of unification as used in unification-based grammar formalisms like HPSG from the perspective of the type theoretical framework. This work has been greatly influenced by work of Jonathan Ginzburg (for example, Ginzburg in prep, Chap. 3). In section 2 we will give a brief informal introduction to our view of type theory with records. The version of type theory that we discuss has been made more precise in Cooper (2005a) and in an implementation called TTR (Type Theory with Records) which is under development in the Oz programming language. In section 3 we will discuss the notion of subtype which records introduce (corresponding to the notion of subsumption in the unification literature). We will then, in section 4, propose that linguistic objects are to be regarded as records whereas feature structures are to be regarded as corresponding to record *types*. Type theory is "function-based" rather than "unification-based". However, the addition of records to type theory allows us to get the advantages of unification without having to leave the "function-based" approach. We show how to do this in section 5 treating some classical simple examples which have been used to motivate the use of unification. Section 6 deals with the way in which unification analyses are used to allow the extraction of linguistic generalizations as principles in the style of HPSG. The conclusion (section 7) is that by using record types within a type theory we can have the advantages of unification-based approaches together with an additional intensionality not present in classical unification approaches and without the disadvantage of leaving the "function-based" approach which is necessary in order to deal adequately with semantics (at least).

2 Records in type theory

In this section¹, we give a very brief intuitive introduction to the kind of type theory we are employing. A more detailed and formal account can be found in Cooper (2005a) and work in progress on the project can be found on <http://www.ling.gu.se/~cooper/records>. While the type theoretical machinery is based on work carried out in the Martin-Löf approach (Coquand et al. 2004; Betarte 1998; Betarte and Tasistro 1998; Tasistro 1997) we are making a serious attempt to give it a foundation in standard set theory using Montague style recursive definitions of semantic domains. There are two main reasons for this. The first is that we think it important to show the relationship between the Montague model theoretic tradition which has been developed for natural language semantics and the proof-theoretic tradition associated with type theory. We believe that the aspects of this kind of type theory that we need can be seen as an enrichment of Montague’s original programme. The second reason is that we are interested in exploring to what extent intuitionistic and constructive approaches are appropriate or necessary for natural language. For example, we make important use of the notion “propositions as types” which is normally associated with an intuitionistic approach. However, we suspect that our Montague-like approach to defining the type theory to some extent decouples the notion from intuitionism. We would like to see type theory as providing us with a powerful collection of tools for natural language analysis which ultimately do not commit one way or the other to philosophical notions associated with intuitionism.

The central idea of records and record types can be expressed informally as follows, where $T(a_1, \dots, a_n)$ represents a type T which depends on the objects a_1, \dots, a_n .

If $a_1 : T_1, a_2 : T_2(a_1), \dots, a_n : T_n(a_1, a_2, \dots, a_{n-1})$, a record:

$$\left[\begin{array}{l} l_1 \\ l_2 \\ \dots \\ l_n \\ \dots \end{array} = \begin{array}{l} a_1 \\ a_2 \\ \dots \\ a_n \end{array} \right]$$

is of type:

$$\left[\begin{array}{l} l_1 \\ l_2 \\ \dots \\ l_n \end{array} : \begin{array}{l} T_1 \\ T_2(l_1) \\ \dots \\ T_n(l_1, l_2, \dots, l_{n-1}) \end{array} \right]$$

A record is to be regarded as a finite set of fields $\langle \ell, a \rangle$, which are ordered pairs of a label and an object. A record type is to be regarded as a finite set of fields $\langle \ell, T \rangle$ which are ordered pairs of a label and a type. The informal notation above suggests that the fields are ordered with types being dependent on previous fields in the order. This is misleading in that we regard record types as sets of fields on which a partial order is induced by the dependency relation. Dependent types give us the possibility of relating the values in fields to each other and play a crucial role in our treatment of both feature structures and semantic objects. Both records and record types are required to be the graphs of functions, that is, if $\langle \ell, \alpha \rangle$ and $\langle \ell', \beta \rangle$ are members of a given record or record type then $\ell \neq \ell'$. A record r is of record type R just in case for each field $\langle \ell, T \rangle$ in R there is a field $\langle \ell, a \rangle$ in r (i.e., with the same label) and a is of type T . Notice that the record may have additional fields not mentioned in the type. Thus a record will generally belong to several record types and any record will belong to the empty record type. This gives us a notion of subtyping which we will explore further in section 3.

Let us see how this can be applied to a simple linguistic example. We will take the content of a sentence to be modelled by a record type. The sentence

a man owns a donkey

corresponds to a record type:

$$\left[\begin{array}{l} x \quad : \quad \text{Ind} \\ c_1 \quad : \quad \text{man}(x) \\ y \quad : \quad \text{Ind} \\ c_2 \quad : \quad \text{donkey}(y) \\ c_3 \quad : \quad \text{own}(x,y) \end{array} \right]$$

A record of this type will be:

$$\left[\begin{array}{l} x \quad = \quad a \\ c_1 \quad = \quad p_1 \\ y \quad = \quad b \\ c_2 \quad = \quad p_2 \\ c_3 \quad = \quad p_3 \end{array} \right]$$

where

a, b are of type *Ind*, individuals

p_1 is a proof of $\text{man}(a)$

p_2 is a proof of $\text{donkey}(b)$

p_3 is a proof of $\text{own}(a, b)$.

Note that the record may have had additional fields and still be of this type. The types ‘ $\text{man}(x)$ ’, ‘ $\text{donkey}(y)$ ’, ‘ $\text{own}(x,y)$ ’ are dependent types of proofs (in a convenient but not quite exact abbreviatory notation – we will give a

more precise account of dependencies within record types in section 3). The use of types of proofs for what in other theories would be called propositions is often referred to as the notion of “propositions as types”. Exactly what type ‘man(x)’ is depends on which individual you choose in your record to be labelled by ‘x’. If the individual a is chosen then the type is the type of proofs that a is a man. If another individual d is chosen then the type is the type of proofs that d is a man, and so on. What is a proof? Martin-Löf considers proofs to be objects rather than arguments or texts. For non-mathematical propositions proofs can be regarded as situations or events. For useful discussion of this see Ranta (1994), p 53ff. We discuss it in more detail in Cooper (2005a).

