Two Levels of Semantic Representation in DenK

René Ahn, Leen Kievit, Gerrit Rentier & Margriet Verlinden
Dialogue-management & Knowledge Acquisition[1]
Institute for Language Technology and Artificial Intelligence (ITK)
Tilburg University, PO Box 90153, 5000 LE Tilburg, The Netherlands
e-mail: {ahn,kievit,rentier,verlinde}@kub.nl

Abstract

In the DenK dialogue-system, we distinguish two levels of semantic representation[2]. We employ *typed feature structures* (Carpenter 1992) at the first representation level, deriving context-independent semantic representations using insights from Head-driven Phrase Structure Grammar (HPSG, Pollard and Sag 1994). At the second representation level, we use a form of *Constructive Type Theory* (Coquand and Huet 1988) to represent both the contextually determined meanings of utterances as well as the information state of the system, the latter containing the context with respect to which an utterance must be interpreted. We discuss both levels of representation, specifically advantages of the levels w.r.t. *underspecification* of semantic ambiguities related to pronouns and to quantifiers.

1 Introduction

The dialogue-system constructed in DenK has a user-interface consisting of a graphical part and a lingual part, cf. Ahn et al. (1994). In the lingual part utterances typed in by the user are changed in several steps from plain English to reactions of the system.

In this paper we discuss two levels of semantic representation that can be discerned in the process which maps English written utterances to updates of

[1]Project sponsored by the Universities of Brabant Joint Research Organization (**SOBU**); participants being the department of Mathematics and Computer Science of the Technical University Eindhoven in collaboration with the Institute for Perception Research (**IPO**) in Eindhoven and the Institute for Language Technology and Artificial Intelligence (**ITK**) in Tilburg.

[2]This notion of 'levels' originates with (Bronnenberg et al. 1979).

the discourse representation. Briefly, in the first step we do the full syntactic analysis, and part of the semantic interpretation without taking into account any contextual knowledge. Here 'context' means both world knowledge and the knowledge of the context of the dialogue, both discourse context and physical context.[3] At this level, all context-dependent interpretation is left *underspecified*. Hence, the level is called Underspecified Logical Form (**ULF**, Kievit 1994).

Of course, such underspecification is not only common-place in natural language processing,[4] also it is of limited use if one does not define how the underspecified representations can be (efficiently) mapped into representations which are suitable for *reasoning*, cf. Allen (1993), Reyle (1992). In DenK discourse representation, world and domain knowledge representation as well as reasoning are expressed in Constructive Type Theory (**CTT**, Coquand and Huet 1988; Martin-Löf 1984).

In this paper, we show how the combination of our architecture and assumptions together with some especially useful properties of ULF and CTT allow for interesting ways around the problem of excessive ambiguity. Specifically, we will discuss how we represent the 16 relevant different interpretations of sentence (1) in 1 ULF, arguably corresponding to 1 CTT segment:

(1) Could *you* tell *me* whether *every* highlighted button controls *some* lens?

In this sentence, *you* and *me* are ambiguous as to whether they concern a male or female referent; also, *you* remains ambiguous between plural and singular.[5] Moreover, the scope of the quantifiers *every* and *some* is ambiguous, as we will discuss below. In a straightforward unification-based grammar[6] of English, using disjunctions over atomic values and using Cooper storage (cf. Cooper 1983) for quantification, this would lead to 16 derived interpretations. This would, for instance, be the case for the grammar of English which is presented in Pollard and Sag (1994).

Obviously, this is not a situation which suggests efficient and effective interpretation. More precisely, interpretation through this *enumerate-and-filter*

[3]In DenK the physical context consists of a graphical representation of a Philips CM-10 Electron Microscope. Both user and system (cooperative assistant) can perceive this domain as well as manipulate objects in this domain.

[4]Perhaps the most well-known form of underspecification is Quasi Logical Form, as defined in the Core Language Engine (**CLE**) Project (Alshawi 1992) (but also see Nerbonne 1992; Reyle 1995). We will have more to say on the comparison between the CLE system and ours in Sect. 2.3.

[5]One might consider omitting agreement features from the grammar to avoid the pronoun-related ambiguities discussed w.r.t. (1). However, arguably, this would induce serious problems since such agreement information can be quite crucial in anaphora resolution; e.g. *himself* vs. *herself* vs. *themselves*.

[6]Unification-based grammar refers to a class of grammar formalisms (or 'constraint-based' formalisms) which are based on attribute-value logics, in which *unification* is a particularly useful operation: see Shieber (1986) for an excellent introduction, or Chap. 1 of Pollard and Sag (1994).

approach (Allen 1993), is not *monotonic* (Alshawi and Crouch 1992). That is, instead of having an interpretation of an utterance gradually become more specified during interpretation, we have a (huge) number of fully specific interpretations of which all but one (!) must be discarded.

In Sect. 2 and Sect. 3 we will discuss our alternative for having a number n of disjunctive feature structures as interpretation for an utterance which is n-ways ambiguous. The specific properties of the representation levels ULF and CTT will be shown to play a crucial role in this respect. In Figure 1 a schema of several levels of the architecture for interpretation in DenK is given. As can be seen, grammatical interpretation is done in a first module, in a stage of structural analysis. This results in expressions in ULF, which can, for the moment, be thought of as boiling down to predicate/argument structure. Extra-grammatical interpretation, which is the part of interpretation that involves processing on the basis of context knowledge, is performed in a second module. The expressions in CTT that result from this module are passed on to the other modules where, by pragmatic processing, the actual behaviour of the system is computed.

