DANDELION

Discourse Functions and Discourse Representation:
An Empirically and Linguistically Motivated,
Interdisciplinarily-Oriented Approach to Natural Language
Texts

ESPRIT Basic Research Project 6665

Deliverable R2.3.1b

A Discourse Grammar for Classifying Demonstrative Nominal Anaphora

Leon Verschuur

November 30, 1995

—— CLS Tilburg/Nijmegen ——

University of Edinburgh GMD-IPSI Darmstadt Universität Saarbrücken

—— Universidad Complutense Madrid ——

Document ID	:	Deliverable R2.3.1b: *A Discourse Grammar for Classifying Demonstrative Nominal Anaphora*
Issue date	:	November 30, 1995
Responsible	:	CLS Tilburg/Nijmegen
Version	:	Final
Status	:	Public
WP number	:	WP2: Grammar integration
Distribution	:	DANDELION Consortium, CEC
Keywords	:	Demonstrative Nominal Anaphora, Dynamic Semantics, Unification Grammars, Definite Clause Grammar

A Discourse Grammar for Classifying Demonstrative Nominal Anaphora

Leon Verschuur

CLS Tilburg/Nijmegen

Abstract:

A formal semantic explanation is given for the fact that superordinate demonstrative noun anaphors can classify their referents, an effect which has been corroborated by experimental investigations (see R1.4.2a). A completed implementation in Definite Clause Grammar is described, while the possibilities for a Typed Feature Structure implementation are indicated.

Contents

1 Introduction

A formal semantic explanation is given for the fact that superordinate demonstrative noun anaphors can *classify* their referents, an effect which has been corroborated by experimental investigations (see [4]). The explanation elaborates the suggestion in R1.4.2a (p.29) that such an anaphor activates co-subordinate class members. It is formalised by letting the demonstrative anaphor introduce such other class members within Muskens' framework of dynamic semantics. Reference to these "shadow" discourse referents is then restricted by the requirement that the referring term holds a subtype relation to the demonstrative term. Also, the classifying use of the demonstrative noun phrase presupposes that it has a proper supertype of the type of its own antecedent. I will give a short overview of the data in section 2, a formal account in section 3, and its implementation by a grammar in section 4. [1]

2 Experimental Data

Experiments reported on in R1.4.2a have corroborated the hypothesis of Maes (see [3]) that superordinate demonstrative noun anaphors classify their referents if they are not needed for reference resolution because the intended reference is already obvious by other means. (They can be used for resolution if there are several possible referents, however.) The Classify Effect consists in a representation of the referent as being an element of the class of individuals denoted by the noun. Thus, a classifying use of demonstrative noun anaphors mentally activates co-subordinate class members, and indeed can be said to *introduce* these members into the discourse. Take an example, similar to one discussed in R1.4.2a:

(1a) Wilhelmina stond ver van het volk.

(1b) Deze koningin/de koningin was formeel.

(1c) Juliana was vriendelijk voor het volk.

(*Wilhelmina was distant from the people. This queen/the queen was formal. Juliana was kind to the people.*)

One experiment (R1.4.2a, Experiment 2) corroborating the Classify Effect yielded that subjects are significantly more inclined to continue sentences like the first two with a third one about a co-subordinate class member if the demonstrative article 'deze' precedes the class denoting noun, than they are if the definite article 'de' precedes it.

[1] Thanks for helpful comments on earlier versions to Jo Calder, Reinhard Muskens, Leo Noordman, Margriet Verlinden, Wietske Vonk.

Another experiment (Experiment 4) showed that subjects read a pair like (1a-b) slower in the demonstrative case than in the definite one, and, subsequently, continuations like (1c) slower in the definite case than in the demonstrative one. A third experiment (Experiment 5), on lexical decision, had the result that nouns for co-subordinate class members were recognised significantly faster if the decision task followed the reading of a sentence with a demonstrative NP, than if it followed the reading of a sentence with a definite one.

The experiments can be interpreted as corroborating the hypothesis that the use of a (not reference resolving) demonstrative introduces the set of co-subordinate class members into the discourse, in the way indefinite noun phrases and proper names do according theories like DRT (see [5]). This hypothesis explains that on the one hand processing pairs like (1a-b) is more complex (and thus takes more time) in the demonstrative case than in the definite one and that subsequently processing a sentence like (1c) is more complex in the definite case than in the demonstrative one. It is explained more specifically using the assumption that a newly used proper name introduces a referent into the discourse if it isn't present yet in the representation thereof, while further occurrences of the same name refer back to that individual. In other words, a proper name presupposes the introduction of an appropriate individual. So according to the hypothesis of the Classify Effect, the presupposition of 'Juliana' in (1c) has been fulfilled by using the demonstrative article in (1b), while it hasn't if the definite article has been used, in which case the processing of 'Juliana' itself has to fulfill the presupposition - and hence has to include introducing the individual in question.