There is an obvious correspondence between this record type and a discourse representation structure (DRS) as characterised in Kamp and Reyle (1993). The characterisation of what it means for a record to be of this type corresponds in an obvious way to the standard embedding semantics for such a DRS which Kamp and Reyle provide.

Records (and record types) are recursive in the sense that the value corresponding to a label in a field can be a record (or record type)². For example,

$$r = \left[\begin{array}{l} f = \left[\begin{array}{l} f = \left[\begin{array}{l} ff = a \\ gg = b \end{array} \right] \\ g = c \end{array} \right] \\ g = \left[\begin{array}{l} h = \left[\begin{array}{l} g = a \\ h = d \end{array} \right] \end{array} \right] \end{array} \right]$$

is of type

$$R = \left[\begin{array}{l} f : \left[\begin{array}{l} f : \left[\begin{array}{l} ff : T_1 \\ gg : T_2 \end{array} \right] \\ g : T_3 \end{array} \right] \\ g : \left[\begin{array}{l} h : \left[\begin{array}{l} g : T_1 \\ h : T_4 \end{array} \right] \end{array} \right] \end{array} \right]$$

given that $a : T_1$, $b : T_2$, $c : T_3$ and $d : T_4$. We can use *path-names* in records and record types to designate values in particular fields, e.g.

$$r.f = \left[\begin{array}{l} f = \left[\begin{array}{l} ff = a \\ gg = b \end{array} \right] \\ g = c \end{array} \right]$$

$$R.f.f.f = T_1$$

The recursive nature of records and record types will be important later in the paper when we use record types to correspond to linguistic feature structures.

Another important aspect of the type theory we are using is that types themselves can also be treated as objects.³ A simple example of how this

can be exploited is the following representation for *a girl believes that a man owns a donkey*. This is a simplified version of the treatment discussed in Cooper (2005a).

$$\left[\begin{array}{l} x : Ind \\ c_1 : \text{girl}(x) \\ c_2 : \text{believe}(x, \left[\begin{array}{l} y : Ind \\ c_3 : \text{man}(y) \\ z : Ind \\ c_4 : \text{donkey}(z) \\ c_5 : \text{own}(y, z) \end{array} \right]) \end{array} \right]$$

The treatment of types as first class objects in this way is a feature which this type theory has in common with situation theory and it is an important component in allowing us to incorporate analyses from situation semantics in our type theoretical treatment.

The theory of records and record types is embedded in a general type theory. This means that we have functions and function types available giving us a version of the λ -calculus. We can thus use Montague's techniques for compositional interpretation. For example, we can interpret the common noun *donkey* as a function which maps records r of the type $[x:Ind]$ (i.e. records which introduce an individual labelled with the label 'x') to a record type dependent on r . We notate the function as follows:

$$\lambda r: [x:Ind] ([c: \text{donkey}(r.x)])$$

The type of this function is

$$P = ([x:Ind])\text{RecType}$$

This corresponds to Montague's type $\langle e, t \rangle$ (the type of functions from individuals (entities) to truth-values). In place of individuals we use records introducing individuals with the label 'x' and in place of truth-values we use record types which, as we have seen above, correspond to an intuitive notion of proposition (in particular a proposition represented by a DRS). Using the power of the λ -calculus we can treat determiners Montague-style as functions which take two arguments of type P and return a record type. For example, we represent the indefinite article by

$$\lambda R_1: ([x:Ind])\text{RecType} \quad \lambda R_2: ([x:Ind])\text{RecType} \quad \left(\begin{array}{l} \text{par} : [x : Ind] \\ \text{restr} : R_1 @ \text{par} \\ \text{scope} : R_2 @ \text{par} \end{array} \right)$$

Here we use $F @ a$ to represent the result of applying function F to argument a .

The type theory includes dependent function types. These can be used to give a classical treatment of universal quantification corresponding to DRT's '⇒'. For example, an interpretation of *every man owns a donkey* can be the following record type:

$$\left[\begin{array}{l} f : (r : \left[\begin{array}{l} x : Ind \\ c_1 : \text{man}(x) \end{array} \right]) \left[\begin{array}{l} y : Ind \\ c_2 : \text{donkey}(y) \\ c_3 : \text{own}(r.x,y) \end{array} \right] \end{array} \right]$$

Records of the type

$$(r : \left[\begin{array}{l} x : Ind \\ c_1 : \text{man}(x) \end{array} \right]) \left[\begin{array}{l} y : Ind \\ c_2 : \text{donkey}(y) \\ c_3 : \text{own}(r.x,y) \end{array} \right]$$

map records r of type

$$\left[\begin{array}{l} x : Ind \\ c_1 : \text{man}(x) \end{array} \right]$$

to records of type

$$\left[\begin{array}{l} y : Ind \\ c_2 : \text{donkey}(y) \\ c_3 : \text{own}(r.x,y) \end{array} \right]$$

Our interpretation of *every man owns a donkey* requires that there exist a function of this type. Why do we use the record type with the label 'f' rather than the function type itself as the interpretation of the sentence? One reason is to achieve a uniform treatment where the interpretation of a sentence is always a record type. Another reason is that the label gives us a handle which can be used to anaphorically refer to the function. This can, for example, be exploited in so-called paycheck examples Karttunen (1969) such as *Everybody receives a paycheck. Not everybody pays it into the bank immediately, though.*

The final notion we will introduce which is important for the modelling of HPSG typed feature structures as record types is that of *manifest field*. This notion is introduced in Coquand et al. (2004). It builds on the notion of singleton type. If $a : T$, then T_a is a *singleton type* and $b : T_a$ iff $b = a$. A *manifest field* in a record type is one whose type is a singleton type, e.g.

$$\left[\begin{array}{l} x : T_a \end{array} \right]$$

written for convenience as

$$\left[\begin{array}{l} x=a : T \end{array} \right]$$

This notion allows record types to be "progressively instantiated", i.e. intuitively, for values to be specified within a record type. A record type that only contains manifest fields is completely instantiated and there will be exactly one record of that type. We will allow dependent singleton types, where a in

T_a can be represented by a path in a record type. Manifest fields are important for the modelling of HPSG-style unification in type theory with records.

