# Annotated Type Theory

for

# Information Representation

P.L.A. Piwek
Institute for Perception Research
P.O. Box 513
NL-5600 MB Eindhoven
The Netherlands
Email: piwek@natlab.research.philips.com

1995

# Table of Contents

**Abstract**

In this paper two formal languages for information representation in the DenK dialogue system are specified. The languages are (1) an extension of constructive type theory (CTT) for the representation of information about the domain of conversation and (2) an annotation language for coding information which enables the system to use and process information which is formulated in the language referred to under point (1).

## Terminology and List of Definitions

### Terminology

| | |
|---|---|
| A | Anchored |
| AP | Asserted Proposition |
| FL | Focus List |
| Gap | alternative name: Placeholder |
| Marker | A marker is a gap or CTT variable |
| RA | Requested Action |
| RB | Requires Binding |
| RI | Requested Information |
| RPU | Responsible Participant User |
| RPS | Responsible Participant System |

### Definitions

# 1 Introduction

The goal of the DenK project[1] (see Ahn et al., 1994) is to build a Cooperative Assistant based on fundamental principles of communication. This Assistant should be able to help people in the interaction with a Physical Domain. The Physical Domain, in principle, may be any (simulated) part of the real world. Typical application domains for the Assistant are complicated devices (e.g. vcrs, television sets, hi-fi installations and electron microscopes). The Assistant should be a generic cooperative interface which combines linguistic and visual interaction. It should be able to interpret questions and commands and generate assertions, clarificatory questions and actions.

One component of the DenK system's Cooperative Assistant is the module *Conduct*. This module determines what (communicative) actions the system has to perform at any given moment. Conduct decides what action to perform on the basis of the actual situation. This situation encompasses the state of the domain and the system's information state. Conduct is implemented as a rule-based system containing rules of the form:

> If it is the case that the current situation is of type X
> Then Do the actions Y.

In order to be able to execute these rules Conduct must be able to determine the type of the current situation. For this purpose it has two abilities: (1) It can observe the Physical Domain. In the DenK project the Assistant will help the User to interact with a simulation of a PHILIPS electron microscope. This means that the Physical Domain consists of the simulation of the microscope. (2) It can inspect its information state by checking (A) what information is part of the information state and (B) what information can be obtained from the information state through deduction.

Currently, the system's information state consists of two components: the Information Context and the Pending Stack. These function, roughly speaking, as the long-term memory and the working memory of the Cooperative Assistant.

There are three types of actions that the system can perform: (1) update its information state with the (underspecified) content of utterances; (2) communicate information; (3) perform actions on the Physical Domain.

In this paper two languages in which part of the contents of the information state are expressed will be specified. These languages are (1) an extension of *Constructive Type Theory* (CTT) to express information about the Physical Domain; (2) an Annotation Language to annotate expressions from CTT for representing information which the system needs to be able to process and use these expressions. As opposed to language no. (1), language no. (2) is not used to describe the Physical Domain. It derives its meaning from the role that it plays in the processing of information. This role is specified in Conduct's behaviour rules (see Kievit & Piwek, 1995) for a short description and Piwek (forthcoming) for a more elaborate account.

This paper consists of three major sections. In section 2 we will describe the architecture of the DenK system. Section 3 contains an informal introduction to CTT and a formal definition of the CTT language which is used in the DenK system. Finally, in section 4, the annotation language will be dealt with.

[1] DenK is a Dutch abbreviation for *Dialogue Handling and Knowledge Transfer*.

## 2 The DENK system and Conduct

The layout of the DENK system is shown in fig. 1. In this section the roles of the modules that are depicted in the picture will be described (a brief explanation of the abbrevations and terms used in this figure is given in table 1). There are three most inclusive modules: (1) the User (for which the term 'module' is perhaps somewhat unfortunate); (2) the GDP (Generalized Display Processor) which simulates the Physical Domain and displays this domain to the User through computer graphics; and (3) the Cooperative Assistant. The Cooperative Assistant consists of a number of submodules.

DABAS2 and the Evaluator are modules for observing and acting upon (a simulation by the GDP of) the Physical Domain.

The Parser, Sem. to ulf and Clues take care of the interpretation of utterances which have been produced by the User. Technically speaking, this means that an utterance by the User is translated into a pair of expressions, each of which belongs to a well-defined formal language. Such pairs are called annotated segments. An annotated segment consists of the following two elements: a CTT segment and an annotation. The segment represents the, possibly underspecified, (semantic) content of the utterance that the User produced. The annotation tells the system what to do with the content. An annotation contains, for instance, information on whether the User's utterance was a question, a command or an assertion. In other words, an annotation contains, amongst other things, information on the utterance's communicative function.

Tactics is the part of the system that can be used to obtain information about the Physical Domain. For this purpose Tactics can reason with information from the Information Context and, if necessary, also appeal to DABAS2 and the Evaluator for observation of the Physical Domain.

Conduct determines what actions the system has to perform at any moment given the situation at that moment. In order to determine what the current situation is it can (1) call Tactics to gather information about the Physical Domain and (2) inspect the contents of the Pending Stack. This stack contains those utterances which have been produced during the dialogue and have not yet been fully processed. Examples of utterances which have not yet been fully processed are: questions which still need to be answered, utterances whose presuppositions have not yet all been checked and assertions which have not yet been accepted by both dialogue participants.

Every utterance, whether produced by the User or the system, passes through the Pending Stack. When an utterance occupies the stack this should be interpreted as meaning that (1) both the User and the system are aware of the fact that the utterance was produced and (2) the utterance has not yet been fully processed by both the User and the system. For instance, if the system asserts something, the assertion will occupy the Pending Stack until the User has signalled to the system that he/she accepts the assertion.[2] As soon as the assertion has been accepted its content becomes part of what the User and the system assume to be their common background (*cf.* Stalnaker, 1974). This common background is part of the Information Context.

There are two modules which have not yet been discussed: Concepts and Verbal Output to User. Concepts is a lexicon which enables Clues (the interpretation module) to translate a representation of an utterance in ULF (Underspecified Logical Form)[3] into an annotated segment. The module Verbal Output to User translates the communicative actions that Conduct produces in the form of annotated segments into English expressions.