I will concentrate our formal theory and its implementation on texts like (1) involving proper names. Cases of demonstrative anaphora in which the subordination is of a categorical nature rather than of a factual one will not be specifically be addressed in the sections to follow; instead I will point out their relation to the subordination of proper names here. An example would be

(2a) De mus draagt een verenpak.

(2b) Deze/de vogel overwintert in Nederland.

(2c) De zwaluw trekt weg.

(*The sparrow wears a feathersuit. This/the bird hibernates in the Netherlands. The swallow leaves.*)

These cases are more difficult to model because the relation of categorical subordination holds typically between nouns occurring within generic (definite or demonstrative) descriptions. Of course, in general also definite descriptions presuppose that an appropriate individual has been introduced; but also in general, and contrary to the case of proper nouns, a newly used definite description can*not* automatically introduce a referent into the discourse if it isn't present yet; it has to be accomodated, and whether that is possible depends upon factors such as background knowledge. Now with these generically used descriptions the background knowledge requirement seems to be simply: to be known as a natural kind. In other words, if the generic description corresponds to such a known kind, the kind can be introduced into the discourse. If it has been introduced this way already, subsequent occurrences of a coreferring description have their presupposition fulfilled and they can refer back to the introduced kind. So it appears that, except for this additional requirement of denoting a known natural kind, generic descriptions can be represented analogously to proper names after all: the descriptions in (2a) and (2c) introduce natural kinds, and the definite one in (2b) refers back to the introduced kind in (2a).

This doesn't however give the result that definite descriptions are (nearly as) easy to model for us as proper names, for although one can express in the lexicon representations whether or not nouns correspond to natural kinds, one can not so easily derive whether a generic interpretation is indeed present in a definite NP; reference to a specific individual remains a possibility also, of course. But assuming that the two kinds can be distinguished on the basis of other factors, it should be clear that an explanation of the Classify Effect involving proper names can indeed be extended to one involving definite descriptions.

As to the restrictions imposed by agreement features, note that whereas in general the features of coreferring terms are just required to be consistent, the requirement imposed by a subordinating demonstrative NP is stronger:

First, its noun has to be a proper supertype of the noun of its antecedent NP (both nouns being considered as types), like in texts (1a-b) and (2a-b). For instance, in

(3a) Er was eens een koningin die ver van het volk stond.

(3a') Er was eens een vrouw die ver van het volk stond.

(3b) Deze koningin was formeel.

(3c) Juliana daarentegen was vriendelijk voor het volk.

(*There was a queen/woman who was distant from the people. This queen was formal. Juliana on the other hand was kind to the people.*)

where the type of 'deze koningin' is identical to, respectively a subtype of, the type of 'een koningin' and 'een vrouw' instead of being a proper supertype of them, the Classify Effect is absent. Indeed, already the sequences (3a-b) and (3a'-b) are, for diverging reasons, hardly acceptable at all.

Note further that an *analysis* of a text like (1a-b) complies with the requirement *only* if the types assigned to the NPs are indeed isomorphic to their nouns (e.g. the type of 'wilhelmina' could not be *woman*).

Second, the type of a subordinating demonstrative NP has to be also a (not necessarily proper) supertype of the type of a subsequent expression referring to individuals of the introduced set. This is obvious in (1c), where this constraint is fulfilled by the assumption that $juliana$ is a proper subtype of $queen$. Also other kinds of noun phrases can refer to a tacitly introduced individual provided that their type is a subtype of the demonstrative's type:

(4a) Wilhelmina stond ver van het volk.

(4b) Deze koningin was formeel.

(4c) Andere koninginnen waren vriendelijk voor het volk.

(*Wilhelmina was distant from the people. This queen was formal. Other queens were kind to the people.*)

The type of 'andere koninginnen' in (4c') is however a subtype of (namely identical to) the type of 'deze koningin', so reference to individuals in the introduced set is possible. If on the other hand the continuation would be

(4c') De koning was vriendelijk voor het volk.

(*The king was kind to the people.*)

the type restriction is violated, no reference to a "shadow individual" is possible, and the text is less continuous: the referent of 'de koning' hasn't been activated yet (unless it had been introduced before (4a) already of course).

Summarising, a discourse model accounting for the Classify Effect of demonstrative NPs (in connection to proper names) should include the following properties:

1. a general dynamics accounting for the very possibility of introducing and referring to individuals as well of sets of them;
2. a semantics of proper names expressing that they, if new, introduce an individual, and, if already having occurred, refer to that individual;
3. a semantics of the demonstrative article 'deze'/'dit' expressing that it refers to an introduced individual, and moreover: that it introduces the set of the *other* individuals belonging to the class expressed by its noun;
4. by contrast, a semantics of the definite article 'de'/'het' expressing that it refers to an introduced individual, and no more;
5. a type system taking care of the agreement constraints on coindexed terms in general, as well as of the subordination relations between the type of a demonstrative NP and the expressions coindexed with it.