3 Dependent types and the subtype relation

We are now in a position to give more detail about the treatment of dependencies in record types. Dependent types within record types are treated as pairs consisting of functions and sequences of path-names providing corresponding to the arguments required by the functions. Thus the type on p. 4 corresponding to *a man owns a donkey* is in official, though less readable, notation:

$$\left[\begin{array}{l} x : Ind \\ c_1 : \langle \lambda v : Ind(\text{man}(v)), \langle x \rangle \rangle \\ y : Ind \\ c_2 : \langle \lambda v : Ind(\text{donkey}(v)), \langle y \rangle \rangle \\ c_3 : \langle \lambda v : Ind(\lambda w : Ind(\text{own}(v, w))), \langle x, y \rangle \rangle \end{array} \right]$$

This enables us to give scope to dependencies outside the object in which the dependency occurs. Thus if, on the model of the type for *a girl believes that a man owns a donkey* on p. 6, we wish to construct a type corresponding to *a girl believes that she owns a donkey* (where *she* is anaphorically related to a *girl*), this can be done as follows:

$$\left[\begin{array}{l} x : Ind \\ c_1 : \langle \lambda v : Ind(\text{girl}(v)), \langle x \rangle \rangle \\ c_2 : \langle \lambda u : Ind(\text{believe}(u, \left[\begin{array}{l} z : Ind \\ c_4 : \langle \lambda v : Ind(\text{donkey}(v)), \langle z \rangle \rangle \\ c_5 : \langle \lambda v : Ind(\text{own}(u, v)), \langle z \rangle \rangle \end{array} \right])), \langle x \rangle \rangle \end{array} \right]$$

There are two kinds of path-names which can occur in such types: *relative path-names* which are constructed from labels, $\ell_1. \dots .\ell_n$, and *absolute path-names* which refer explicitly to the record, r in which the path-name is to be evaluated, $r.\ell_1. \dots .\ell_n$. A *dependent record type* is a set of pairs of the form $\langle \ell, T \rangle$ where ℓ is a label and T is either a type, a dependent record type or a pair consisting of a function and a sequence of path-names as characterized above. An *anchor* for a dependent record type T of the form

$$\left[\begin{array}{l} \ell_1 : T_1 \\ \dots \\ \ell_n : T_n \end{array} \right]$$

is a record, h , such that for each T_i of the form $\langle f, \langle \pi_1, \dots, \pi_m \rangle \rangle$, π_i is either an absolute path or a path defined in h , and for each T_i which is a dependent record type, h is also an anchor for T_i . In addition we require that the result

of anchoring T with h as characterized below is well-defined, i.e., that the anchor provides arguments of appropriate types to functions and provides objects of appropriate types for the construction of singleton types as required by the anchoring. The result of *anchoring* T with h , $T[h]$ is obtained by replacing

1. each T_i in T of the form $\langle f, \langle \pi_1, \dots, \pi_m \rangle \rangle$ with $f(\pi_1[h] \dots (\pi_m[h]))$ (where $\pi_i[h]$ is the value of $h.\pi_i$; if π_i is a relative path-name and the value of π_i if π_i is an absolute path-name)
2. each T_i in T which is a dependent record type with $T_i[h]$
3. each basic type which is the value of a path π in T and for which $h.\pi$ is defined, with T_a , where a is the value of $h.\pi$, i.e. the singleton type obtained from T and the value of $h.\pi$.

A dependent record type T is said to be *closed* just in case each path which T requires to be defined in an anchor for T is defined within T itself. It is the closed dependent record types which belong to our type universe. If T is a closed dependent record type then $r : T$ if and only if $r : T[r]$.

Let us return to our type above for *a girl believes that she owns a donkey*. An anchor for this type is

$$\left[\begin{array}{l} x = m \end{array} \right]$$

(where m is an object of type *Ind*) and the result of anchoring the type with this record is

$$\left[\begin{array}{l} x=m : \text{Ind} \\ c_1 : \text{girl}(m) \\ c_2 : \text{believe}(m, \left[\begin{array}{l} z : \text{Ind} \\ c_4 : \langle \lambda v : \text{Ind}(\text{donkey}(v)), \langle z \rangle \rangle \\ c_5 : \langle \lambda v : \text{Ind}(\text{own}(m, v)), \langle z \rangle \rangle \end{array} \right]) \end{array} \right]$$

Notice that the anchor has no effect on dependencies with scope within the argument type corresponding to *that she owns a donkey* but only the dependency with scope external to it.

We now turn our attention to the subtype relation. Record types introduce a notion of subtype which corresponds to what is known as subsumption in the unification literature. The subtype relation can be characterized model theoretically as follows:

If T_1 and T_2 are types, then T_1 is a subtype of T_2 ($T_1 \sqsubseteq T_2$) just in case
 $\{a \mid a :_M T_1\} \subseteq \{a \mid a :_M T_2\}$ for all models M .

where the right-hand side of this equivalence refers to sets in the sense of classical set theory and models as defined in Cooper (2005a). These models assign sets of objects to the basic types and sets of proofs to proof-types constructed from predicates with appropriate arguments, e.g. if k is a woman according to model M then M will assign a non-empty set of proofs to the type $woman(k)$. Such models correspond to sorted first-order models. If this notion of subtype is to be computationally useful, we need some way of computing whether two types stand in the subtype relation without having to compute the sets of objects which belong to those types in all possible models. Thus we define another relation \sqsubseteq_c which is computed without reference to the models.