[2]Acceptance does not have to be explicitly voiced. If the user's next utterance is unrelated to the system's

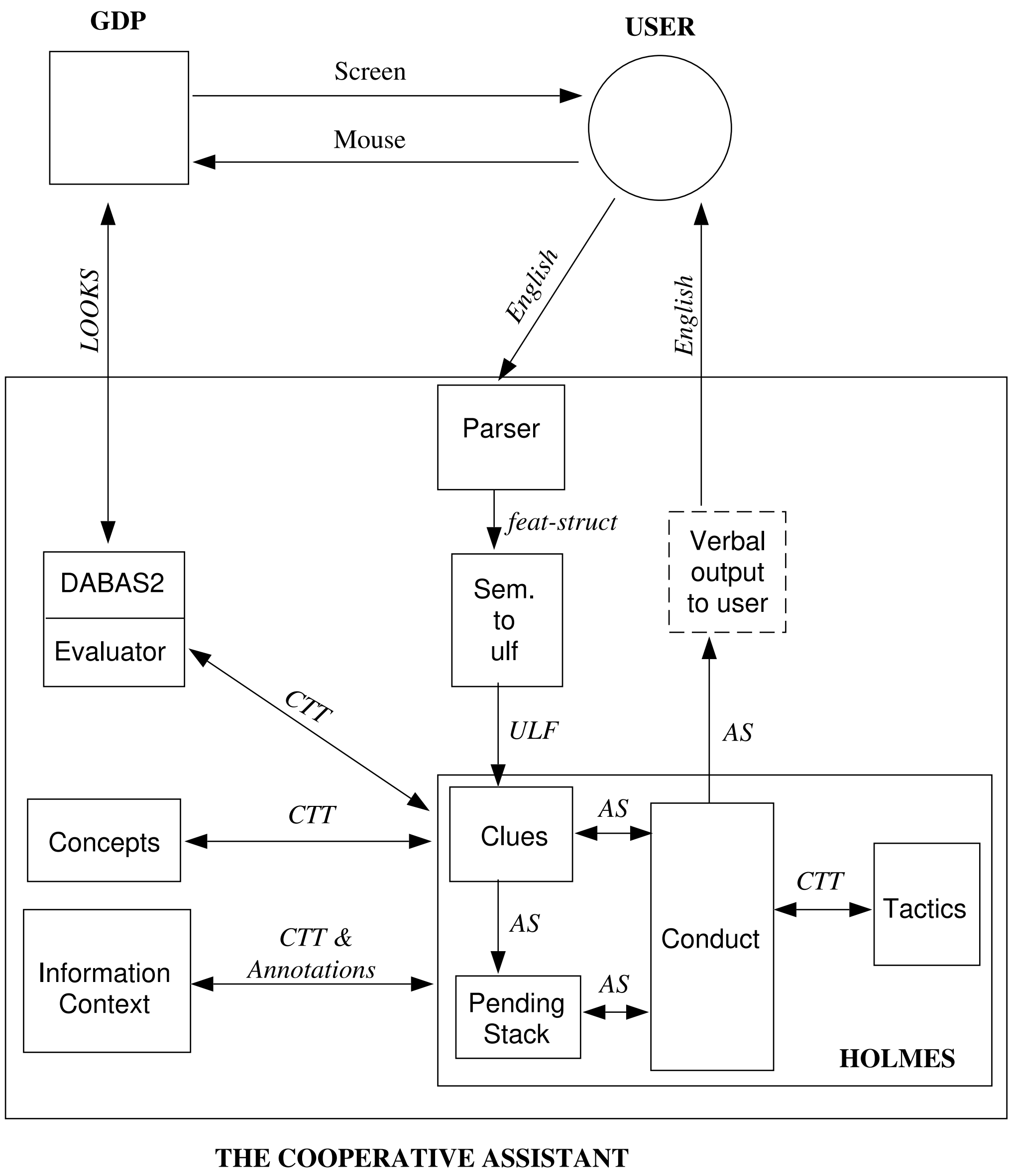


Figure 1: The DenK system

| | |
|---|---|
| AS | Annotated Segment |
| Clues | Interpretation Module |
| Conduct | Component containing the Behaviour Rules |
| CTT | Constructive Type Theory |
| Concepts | Conceptual Lexicon |
| Dabas2 & Evaluator | Component for observation of and action on the Physical Domain which is simulated by the GDP |
| Feat-Struct | Feature Structures. For parsing and representing the English phrases that the User produces (see Rentier, 1995). The grammatical theory that will be used is a variation on HPSG (Head-driven Phrase Structure Grammar). |
| GDP | Generalized Display Processor |
| LOOKS | Language for Object-Oriented Kinematic Specification. For the communication between the perception module of the DenK system (DABAS2 and the Evaluator) and the Physical Domain (as it is simulated by the GDP). |
| Pending Stack | This is the working memory of Conduct and Clues |
| Sem. to ULF | Translates feature structures into ULF |
| Tactics | Inference Engine |
| ULF | Underspecified Logical Form. For the representation of the contents of ambiguous sentences (see Kievit, 1994). |
| Verbal output to User (optional) | translates annotated segments into English. |

Table 1: List of Abbreviations and Terms occurring in Fig. 1

# 3 Constructive Type Theory

## 3.1 A Short Introduction to CTT

We will give an outline of CTT (Barendregt, 1992; Ahn & Kolb, 1990) by comparing it with Kamp's Discourse Representation Theory (DRT) (Kamp, 1981).

In DRT contexts are modelled as so-called Discourse Representation Structures (DRSs). A DRS consists of a set of discourse referents and a set of conditions. The discourse referents can be seen as pegs, and the conditions as assignments of properties to these pegs. For example, the sentence *A horse neighs* yields a DRS containing a fresh referent and two conditions: $[x \mid horse(x), neigh(x)]$. The conditions attach the properties *is a horse* and *neighs* to the referent.

In CTT a context is modelled as a sequence of so-called introductions. Introductions are of the form $V : T$, where $V$ is a variable and $T$ is the type of the variable. The introductions are ordered because the type $T$ may depend on introductions which preceded $V : T$. We will give an example of this shortly.

The variable V in an introduction $V : T$ (where $T$ itself is of the type $Set$, i.e. $T : Set$) can be seen as corresponding to a discourse referent in DRT. So, if we want to introduce a referent for an entity from the set of horses we write $x : horse$. The type $horse$ may only be used in the introduction $x : horse$ if $horse : Set$ is already part of the context (in other words, the introduction $horse : Set$ has to precede the introduction $x : horse$). This way, one introduction can depend on another introduction.

DRT's conditions correspond to introductions $V : T$ where $T$ is of the type $Prop$ (short for proposition). Thus the introduction $y : neigh \cdot x$ corresponds to the condition $neigh(x)$. The type $neigh \cdot x$ (of type $Prop$) is obtained by applying the type $neigh$ to the object $x$. It depends on the introductions of $x$ and $neigh$. We already discussed the introduction of $x$, so let us have a look at $neigh$. Since $neigh \cdot x$ should be of the type $Prop$, $neigh$ must be a (function) type from the set of horses into propositions, i.e. $neigh : horse \rightarrow Prop$.

The introduction $y : neigh \cdot x$ also involves the variable $y$ (of the type $neigh \cdot x$). $y$ is said to be an inhabitant of $neigh \cdot x$. What could possibly be the inhabitants of propositions? Curry (Curry and Feys, 1958) came up with the idea that propositions can be seen as classifying proofs (this is known as the propositions as types – proofs as objects paradigm). The introduction tells us that there is a proof $y$ for the proposition $neigh \cdot x$.