3 Representation in a Discourse Model

A discourse representation formalism particularly well-suited for connection to a grammar is Muskens Logic of Change ([6]). It makes it possible to construct discourse representations by functional application of λ-terms (representing lexical meanings), to other λ-terms, just like in standard Montague Grammar. The result is for a sentence or text always a proposition, which in this formalism amounts to a relation between *states*. The concept of a state derives from dynamic logic, where the semantics of programming languages was given in terms of transitions of machine states. One could in our case of natural language semantics also think of it as the mental state the hearer is in. Essential is that the values of the variables may vary depending on the state. The general idea of dynamic logic was that a machine state before execution of a (piece of) program may differ from its state after execution in that the values of certain variables have changed, and to take that change as the meaning of the program. The idea of a dynamic natural language semantics, first developed by Groenendijk and Stokhof, ([2]) is that a state before interpretation of a sentence or text may differ from the state after interpretation in that certain variables refer to different individuals than they did earlier, and to take the change as the meaning of the sentence or text. So this dynamic semantics makes it in the first place possible to express which (if any) individuals have been introduced by a text and to which a reference is possible; in addition the ordinary, static meaning of a text is derivable by the fact that the dynamic proposition also expresses which properties all mentioned individuals have been assigned.

The role of these dynamic variables is played in the Logic of Change by so-called *stores* $v_1, v_2, ..$; their values depend on states, so they are function constants mapping a state $A, B, ..$ onto an individual. States themselves are considered to be primitive objects. Stores are notated v_n, the index n corresponding to the index associated to a noun phrase. A third concept in the Logic of Change is a notion of maximal difference between states A and B, notated originally as $A[v_i]B$ but here as $diff(A)(v_i)(B)$. This means that state A differs from state B possibly in store v_i, whereas all other stores have in B the same value as they had in A.

States, stores, and the standard logical apparatus of the connectives and quantifiers are added to the language of typed λ-calculus. In this language all these concepts are viewed as terms to which types are assigned: atomic types, or complex types constructed out of atomic types by the function type constructor '$\rightarrow$' ('$a \rightarrow b$' meaning "the set of functions from objects of type a to objects of type b"). A term with a functional type $a \rightarrow b$ can be applied to an argument of type a to yield a complex term with a type b. The type theory is called *two-sorted* because there are, besides the truthvalue type t, two atomic types for two kinds of primitive objects: objects of type e, the individuals, and of type s, the states. Stores have the type $s \rightarrow e$. All other type assignments are isomorphic to those standardly given in Montague Grammar: the truthvalue type t in the standard type assignments is consequently replaced by

$s \rightarrow (s \rightarrow t)$. For instance a verb, say 'walk', of the former type $e \rightarrow t$ now gets the type $e \rightarrow (s \rightarrow (s \rightarrow t))$. The latter is possible because 'walk' no longer translates into a simple predicate $walk$, but - $walk$ still being of type $e \rightarrow t$, and x of type e - as $\lambda x.\lambda A.\lambda B.walk(x) \wedge A = B$. This implies that the verb itself has no dynamic effect; if applied to a constant c for instance the result would be

$$\lambda A.\lambda B.walk(c) \wedge A = B$$

that is, the identity relation between states A and B, such that c walks. The output state of the proposition remains the same, and there is only the condition that the constant c (denoting independently of state) denotes a walking individual. It is indefinite terms and proper names which actually change states. Thus, the representation

A$_1$ man walks. He$_1$ whistles.

to take an indefinite example, would be

$$\lambda A\lambda B.diff(A)(v_1)(B) \wedge man(v_1(B)) \wedge walk(v_1(B)) \wedge whistle(v_1(B))$$

that is, the relation between A and B, such that B differs from A at most in store v_1, and this store in B denotes a man who walks and whistles. Representations like this are obtained by β-reduction and a predicate logic simplification of a λ-term built by functional applications by a grammar (see for all this the next section).

The basic apparatus of the Logic of Change needs obviously to be extended if we want to express that demonstrative noun phrases can introduce *sets* of individuals rather than just individuals. Therefore we add a second kind of store, called superstores, to the inventory, notated as $\mathcal{S}_1$, $\mathcal{S}_2$,... These are function constants which, applied to a state, yield a set of stores (that is, yield a truthvalue if applied to a store). Accordingly their type is $s \rightarrow ((s \rightarrow e) \rightarrow t)$. Just as the $diff$ predicate can express a maximal difference between two states in some store, it can now also express a maximal difference between two states in some superstore, that is a set of stores; of course such a set needs in particular representations to be well-restricted. The idea is that

(1b') Deze$_1$ koningin is formeel.