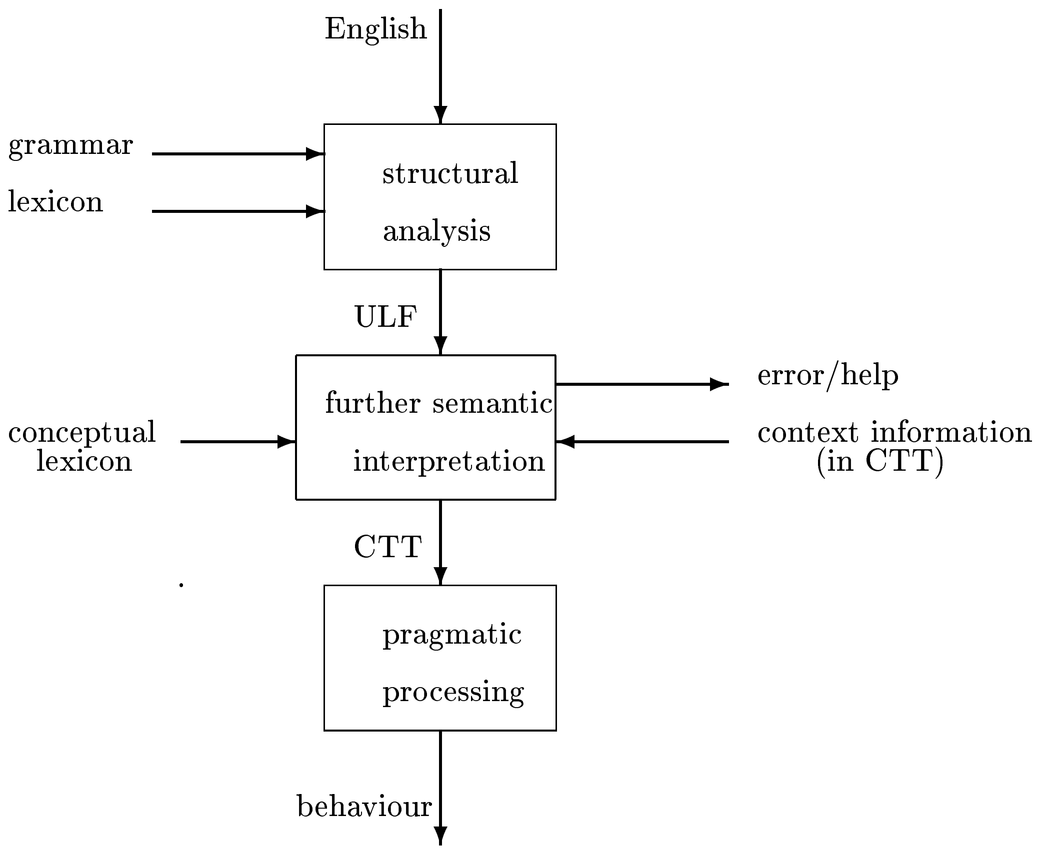


Figure 1: Architecture for interpretation in DenK.

2 Underspecification and Typed Feature Structures

In DENK, we have chosen to use a constraint-based grammar which is in important ways similar to Head-Driven Phrase Structure Grammar (HPSG; Pollard and Sag 1994). The theoretical choice for HPSG practically implies the use of a parser that handles typed feature structures. In fact, we use the Attribute Logic Engine (ALE, Carpenter and Penn 1994) for parsing.[7] The organization of the types[8] of a typed feature formalism in a type hierarchy allows for a way around disjunction over atomic values.

2.1 Trading Disjunctions for Underspecification

Constraint-based linguistic formalisms take as their informational domain a system of objects based on *features* (or 'attributes') and *values*. A set of feature-value pairs is called a *feature structure*. A value can be either atomic or complex; a complex value is again a feature structure.[9] In *typed* feature structures then, each structure has a type (as a left-subscript in the attribute-value matrix) which generally determines the appropriate features in that structure. For instance, some type *index* for the representation of agreement features might 'type' some feature structure which looks as follows:

(2) $${}_{index}\begin{bmatrix} \text{GENDER} & \text{male} \\ \text{NUMBER} & \text{singular} \\ \text{PERSON} & \text{third} \end{bmatrix}$$

Moreover, the atomic values are typed as well, in fact, values *are* types. So in structure (2) MALE, SINGULAR and THIRD are types. The nice thing about these types is that all possible values of a certain feature, e.g. GENDER, are hierarchically ordered. In tree (3) the type hierarchy for appropriate values of GENDER is given. As can be seen the most general (appropriate) type for GENDER is *gend*. This means, if nothing more is known of an object than that it does have some gender, than GENDER is given the value *gend*. The logic underlying this formalism makes such a hierarchy a *bounded complete partial order*, which has pleasant computational properties.[10]

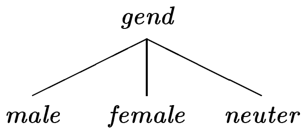
[7]Cf. Rentier (ms.) for a detailed description of a proto-type fragment; also, Rentier (1995).

[8]**N.B.**: Please note that types in typed feature logic should not be associated with types in Type Theory. In HPSG a type is a symbol which gives a name to each ontological category in HPSG. HPSG types have no combinatory potential. A type in Type Theory on the other hand is a mathematical object which may have internal structure, and which defines the combinatory potential of objects.

[9]For more about feature structures the reader is once more referred to Shieber (1986).

[10]Unification can be performed in quasi-linear time in the size of the input, cf. Carpenter (1992).

(3)



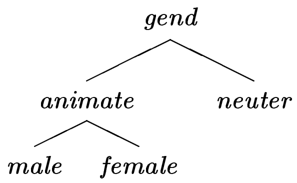
Now, when features of an object can have a number of values, like for the word *you* in example (1), an *untyped* feature structure with agreement features would contain a disjunctive value for each ambiguity, see structure (4).

(4)
$$\begin{bmatrix} \text{GENDER} & \text{male OR female} \\ \text{NUMBER} & \text{singular OR plural} \\ \text{PERSON} & \text{second} \end{bmatrix}$$

Two disjunctions like this in a structure are generally at least as computationally 'expensive' as 4 separate disjunctive structures.

Thanks to the typed features the value of NUMBER can be replaced by the most general type, usually having (nearly) the same name as the feature: *numb*. The feature GENDER here, can have only two of the three values. For cases like this, where a number of the values are allowed but not all, a somewhat more complex type hierarchy gives the opportunity to underspecify the ambiguous feature. For GENDER this means that we create a more general type for *male* and *female*, called *animate*. The resulting type hierarchy is given in tree (5), and the corresponding new feature structure for *you* in (6). This way the subject and direct object of sentence (1) can both be represented by one feature structure and as a consequence by one ULF expression. The (simple) relation between feature structures and ULF is explained in Sect. 2.3.

(5)



(6)
$$_{index}\begin{bmatrix} \text{GENDER} & \text{animate} \\ \text{NUMBER} & \text{numb} \\ \text{PERSON} & \text{second} \end{bmatrix}$$

2.2 Constraint-Based Context-Independent Event Semantics

In the way we described above we reduced the number of analyses for the example (1) from 16 to 2. But the ambiguity caused by the different possible interactions of the scopes of the two quantifiers in the sentence remains if we would still represent the semantics of the sentence as in (Pollard and Sag

1994). However, in contrast to the approach favoured and laid out by Pollard and Sag, we do not use a combination of situation semantics and quantifier storage/retrieval, but have opted for a constraint-based *context-independent event semantics* at the level of typed feature structures.

First of all, this means that we do not view reality as existing of parameterized states of affairs (Barwise and Perry 1983), but as events in the sense of Davidson (1967). In fact, in this choice our DENK grammar closely resembles Unification Categorial Grammar (UCG; Zeevat et al. 1987). Since we do not incorporate any context information in the feature structures (as explained, we choose to deal with context after parsing the sentence), we also abandon the feature CONTEXT which ranges over presuppositions and the like, cf. Chapters 1 and 8 of Pollard and Sag (1994)[11]. We will illustrate our approach with our treatment of a quantifier scope ambiguity which is analogous to example (1).