If a person claims that he has a proof for a proposition, this comes very close to saying that the proposition is true. There is, however, an essential difference. A proof, as opposed to "the truth", is to some extent a subjective thing. This is so, because a proof is built up of premisses. People sometimes disagree on the premisses that may be used to prove something, and consequently also disagree on what counts as a proof. CTT allows us to model individuals who work with different premisses; these agents simply use different CTT contexts to prove things.

In DRT the proposition *All horses neigh* is translated into the implicative condition $[x|horse(x)] \Rightarrow [\,|\,neigh(x)]$. In CTT this proposition corresponds to the type $(\Pi x : horse.neigh \cdot x)$, which is a so-called dependent function type. It is a function from the type $horse$ into the type $neigh \cdot x$. The range of this function $(neigh \cdot x)$ depends on the object $x$ to which it is applied. Suppose that we have an inhabitant $f$ of this function type, i.e. $f : (\Pi x : horse.neigh \cdot x)$. Then we have a function which, when it is applied to any object, say $y$, of type $horse$, yields an inhabitant of the type $neigh \cdot y$. In other words, $f$ is a constructive proof for the

assertion, the System assumes that the assertion has been accepted.

[3] See Kievit (1994) for a specification of the ULF language.

proposition that *All horses neigh.*[4]

Finally, we would like to illustrate how one can reason with the contexts of CTT. CTT encompasses a number of derivation rules with which one can determine the type of an object in a given context. These rules can also be used to search for an object belonging to a particular type. One of the rules bears a close resemblance to Modus Ponens in Predicate Logic, but it also shows elements of function application in it:[5]

> IF in context $\Gamma$ it holds that $F : (\Pi x : A.B)$ and $a : A$
> THEN in context $\Gamma$ it holds that $F \cdot a : B[x := a]$

**Example 1** If the context contains the introductions $x : horse$ and $f : (\Pi y : horse.neigh \cdot y)$, then we can use this rule to find an inhabitant of the type $neigh \cdot x$. The rule tells us that we should apply the function $f$ to the object $x$ and substitute $x$ for $y$ in $neigh \cdot y$. This yields the complex proof object $f \cdot x$ of the proposition $neigh \cdot x$.

### 3.2 Formal Specification: The Kernel

In this section a formal specification of CTT, in the spirit of Barendregt (1992), is given. A ProLog-implementation of this specification can be found in the appendix (section 6 on page 21).

Intuitively, types can be seen as representing the concepts that the Cooperative Assistant has mastered. Mastering a concept boils down to acquiring the ability to recognize objects falling under the concept (i.e. inhabitants of the type). Whereas human agents acquire concepts through interaction with and training in the environment, our artificial agent was constructed with the appropriate concepts. Human beings use concepts in virtue of their causal links to reality (perception and action). In the DENK system these causal links are substituted by computational links: the Assistant can compute whether or not an object (in a Physical Domain simulated in an object-oriented language) belongs to a type by applying an evaluation function.

In the following definitions $V$ is a set of variables such that $V \cap \{*_p, *_t, *_e, \Box\} = \emptyset$

**Def. 1 (sorts)** $\mathcal{S} = \{*_p, *_t, *_e, \Box\}$

The four special types $*_p, *_t, *_e$ and $\Box$ are called *sorts*. $*_p$ and $*_t$ correspond to what we called *Prop* and *Set* above. $*_e$ stands for type of *events*. All three $*$s are themselves of the sort $\Box$ (see the axioms in definition 7 on page 8).

**Def. 2 (pseudo-terms)** $T := \mathcal{S} \mid V \mid (T \cdot T) \mid \lambda V : T.T \mid \Pi V : T.T$

**Def. 3 (introductions)** $I := V : T$

**Def. 4 (pseudo-contexts)** $\Gamma := \epsilon \mid \Gamma, I$

In other words, a pseudo-context is a sequence of introductions. $\Gamma, I$ is the sequence which is obtained by appending $I$ to the sequence $\Gamma$. $\epsilon$ is the empty sequence.

[4]In Ahn & Kolb (1990) we find a somewhat different notation for function types. For instance, in their notation *All horses neigh* corresponds to $[x : horse] \Longrightarrow [f \cdot x : neigh \cdot x]$. This notation reveals the similarities between dependent function types and the implicative conditions of DRT.

[5]In this rule $T[x := a]$ stands for a $T$ such that all occurrences of $x$ in $T$ have been substituted by $a$.

**Def. 5 (pseudo-judgements)** $J := \Gamma \vdash T : T$

**Def. 6 (rules)** $\mathcal{R}_\Pi \subseteq \{\langle x, y\rangle \mid x \in \mathcal{S}, y \in \mathcal{S}\}$

These rules should not be confused with the deduction rules which we will give below. The rules in $\mathcal{R}_\Pi$ are used by a deduction rule called formation ($form$). This deduction rule can be used to construct function types. The choice of $\mathcal{R}_\Pi$ determines what kind of functions can be constructed. For instance, if $\langle *_t, *p\rangle \in \mathcal{R}_\Pi$, then predicates can be constructed, i.e. functions from objects to propositions.

**Def. 7 (deduction rules)** *The Rules which axiomatize the relation $\vdash$ are specified below. The usual notions of fresh variable ($\Gamma$-fresh: a variable which does not occur in $\Gamma$) and substitution (The replacement of all occurences of a variable in a term with another variable) are used. The notion of beta-equality ($=_\beta$) which we will use will be defined below.*

$$(axioms) \qquad \epsilon \vdash *_e : \Box \qquad \epsilon \vdash *_t : \Box \qquad \epsilon \vdash *_p : \Box$$

$$(start) \qquad \frac{\Gamma \vdash A : s}{\Gamma, x : A \vdash x : A} \qquad x \text{ is } \Gamma\text{−fresh}$$

$$(weaken) \qquad \frac{\Gamma \vdash A : B \quad \Gamma \vdash C : s}{\Gamma, x : C \vdash A : B} \qquad x \text{ is } \Gamma\text{−fresh and } s \in \mathcal{S}$$

$$(form) \qquad \frac{\Gamma \vdash A : s_1 \quad \Gamma, x : A \vdash B : s_2}{\Gamma \vdash (\Pi x : A.B) : s_2} \qquad \langle s_1, s_2\rangle \in \mathcal{R}_\Pi$$

$$(intro) \qquad \frac{\Gamma, x : A \vdash b : B \quad \Gamma \vdash (\Pi x : A.B) : s}{\Gamma \vdash (\lambda x : A.b) : (\Pi x : A.B)} \qquad s \in \mathcal{S}$$

$$(elim) \qquad \frac{\Gamma \vdash F : (\Pi x : A.B) \quad \Gamma \vdash a : A}{\Gamma \vdash F \cdot a : B[x := a]}$$

$$(conv) \qquad \frac{\Gamma \vdash A : B \quad \Gamma \vdash B' : s \quad B =_\beta B'}{\Gamma \vdash A : B'} \qquad s \in \mathcal{S}$$