The approach taken to this in the implementation TTR is to instantiate (dependent) record types, R , recursively as an anchor for R introducing arbitrary formal objects guaranteed to be of the appropriate type. Basic types and types constructed with predicates are instantiated to arbitrary formal objects guaranteed to be of the type (in the implementation, pairings of gensym atoms with the type); singleton types are instantiated to the object used to define the type; record type structures are instantiated to records containing the instantiations of the types (or anchors for the dependent record types) in each field; similar instantiations are given for other complex types. Thus the instantiation of the record type

$$\left[\begin{array}{l} f : T_1 \\ g=b : T_2 \\ h : \left[\begin{array}{l} i : r_1(f,g) \\ j : r_2(g,f) \end{array} \right] \end{array} \right]$$

can be represented as:

$$\left[\begin{array}{l} f = a0\#T_1 \\ g = b \\ h = \left[\begin{array}{l} i = a1\#r_1(a0\#T_1, b) \\ j = a2\#r_2(b, a0\#T_1) \end{array} \right] \end{array} \right]$$

We use $Inst(T)$ to represent such an instantiation of type T . $T_1 \sqsubseteq_c T_2$ just in case $Inst(T_1) : T_2$. One advantage of this approach is that the computation of the subtype relation will be directly dependent on the of-type relation. If T_1 is a record type containing a superset of the fields of the record type T_2 then $T_1 \sqsubseteq_c T_2$ as desired for the modelling of subsumption in unification systems. Thus, for example,

$$\left[\begin{array}{l} f:T_1 \\ g:T_2 \\ h:T_3 \end{array} \right] \sqsubseteq_c \left[\begin{array}{l} f:T_1 \\ g:T_2 \end{array} \right]$$

This method of computing the subtype relation appears to be sound with respect to the models but not necessarily complete since it does not take account of the logic associated with predicates. For example, later in the paper we will make use of an equality predicate. The predicate eq is such that $a : eq(T, x, y)$ iff $a = \langle x, y \rangle$, $x, y : T$, and $x = y$. Now consider that the type

$$\begin{bmatrix} x:T \\ y:T \\ c:r(x) \end{bmatrix}$$

is not a subtype of

$$\begin{bmatrix} x:T \\ y:T \\ c:r(y) \end{bmatrix}$$

whereas according to the model theoretic definition

$$\begin{bmatrix} x:T \\ y:T \\ c:r(x) \\ d:eq(T, x, y) \end{bmatrix} \sqsubseteq \begin{bmatrix} x:T \\ y:T \\ c:r(y) \end{bmatrix}$$

since anything of the first type must also be of the second type. The instantiation of the first type

$$\begin{bmatrix} x=a0\#T \\ y=a1\#T \\ c=a2\#r(a0\#T) \\ d=a3\#eq(T, a0\#T, a1\#T) \end{bmatrix}$$

will not, however, be computed as being of the second type unless we take account of the import of the equality predicate. This is easily fixed, for example by normalizing the instantiation so that all arbitrary objects which are required to be identical are represented by the same symbols, in this case, for example, substituting $a0$ for all occurrences of $a1$:

$$\begin{bmatrix} x=a0\#T \\ y=a0\#T \\ c=a2\#r(a0\#T) \\ d=a3\#eq(T, a0\#T, a0\#T) \end{bmatrix}$$

This will then have the desired effect on the computation of the subtype relation. However, there is no guarantee that it will always be possible to give a complete characterization of the subtype relation if the logic of the predicates is incomplete. However, we should not let this stop us exploiting those inferences about subtyping which we can draw in a computational implementation.

4 Records as linguistic objects

We will consider linguistic objects to be records. Here is a simple linguistic object which might correspond to the word *man*.

$$\left[\begin{array}{l} \text{phon} = [\text{man}] \\ \text{cat} = \text{n} \\ \text{agr} = \left[\begin{array}{l} \text{num} = \text{sg} \\ \text{gen} = \text{masc} \\ \text{pers} = \text{third} \end{array} \right] \end{array} \right]$$

It is a record with three fields. The field for phon(ology) has as value a (singleton) list of words (following the traditional HPSG simplifying assumption about phonology). For the cat(egory) field we will use atomic categories like n(oun), although nothing in our approach excludes the complex categories normally used in HPSG analyses. We include three agr(eement) features: num(ber), in this case with the value s(in)g(ular), gen(der), in this case with the value masc(uline) and pers(on) in this case with the value third (person).

Not all words correspond to a single linguistic object. For example, the English word *fish* can be singular or plural and masculine, feminine or neuter. This means that there will be six records corresponding to the single record for *man*. Here are three of them:

$$\left[\begin{array}{l} \text{phon} = [\text{fish}] \\ \text{cat} = \text{n} \\ \text{agr} = \left[\begin{array}{l} \text{num} = \text{sg} \\ \text{gen} = \text{neut} \\ \text{pers} = \text{third} \end{array} \right] \end{array} \right]$$

$$\left[\begin{array}{l} \text{phon} = [\text{fish}] \\ \text{cat} = \text{n} \\ \text{agr} = \left[\begin{array}{l} \text{num} = \text{pl} \\ \text{gen} = \text{neut} \\ \text{pers} = \text{third} \end{array} \right] \end{array} \right]$$

$$\left[\begin{array}{l} \text{phon} = [\text{fish}] \\ \text{cat} = \text{n} \\ \text{agr} = \left[\begin{array}{l} \text{num} = \text{sg} \\ \text{gen} = \text{masc} \\ \text{pers} = \text{third} \end{array} \right] \end{array} \right]$$

Now let us consider *types* of linguistic objects. Nouns correspond to objects which have a phonology, the category *n* and the agreement features for number, gender and person. That is we can define a record type *Noun* as

follows:

$$Noun \equiv \left[\begin{array}{l} \text{phon} : Phon \\ \text{cat=n} : Cat \\ \text{agr} : \left[\begin{array}{l} \text{num} : Number \\ \text{gen} : Gender \\ \text{pers} : Person \end{array} \right] \end{array} \right]$$

where:

$Phon \equiv [Lex]$ (i.e. type of list of objects of type Lex)

the, a, fish, man, men, swim, swims, swam, ... : Lex

n, det, np, v, vp, s, ... : Cat

sg, pl : $Number$

masc, fem, neut : $Gender$

first, second, third : $Person$

We can further define types for determiners, verbs and agreement:

$$Det \equiv \left[\begin{array}{l} \text{phon} : Phon \\ \text{cat=det} : Cat \\ \text{agr} : \left[\begin{array}{l} \text{num} : Number \\ \text{gen} : Gender \\ \text{pers} : Person \end{array} \right] \end{array} \right]$$

$$V \equiv \left[\begin{array}{l} \text{phon} : Phon \\ \text{cat=v} : Cat \\ \text{agr} : \left[\begin{array}{l} \text{num} : Number \\ \text{gen} : Gender \\ \text{pers} : Person \end{array} \right] \end{array} \right]$$

$$Agr \equiv \left[\begin{array}{l} \text{num} : Number \\ \text{gen} : Gender \\ \text{pers} : Person \end{array} \right]$$

Now we can define the type of linguistic objects corresponding to the word *man*.

$$Man \equiv \left[\begin{array}{l} \text{phon}=[\text{man}] : Phon \\ \text{cat=n} : Cat \\ \text{agr} : \left[\begin{array}{l} \text{num=sg} : Number \\ \text{gen=masc} : Gender \\ \text{pers=third} : Person \end{array} \right] \end{array} \right]$$

This type identifies a unique linguistic object (namely the record corresponding to *man* which we introduced above). It is a singleton (or fully specified) type. It is also a *subtype* (or specification) of *Noun* in the sense that if an object is of type *Man* it is also of type *Noun*. We define the type for the plural *men* in a similar way.

$$Men \equiv \left[\begin{array}{l} \text{phon}=[\text{men}] : Phon \\ \text{cat}=\text{n} : Cat \\ \text{agr} : \left[\begin{array}{l} \text{num}=\text{pl} : Number \\ \text{gen}=\text{masc} : Gender \\ \text{pers}=\text{third} : Person \end{array} \right] \end{array} \right]$$

The type *Fish* corresponding to the noun *fish* is a less specified type, however:

$$Fish \equiv \left[\begin{array}{l} \text{phon}=[\text{fish}] : Phon \\ \text{cat}=\text{n} : Cat \\ \text{agr} : \left[\begin{array}{l} \text{num} : Number \\ \text{gen} : Gender \\ \text{pers}=\text{third} : Person \end{array} \right] \end{array} \right]$$

The objects which are of this type will be the six records which we identified earlier. *Fish* is also a subtype of *Noun*.

We can also define two types *IndefArt* and *DefArt* corresponding to the indefinite and definite articles which have different degrees of specification:

$$IndefArt \equiv \left[\begin{array}{l} \text{phon}=[\text{a}] : Phon \\ \text{cat}=\text{det} : Cat \\ \text{agr} : \left[\begin{array}{l} \text{num}=\text{sg} : Number \\ \text{gen} : Gender \\ \text{pers}=\text{third} : Person \end{array} \right] \end{array} \right]$$

$$DefArt \equiv \left[\begin{array}{l} \text{phon}=[\text{the}] : Phon \\ \text{cat}=\text{det} : Cat \\ \text{agr} : \left[\begin{array}{l} \text{num} : Number \\ \text{gen} : Gender \\ \text{pers}=\text{third} : Person \end{array} \right] \end{array} \right]$$

Both of these are subtypes of *Det*. *IndefArt* is specified for both number and person whereas *DefArt* is only specified for person.

Similar differences in specification arise in verbs:⁴

$$Swims \equiv \left[\begin{array}{l} \text{phon}=[\text{swims}] : Phon \\ \text{cat}=\text{v} : Cat \\ \text{agr} : \left[\begin{array}{l} \text{num}=\text{sg} : Number \\ \text{gen} : Gender \\ \text{pers}=\text{third} : Person \end{array} \right] \end{array} \right]$$

$$Swim \equiv \left[\begin{array}{l} \text{phon}=[\text{swim}] : Phon \\ \text{cat}=\text{v} : Cat \\ \text{agr} : \left[\begin{array}{l} \text{num}=\text{pl} : Number \\ \text{gen} : Gender \\ \text{pers}=\text{third} : Person \end{array} \right] \end{array} \right]$$

$$Swam \equiv \left[\begin{array}{ll} \text{phon}=[\text{swam}] & : \textit{Phon} \\ \text{cat}=\textit{v} & : \textit{Cat} \\ \text{agr} & : \left[\begin{array}{ll} \text{num} & : \textit{Number} \\ \text{gen} & : \textit{Gender} \\ \text{pers}=\textit{third} & : \textit{Person} \end{array} \right] \end{array} \right]$$

These three types are all subtypes of *V*.

5 A type theoretical approach to unification phenomena

The types that we introduced in section 4 lay the basis for a kind of unification phenomenon in language which has been discussed in the classical literature on unification approaches to natural language grammar (Shieber 1986, e.g.). The sentence (1) is underspecified with respect to number.

- (1) The fish swam

Note, however, that either all the words are singular or all the words are plural. It cannot be the case that *fish* is regarded as singular while *swam* is regarded as plural, for example. This is because there are requirements that the determiner and the noun agree in number and that the subject noun-phrase and the verb agree in number. In a unification-based grammar this is expressed by requiring that the number features of the relevant phrases unify. In the terms of our previous discussion it means that there are two linguistic objects corresponding to (1) rather than eight. Note that the sentences in (2) are all singular.

- (2) a. a fish swam
 b. the man swam
 c. the fish swims

However, the “source” of the singularity is different in each case. It is the fact that *a*, *man*, and *swims* respectively are specified for singular, together with the requirements that all the number features unify which have as a consequence that all the words are specified for singular in the single linguistic object corresponding to each of these sentences. Unification is regarded as a useful tool in the linguistic analysis because it reflects the lack of “directionality” of the agreement phenomenon, that is, as long as one of the words is specified for number they all have to be specified for the same number. Unification is traditionally regarded as partial, that is, it can fail and this is

used to explain why the strings of words in (3) are not sentences of English, that is, they do not correspond to linguistic objects allowed by the grammar of English.