Quantifier scope ambiguity is one of the more stubborn problems in dialogue systems, cf. Calder et al. (1987), as it is in most applications of natural language processing. Consider (7a)—which contains the same quantifier scope ambiguity as (1)—and its two predicate logic interpretations:

(7) a. Every button controls a lense
b. $\forall y.BUTTON(y) \rightarrow \exists z.LENSE(z) \wedge CONTROL(y, z)$
c. $\exists z.LENSE(z) \wedge \forall y.BUTTON(y) \rightarrow CONTROL(y, z)$

The more general reading of (7a) in (7b) concerns the case that for every button, there is a lense which is controlled by this button. The more restricted reading in (7c) concerns a unique lense which is controlled by every button. Since Montague (1974), in formal semantics it is generally the issue how to derive both readings of an ambiguous sentence by separate derivations. But arguably not for all kinds of ambiguity should all readings correspond to additional derivations c.q. structures. As pointed out by Kempson and Cormack, quantifier scope ambiguity is an entirely different kind of ambiguity than structural ambiguity, as in (8) for instance (cf. Kempson and Cormack 1981):

(8) a. They saw her duck
b. Young men and women cause trouble

Kempson and Cormack argue convincingly that for structural ambiguities as in (8), but *not* for quantificational ambiguity as in (7) more than one representation should be derived. This assumption is reflected in our approach; we do not follow the Montagovian tradition of deriving a disjunction of all possible interpretations for a quantificationally ambiguous sentence. Consequently we do not make use of quantifier storage or retrieval in syntax, so-called "Cooper-storage" (Cooper 1983), though this is very well possible in HPSG, cf. Chapter 8 of Pollard and Sag 1994.

[11]See section 3.5 where we treat presupposition as accommodation at discourse level.

In such an approach one derives n disjunctive interpretations for n possible quantifier scopings. Instead, in DENK we would derive[12] only one interpretation, e.g. the typed feature structure in Figure 2 as a context-independent semantic interpretation for (7a).

Feature structures of type *ulf* introduce three features: QUANT, VAR and RESTR. The value of QUANT is a quantifier which we take to represent quantification over an event or object. Such events and objects correspond to a variable, the value of VAR, which intuitively also corresponds to a discourse referent in the sense of Kamp (1984).

Also, with common nouns as with events, RESTR takes as its value feature structures which have features RELN and INST, indicating the nature of the relation and an indication of the instance of this relation.[13] If the relation involves thematic dependents, then these are represented on a list called ARGS, for 'arguments'. Since ARGS takes values of type *ulf*, these typed feature structures are recursively defined.

2.3 The Role and Status of Underspecified Logical Form

In our view, then, the parser is merely responsible for deriving a context-neutral interpretation of an utterance, where quantifier scope ambiguities are underspecified.[14] The interface expressions between grammatical interpretation and extra-grammatical interpretation are referred to as Underspecified Logical Form. These expressions are linearizations of values of feature structures of type *ulf*.[15] E.g., the ULF for (7a) would be a linearization of the typed feature structure values of Figure 2, to be represented as follows:

(9)
```
ulf(event,X,[tense:pres],[control(X,[
   ulf(every,Y,[pers:3d,num:sg,gend:neuter],[button(Y,[])]),
   ulf(a,Z,[pers:3d,num:sg,gend:neuter],[lens(Z,[])])
                                          ])])
```

This will be input for a translation process, which translates this to the most general reading of the sentence in a Type Theoretic representation (cf. Sect. 3.5).

The use of ULF is merely as an interface between the level of *structural analysis* and subsequent levels of *knowledge representation and reasoning*. At the moment we are developing extended versions of ULF as defined in Kievit

[12] Our grammar is syntactically inspired mostly by Chap. 9 of Pollard and Sag (1994), cf. Rentier (1995).

[13] Through structure-sharing, INST takes a variable that is token-identical to the value of VAR.

[14] It is also our intent to devise ways in which to underspecify structural ambiguities (typically, ambiguous PP-attachment, cf. Kievit () for relevant discussion) at ULF. This is, however, work in progress; also, the context-independent interpretation of operators remains future work.

[15] This view arose in the PLUS project, cf. Geurts and Rentier (1993) for further discussion.

$$
{}_{\mathit{ulf}}\begin{bmatrix}
\text{QUANT} & \text{event} \\
\text{VAR} & \boxed{1}\begin{bmatrix}\text{TNS} & \text{pres}\end{bmatrix} \\
\text{RESTR} & \left\{ {}_{\mathit{restr}}\begin{bmatrix}
\text{RELN} & \text{controls} \\
\text{INST} & \boxed{1} \\
\text{ARGS} & \left\langle
{}_{\mathit{ulf}}\begin{bmatrix}
\text{QUANT} & \text{every} \\
\text{VAR} & \boxed{2}\begin{bmatrix}\text{GENDER} & \text{neuter} \\ \text{NUMBER} & \text{sing} \\ \text{PERSON} & \text{third}\end{bmatrix} \\
\text{RESTR} & \left\{ {}_{\mathit{restr}}\begin{bmatrix}\text{RELN} & \text{button} \\ \text{INST} & \boxed{2} \\ \text{ARGS} & \langle\ \rangle\end{bmatrix} \right\}
\end{bmatrix},
{}_{\mathit{ulf}}\begin{bmatrix}
\text{QUANT} & \text{a} \\
\text{VAR} & \boxed{3}\begin{bmatrix}\text{GENDER} & \text{neuter} \\ \text{NUMBER} & \text{sing} \\ \text{PERSON} & \text{third}\end{bmatrix} \\
\text{RESTR} & \left\{ {}_{\mathit{restr}}\begin{bmatrix}\text{RELN} & \text{lens} \\ \text{INST} & \boxed{3} \\ \text{ARGS} & \langle\ \rangle\end{bmatrix} \right\}
\end{bmatrix}
\right\rangle
\end{bmatrix} \right\}
\end{bmatrix}
$$

Figure 2: Feature structure for *Every button controls a lens.*

(1994), and a mapping to DENK's information state representation level, CTT, which is discussed in Sect. 3.

Our work for DENK is in important ways based on the Core Language Engine (CLE, Alshawi 1992) project, though we try to develop ideas further and make the approach more efficient. Like the CLE, we use ambiguous representations to represent the results of structural analysis; and also like the CLE project, we use an event based semantics at both levels. But unlike the CLE project, we have chosen a lexicalist and principle based theory of syntax and semantics, namely HPSG. The advantages of this choice are a.o. that we can freely benefit from insights that have been and will be developed in this theory. Additionally the types in typed feature logic allow for underspecification of disjunctions as discussed in the previous subsection, which is not possible using untyped feature logics.[16]

Also, in the CLE, QLF's are translated to Logical Form; however, the CLE's LF is a form of predicate logic, extended to deal with sortal constraints and events, cf. Alshawi (1992). In contrast, in DENK we use segments of Type Theory, as we will discuss in Sect. 3.2, which has several advantages. For one, the fact that the CLE represents interpretations in predicate logic means that their LF representation inherits most if not all problems which are inherent to predicate logic. E.g., predicate logic by itself is not a dynamic semantic theory in the sense that is common since Kamp (1984). This means, for instance, that coreference across sentence boundaries is not provided for, and that there are problems concerning the binding of existentially bound variables in so-called *donkey*-sentences:

(10) a. A man_1 walks in the park. He_1 whistles.
 b. If a farmer owns a donkey_1, (then) he beats it_1

Such problems have been successfully addressed by Discourse Representation Theory (DRT, Kamp 1984; Kamp and Reyle 1993) and its successors. Since CTT can be seen as a generalization of DRT, cf. Ahn and Kolb (1990), our approach does capture the advantages of a dynamic semantics, counter to the CLE's.