would be represented as the following - given that the NP here does not only refer back to a queen introduced earlier, but also introduces the set of other queens (fo abbreviates $formal$):

$$\lambda A\,\lambda B.\,queen(v_1(A)) \wedge fo(v_1(A)) \wedge diff(A)(\mathcal{S}_1)(B) \wedge \forall v_n((\mathcal{S}_1(B)(v_n)) \leftrightarrow (queen(v_n(B)) \wedge n > 1))$$

Here the superstore $\mathcal{S}_1$ denotes in B all stores denoting a queen and the index of which is greater than 1, and no other stores. So the proposition is a relation between a state

A in which v_1 refers to a particular queen, and a state B differing from A at most in all stores denoting the other queens. Note that $v_1(B)$ still refers to the same queen as $v_1(A)$ did, due to the fact that $diff$ stands for maximal difference. But if further text would for instance add a term $v_2(B)$ to the representation, then this individual, given that its type would be a subtype of the type *queen*, will not have been introduced explicitly (otherwise it would have been $v_2(C)$ for some C other than A, B actually), and necessarily becomes a member of the tacitly introduced set of queens $\mathcal{S}_1(B)$. If the type of such a term $v_2(B)$ is *not* a subtype of *queen*, then it is *not* a member thereof and it refers to an individual having been introduced earlier in the discourse. I will demonstrate the various possibilities in section 4.

Let us now turn to a type system capable to express the supertype requirements of the classifying NPs. To begin with, I will in general adopt the general unification grammar approach that agreement constraints are fulfilled if the agreement features of coindexed individuals are unifiable (see for instance [8]); that is, if there is a most general value which is more specific than (or identical to) the values of the features in question. As in typed feature structure formalisms, I frame these values as types. They will for the present purposes be isomorphic to the orthography of the expressions they are the agreement value of; one sees in connection to example (1a-b) that a type on a less specific level than that of the term (e.g. *woman* for 'wilhelmina') can become to coarse in the special case of demonstratives - this wouldn't be a subtype of *queen*.

Such a type system computing unifications of types of individuals cannot be embedded into the standard deduction rules of type assignment for the typed λ-calculus: this is possible only for parses of simple sentences. The point is that as soon as several simple propositions, themselves typed to $s \rightarrow (s \rightarrow t)$ by an appropriate type system for λ-terms, are linked by the dynamic conjunction, the result can be typed (to the same type) by that system also - no matter how the types of the coindexed terms for individuals *within* the separate propositions diverge. Therefore no reasoning with arrow types is desired here; instead I have chosen to collect sets of type assignments on particular individuals, and to check whether they are consistent.

This can be considered as being analogous to an implementation in a typed feature structure formalism including a notion of reentrancy. The latter would in connection to this problem obviously have its advantages, as was shown in [9]: our dynamic λ-terms can occur as complex AVMs, and the agreement values as types: they can, in such a formalism (such as ALE), be checked on unifiability no matter how deeply embedded in the structure they are. As said, also the super- or subtype relation is needed for our purposes; although I haven't made an implementation in ALE yet, that is likely to be possible as well: ALE allows expressing explicitly required subtype relations within the definite clause attachments to grammar rules (since the compilation of the subtype relation precedes the compilation of the grammar, see [1], p.56). Now in the present DCG-implementation not only the required subtype relations, but also the required unifiability relations themselves are attached as definite clauses: clauses checking whether the sets of type assignments on individuals are consistent.

This check is performed on the basis of the general subtype relation, which is in turn the transitive and reflexive closure of the declared immediate subtype relations. Consistency of a set of "normal" agreement types is computed by finding their unifier. However, agreement types of classifying demonstratives are required to be a proper supertype of their antecedent's type, and a supertype of the subsequent terms they classify. The latter can be ensured due to my assumption that the type of "shadow individuals" is added to the Prolog database (see section 4); but for the former, the types of demonstratives have to be marked, in order that they, being collected in the total set of type assignments, express this special requirement. For this purpose I propose to close the set of basic types under a new type constructor $classifier$, so that if t is a type, $classifier(t)$ is a type as well; consistency of a set of types including a $classifier(t)$ is computed by checking whether t is a proper supertype of all other assigned types. The model-theoretic interpretation of $classifier(t)$ is identical to that of t; thus the constructor merely expresses how t must be related to other types of the same individual, which is admittedly a deviating use for a type. How all this works out we will see in more detail in the next section.

4 Grammar and Implementation

The Prolog program PsychoGram, delivered as Dandelion P2.3.1, can analyse simple texts of arbitrary length which exemplify NP-anaphora, and more in particular the Classify Effect yielded under certain conditions by demonstrative nominal anaphors. The program consist in a DCG grammar of conditioned rewrite rules, β-reduction and predicate logical reduction of semantic representations, a type system checking the consistency of the representations, and a module transcribing the result into various LaTeX formats and previewing them on the screen. Compared to standard implementations of Montague Grammar (see [7]), the essential extensions are that the λ-terms also express state transitions; that there is a further (predicate logical) reduction needed to simplify the β-reduced terms; and that there is a type system checking consistency of coindexed terms, as well as expressing the conditions of the Classify Effect.