- (3) a. *a fish swim
 b. *the man swim
 c. *a men swims

On our type theoretical view the intuitive notion of unification is related to meet (as in the meet, or conjunction, of two types) and equality. In the definition of our type theory in Cooper(2005b) we introduce meet-types in the following way:

If T_1 and T_2 are types, then $T_1 \wedge T_2$ is also a type.

$a : T_1 \wedge T_2$ iff $a : T_1$ and $a : T_2$

If T_1 and T_2 are record types then there will always be a record type (not a meet) T_3 which is equivalent to $T_1 \wedge T_2$ (in the sense that $a : T_3$ iff $a : T_1 \wedge T_2$).

Let us consider some examples:

$$\begin{aligned} [f:T_1] \wedge [g:T_2] &\approx \begin{bmatrix} f:T_1 \\ g:T_2 \end{bmatrix} \\ [f:T_1] \wedge [f:T_2] &\approx [f:T_1 \wedge T_2] \end{aligned}$$

Below we present some informal pseudocode for a function μ which will simplify meets of records types, returning an equivalent record type. The algorithm is similar in essential respects to the graph unification algorithm used in classical implementations of feature based grammar systems (Shieber 1986). One important respect in which it differs from the classical unification algorithm is that it never fails. In cases where the corresponding unification would have failed it will return a record type which is equivalent to the distinguished empty type \perp .⁵ Another way in which it differs from the classical unification algorithm is that it applies to all types, reducing meets of records types to non-meet types and recursively performing this reduction within record types and otherwise returning the original type. The algorithm that is informally presented here is a simplification of the one that is implemented in TTR which has some additional special cases and also has an additional level of complication for the handling of dependent types, using the technique of environments which we referred to in section 3. In order to understand the intention of this pseudocode it is important to remember that record types are considered to be finite sets of ordered pairs (representing the fields) as

described above in section 2. When we write $\text{Map}(T, \lambda, v[\Phi])$ we mean that each field $\langle \ell, T' \rangle$ in T is to be replaced by the result of applying the function $\lambda, v[\Phi]$ to ℓ and T' . When we say that $T.l$ is defined we mean that for some $T', \langle \ell, T' \rangle \in T$. We assume that the type theory will define an incompatibility relation which holds between certain basic types such that if T_1 and T_2 are incompatible then there will be no a such that $a : T_1$ and $a : T_2$. For example, one might require that all basic types are pairwise incompatible.

```

 $\mu(T) =$ 
if for some  $T_1, T_2, T = T_1 \wedge T_2$  then
  let
     $T_1' = \mu(T_1)$ 
     $T_2' = \mu(T_2)$ 
  in
    if  $T_1' \sqsubseteq T_2'$  then  $T_1'$ 
    elseif  $T_2' \sqsubseteq T_1'$  then  $T_2'$ 
    elseif  $T_1'$  and  $T_2'$  are incompatible, then  $\perp$ 
    elseif  $T_1'$  and  $T_2'$  are record types then
       $\text{Map}(T_1', \lambda, v[\text{if } T_2'.l \text{ is defined then } \langle l, \mu(v \wedge T_2'.l) \rangle \text{ else } \langle l, v \rangle])$ 
       $\cup$ 
       $(T_2' - \{\langle l, v \rangle \in T_2' \mid T_1'.l \text{ is defined}\})$ 
    else  $T_1' \wedge T_2'$ 
  end
end
elseif  $T$  is a record type, then
   $\text{Map}(T, \lambda, v[\langle l, \mu(v) \rangle])$ 
else  $T$ 
end

```

If we know $a : T_1$ and $b : T_2$ and in addition know $a = b$ then we know $a : T_1 \wedge T_2$ and $a : \mu(T_1 \wedge T_2)$. Intuitively, equality of objects corresponds to meet (or “unification”) of types. We can exploit this in characterizing a type *NP* which allows noun-phrases consisting of a determiner and a noun which agree in number.

$NP \equiv$

$$\left[\begin{array}{l} \text{phon}=\text{append}(\text{daughters.first.phon}, \text{daughters.rest.first.phon}):Phon \\ \text{cat}=\text{np}:Cat \\ \text{daughters}: \left[\begin{array}{l} \text{first} : Det \\ \text{rest} : \left[\begin{array}{l} \text{first} : Noun \\ \text{rest=nil} : [Sign] \end{array} \right] \end{array} \right] \\ \text{agr}=\text{daughters.rest.first.agr}:Agr \\ \text{c}:eq(Number, \text{daughters.first.agr.num}, \text{daughters.rest.first.agr.num}) \end{array} \right]$$
 Here *Sign* is to be thought of as a recursively defined type defined by:

1. if $a : Det$ then $a : Sign$
2. if $a : Noun$ then $a : Sign$
3. if $a : NP$ then $a : Sign$

and so on for other types corresponding to word and phrase types

...

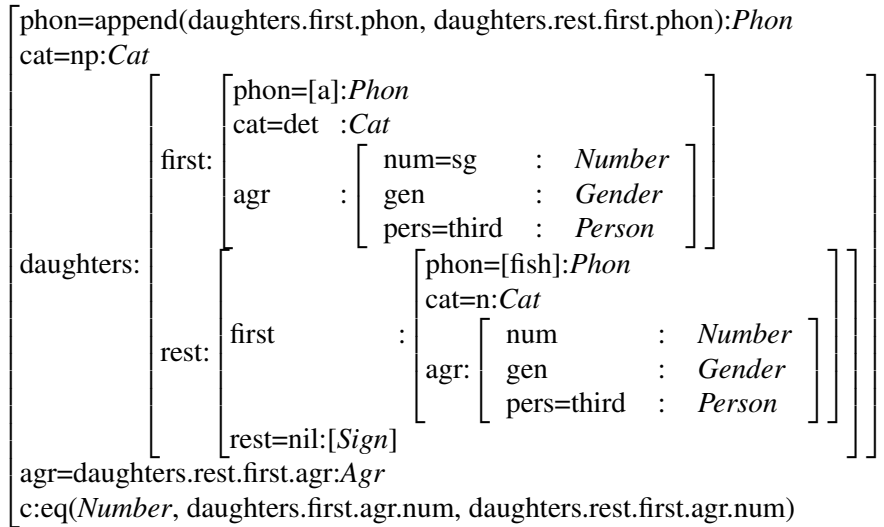
- n. no object is of type *Sign* except as required by the above clauses

The predicate *eq* is as defined in section 3.