Finally, due to interesting properties of CTT we do not need to expand ambiguous ULFs like (9) into many different representations at the level of context-dependent interpretation. In an approach as the CLE's, one will still derive several readings for ambiguous expressions. Though there will be, e.g., but 1 QLF for (7a), there will still be interpretations similar to (7b) and (7c) for (7a) at LF. So n possible interpretations of quantifier scope ambiguity still correspond to n disjunctive interpretations, albeit at a different level. In contrast, CTT allows for truly monotonic interpretation in the sense of Alshawi and Crouch (1992). As we will discuss in Sect. 3.5, a sentence like (7a) will, in our approach, correspond to but 1 ULF and also to but 1 CTT segment.

[16] However, there we may simply use some abbreviation for a disjunction: clearly, this works as well, but without the theoretical and the efficiency gains.

Discourse and other knowledge may urge the system to make this reading more specific; this is also discussed.

In sum, though there are some obvious similarities between the CLE approach and the approach to natural language interpretation in DenK, it must be clear that we may discern many interesting differences. We turn to the level of full semantic interpretation.

3 Constructive Type Theory and Underspecified Quantifier Scope

The idea behind the kind of semantics that we want to have in the DenK system is closely related to the general aim of the project. That is, in DenK we are trying to develop a *cooperative assistant* to help a user accomplish some tasks in a specific domain. The user wants to perform some task, and the system assists the user. To make this cooperation possible, the user and the system must communicate about the task. This means a *cooperative dialogue* takes place, in which both dialogue partners can ask for information, and supply information, thereby enlarging the knowledge that the partners have in common. By doing this they will work together on achieving the user's goal.

3.1 Cooperative Dialogues and Knowledge Representation

To model such a dialogue, we need to formalize what goes on when a dialogue takes place. Since we are interested in the information exchange aspect of dialogues, we will give a precise description of the process of information exchange. Exchange of information presupposes the presence of information: both agents in a dialogue have some information. The information that one agent has at a certain point in time will be called an *information state*. This information state contains different kinds of information, for instance information about the world, about the task that needs to be performed, and, for a truly cooperative dialogue, about the knowledge of the partner. The latter kind of information enables the agent to respond adequately, since he knows what the partner knows and doesn't know (cf. Beun (1993) for discussion).

The information that an agent has can originate with different sources. Either the agent knows something because he saw it in the outside world, or he knows it because it was communicated to him and he chose to believe it, or he deduces it on the basis of information he already possesses.

The primary operation that an agent has for internalizing information about the outside world is *categorization*. We assume agents have some internal technique that allows them to classify what they perceive. To put it in another way: if an agent perceives something, he will immediately recognize what kind of thing he is seeing. This means the primary relation in an information state must be a classification relation.

However, 'new' information can only be understood by an agent, if he already has some knowledge about the things that the new information is about. In some way, new information must have some connection to the already present knowledge. For it is possible that an agent perceives an object that he cannot classify: this would mean that such an object was inconceivable to the agent, given his information state. However, the fact that an object is unconceivable to an agent at one time, does not preclude the possibility that he might later learn to recognize the object. Information states are inherently dynamic, and objects that cannot be categorized at a certain time, may be categorizable at some other time.

As we saw, new information can also become available through communication with other agents, or from reasoning about the information the agent already had. In the latter case, we cannot say that the agent has new information, but rather that he is exploring the consequences of the information that he has. So what is happening is that information that was *implicit* in an agent's information state, becomes *explicit.* The agent may even realize that his information is inconsistent, which he did not know before.

If new information is understandable for the agent, he can choose to add it to his knowledge. This would mean he believes the information, and *extends* his information state. It could turn out that the agent was wrong in accepting the information, for instance if he accepted it from another agent, or his observation of the external world was for some reason wrong. As we have seen, this inconsistency may not be recognized immediately by the agent.

Now we have seen some features that we think are important for formalizing information states. The knowledge representation formalism that we use to formalize these states is Constructive Type Theory (CTT) (Coquand and Huet 1988; Martin-Löf 1984). This is a mathematical formalism that has been used to formalize theories in such a way that proofs in such theories can be checked automatically.

The application of CTT as a formalism for natural language semantics is relatively new. A related Type Theory, Martin-Löfs Theory of Types (MLTT, Martin-Löf 1984), was first recognized by Göran Sundholm as a formalism that could effectively solve some of the problems with *donkey*-sentences (cf. Kamp 1984) in natural language (Sundholm 1986). Aarne Ranta (Ranta 1991) introduced a set of rules for generating English sentences from MLTT expressions. A way of introducing *generalized quantifiers* in MLTT was investigated by Sundholm in Sundholm (1989). The dynamic aspect of these type theories was made explicit in Ahn and Kolb (1990), where a translation of DRT into CTT was given. An extension to CTT which is important for the DenK project, is given in Borghuis (1994). Here, a modal extension of CTT is presented.

3.2 CTT Representations for Natural Language Expressions

The form of Type Theory we use in DENK, is this modal extension of CTT (cf. Borghuis 1994), which allows us to model the knowledge of the system about the knowledge of the user. As we saw this is necessary since we want the system to react in a cooperative fashion. Another addition is the inclusion of Σ types, which were introduced in Martin-Löf (1984).

The building blocks of Type Theory are terms. The basic relation in CTT is the typing relation. This is a relation between two terms. If the typing relation holds between a term A and a term T, we write

$$A : T$$

We call A an *inhabitant* of T, and we call T the *type* of A. The expression $A : T$ itself is called a *statement*.

This typing relation is our formalization of the result of the categorization operation that we assumed agents to possess. The types correspond to the result of the categorization procedure. We noted that this categorization can only be successful if an agent can conceive of the kind of thing he is trying to categorize. In CTT this property falls out automatically. Information states are formalized as sequences of statements, built up according to a certain set of rules. Such a finite sequence of statements is usually called a *context*. For every statement $A : T$ in this context, it must be the case that the type T has some connection with the part of the context that precedes the statement. In fact, the type must either occur in the preceding context in a statement of the form $T : K$, i.e. it must occur as a term itself, with some type, or the type T must be *derivable* as a term with a type K from the material in the context, using the rules of Type Theory (Barendregt 1992).

Obviously, it cannot be the case that every type must have a derivable type itself, since the preceding context is assumed to be finite. Some types are always considered well-formed. They are introduced by means of axioms. These special types are usually called *sorts*. We will use two sorts here, $*_t$ and $*_p$, which may be read as 'the sort of types' and 'the sort of propositions' respectively.

It is now time for a small example. Imagine an agent who is able to recognize objects in the outside world that are dogs. What this means in CTT, is that there is a term of sort $*_t$ that corresponds to the category of objects 'dog'. Call this term 'dog', then we can write down the information state of the agent as a CTT context:

(11) $[\ \text{dog}{:}*_t\ ]$

Such inhabitants of sort $*_t$ are usually called *types*, which, admittedly, may be confusing. The presence of such a type in a context represents the capacity of the agent to recognize objects in the outside world of that type.