After loading the .pl-file under Sicstus Prolog, texts with indexed noun phrases can be offered as follows:

```
?- text([een(1),man,wandelt,'.',hij(1),fluit], Semantics, Types).
```

for *A man walks. He whistles.* (Instead of the period one can also use the conjunctions 'en' and 'maar'.) The definition of `text` is

```
text(Text,Sem,Type) :- abolish(shadow/1), asserta(shadow(null)),
                 s(Sem,Type,[0,_],Text,[]),
```

```
typeinfo(Type,Bool),
maketex(Text,Sem,Type,Bool).
```

The use of `shadow` is that of keeping score of the type assigned to the set of "shadow" individuals tacitly introduced by a classifying demonstrative. The `null` type expresses, as a starting point of each text, that no current shadow individuals exist. This way of implementing, with `shadow` as a dynamic predicate, has the disadvantage that it can yield only one Classify Effect per text; on the other hand, allowing more than one would assume rather long input texts, as well as a much complexer program: it would involve passing a "shadow type" from constituent to constituent. Apart from this complexity issue, there is the as yet unsolved empirical question making explicit the conditions under which a particular Classify Effect disappears again).

The `s` predicate tests whether the list `Text` can be parsed as a complex sentence with a semantics substituted for `Sem` and a type list substituted for `Types`. Its third argument is a pair of indices (old, which is for a new text `0`, and new) which will be used for deciding whether a proper name is introducing or referring back. Then `typeinfo` calls a `typecheck` and, if the latter succeeds, gets a positive second `Bool` argument; if it fails, a negative one. The boolean value is used by the next predicate to decide on whether the list of types is to presented as a consistent one or not: `maketex` produces a LaTeX file, runs LaTeX on it and starts the previewer (according to the program xdvi but this may have to be adapted of course) showing the .dvi-file. As a result one sees the list of the original text input; a reduced and formatted λ-term representing the discourse; a second, pretty printed format which incrementally presents the successive state transitions and in which stores they differ; and the list of types of discourse referents together with the boolean result of the typecheck. For instance, the query given yields the following on the xdvi-previewer:

[een(1),man,wandelt,en,hij(1),fluit]

$\lambda A\ \lambda B\ diff(A)(v_1)(B) \wedge man(v_1(B)) \wedge walk(v_1(B)) \wedge whistle(v_1(B))$

A
$> v_1$
$B \quad man(v_1(B)) \quad walk(v_1(B)) \quad whistle(v_1(B))$

[v(1):man,v(1):male] $\Longrightarrow$ This list of types is consistent.

As a more complex example, consider the text:

(5) Een(1) man wandelt en een(2) meisje wandelt. De(1) man begeleidt haar(2).

Hij(1) koopt een(3) ijsje en zij(2) eet het(3).

(A man walks and a girl walks. The man accompanies her. He buys an icecream and she eats it.)

Then this is the output of a corresponding Prolog query:

[een(1),man,wandelt,en,een(2),meisje,wandelt,.,de(1),man,begeleidt,haar(2),., hij(1),koopt,een(3),ijsje,en,zij(2),eet,het(3)]

$\lambda A\, \lambda B\, \exists C(diff(A)(v_1)(C) \wedge man(v_1(C)) \wedge walk(v_1(C)) \wedge \exists D(diff(C)(v_2)(D) \wedge girl(v_2(D)) \wedge walk(v_2(D)) \wedge man(v_1(D)) \wedge accompany(v_2(D))(v_1(D)) \wedge diff(D)(v_3)(B) \wedge icecream(v_3(B)) \wedge buy(v_3(B))(v_1(D)) \wedge eat(v_3(B))(v_2(B))))$

A
$> v_1$
$C \quad man(v_1(C)) \quad walk(v_1(C))$
$> v_2$
$D \quad girl(v_2(D)) \quad walk(v_2(D)) \quad man(v_1(D)) \quad accompany(v_2(D))(v_1(D))$
$> v_3$
$B \quad icecream(v_3(B)) \quad buy(v_3(B))(v_1(D)) \quad eat(v_3(B))(v_2(B))$

[v(1):man,v(2):girl,v(1):man,v(2):female,v(1):male,v(3):icecream,v(2):female,v(3):neuter]
$\Longrightarrow$ This list of types is consistent.

The pretty-printed format is equivalent to the linear formula if one reads the first and the last state variables as expressing the two λ-abstractions with which the formula begins, all intermediate states as existential quantifications following the λ-abstractions, and the $> X$ symbols as maximal differences in X.

In addition to this .dvi-output, the program of course returns the substitutions of `Sem` and `Types` in Prolog format in the Sicstus window.

I will give by way of example most grammar rules needed for text (1) in a moment. First, a general remark on the order of the two reductions of the semantic representations, β-reduction and certain predicate logical simplifications.