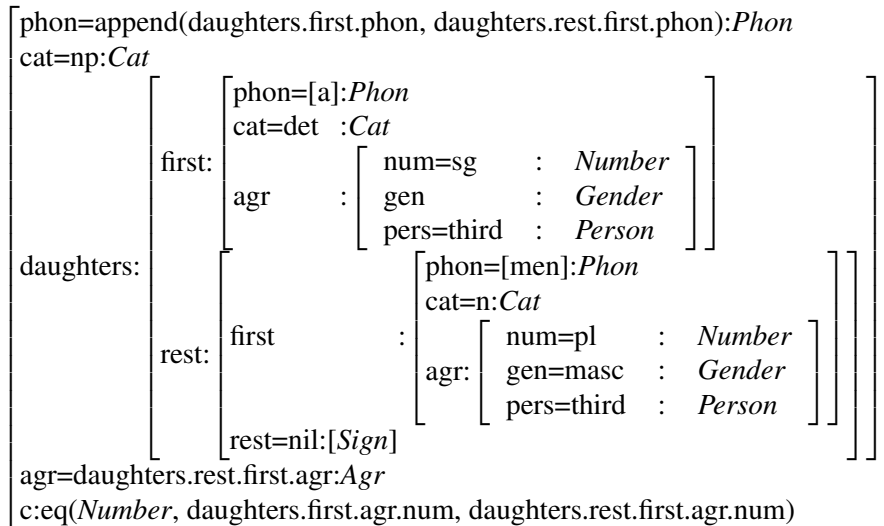
The type *NP* can be further specified to a type where the first daughter is of type *DefArt* and the second daughter is of type *Man* since these types are subtypes of *Det* and *Noun* respectively.

$$\left[\begin{array}{l} \text{phon}=\text{append}(\text{daughters.first.phon}, \text{daughters.rest.first.phon}):Phon \\ \text{cat}=\text{np}:Cat \\ \text{daughters}: \left[\begin{array}{l} \text{first}: \left[\begin{array}{l} \text{phon}=[\text{the}]:Phon \\ \text{cat}=\text{det} : Cat \\ \text{agr} : \left[\begin{array}{l} \text{num} : Number \\ \text{gen} : Gender \\ \text{pers}=\text{third} : Person \end{array} \right] \end{array} \right] \\ \text{rest}: \left[\begin{array}{l} \text{first} : \left[\begin{array}{l} \text{phon}=[\text{man}]:Phon \\ \text{cat}=\text{n}:Cat \\ \text{agr}: \left[\begin{array}{l} \text{num}=\text{sg} : Number \\ \text{gen}=\text{masc} : Gender \\ \text{pers}=\text{third} : Person \end{array} \right] \end{array} \right] \\ \text{rest=nil}: [Sign] \end{array} \right] \end{array} \right] \\ \text{agr}=\text{daughters.rest.first.agr}:Agr \\ \text{c}:eq(Number, \text{daughters.first.agr.num}, \text{daughters.rest.first.agr.num}) \end{array} \right]$$

Note that this type represents that the singularity of the phrase has its source in *man*. Similarly we can create the type corresponding to *a fish* where the source of the singularity is the determiner.



A difference between our record types and feature structures is that the record types preserve the information of the source of the number information in these examples whereas in feature structures this information is lost once the feature structures have been unified. An additional difference is that we are able to form types corresponding to ungrammatical phrases such as **a men*.



This is a well-formed type but one that cannot have any elements since *sg* and *pl* are not identical. This type is thus equivalent to \perp . Such types might be usefully exploited in robust parsing. Note that even though the type is empty it contains a great deal of information about the phrase. In particular if our types were to include information about the meaning or content of a phrase it might be possible to extract information about the meaning of a phrase even though it does not actually correspond to any well-formed linguistic object. This could potentially be exploited in a way similar to that suggested by Fouvry (2003) for weighted feature structures.

6 Using unification to express generalizations

Unification is also exploited in unification grammars to extract generalities from individual cases. For example, the noun-phrase agreement phenomenon that we discussed in section 5 requires that the agreement features on the noun be the same as those on the NP. This is an instance of the head feature principle which requires that the agreement features of the mother be the same as those of the head daughter. If we identify the head daughter in phrases then we can extract this principle out by creating a new type *HFP* which corresponds to the head feature principle.

$$HFP \equiv \left[\begin{array}{ll} \text{hd-daughter} & : \textit{Sign} \\ \text{agr=hd-daughter.agr} & : \textit{Agr} \end{array} \right]$$

We also define a new version of the type *NP*, *NP'*, which identifies the head daughter but does not contain the information corresponding to the head feature principle.

$$NP' \equiv \left[\begin{array}{ll} \text{phon=append(daughters.first.phon, daughters.rest.first.phon)} & : \textit{Phon} \\ \text{cat=np} & : \textit{Cat} \\ \text{hd-daughter=daughters.rest.first} & : \textit{Noun} \\ \text{daughters:} \left[\begin{array}{ll} \text{first} & : \textit{Det} \\ \text{rest} & : \left[\begin{array}{ll} \text{first} & : \textit{Noun} \\ \text{rest=nil} & : [\textit{Sign}] \end{array} \right] \end{array} \right] \\ \text{agr} & : \textit{Agr} \\ \text{c:eq(Number, daughters.first.agr.num, daughters.rest.first.agr.num)} & \end{array} \right]$$

The record type that characterizes noun-phrases is now $\mu(NP' \wedge HFC)$.