Now, if this agent sees an actual object, and classifies it as being of this type 'dog', he can add this knowledge by creating a representation of the individual object (a referent of that dog). Let us assume that the dog is called Max, then we can use 'max' as the referent for this individual dog. The agent sees Max, recognizes him as a dog, and a statement that signals this recognition is added to the information state, which now becomes:

(12) $[\ \text{dog}{:}*_t,\quad \text{max}{:}\text{dog}\ ]$

This context is well-formed, since 'dog' was introduced as a type (a term of sort $*_t$) before an inhabitant of the type was introduced.[17]

3.3 Propositions and Predicates as Types

Now, the types in CTT that we use here to classify objects, serve a number of goals. The type of an inhabitant tells us with which kinds of other objects we can combine the inhabitant. It also tells us, what the type will be of the resulting object of such a combination.

Compare this for instance to the lambda calculus that is used in Montague Semantics (Montague 1974; Gamut 1991). There we have for instance interpretations of adjectives like *old*, usually written as **old'**, with a type (e, t), i.e. a predicate on entities, which is a function, that, when applied to something of type e, returns a truth-value. These kinds of objects can also be found in CTT. We can interpret *old* as a predicate, but here this would not be a function from entities to truth-values, but rather a function from some type to propositions. However, before we can show this, we need to take a look at what constitutes a proposition in CTT.

The inhabitants of the sort $*_t$ are the types of objects that we want to talk about in the information state. We also need another sort, denoted by $*_p$, the inhabitants of which will be representations of propositions. As one of the easiest examples of this we consider the sentence *It is raining*. [18] The sort $*_p$ itself does not have to be introduced in the context. But as was the case for the type 'dog', we must introduce the concept of raining before we can add information about whether it is raining into our agent's information state. This is done in CTT by introducing a constant that represents the proposition *It is raining* as an inhabitant of sort $*_p$:

(13) $[\ \text{rain}{:}*_p\ ]$

[17] Note that the classification procedure must be a function. We cannot have two different types for a single inhabitant. Suppose the agent has some knowledge about dogs, he may also know that Max is a retriever. But we cannot formalize this in CTT by simply adding a type 'retriever:$*_t$' and the statement 'max : retriever'. This means that one must be very careful when formalizing information states, what types to choose and how one wants to classify objects. We will not go into this here but will address this problem in future work.

[18] What is easy about this proposition is that it does not involve a predicate that needs arguments: 'raining' does not need an agent, or indeed any other thematic dependent.

Note that this does not represent that the agent believes that it is raining. It merely reflects the fact that the agent can recognize instances of situations that involve raining. If we want to add *It is raining* to an agent's information state as a fact, we need to find an inhabitant of 'rain'. This means that we treat 'rain' as a type, and introduce some new constant, which is given that type:

(14) $[\ \mathrm{rain}{:}*_p,\ \ \mathrm{p}{:}\mathrm{rain}\ ]$

So, 'p' is added as a constant of type 'rain'. But how should we interpret this 'p'? What does it represent?

It is well-known in mathematical logic, that there is a precise connection between Natural Deduction type proof systems for logic, and certain type-theories. This connection is usually called the *Curry-Howard-DeBruijn isomorphism.* Because of this connection between type-theory and various logics, we can interpret certain types as propositions, and their inhabitants as representations of *proofs* of those propositions. This is known as the *Propositions-as-Types* interpretation of Type Theory.

So, in our example in (14), 'p' can be seen as a proof of the fact that it rains. Now it will become clear that if an agent's information state contains a proof of a proposition, this is interpreted as saying that the agent believes the proposition. This is why 'p:rain' has the desired interpretation that the agent believes that it is raining.

Predicates can be interpreted as functions from some object to a proposition. To build types of functions, we have a special constructor in CTT, the Π constructor. Suppose we want to interpret the expression 'old' in Type Theory. This predicate should be applicable to dogs. This means 'old' must be a function from the type 'dog' to a proposition (the proposition has type $*_p$, since that is the sort of propositions). Using the Π constructor, this predicate 'old' can be introduced into the information state as follows:

(15) $[\ \mathrm{dog}{:}*_t,\ \ \mathrm{max}{:}\mathrm{dog},\ \ \mathrm{old}{:}(\Pi\, x{:}\mathrm{dog}.*_p)\ ]$

Note that, here, for 'old:($\Pi\, x$:dog.$*_p$)', we could also have written 'old:dog$\rightarrow *_p$'. But in general this is not so, since the types that are build with Π allow the result type to be dependent on the inhabitant that the function is applied to. In the case of 'old', the result of applying the function to some argument, will always have the type $*_p$.

Of course, since 'old' is now a function from dogs to propositions, this function, when applied to 'max' should yield a proposition. And indeed the following is derivable in CTT:

(16) $[\ \mathrm{dog}{:}*_t,\ \ \mathrm{max}{:}\mathrm{dog},\ \ \mathrm{old}{:}(\Pi\, x{:}\mathrm{dog}.*_p)\ ]\ \vdash\ \mathrm{old}(\mathrm{max}){:}*_p$

Again, the fact that this is derivable in an agent's information state, does not mean that the agent believes the propositions. It only means that he acknowledges that this is something which might be true or false. If in fact, Max is an old dog, and the agent knows this, this would again be indicated by the presence of an inhabitant of 'old(max)'.

3.4 Universal and Existential Quantification in CTT

The meanings of sentences, which typically involve predicates as well, would be represented in a similar way. The meaning of the verb 'run' for instance, can also be defined as a function from dogs to propositions. Let us introduce another dog in the agent's information state, and also the concept of 'running':[19]

(17) [dog:$*_t$, max:dog, old:($\Pi\, x$:dog.$*_p$), lassie:dog, run:($\Pi\, x$:dog.$*_p$)]

Now we may turn to the interpretation of quantifiers.

We begin by looking at universal quantification. A sentence like *Every dog runs* should have an interpretation such that we can derive for every individual dog, that this particular dog runs. This implies that there must be a function which returns for every dog x, an inhabitant of 'run(x)'. Such a function is again an inhabitant of a Π type. So we will use a *dependent*[20] Π type to represent the meaning of *Every dog runs*. The interpretation of *Every dog runs* will then read as follows:

(18) $\Pi\, x$:dog.run(x)

Since this Π type itself represents a proposition, it must be of the sort $*_p$. We do not need to add this explicitly in the information state, since it follows from some axioms in CTT. That is, the following may be derived in CTT:

(19) [dog:$*_t$, max:dog, old:($\Pi\, x$:dog.$*_p$), lassie:dog, run:($\Pi\, x$:dog.$*_p$)] $\vdash$ ($\Pi\, x$:dog.run(x)):$*_p$

Again, to assert the truth of this proposition in the agent's information state, an inhabitant of this type must be added:

(20) [dog:$*_t$, max:dog, old:($\Pi\, x$:dog.$*_p$), lassie:dog, run:($\Pi\, x$:dog.$*_p$), q:($\Pi\, x$:dog.run(x))]

This proof object 'q' is a function, which for every argument x (of type 'dog'), returns an object of the type 'run(x)'. This means that the interpretation of *Every dog runs* makes the following derivable:

(21) [dog:$*_t$, max:dog, old:($\Pi\, x$:dog.$*_p$), lassie:dog, run:($\Pi\, x$:dog.$*_p$) q:($\Pi\, x$:dog.run(x))] $\vdash$ q(max):run(max)

[19] For expository reasons, the examples concern a perhaps somewhat peculiar information state, where running is categorized as something which is only typical for dogs. More 'realistic' information states are left up to the imagination of the reader, as is so often the case with expositions of formal semantics.