**Def. 8 (beta-reduction)** $(\lambda x : A.M) \cdot N \rightarrow_{1\beta} M[x := N]$

**Def. 9 (congruence)** $P \equiv_\alpha Q$ *iff $Q$ has been obtained from $P$ by a finite (perhaps empty) series of changes of bound variables (i.e. variables which are within the scope of a lambda, as the variable $x$ in $\lambda x.x$, which can renamed into $y$, which yields $\lambda y.y$).*

**Def. 10 (beta-equality)** $P =_\beta Q$ *iff $Q$ is obtained from $P$ by a finite (perhaps empty) series of $\beta$-reductions and reversed $\beta$-reductions and changes of bound variables: $P_0 =_\beta P_n$ iff there exist $P_1, \ldots P_{n-1}$ such that for all $0 \leq i \leq n-1$: ($P_i \rightarrow_{1\beta} P_{i+1}$ or $P_i \leftarrow_{1\beta} P_{i+1}$ or $P_i \equiv_\beta P_{i+1}$).*

**Def. 11 (legal term)** *A pseudo-term $E$ is a (legal) term if a pseudo-context $\Gamma$ and a pseudo-term $T$ exist so that $\Gamma \vdash E : T$ is derivable.*

**Def. 12 (legal context)** *A pseudo-context is a (legal) context if there are two terms $E$ and $T$ such that $\Gamma \vdash E : T$ is derivable.*

**Def. 13 (legal judgement)** *A pseudo-judgement is a (legal) judgement if it is derivable.*

**Theorem 1 (subject reduction)** *If $\Gamma \vdash E : T$ and $E \rightarrow_\beta E'$ then $\Gamma \vdash E' : T$. ($E_0 \rightarrow_\beta E_n$ iff there exist $E_1, \ldots, E_{n-1}$ such that for all $0 \leq i \leq n-1$: ($P_i \rightarrow_{1\beta} P_{i+1}$).*

## 3.3 Extensions of CTT

### 3.3.1 Segments

**Def. 14 (segment)** *A pseudo-context $S$ is a segment given a legal context $\Gamma$, if $\Gamma$, $S$ is a legal context.*

**Def. 15 (negative segment)** *A negative segment is a segment in which some CTT variables have been replaced by so-called gaps (alternative name: placeholders).*

**Example 2** Suppose $[x : T_1, y : T_2, z : T_3]$ is a segment (given some context $\Gamma$). Then we can obtain the negative segment $[P : T_1, y : T_2, Q : T_3]$ by replacing the variables $x$ and $z$ by the gaps $P$ and $Q$. Gaps are, just like ProLog-variables, marked with capital letters. This is not accidental. In fact, on the implementation level, negative segments will be implemented as ProLog queries. For instance, the negative segment $X : walk \cdot john$ is interpreted as a query asking for inhabitants (i.e. proofs) of the proposition $walk \cdot john$.

### 3.3.2 Definitions

The use of definitions in CTT was proposed by N.G. de Bruijn (see De Bruijn, 1980). After an inhabitant $E$ for a particular type $T$ has been constructed, a definition can be used to store it: a fresh variable is defined as an abbreviation of $E$: $x = E : T$. $x$ can subsequently be used as an inhabitant of type $T$. This means that if we later on need an inhabitant of type $T$, we can use the variable $x$, and do not again have to go through a (perhaps cumbersome) construction of $E$.

Since definitions are part of the context, and contexts consist of introductions, the notion of an introduction has to be redefined:

**Def. 16 (introductions′)** $I := V : T \mid V = T : T$

Furthermore, we need to add extra start en weakening rules:

$$(def.start) \qquad \frac{\Gamma \vdash A : s}{\Gamma, x = E : A \vdash x : A} \qquad (\Gamma, x = E : A) \subseteq \Gamma_{base} \text{ and } x \text{ is } \Gamma\text{−fresh}$$

$$(def.weaken) \qquad \frac{\Gamma \vdash A : B \quad \Gamma \vdash C : s}{\Gamma, x = E : C \vdash A : B} \qquad (\Gamma, x = E : C) \subseteq \Gamma_{base}; \ x \text{ is } \Gamma\text{−fresh and } s \in \mathcal{S}$$

Note that we did not choose for the following alternative (perhaps more standard) version of the *def. start* rule (cf. Severi and Poll, 1993 whose weakening rule also differs from ours):

$$(def.start) \qquad \frac{\Gamma \vdash E : A}{\Gamma, x = E : A \vdash x : A} \qquad x \text{ is } \Gamma\text{−fresh}$$

The use of this version of the start rule means that definitions are only a matter of cosmetics: whenever an inhabitant has to be found for some type $T$, and there is a definition $x = E : T$, the rule tells us that $x$ is an inhabitant if we can derive that $E$ is an inhabitant. This means that we have to go through all the work of proving that $E$ is indeed an inhabitant of $T$.

The rule which we have adapted makes substantial use of the definition. If we have $\Gamma, x = E : T$, then we can, without checking $E$, derive $x : T$ in $\Gamma$. Of course, we would like some constraint on the introduction of $E$. For this purpose the two rules are tagged with the clause $\Gamma, x = E : A \subseteq \Gamma_{base}$ and $\Gamma, x = E : C \subseteq \Gamma_{base}$, respectively.

The context $\Gamma_{base}$ is the legal context from which we start if we want to construct an inhabitant or assign a type to an inhabitant. For instance, if we want to construct an inhabitant of $walk \cdot mary$ our goal is to find a substitution $S$ for $P$ such that: $\Gamma_{base} \vdash P : walk \cdot mary[S]$ ($\Gamma_{base}$ will, for instance, have to contain the introduction $mary : woman$ and $woman : *_t$).

A definition $x = E : T$ may be added (from outside) to $\Gamma_{base}$ if $x$ is $\Gamma$-fresh and $\Gamma_{base} \vdash E : T$ (we assume that we start off with a $\Gamma_{base}$ which contains no definitions). This way we make sure that the $E$ in the definition makes sense. On the other hand, we avoid having to derive $E : T$ every time we want to find an inhabitant of $T$. The only point at which we require that $E : T$ has to be derived is when the definition $x = E : T$ is added to $\Gamma_{base}$. This way, the decision when to add a definition has become an entirely pragmatic one, and is not part of the deduction rules.

Definitions can, amongst other things, be used to connect new variables which come from a new utterance to the context. A new variable can also be connected to the context through substitution.