β-reduction is the reflexive and transitive closure of the relation

$$(A) \quad (\lambda v.\alpha)(\beta) \longrightarrow \alpha[v := \beta]$$

the right hand expressing "α with β substituted for all occurrences of v". This was implemented using the Pereira and Shieber implementation (see [7]) of the basic case of application which reads

```
beta_reduce((X^T)@X,T).
```

(In general, abstraction, application, conjunction, and equivalence are expressed by the Prolog operators

```
 ^ @ * <=>
```

respectively.) Now to see what the predicate logical reduction is needed for, take for instance the Logic of Change representations of 'een' and of 'man':

$$\lambda P\lambda Q\lambda A\lambda B.\exists C(\exists D(diff(A)(v_n)(C) \wedge P(v_n(C))(C)(D) \wedge Q(v_n(C))(D)(B)$$

$$\lambda x\lambda A\lambda B.man(x) \wedge A = B$$

The β-reduction of the functional application of the one to the other is

$$\lambda Q\lambda A\lambda B.\exists C(\exists D(diff(A)(v_n)(C) \wedge man(v_n(C)) \wedge C = D \wedge Q(v_n(C))(D)(B)$$

If this in turn is applied to the representation for 'wandelt' (the one given earlier for 'walk', which is analogous to the one for 'man'), we get

$$\lambda A\lambda B.\exists C(\exists D(diff(A)(v_n)(C) \wedge man(v_n(C)) \wedge C = D \wedge walk(v_n(C)) \wedge D = B$$

This is equivalent to the much more perspicuous

$$\lambda A\ \lambda B.\ diff(A)(v_1)(B) \wedge man(v_1(B)) \wedge walk(v_1(B))$$

The equivalence holds by the validity of

$$(B)\quad (\exists x\exists y(\phi \wedge (x = y) \wedge \psi)) = (\exists x(\phi[y := x] \wedge \psi[y := x])$$

and of

$$(C)\quad (\lambda x\exists y(\phi \wedge (x = y) \wedge \psi)) = (\lambda x.(\phi[y := x] \wedge \psi[y := x])$$

which can be used in reduction rules as reductions from the left side to right side (the proof of *(C)* is in fact a proof of identity of sets). I have implemented them by making use of Prolog unification of the variables. This leads us back to the grammar rules - take for instance the entries of the three words of our little sentence:

```
det(P^Q^I^J^
     (exists(K,(exists(H,(diff@I@v(N)@K)*(P@(v(N)@K)@K@H)*(Q@(v(N)@K)@H@J))))),
    [v(N):_],
    [_,N])
--> [een(N)].

noun(X^I^J^(man@X*(I=J)),
     [v(_):man])
--> [man],
```

```
    {I=J}.

iv(X^I^J^(walk@X*(I=J)),
    [],
    [C,C])
--> [wandelt],
    {I=J}.
```

Both the noun and the verb assume a call to `I=J`, which means that Prolog factually unifies these variables within the semantic representation. This leads initially to representations with sequences of binders (abstraction and quantification) of the same variables; so the implementation of predicate logical reduction can now consist in removing all variable equivalences, as well as all quantifications over A following an earlier quantification or abstraction over A. The reason why this is always permitted is that identity of bindings can in this grammar really *only* be caused by explicit unification of two formerly distinct variables.

This simple way of implementing predicate logical reductions has the consequence that the phrasal rules become slightly more complex; for the explicit unification of variables only gives results which are guaranteed to be sound if *is combined with* the clauses removing the iterated bindings. In other words, the more modular implementation of first composing a representation and afterwards β-reducing and PL-reducing it is not possible: PL-reduction is necessary before each new functional application step. For instance the NP-rule runs:

```
np(Snp,
   [v(N):T],
   [_,N])
--> det(Sdet,
        [v(N):_],
        [_,N]),
    noun(Snoun,
         [v(_):T]),
    {beta_reduce((Sdet)@Snoun,Snpi),
     pl_reduce(Snpi,Snp)}.
```

So the NP semantics is already fully reduced before it will be (by an `s`-rule) applied to a VP semantics. This is the general idea behind all phrasal rules.

Now let us first make some further remarks about the four rules presented yet. Note that the `noun` entry lacks a third argument for a pair of indices, because nouns themselves can not change the index; this is done by the determiner. The `iv` category in general *can* change the index (namely in case of complex IV's consisting of a transitive verb and a NP), hence this argument is also present in the atomic entry for 'walk' - although being occupied by the identity pair. A similar remark can be made as to the presence in the `iv` rule of the empty list of type assignments to the individuals: it is non-empty for the complex `iv` cases (see below). With respect to this second argument, the type assignments, note that by the `np` rule the index `N` of the individual in question is inherited from the determiner, and its type from the noun.