[20] A dependent Π type is a Π type where the variable occurs in the 'result' type.

So, by combining the proof 'q' for *Every dog runs* with the individual dog 'max', we get a proof object of type 'run(max)', i.e. the agent can derive a proof of the proposition *Max runs*. The same is of course true for *Lassie runs*.

Existential quantification is expressed by using the Σ type operator. On the surface Σ types resemble Π types. However, the inhabitants of Σ types are not functions, but *pairs* of objects. As an example, consider the interpretation of *A dog runs*, which gets interpreted using a Σ type:

(22) $\Sigma\, x{:}\mathrm{dog}.\mathrm{run}(x)$

We will call 'x:dog' the *abstraction* part of the Σ type, and 'run(x)' will be called the *body*. This type itself has the sort $*_p$, since it represents a proposition.

From the fact that in our context we had proof of *Every dog runs*, we should be able to derive a proof object of *Some dog runs*. As we have said, such inhabitants of Σ types are pairs. The first element must be an individual of the type stated in the abstraction part of the Σ type; in this example we need some individual dog. The second element of the pair must be a proof that the property indicated in the body part of the Σ type, which in this case is 'run', holds of the first element of the Σ type. Suppose we take for our first element the individual dog 'lassie'. Now, we need for our second element an inhabitant of 'run(lassie)', i.e. a proof object for *Lassie runs*. But we have seen above how we can use the proof of *Every dog runs* to get proofs for *x runs* for every dog x, namely by applying the proof object 'q' of *Every dog runs* to an individual dog. So 'q(lassie)' is a proof of *Lassie runs* and this can be used as the second element in our pair. Indeed, in CTT the following is derivable:

(23) $[\ \mathrm{dog}{:}*_t,\ \mathrm{max}{:}\mathrm{dog},\ \mathrm{old}{:}(\Pi\, x{:}\mathrm{dog}.*_p),\ \mathrm{lassie}{:}\mathrm{dog},\ \mathrm{run}{:}(\Pi\, x{:}\mathrm{dog}.*_p),$
$\mathrm{q}{:}(\Pi\, x{:}\mathrm{dog}.\mathrm{run}(x))\ ]\ \vdash\ \langle\ \mathrm{lassie},\ \mathrm{q}(\mathrm{lassie})\ \rangle{:}(\Sigma\, x{:}\mathrm{dog}.\mathrm{run}(x))$

Note that there are also projection functions available in CTT, which allow us to recover the individual elements of a pair that is an inhabitant of some Σ type. These elements may be relevant when we need representatives of a class of objects with some property. They also play a role in our account of quantifier scopes in CTT, as is shown in the next section.

3.5 Underspecification of Quantifier Scopes in CTT

So now we have seen how the meaning of some simple sentences can be represented in CTT. In the DENK system, these representations are created by mapping the ULF expressions that were the result of the structural analysis of a sentence, onto CTT expressions.[21] In the case of quantifier scope ambiguities like in (24), repeated here from (7), we map it to a representation of the most general reading of the sentence (equivalent to (24b)):

[21]The ULF expressions are related to CTT terms by means of a constraint-based mapping. The constraints originate from several sources:

- The conceptual lexicon gives the possible CTT terms that a ULF is mapped onto,

(24) a. Every button controls a lense

b. $\forall y.BUTTON(y) \rightarrow \exists z.LENSE(z) \wedge CONTROL(y, z)$

c. $\exists z.LENSE(z) \wedge \forall y.BUTTON(y) \rightarrow CONTROL(y, z)$

Context and reasoning will then optionally allow us to decide whether the interpretation should be 'narrowed down' to the more restricted reading (equivalent to (24c)). It is however equally possible to reason on the basis of the former representation, without having to resolve the ambiguity.

In natural language processing systems (NLP systems), it is ultimately reasoning with respect to context which is able to resolve quantifier scope ambiguity. If, indeed, quantifier scope resolution is possible at all at the current point in discourse. As argued by Allen (1993) and Reyle (1992), it is imperative that an NLP system is able to have underspecified representations of expressions which are ambiguous w.r.t. quantifier scope. Consider the tiny discourse in (25):

(25) Every toddler climbed a tree. They were laughing and screaming. The tree collapsed, and all toddlers fell out.

The first sentence is ambiguous in the same way as (24). The resolution of the ambiguity in the first expression is resolved by the third expression; by the singular definite use of *tree* in the third expression it becomes clear that in the first expression the existentially quantified tree should have wide scope. This kind of reasoning for quantifier scope resolution obviously takes place at the extra-grammatical level. Furthermore, Allen (1993) argues that sometimes quantifier scopes in discourse do not get resolved at all, since the resolution might be irrelevant and Gricean maximes apply.

The 'strengthening on demand' strategy can be implemented in CTT in quite a straightforward way, since the functional relations between universal and existential quantifiers can be made explicit[22], and additional restrictions on these functions can be imposed whenever one pleases. As an example, let us look at the small discourse in (25). We need a context to start with:

(26) [tree:$*_t$, toddler:$*_t$, mike:toddler, bob:toddler, jim:toddler, climb:(Π x:toddler.(Π x:tree.$*_p$)), collapse:(Π x:tree.$*_p$)]

So, 'toddler' and 'tree' are types, there are three toddlers, 'climb' is a function from toddlers to a function from trees to propositions, i.e. a 2-place predicate,

- Since we are using a strictly typed system, we immediately get a form of selectional/typing restrictions, which restricts the number of possible CTT translations for terms,
- The dialogue context defines a saliency scale on concurring translations for terms.

[22]For non-first-order quantifiers, such a strategy is not generally available, as it is not guaranteed that there will be a 'weakest' reading to start from. In such cases, the ambiguity must be resolved as part of the process of converting ULF's to CTT, for instance by enumeration and checking for inconsistencies.

and 'collapse' is a 1-place predicate on trees. Now, *Every toddler climbed a tree* is translated (ignoring tense) into (27):

(27) u:(Π a:toddler.(Σ b:tree.climb(a)(b)))

From its type, we can see that the proof term 'u' of this proposition must be a function that, given a toddler 'x', returns an object of type 'Σ b:tree.climb(x)(b)'. But this is the representation of the proposition 'x climbs a tree'.

So in fact 'u' is a function that for every toddler returns a proof that this toddler climbed a tree. Look at these propositions of the form 'x climbs a tree'. Their CTT representation contains a Σ operator. This means that proofs of such propositions are actually pairs, the first member of which is the actual tree, call it 'y', that toddler 'x' climbed, and the second member is a proof that toddler 'x' climbed tree 'y'.

Now, we can create a function that, given a toddler 'x', returns the tree 'y' that that toddler climbed. It is clear that we need the proof object 'u' to do this. The function is constructed as follows: imagine we apply the function 'u' to a hypothetical toddler x. This gives us 'u(x)' of type 'Σ b:tree.climb(x)(b)'. This 'u(x)' is an inhabitant of a Σ type, so it must be a pair. This means we can use a projection which we shall call π_1 to get the first element of this pair (which is the individual tree that we want). We get π_1(u(x)) of type 'tree'. We now abstract away from our hypothetical toddler x, by means of lambda-abstraction in the proof object, and Π abstraction in the type (since we create a function).