**Example 3** (substition) Suppose we have a context $[\ldots, c12 : lens, \ldots]$ and a negative segment $[X : lens, p : on \cdot X]$. In that case the following *substitution* connects the segment to the context: $X := c12$. If we apply this substitution to the segment we get $[c12 : lens, p : on \cdot c12]$. We can update the context with this segment by appending to the context only those introductions that do not already occur in the context (provided that the requirements on context updates which we will describe in section 3.3.6 on page 13 have been fulfilled). This results in: $[\ldots, c12 : lens, \ldots, p : on \cdot c12]$.

**Example 4** (definitions) Suppose we have a context $[\ldots, c12 : lens, \ldots]$ and a segment $[x : lens, p : on \cdot x]$. In that case the following *definition* connects the segment to the context: $x = c12 : lens$. We can update the context with this segment by first replacing $x : lens$ by $x = c12 : lens$ and next appending all the introductions in the segment to the context. This results in: $[\ldots, c12 : lens, \ldots, x = c12 : lens, p : on \cdot x]$.

There is one reason for using definitions as means for connecting segments to contexts: definitions are appended to the right side of the context whenever a CTT variable which is a member of the context is referred to. This means that the order in which CTT variables are referred to is recorded by the context.

### 3.3.3 Subtyping

In CTT, the type of a predicate, say $p$, is a function from some type $A$ into the type of propositions $*_p$, (i.e., $p : A \to *_p$). This means that a predicate can only be applied to objects of one particular type. Sometimes, however, it may be convenient to be able to apply a predicate to objects of different types. We may, for instance, want to have a CTT equivalent of the natural language predicate *red*. The natural language predicate *red* can be applied to cars, carpets, books, and so on. In pure CTT, natural language predicates like *red* can be represented with the help of polymorphic types. A polymorphic predicate is of the type $\Pi T : *_t.T \to *_p$. This way the predicate is no longer directly applicable to an object. It first has to be applied to the type of the object. This yields a predicate in the orginal sense, i.e., a function from the type of the object into $*_p$.

**Example 5** Take the polymorhic predicate $red : \Pi T : *_t.T \rightarrow *_p$ and an object $x : car$. Before the predicate can be applied to object $x$, we first have to apply it to type of the object. This results in $red \cdot car : car \rightarrow *_p$. Now we can apply $red \cdot car$ to the object $x$, in order to obtain an object of the type proposition: $red \cdot car \cdot x : *_p$.

There is a different way of obtaining predicates which can be used for more than one type of objects. It is called subtyping (see, e.g., Ahn (1995) for subtyping applied to the electronmicroscope domain). The following rule, which is given after extending the notion of an introduction, is most important with respect to subtyping (of course, we also need to add new start and weaken rules):

**Def. 17 (introduction$''$)** $I := V : T \mid (V = T) : T \mid (V < V) : *$ *(where* $*$ *ranges over* $*_e$, $*_t$ *and* $*_p$*)*

$$\frac{\Gamma \vdash c : C \quad \Gamma \vdash C < A : * \quad \Gamma \vdash F : (\Pi x : A.B)}{\Gamma \vdash F \cdot c : B[x := c]}$$

This rule says that if $c$ is of type $C$ and $C$ inherits from type $A$, then all function applicable to objects of type $A$ are also applicable to objects of type $C$.

**Example 6** Suppose we need to give an interpretation of the indefinites occurring in the following small discourse: *I see an animal. It is a rabbit.* In line with DRT, we will treat indefinites as introducing a new discourse referent (i.e. CTT variable). Furthermore, we will assume that natural language nouns are systematically interpreted as types. Then we can write:

$$[d : animal, k : rabbit, p : equal \cdot animal \cdot d \cdot k]$$

This is a proper representation of the indefinite *an animal* and the sentence *It is a rabbit*, if we assume the use of subtyping ($rabbit < animal : *_t$) and polymorphyic types (the predicate $equal$ is assigned the polymorphic type $\Pi T : *_t.T \rightarrow T \rightarrow *_p$). Subtyping is necessary in order to ensure that the object $k$ inherits all the animal properties of $d$ and that the predicate $equal$ can be applied to both animals and rabbits.

Subtyping can also be used for the predicate *equal*. Since equality can be tested for any two objects which have the same type or stand in the subtype relation, we then need a type which is the supertype of all the atomic types. Let us call this type *top*. Consequently, *equal* can be defined as a two-place predicate for objects of the type *top* ($equal : top \rightarrow top \rightarrow *_p$). There is one disadvantage to this approach: it only allows for the application of *equal* to atomic types, because subtyping is only defined for atomic types. Sometimes it may, however, be necessary to express equality between objects of a non-atomic type, like functions. In this respect a polymorphic equality predicate is superior to an equality predicate which makes use of subtyping.

### 3.3.4 $\Sigma$-types

$\Sigma$-types (Löf, 1984) are useful for framing a dynamic and compositional semantics in CTT (see Ranta, 1994). The inhabitant of a $\Sigma$-type is a pair of objects. Suppose the $\Sigma$-type is $(\Sigma x : A.B)$ (where $A : *_t$ and $B : *_p$), then a pair inhabiting $(\Sigma x : A.B)$ consists of an object of type $A$ and a proposition of type $B$. Thus the assertion of *A man walks* can be translated into $p : (\Sigma x : man.walk \cdot x)$. The operators $\pi_1$ and $\pi_2$ can be applied to $p$ in order to obtain an object of type *man* and a proof that the man walks, respectively.

With a combination of $\Pi$ and $\Sigma$-types the renowned donkey sentence (*If a man owns a donkey, he beats it.*) can be represented in CTT:[6]

$$(\Pi p : (\Sigma x : man.(\Sigma y : donkey.own \cdot x \cdot y)).beat \cdot (\pi_1 \cdot p) \cdot (\pi_1 \cdot (\pi_2 \cdot p)))$$

This formula should be read as follows: If we have a proof that there is a man and a donkey and the man owns the donkey ($\Sigma x : man.(\Sigma y : donkey.own \cdot x \cdot y)$), then we have a proof that the man beats the donkey: $beat \cdot (\pi_1 \cdot p) \cdot (\pi_2 \cdot (\pi_1 \cdot p)))$.

To accommodate $\Sigma$-types we need to extend the definitions of beta-reduction, rules and pseudo-terms:

**Def. 18 (beta-reduction′)** *Beta-reduction ($\rightarrow_{1\beta}$) is defined as follows:*

- $(\lambda x : A.M) \cdot N \rightarrow_{1\beta} M[x := N]$;
- $\pi_1(\langle A, B\rangle) \rightarrow_{1\beta} A$;
- $\pi_2(\langle A, B\rangle) \rightarrow_{1\beta} B$.