Here are some other phrasal rules, needed for for instance text (1):

```
s(Ss,I,[C1,C3])
--> s1(S1s,I1,[C1,C2]),
    co(Cos),
    s(S2s,I2,[C2,C3]),
    {beta_reduce(Cos@S1s,Red1),
     beta_reduce(Red1@S2s,Sstmp),
     pl_reduce(Sstmp,Ss),
     append(I1,I2,I)}.

s(Ss,I,[C1,C3])
--> np(Snp,I1,[C1,C2]),
    iv(Sverb,I2,[C2,C3]),
    {beta_reduce(Snp@Sverb,Sstmp),
     pl_reduce(Sstmp,Ss),
     append(I1,I2,I)}.

iv(Siv,I,[C2,C3])
--> tv(Stv,[]),
    np(Snp,I,[C2,C3]),
    {beta_reduce(Stv@Snp,Sivtmp),
     pl_reduce(Sivtmp,Siv)}.
```

The rule for `s1` is identical to the second rule for `s`. They are for parsing simple sentences, calling possibly the complex `iv` rule given also; in addition to building simplifying semantic represetations they collect all lists of type assignments into a single list, and also keep score of the current index. The first rule is for parsing texts as conjoined sentences. It calls the following lexical rule for conjunction:

```
co(A^B^I^J^(exists(H,(A@I@H)*(B@H@J))))
--> [.].
```

(variants rewriting to e.g. 'en' are possible also, of course).

We now come to the "heart" of the explanation of the Classify Effect. First, recall that the grammar assumes that the user provides all NPs used in his text explicitly with increasing integers as indices (on the determiner or on the proper name). The third argument keeps score of the current index, and is used to decide whether a proper name must be analysed as introducing an individual - in case its index is higher than the current one; or whether it refers back to an individual - in case its index is not higher than the current one, *or* in case it can refer to a shadow individual, which is not indexed, but which must have a supertype of the type of the name. So this leads to the following triple entry for a proper name (reduction to less entries with the help of disjunction are probably possible):

```
np(P^I^J^(((v(N)@I)=wilhelmina)*P@(v(N)@I)@I@J),
   [v(N):wilhelmina],
   [_,N])
--> [wilhelmina(N)],
    {\+shadow(null),
     shadow(Type),
     subtype(wilhelmina,Type),!}.

np(P^I^J^(((v(N)@I)=wilhelmina)*P@(v(N)@I)@I@J),
   [v(N):wilhelmina],
   [C1,N])
--> [wilhelmina(N)],
    {N=<C1,!}.

np(P^I^J^(exists(K,(diff@I@v(N)@K)*(v(N)@K)=wilhelmina)*P@(v(N)@K)@K@J),
   [v(N):wilhelmina],
   [_,N])
--> [wilhelmina(N)].
```

Note that in the first case the extra conditions are that there are shadow individuals of a type `Type` and that the name's type is a subtype thereof. In the second case the condition is that the name's index is less or equal to the current one. The semantics of both rules expresses the reference to a formerly introduced individual. The third rule then, only being applicable if the other two fail, has a semantics expressing the introducing use.

Here are the rules for three determiners, indefinite, definite, and demonstrative respectively:

```
det(P^Q^I^J^
     (exists(K,(exists(H,(diff@I@v(N)@K)*(P@(v(N)@K)@K@H)*(Q@(v(N)@K)@H@J))))),
    [v(N):_],
    [_,N])
--> [een(N)].

det(P^Q^I^J^(exists(K,(P@(v(N)@K)@I@K)*(Q@(v(N)@K)@K@J))),
    [v(N):_],
    [_,N])
--> [de(N)].
```

```
det([P,S],
    P^Q^I^J^(exists(K,exists(L,exists(M,(P@(v(N)@K)@I@K)*
     (Q@(v(N)@K)@K@L)*(diff@L@s(N)@M)*
     (forall(v_n,(((s(N)@M@v_n)<=>((S@(v_n@M)@M@J)*(test(N))))))))))),
    [v(N):_],
    [_,N])
--> [deze(N)].
```

The semantics of the first introduces an individual and the second refers back to one; both allow the possibility that the possibly complex predicates P and Q, corresponding to a noun and a verb, give rise to new, different states on the basis of their containing further indefinite terms.

The third rule is somewhat more complex due to a specific programming issue: that fact that `beta-reduction` is currently not complete for cases in which an abstracted variable of a high type binds occurrences on several sides of a connective. This is the case here for P; so I have replaced its last occurrence by S, and added a first argument $[P, S]$ meant to express that these are alphabetic variants of each other. See for its use the `np` rule below. The semantics expresses the introduction of a set of shadow individuals described earlier; `test(N)` abbreviates `n>N`. A separate `np` rule for demonstrative NPs reads:

```
np(Snp,
   [v(N):classifier(T)],
   [_,N])
--> det([Snoun,S],
        Sdet,
        [v(N):_],
        [_,N]),
    noun(Snoun,
         [v(_):T]),
    {copy_term(Snoun,S),
     beta_reduce((Sdet)@Snoun,Snptmp),
     pl_reduce(Snptmp,Snp),
     abolish(shadow/1),asserta(shadow(T))}.
```

One sees that the new extra argument of `det` is indeed conditioned to contain two alphabetic variants by the `copy-term` condition. (Attempts to use this system predicate into the `beta-reduction` definition itself haven't been succesful for some reason.) The type assignment guarantees that the type originating from the noun is a proper supertype of the type of the antecedent(s) of the demonstrative phrase. The two last conditions (within curly brackets) establish the type of the individuals tacitly introduced by the phrase.