This means we finally get (28):

(28) (λ x:toddler.π_1(u(x))):(Π x:toddler.tree)

Let us call this function 'p'.

We were trying to interpret the small discourse (25). The second sentence does not tell us anything new about the preferred scoping of the first sentence, but when we come to the third sentence, we discover that in fact, there should be some unique salient tree. We take this to imply that we have a *presupposition* that there is some unique, salient tree in the preceding discourse; however, the only representation we have to our disposal is our function 'p'.

But what can be done, is that we *accommodate* this presupposition by demanding that 'p' be a constant function, i.e. for every toddler 'x', 'p' should return the same tree. It is possible to define a predicate over functions, 'constant', which demands exactly this.[23] The only thing that is needed then

[23]The definition of this predicate is actually somewhat more complicated than may be evident from the main text. Since functions can have all kinds of domains and codomains, we either need a predicate 'constant' for every possible domain-codomain pair, or we must make it a *polymorphic predicate*, i.e., we also abstract over the type of the function. In the example, the function has type '(Π x:toddler.tree)'. If we would add a different predicate 'constant' for every domain-codomain pair, the type of 'constant' would simply be '(Π f:(Π x:toddler.tree).$*_p$)', i.e. a function from functions of toddlers to trees, to propositions.

to accommodate the presupposition is to subsequently add a proof object to indicate that 'p' is constant:

(29) acc:constant(p)

The constant value that 'p' returns can subsequently be used to represent the meaning of *The tree collapsed*:

(30) u3:collapse(p(jim))

Note that it doesn't matter which toddler we use as an argument to 'p'.
So we see how the use of explicit proof objects, which are inhabitants of types that represent propositions, gives us precisely the tool we need to handle the resolution, if necessary and possible, of scopal ambiguity via presupposition accommodation.

This kind of approach to quantifier scope ambiguity, using explicit functions, the properties of which are determined as the need arises, is also explored in the work of Van der Does and Verkuyl (to appear).

4 Conclusion and Future Work

We have discussed some[24] properties of the two main levels of semantic representation in DENK, specifically those properties which turn out to be useful w.r.t. the handling of potentially massive ambiguity. Of course, this short

In case we add a polymorphic predicate 'constant', we must abstract over the type of objects in the domain and the type of objects in the codomain of the function, so 'constant' would have the type '$(\Pi\ s{:}*_t.(\Pi\ t{:}*_t.(\Pi\ f{:}(\Pi\ x{:}s.t).*_p)))$', which can be read as: for all types s and t, from functions f from s to t, to propositions. Note that in this case the accommodation statement must take into account the polymorphic nature of the predicate, and it would have to be changed to:

acc:constant(toddler)(tree)(p)

Another problem is the way in which such a predicate 'constant' can be defined such that we can actually use it to derive some information from it. There are two straightforward possibilities for a definition of constant (restricting ourselves to the non-polymorphic case for reasons of clarity):

- 'constant' = $(\lambda f{:}(\Pi\ x{:}\text{toddler.tree}).\ (\Sigma\ c{:}\text{tree}.(\Pi\ t{:}\text{toddler}.\ \text{equal}(f(t),c))))$
- 'constant' = $(\lambda f{:}(\Pi\ x{:}\text{toddler.tree}).\ (\Pi\ t_1{:}\text{toddler}.(\Pi\ t_2{:}\text{toddler}.\ \text{equal}(f(t_1),f(t_2)))))$

Given a suitable definition of equality, it is worth noting that in the first option, we are using a Σ type. This means that an inhabitant of 'constant(p)', as in example (29), must actually be a pair, the first element of which is precisely the constant value of the function. The solution to getting the constant value that is taken in the main text, is by using the function applied to an arbitrary inhabitant of the domain type of the function, but note that we may not even know an inhabitant of the domain type. In that case the constant value of the function could not be represented in this way.

[24] Most certainly we do not pretend to have discussed either formalism thoroughly: neither do we pretend that the properties discussed here are the most interesting properties of either formalism.

investigation into the practical and theoretical advantages of using ULF and CTT for semantic representation cannot be more than preliminary. Many formal and practical issues are left open, and structural ambiguity as well as semantic vagueness have not been discussed here.

Moreover, the mapping from ULF to CTT has been left undiscussed: future work will define this, conceivably along lines as discussed in a.o. Pulman () and Crouch (1995). Also, we have mentioned neither the principles[25] nor any specifics of the DenK grammar fragment. Thus we left it fairly implicit how utterances of the user are transformed to typed feature structures containing a feature structure of type *ulf*. Both mappings, from English to ULF and from ULF to CTT, will be topic of future work by (subgroups and individuals of the group of) the authors of the present paper.

However, we might conclude from the foregoing presentation that both representation formalisms have interesting properties to start out with. The use of HPSG and typed feature structures brings a principled approach to syntax as well as a formally well-founded and efficient formalism for structural analysis. As we have seen in Sect. 2.1, at the level of typed feature structures we may use a straightforward method to avoid disjunction over atomic values. By excluding all context-information from this level, we get a radically clear distinction between context-dependent and context-independent semantic interpretation, cf. Sect. 2.2. This level, ULF, can be linearized and viewed as an interface between structural analysis and actual knowledge representation and reasoning.[26]

Furthermore, we have shown in Sect. 3 how CTT makes available a powerful knowledge representation formalism, which allows for reasoning, and all kinds of interesting objects. As discussed in Ahn and Kolb (1990), this formalism can be seen as a generalization of Discourse Representation Theory (DRT, Kamp 1984; Kamp and Reyle 1993), and thus as a generalization of one of the prevailing theories of dynamic semantics.[27]

Also, cf. Sect. 3.5, the formalism allows for an interesting method for underspecification of quantifier scopes. This is currently an important issue on the agenda of many computational semanticists (cf. a.o. Alshawi (1992), Nerbonne (1992), Allen (1993), Bos (1994), Reyle (1995)).

An interesting property of our approach is that no readings of a sentence containing quantifier scopes have to be destroyed. It is merely so that discourse or other knowledge is allowed to command that the more general reading be 'narrowed down' to a more restricted reading. Thus, at least as far as quantifier scope is concerned, our approach allows for *monotonic interpretation* in the sense of Alshawi and Crouch (1992).

[25] But cf. Rentier (1995) for a first, tentative approximation.

[26] A tentative definition of ULF has been described in Kievit (1994).

[27] However, it differs from these and other approaches in many interesting ways: cf. Ahn () for a preliminary discussion of ideas behind the use of CTT for representing knowledge of natural language processing agents, as well as Sect. 3.1. For discussion of proof theory and CTT, cf. Helmink and Ahn (1988).

Recapitulating, our two-level approach seems to allow for interesting methods of handling ambiguity resolution, discourse representation and reasoning on the basis of discourse representations. We hope to extend this approach both in breadth, by addressing more phenomena,[28] as in depth, by formalizing, implementing and describing the levels and mappings in full detail.

References

Ahn, R. Dynamic knowledge states in type theory. In *(Bunt et al. 1994)*, 1–10.