**Def. 19 (rules′)** $\mathcal{R}_\Pi \subseteq \{\langle x, y\rangle \mid x \in \mathcal{S}, y \in \mathcal{S}\}$, $\mathcal{R}_\Sigma \subseteq \{\langle x, y\rangle \mid x \in \mathcal{S}, y \in \mathcal{S}\}$

**Def. 20 (pseudo-terms′)** $T := \mathcal{S}|V|(T{\cdot}T)|\lambda V : T.T|\Pi V : T.T|\langle T, T\rangle|\Sigma V : T.T|\pi_1 T|\pi_2 T$

Furthermore, the rules $\Sigma$-form, $\Sigma$-intro, $\pi_1$ and $\pi_2$ have to be added to the set of rules of definition 7 on page 8.

$$(\Sigma\text{-form}) \qquad \frac{\Gamma \vdash A : s_1 \quad \Gamma, x : A \vdash B : s_2}{\Gamma \vdash (\Sigma x : A.B) : s_2} \qquad \langle s_1, s_2\rangle \in \mathcal{R}_\Sigma$$

$$(\Sigma\text{-intro}) \qquad \frac{\Gamma \vdash a : A \quad \Gamma \vdash b : B[x := a] \quad \Gamma \vdash (\Sigma x : A.B) : s}{\Gamma \vdash \langle a, b\rangle : (\Sigma x : A.B)} \qquad s \in \mathcal{S}$$

$$(\pi_1) \qquad \frac{\Gamma \vdash p : (\Sigma x : A.B)}{\Gamma \vdash (\pi_1 p) : A}$$

$$(\pi_2) \qquad \frac{\Gamma \vdash p : (\Sigma x : A.B)}{\Gamma \vdash (\pi_2 p) : B[x := (\pi_1 p)]}$$

### 3.3.5 Modalities

The declarative information that the Cooperative Assistant has at its disposal is stored in one CTT context (the Information Context) which is divided into two parts: *common* and *private*. *Common* is defined as the place where the Assistant stores the (declarative) information which it *believes* has been recognized to be shared by the dialogue participants. *Private* contains all the information that the Assistant does not believe to share with the User.

The two subcontexts are taken as primitives. This means that we do not model modalities in the object language (CTT) itself. The CTT language does not contain operators like $C$, $P$ or $B$ (for common, private and belief).

[6] *Cf.* Sundholm (1986).

CTT does not exclude the incorporation of epistemic operators into the object language. Borghuis (1995) has shown how modalities can be modelled explicitly in CTT. Borghuis extended CTT with deduction rules for entering and exiting subcontexts (in the following two rules $B_A$ is an operator for *person A believes that*):

> $B$–import: **IF** $E : B_A\beta$ is derivable in an A-context $\Gamma$, **THEN** $imp \cdot E : \beta$ is derivable in an $A$-context subordinate to $\Gamma$.
>
> $B$–export: **IF** $E : \beta$ is derivable in A context subordinate to some context $\Gamma$, **THEN** $exp \cdot B_A\beta$ is derivable in $\Gamma$.

The difference between this approach and the approach involving the separation of contexts into parts is that this approach allows the Assistant to reason about propositional attitudes within CTT contexts. Currently, the modalities *common* and *private* are used by the behaviour module Conduct in a fairly primitive way, which does not require any real CTT reasoning about these modalities. Of course, if we had allowed the User to use, for instance, belief sentences, this would have been different, because in the DenK system the content of the user's utterances is translated into CTT.

#### 3.3.6 Context Update

In the system which we have described so far a context can only be extended by applying the *start* or *weakening* rules (*cf.* definition 7 on page 8) or by adding a definition or subtype introduction to the context .

The Cooperative Assistant maintains a context which contains all the information that has been exchanged during the dialogue (this context forms a part of the Information Context). The Assistant updates this context whenever the User or he himself has said something. This way information is introduced into the context 'from outside'. Borghuis (1995) formulated two requirements which an introduction $V : T$ has to meet if it is to be appended to a context $\Gamma$:

1. $\Gamma \vdash T : s$ (where $s \in \mathcal{S}$).
2. $V$ is fresh in $\Gamma$.

This way the extension of the context with the introduction $V : T$ looks as if it could have been obtained by either *start* or *weakening*.

# 4 Annotations

The annotation language is a means for annotating (negative) segments and CTT contexts. The annotations contain information which guides Clues and Conduct in their processing of (negative) segments and their use of contexts. Clues and Conduct can make use of so-called *services*, which (amongst other things) can be used to consult information that has been recorded in the annotation language. A preliminary specification of these services can be found in Kievit and Piwek (1995). A more extensive description of the services that Conduct uses is in preparation (Piwek, forthcoming).

## 4.1 The Annotation Language

In the preceding sections we have spoken about CTT variables and gaps. Below, we will use the term *markers* as a collective noun for CTT variables and gaps.

**Def. 21 (feature-value pairs)** *A feature-value pair is a pair $\langle A, B\rangle$, where $A$ is a feature and $B$ its value. Let us, for now, assume that only sets of markers and lists of markers are allowed as values.*

At the moment, it is not yet possible to specify precisely what values features are allowed to have.[7] The choice of a particular type of value depends to a great extent on the kind of services that we need to inspect the content of these features and on the kind of information they are supposed to record. For instance, if we want to record that one object depends upon another we probably need relations, i.e. set of pairs of objects. We may even discover that for some purposes sets of n-tuples of objects are required or lists of lists of ... of lists.

**Def. 22 (annotation)** *An annotation is a set of feature-value pairs.*

**Def. 23 (annotated segment)** *An annotated segment is a pair $\langle S, A\rangle$ where $S$ is a (negative) segment and $A$ is an annotation.*

**Def. 24 (annotated context)** *An annotated context is a pair $\langle C, A\rangle$ where $C$ is a context, and $A$ is an annotation.*

## 4.2 Annotations for Segments

The DenK system's interpretation modules construct annotated segments out of the User's natural language input (the Parser, Sem. to ulf and Clues). They are stored in the Pending Stack, where they are operated upon by the modules Clues and Conduct. It is the module Conduct that determines what operations have to be carried out on the annotated segments. It may, for instance, ask the module Clues (see Beun & Kievit, 1995) to resolve the anaphoric material that an annotated segment contains or decide to update the context with the content of an annotated segment.

It is important to note that it is not only the User's natural language input that is stored in the Pending Stack: the communicative acts that the Assistant produces pass through it. In the latter case, however, the whole interpretation process is skipped. The reason for also storing the Assistant's utterances (temporarily) in the pending stack is that this stack indicates what utterance is currently receiving both the User's and the Assistant's attention. Therefore, feedback signals (e.g. questions for clarification like *Which lens do you mean?*) are interpreted with respect to the contents of the pending stack. Furthermore, a temporary store of utterances is needed to bridge the gap between the time of utterance and the time at which an utterance is accepted by the addressee and subsequently added by both speaker and addressee to their common background.