7 Conclusions

We have shown how a type theory with records gives us a notion of subtyping corresponding to subsumption in the unification literature and a way of reducing meets of record types to record types which is similar to the graph unification used in unification-based grammar formalisms. Using record types instead of feature structures gives us a kind of intensionality which is not available in feature structures. This intensionality allows us to distinguish equivalent types which preserve information which is lost in the unification of feature structures, such as the source of grammatical information associated with a phrase. This intensionality can also be exploited by associating empty types with ungrammatical phrases. Such types may contain information which could be used in robust parsing. While it may appear odd to refer to this property of as “intensionality” in the context of parsing, we do so because it is the same kind of intensionality which is important for our approach to the semantic analysis of attitudes such as *know* and *believe*. The type theory provides us with a level of abstraction which permits us to make generalizations across phenomena in natural language that have previously been treated by separate theories. Finally, this approach to unification is embedded in a rich “function-based” type theoretical framework which provides us with the kind of tools that are needed for semantics while at the same time allowing us to import unification into our semantic analysis.

Notes

This work was supported by Swedish Research Council projects numbers 2002-4879 *Records, types and computational dialogue semantics* and 2005-4211 *Library-based Grammar Engineering*. I am grateful to Thierry Coquand, Dan Flickinger, Jonathan Ginzburg, Erhard Hinrichs, Bengt Nordström and Aarne Ranta for discussion in connection with this work.

1. This section contains revised material from Cooper (2005a)
2. There is a technical sense in which this recursion is non-essential. These records could also be viewed as non-recursive records whose labels are sequences of atomic labels. See Cooper (2005a) for more discussion.

3. In order to do this safely we stratify the types. We define the type system as a family of type systems of order n for each natural number n . The idea is that types which are not defined in terms of other types are of order 0 and that types which are defined in terms of types of order n are of order $n + 1$. We will not discuss this in detail here but rely on the discussion in Cooper (2005a). In this paper we will suppress reference to order in the specification of our types.
4. We are making the simplifying assumption that all the verb forms represented here are third person.
5. Any record type which has \perp in one of its fields will be such that there are no records of that type and thus the type will be equivalent to \perp .

References

- Barwise, Jon and John Perry
 1983 *Situations and Attitudes*. Bradford Books. MIT Press, Cambridge, Mass.
- Betarte, Gustavo
 1998 *Dependent Record Types and Algebraic Structures in Type Theory*. Ph.D. thesis, Department of Computing Science, Göteborg University and Chalmers University of Technology.
- Betarte, Gustavo and Alvaro Tasistro
 1998 Extension of Martin-Löf's type theory with record types and subtyping. In Giovanni Sambin and Jan Smith, (eds.), *Twenty-Five Years of Constructive Type Theory*, number 36 in Oxford Logic Guides. Oxford University Press, Oxford.
- Cooper, Robin
 2005a Austinian truth, attitudes and type theory. *Research on Language and Computation*, 3: 333–362.
 2005b Records and record types in semantic theory. *Journal of Logic and Computation*, 15(2): 99–112.
- Coquand, Thierry, Randy Pollack, and Makoto Takeyama
 2004 A logical framework with dependently typed records. *Fundamenta Informaticae*, XX: 1–22.
- Fouvry, Frederik
 2003 Constraint relaxation with weighted feature structures. In *IWPT 03, International Workshop on Parsing Technologies*. Nancy (France).
- Ginzburg, Jonathan
 in prep Semantics and interaction in dialogue. Draft available from <http://www.dcs.kcl.ac.uk/staff/ginzburg/papers.html>.
- Kamp, Hans and Uwe Reyle
 1993 *From Discourse to Logic*. Kluwer, Dordrecht.
- Karttunen, Lauri
 1969 Pronouns and variables. In R.I. Binnick, A. Davison, G.M. Green, and J.L. Morgan, (eds.), *Papers from the Fifth Regional Meeting of the Chicago Linguistic Society*, pp. 108–115. Department of Linguistics, University of Chicago, Chicago, Illinois.
- Kohlhase, Michael, Susanna Kuschert, and Manfred Pinkal
 1996 A type-theoretic semantics for λ -DRT. In P. Dekker and M. Stokhof, (eds.), *Proceedings of the 10th Amsterdam Colloquium*, pp. 479–498. ILLC, Amsterdam.
- Larsson, Staffan
 2002 *Issue-based Dialogue Management*. Ph.D. thesis, University of Gothenburg.
- Mönnich, Uwe
 1985 Untersuchungen zu einer konstruktiven Semantik für ein Fragment des Englischen. Habilitationsschrift, Universität Tübingen.

- Montague, Richard
1974 *Formal Philosophy: Selected Papers of Richard Montague*. Yale University Press, New Haven. Ed. and with an introduction by Richmond H. Thomason.
- Muskens, Reinhard
1996 Combining Montague semantics and discourse representation. *Linguistics and Philosophy*, 19(2): 143–186.
- Ranta, Aarne
1994 *Type-Theoretical Grammar*. Clarendon Press, Oxford.
- Sag, Ivan A., Thomas Wasow, and Emily M. Bender
2003 *Syntactic Theory: A Formal Introduction*. CSLI Publications, Stanford, 2nd edition.
- Shieber, Stuart
1986 *An Introduction to Unification-Based Approaches to Grammar*. CSLI Publications, Stanford.
- Sundholm, Göran
1986 Proof theory and meaning. In Dov Gabbay and Franz Guentner, (eds.), *Handbook of Philosophical Logic, Vol. III*. Reidel, Dordrecht.
- Tasistro, Alvaro
1997 *Substitution, record types and subtyping in type theory, with applications to the theory of programming*. Ph.D. thesis, Department of Computing Science, University of Gothenburg and Chalmers University of Technology.
- van Eijck, Jan and Hans Kamp
1997 Representing discourse in context. In Johan van Benthem and Alice ter Meulen, (eds.), *Handbook of Logic and Language*. North Holland and MIT Press.