Ahn, R., Beun, R. J., Borghuis, T., Bunt, H., and Van Overveld, C. (1994). The DENK-architecture: a fundamental approach to user-interfaces. *Artificial Intelligence Review Journal*, 8(2). to appear.

Ahn, R., and Kolb, H. P. (1990). Discourse representation meets constructive mathematics. In Kálmán, L., and Pólos, L., editors, *Papers from the Second Symposium on Logic and Language*. Akadémia Kiadó. Budapest, Hungary.

Allen, J. F. (1993). Natural language, knowledge representation and logical Form. In Bates, M., and Wieschedel, R. M., editors, *Challenges in Natural Language Processing*. Cambridge University Press, USA.

Alshawi, H., editor (1992). *The Core Language Engine*. Cambridge, Massachusetts: MIT Press, USA.

Alshawi, H., and Crouch, R. (1992). Monotonic semantic interpretation. In *Proceedings 30th Annual Meeting of the Association for Computational Linguistics*, 32–39. Newark, Delaware, USA.

Barendregt, H. P. (1992). Lambda calculi with types. In Abramsky, S., Gabbay, D., and Maibaum, T., editors, *Handbook of Logic in Computer Science*. Oxford: Oxford University Press, UK.

Barwise, J., and Perry, J. (1983). *Situations and Attitudes*. Cambridge, Massachusetts: MIT Press, USA.

Beun, R. J. (1993). Rules in dialogue. In *Proceedings NATO-ASI 'Basics of Human-Machine Communication for the Design of Educational Systems'*. Veldhoven, the Netherlands.

Borghuis, T. (1994). *Coming to Terms with Modal Logic*. PhD thesis, University of Eindhoven, Eindhoven, the Netherlands.

Bos, J. (1994). Underspecified predicate logic. paper delivered at CLIN V.

[28]For a tentative definition of the relevant fragment, cf. Verlinden and Rentier (1995).

Bronnenberg, W., Bunt, H., Scha, R., Schoenmakers, W., and Van Utteren, E. (1979). The question answering system PHLIQA1. In Bolc, L., editor, *Natural Communication with Computers*, vol. II. München, Wien: Carl Hanser Verlag.

Bunt, H., Muskens, R., and Rentier, G., editors (1994). *Proceedings of the International Workshop on Computational Semantics.* ITK. Tilburg University, the Netherlands. 239 pages.

Calder, J., Klein, E., Moens, M., and Zeevat, H. (1987). Problems of dialogue parsing. Research Paper EUCCS/RP-1, Centre for Cognitive Science, Edinburgh University, Scotland.

Carpenter, B. (1992). *The Logic of Typed Feature Structures.* Tracts in Theoretical Computer Science. Cambridge: Cambridge University Press, USA.

Carpenter, B., and Penn, G. (1994). *The Attribute Logic Engine Users Guide.* Carnegie Mellon University, Pittsburgh, USA. manuscript.

Cooper, R. (1983). *Quantification and Syntactic Theory*, vol. 21 of *Synthese Language Library.* Dordrecht, the Netherlands: Reidel.

Coquand, T., and Huet, G. (1988). The calculus of constructions. *Information and Computation*, 76:95–120.

Crouch, R. (1995). Ellipsis and quantification: A substitutional approach. In *Proceedings of EACL 1995.* Dublin, Ireland.

Davidson, D. (1967). The logical form of action sentences. reprinted in his *"Essays on Actions and Events"*, Clarendon Press, Oxford, 1980, UK.

Gamut, L. (1991). *Logic, Language and Meaning.* Chicago: the University of Chicago Press.

Geurts, B., and Rentier, G. (1993). Quasi logical form in plus. internal report ESPRIT P5254 (PLUS), ITK, Tilburg University.

Helmink, L., and Ahn, R. (1988). Goal-directed proof construction in type theory. In Huet, G., and Plotkin, G., editors, *Logical Frameworks.* USA: Cambridge University Press.

Kamp, H. (1984). A theory of truth and semantic interpretation. In Groenendijk, J., Janssen, T., and Stokhof, M., editors, *Truth, Interpretation and Information.* Dordrecht: Foris Publications.

Kamp, H., and Reyle, U. (1993). *From Discourse to Logic.* Dordrecht, The Netherlands: Kluwer.

Kempson, R., and Cormack, A. (1981). Ambiguity and quantification. In *Linguistics and Philosophy*, vol. 4, 259–309. Dordrecht, the Netherlands: Reidel.

Kievit, L. Representing structural syntactic ambiguity. In *(Bunt et al. 1994)*, 131–140.

Kievit, L. (1994). Proto-ulf. DENK Research Report 94/02, ITK, Tilburg University, the Netherlands.

Martin-Löf, P. (1984). *Intuitionistic Type Theory.* Bibliopolis.

Montague, R. (1974). The proper treatment of quantification in ordinary english. In Thomason, R., editor, *Formal Philosophy: selected papers of Richard Montague.* New Haven, USA: Yale University Press.

Nerbonne, J. (1992). Constraint-based semantics. In Dekker, P., and Stokhof, M., editors, *Proceedings of the 8th Amsterdam Colloquim*, 425–445. University of Amsterdam. The Netherlands.

Pollard, C., and Sag, I. A. (1994). *Head-driven Phrase Structure Grammar.* University of Chicago Press and CSLI Publications.

Pulman, S. A computational theory of context dependence. In *(Bunt et al. 1994)*, 161–170.

Ranta, A. (1991). Intuitionistic categorial grammar. *Linguistics and Philosophy*, 14:203–239.

Rentier, G. (1995). Questions and left dislocation in DENK's hpsg Fragment. DENK Research Report 95/05, ITK, Tilburg University, the Netherlands.

Rentier, G. (ms.). Head-driven phrase structure grammar and underspecified logical form: Documentation of plug 1.0. DENK-document, ITK, Tilburg University, the Netherlands.

Reyle, U. (1992). Dealing with ambiguities by underspecification: A first order calculus for unscoped representations. In Dekker, P., and Stokhof, M., editors, *Proceedings of the 8th Amsterdam Colloquim*, 493–513. University of Amsterdam. The Netherlands.

Reyle, U. (1995). On reasoning with ambiguities. In *Proceedings of EACL 1995.* Dublin, Ireland.

Shieber, S. (1986). *An Introduction to Unification-Based Approaches to Grammar*, vol. 4 of *CSLI Lecture Notes.* USA: Stanford: CSLI.

Sundholm, G. (1986). Proof theory and meaning. In Gabbay, D., and Guenther, F., editors, *Handbook of Philosophical Logic*, vol. 3. Reidel Publishing Company.

Sundholm, G. (1989). Constructive generalized quantifiers. *Synthese*, 79:1–12.

Van der Does, J., and Verkuyl, H. (to appear). Quantification and predication. In Deemter, K. V., and Peters, S., editors, *Semantic Ambiguity and Underspecification*. Stanford, Ca.: CSLI Publications.

Verlinden, M., and Rentier, G. (1995). Towards a definition of the DenK natural language fragment. DenK Research Report 95/07, ITK, Tilburg University, the Netherlands.

Zeevat, H., Klein, E., and Calder, J. (1987). Unification categorial grammar. Research Paper EUCCS/RP-21, Centre for Cognitive Science, Edinburgh University, Scotland.