The module Conduct uses annotations to determine what kind of actions it has to perform. For instance, when the annotation of a segment indicates that the annotated segment represents a question, Conduct may decide to generate an answer.

We will now give a short overview of the kind of features that Conduct makes use of. All these features have a *set* of markers as their value. Conduct uses three services for manipulating and inspecting features and some other services which can perform tests on the values of features (The value of a feature can be obtained through the first service):

[7] At the time that this document was written, the full set of services was not yet known. Thus a definitive specification of the possible values for features could not yet be given. Furthermore, on the implementation level we will select feature values on the basis of how fast they can be inspected by the services of Clues and Conduct.

1. A function which returns the value (which is a set of markers) of a feature belonging to a specific annotated segment:
$return - value(+Feature, +AnnotatedSegment, -Value)$.

2. A procedure to remove a marker from the value of a particular feature.
$Remove(+Marker, +Feature, +AnnotatedSegment)$.

3. A procedure to add a marker to the value of a particular feature.
$Add(+Marker, +Feature, +AnnotatedSegment)$.

4. There are a number of operations which can be performed on sets (i.e. feature-values): (A) Check whether the set is empty or not: $empty - set(+Set, -TruthValue)$. (B) operation for taking the union and intersection of sets:
$union(+Set_1, +Set_2, -Set_3)$,
$intersection(+Set_1, +Set_2, -Set_3)$.

**AP** (Asserted Proposition). This feature has as its value the set of those proof objects belonging to a given annotated segment (i.e. markers whose type is an inhabitant of $*_p$) which have been used to posit a proof for a proposition, in other words to assert the proposition. This way these markers can be distinguished from proof objects which are used to pose a question, i.e. whether the marker can be connected to a proof object which is already part of the context. Such markers may arise out of yes/no-questions. See Piwek (forthcoming) for a treatment of yes/no-questions in terms of annotated segments.

**RPU** and **RPS** (for, respectively, Responsible Participant User and Responsible Participant System). These features indicate which dialogue participant introduced a particular marker (by uttering a particular sentence). In our model we keep track of this information because both what the system says and what the User says is stored in the *Pending Stack*. In order to process the contents of the *Pending Stack* it is crucial that the system knows what was said by itself and what by the User.

Furthermore, the information can be helpful when a contradiction arises: if the system arrived at a contradiction by combining its own information and that from the User, then it can choose whether it judges its own information or that of the User to be more reliable. Based on this judgement it can discard the information which came from either the system or the User. The judgement may be derived from an assesment of the level of expertise of the system and the User (concerning the Physical Domain). In the current implementation of the DenK system, the system is rated 'expert' and the User 'beginner'.

**RB** (Requires Binding). The value of this feature consists of those markers which come from anaphoric expressions. It indicates that the markers have to be connected to the context. In case the marker is a gap, it will be substituted by a CTT variable from the context. If it is a variable a definition will be added which defines the variable in terms of a variable which was already part of the context.

If the value of the RB feature is not empty, Conduct calls Clues (in particular that part of Clues which is named *resolve*) in order to connect the markers which are members of the value to the context. If a marker has been resolved (i.e. connected to the context), then Clues removes the marker from the value of the feature RB.

RB is used to indicate the anaphoric parts of an utterance, stemming from, amongst other things, definites and pronouns. Ellipsis and in particular VP ellipsis (sometimes also called VP anaphora) is also represented with the help of the RB feature. An expression like *The lens* will be translated into the following annotated segment: $\langle [X : lens, Q : lens \to *_p, P : Q - X], [RB = \{X, Q\}, \ldots] \rangle$. (The dots indicate that we have not indicated all the features. In particular, additional features that Clues uses to facilitate the resolution of anaphors have been ommited. Clues may, for instance, need special features to be able to distinguish between definite articles and demonstratives. It will also need to know whether one referent modifies another referent, as in *The lens in the condenser system.* Here the fact that $q$ is a member of RB means that a predicate of type $lens \to *_p$ has to be found and substituted for $Q$. Suppose that *The lens* is uttered as a response to the question *Which lens is on?*, then the representation of the predicate *be on* would be a good substitute for the marker $Q$.

**RA** (Requested Action). We use the feature RA to indicate which markers have a type (which is itself of type $*_e$) that stands for an action that the Assistant has to perform. As soon as the action has been performed the Assistant substitutes a fresh CTT variable for the marker. This way, when the context is updated with the annotated segment, it will be extended with the information that the action has been carried out.

**Example 7** Suppose the system is confronted with the command: 'Switch off the lens!' In that case the following annotated segment will be pushed onto the pending context:[8]

$$\langle [X : lens, P : switchoff \cdot X], [RB = \{X\}, RA = \{P\}, RPU = \{X, P\}, RPS = \emptyset] \rangle$$

Clues will first resolve the definite:

$$\langle [l12 : lens, P : switchoff \cdot l12], [RB = \emptyset, RA = \{P\}, RPU = \{l12, P\}, RPS = \emptyset] \rangle$$

Now it is Conduct's turn. Conduct will try to carry out the act which is described by the annotated segment. If this succeeds, then the aforementioned segment will have been replaced by:

$$\langle [l12 : lens, p312 : switchoff \cdot l12], [RB = \emptyset, RA = \emptyset, , RPU = \{l12\}, RPS = \{p312\}] \rangle$$

This means that the system now has the fresh proof *p312* of the proposition: *Lens l12 has been switched off* (the fact that the System did the switching is recorded by the RPS feature). It can now give some feedback to the User (e.g. 'Done') and move the segment to the common background (meaning that both the system and the User are mutually aware of the fact that the system switched off the lens). If the system does not succeed in performing the action, the system will have to initiate a subdialogue. In this subdialogue it will have to explain to the User why the action could not be carried out.

**RI** (Requested Information). Markers which are members of the value of this feature normally come from wh-questions. The feature indicates that a substitution (i.e. a CTT variable from the context) has to be found for the marker such that the segment in which the marker occurs is provable. Subsequently, the substitution has to be communicated to the User.

**Example 8** Let us give an example of a behaviour rule of Conduct which uses the feature RI. Remember that Conduct's rules are of the form

> **If it is the case that** the current situation is of type X
> **Then Do** the actions Y.

The rule which we will describe is a generation rule:[9] it tells the assistant in which situations it has to generate an utterance. The type of these situations is the following: if the User has asked a question Q then the system should compute a response R to this questionand perform the action of uttering R.

[8] Notice that in the semantic representation of the command the covert subject, i.e., the addressee of the command, has been ommited. The speaker and addressee of the command can, however, be retrieved from the information stored in the annotation: in this case, from the value of the feature RPU we can deduce that the command was issued by the user, and therefore, directed at the system.

[9] More rules can be found in Piwek (forthcoming).

```
inspect(AS1)
return-value(RI,AS1,V1)                  wh-response(AS1,AS2)
return-value(RPU,AS1,V2)         ⟹      utter(AS2)
intersection(V1,V2,V3)
empty-set(V3, false)
```

With the service $inspect(-AS)$ Conduct can *read* the annotated segment that is on the top of the pending stack. The services $inspect$, $return-value$, $intersection$ and $empty-set$ are used to check whether the intersection of the values of the features $RI$ and $RPU$ is non-empty, in other words, whether the top element of the pending stack is a wh-question uttered by the User. $wh-response(AS1, AS2)$ means that $AS2$ is a response to $AS1$. It can best be regarded as a ProLog predicate which may be defined as follows:

```
wh-response(A,B) :-
        informational-answer(A,C),
        linguistic-answer(C,B).
```

This rule says that B is a response to A if C is the informational answer to A and B is the linguistic answer corresponding to the informational answer C. For instance, the question *Which lens is on?*, may have the informational answer $l6 : lens$, i.e. the lens which is internally represented by the Assistant with the variable $l6$. The linguistic answer may be something like *The condenser lens*.

If all the tests have been passed, then the Assistant may communicate $AS2$ to the user ($utter(AS2)$). The tests are used to determine the type of the current situation (*cf.* the general form of the behavior rules). This also holds for the test $wh - response(A, B)$. Earlier on we claimed that the Assistant could only inspect the Information Context and the Physical Domain in order to find out the situation type. To account for the use of $wh - response(A, B)$ we need to add a storage containing semantic and pragmatic rules. For instance, rules about the relation between questions and informational answers or between informational and linguistic answers.

### 4.3 Annotations for Contexts

Like segments, contexts can also be annotated. The annotation of a context contains instructions on how to use the information that is part of the context. The services that use contexts are still being developed, and therefore the following examples of features for annotating contexts should be seen a preliminary exploration. All the features which we will discuss have a set of CTT variables as their value, except for the feature **FL**.

**C** and **P** (common and private). Each variable which is part of a context is either a member of the value of C or P. These features divide the context into two parts: the common and the private context (*cf.* section 3.3.5 Modalities on page 12).

**A** (anchored). The value of this feature is a set of variables. Whenever a variable is a member of the value of this feature, this variable is connected to an object from the Physical Domain. This means that the interpretation of the variable has been fixed. The notion of an anchor comes from situation semantics (Barwise & Perry, 1983), but has also been used in Discourse Representation Theory (Kamp, 1990) and DenK (Ahn, 1994).

**FL** (focus list). The value of this feature is a list of CTT variables. This list records when a particular object has been referred to. It is only needed when we use substitution. Whenever a gap is replaced by a variable from the context (with a type inhabiting $*_t$), that variable is appended to the Focus List. If we use definitions, this information can be recorded in the context itself, because whenever a variable (introduced in the discourse) is connected to a variable from the context, a definition containing that variable is appended to the right side of the context.

**RPS** and **RPU** (responsible participant (=) system, responsible participant (=) User). These features can be used to record which participant introduced a particular proof object into the context.

♠

**Acknowledgements** I would like to thank René Ahn and Tijn Borghuis for the constructive discussions we had on the contents of this paper, and Robbert-Jan Beun for his helpful comments on an earlier version of this paper.

## 5 References

Ahn, R. (1994). Communicating Contexts: A Pragmatic Approach to Information Exchange. to be published in *the proceedings of the BRA workshop : Types of Proofs and Programs*, Baastad, Sweden, June 1994.

Ahn, R. (1995). Logical Model of the Electron Microscope. *DenK-report*, *95/15*, SOBU, Tilburg.

Ahn, R. & H-P. Kolb (1990). Discourse Representation meets Constructive Mathematics. In: *Papers from the Second Symposium on Logic and Language* (László Kálmán & László Pólos, eds.), Akademiai Kiado, Budapest.

Ahn, R., R.J. Beun, T. Borghuis, H.C. Bunt, C.W.A.M. van Overveld (1994). The DenK-architecture: a fundamental approach to User-interfaces. *IPO-manuscript*, *1009/III*, Accepted for *Artificial Intelligence Review.*

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

Barwise, J. & J. Perry (1983). *Situations and Attitudes.* MIT Press, Cambridge.

Beun, R-J. & L. Kievit (1995). Resolving definite expressions in DenK. *DenK Report 95/16.*

Borghuis,T. (1995). Coming to Terms with Modal Logic: On the interpretation of modalities in typed $\lambda$-calculus. Dissertation, Eindhoven University.

Curry, H.B. & R. Feys (1958). *Combinatory Logic.* Vol. 1, North-Holland, Amsterdam.

De Bruijn, N.G. (1980). A Survey of the Project Automath. In: *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalisms* (J.R. Seldin & Hindley, J.P., eds.), Academic Press, 589-606.

Kamp, J.A.W. (1981). A Theory of Truth and Semantic Interpretation, In: *Formal Methods in Study of Language* (Groenendijk, J., T. Janssen & M. Stokhof, eds.), Mathematical Centre Tracts 136, Mathematisch Centrum, Amsterdam.

Kamp, J.A.W. (1990). Prolegomena to a Structural Account of Belief and Other Attitudes. In: *Propositional Attitudes: The Role of Content in Logic, Language and Mind* (C.A. Anderson & J. Owens, eds.), CSLI Lecture Notes, *20*, 27–90.

Kievit, L.A. (1994). Defining Pro-Ulf. *DenK-report*, *94/02*, SOBU, Tilburg.

Kievit, L.A. and P.L.A. Piwek (1995). Services for Sherlock. *Denk Report*, *95/13*, SOBU, Tilburg (IPO Rapport no. 1075, Eindhoven).

Piwek, P.L.A. (1994). Issues in Response Generation, *IPO Rapport*, *1012*, Eindhoven.

Piwek, P.L.A. (forthcoming). Dialogue Dynamics (working title).

Ranta, A. (1994). *Type-theoretical grammar.* Clarendon Press, Oxford.

Rentier G. (1995). Questions and Left Dislocation in DenK's HPSG Fragment, *Denk report*, *95/05*, SOBU, Tilburg.

Severi, Paula and Erik Poll (1993). Pure Type Systems with definitions. In: *Logical Foundations of Computer Science'94*, LNCS 813, Springer Verlag.

Stalnaker, R. (1974). Pragmatic Presuppositions. In: *Semantics and Philosophy* (Munitz, Milton K. and Peter K. Unger, eds.), New York University Press, New York.

Sundholm, G. (1986). Proof Theory and Meaning. In: *Handbook of Philosophical Logic, Vol. III* (Gabbay, D. and F. Guenthner, eds.), D. Reidel, Dordrecht, 471-506.

# 6 Appendix: Texts and Contexts by René Ahn