5 Results and Conclusion

I can now present the formatted output of a little demo accounting for the Classify Effect in text (1). The Effect becomes obvious if this text is divided into several subtexts, yielding the following representations:

1.[wilhelmina(1),stond-ver-van,het(2),volk,.,de(1),koningin,was-formeel]

$\lambda A\ \lambda B\ diff(A)(v_1)(B) \wedge v_1(B) = wilhelmina \wedge people(v_2(B)) \wedge distantfrom(v_2(B))(v_1(B)) \wedge queen(v_1(B)) \wedge formal(v_1(B))$

A
$> v_1$
$B \quad v_1(B) = wilhelmina \quad people(v_2(B)) \quad distantfrom(v_2(B))(v_1(B)) \quad queen(v_1(B)) \quad formal(v_1(B))$

[v(1):wilhelmina,v(2):people,v(1):queen] $\Longrightarrow$ This list of types is consistent.

2.[wilhelmina(1),stond-ver-van,het(2),volk,.,deze(1),koningin,was-formeel]

$\lambda A\ \lambda B\ \exists C(diff(A)(v_1)(C) \wedge v_1(C) = wilhelmina \wedge people(v_2(C)) \wedge distantfrom(v_2(C))(v_1(C)) \wedge queen(v_1(C)) \wedge formal(v_1(C)) \wedge diff(C)(\mathcal{S}_1)(B) \wedge \forall v_n((\mathcal{S}_1(B)(v_n)) \leftrightarrow (queen(v_n(B)) \wedge n > 1)))$

A
$> v_1$
$C \quad v_1(C) = wilhelmina \quad people(v_2(C)) \quad distantfrom(v_2(C))(v_1(C)) \quad queen(v_1(C)) \quad formal(v_1(C))$
$> \mathcal{S}_1$
$B \quad \forall v_n((\mathcal{S}_1(B)(v_n)) \leftrightarrow (queen(v_n(B)) \quad \wedge n > 1))$

[v(1):wilhelmina,v(2):people,v(1):classifier(queen)] $\Longrightarrow$ This list of types is consistent.

3.[wilhelmina(1),stond-ver-van,het(2),volk,.,de(1),koningin,was-formeel,.,juliana(3),was-vriendelijk-voor,het(2),volk]

$\lambda A\ \lambda B\ \exists C(diff(A)(v_1)(C) \wedge v_1(C) = wilhelmina \wedge people(v_2(C)) \wedge distantfrom(v_2(C))(v_1(C)) \wedge queen(v_1(C)) \wedge formal(v_1(C)) \wedge diff(C)(v_3)(B) \wedge v_3(B) = juliana \wedge people(v_2(B)) \wedge kindto(v_2(B))(v_3(B)))$

A
$> v_1$
$C \quad v_1(C) = wilhelmina \quad people(v_2(C)) \quad distantfrom(v_2(C))(v_1(C)) \quad queen(v_1(C)) \quad formal(v_1(C))$
$> v_3$
$B \quad v_3(B) = juliana \quad people(v_2(B)) \quad kindto(v_2(B))(v_3(B))$

[v(1):wilhelmina,v(2):people,v(1):queen,v(3):juliana,v(2):people] $\Longrightarrow$ This list of types is consistent.

4.[wilhelmina(1),stond-ver-van,het(2),volk,.,deze(1),koningin,was-formeel,.,juliana(3),was-vriendelijk-voor,het(2),volk]

$\lambda A\ \lambda B\ \exists C(diff(A)(v_1)(C) \wedge v_1(C) = wilhelmina \wedge people(v_2(C)) \wedge distantfrom(v_2(C))(v_1(C)) \wedge queen(v_1(C)) \wedge formal(v_1(C)) \wedge diff(C)(\mathcal{S}_1)(B) \wedge \forall v_n((\mathcal{S}_1(B)(v_n)) \leftrightarrow (queen(v_n(B)) \wedge n > 1)) \wedge v_3(B) = juliana \wedge people(v_2(B)) \wedge kindto(v_2(B))(v_3(B)))$

A
$> v_1$
$C \quad v_1(C) = wilhelmina \quad people(v_2(C)) \quad distantfrom(v_2(C))(v_1(C)) \quad queen(v_1(C)) \quad formal(v_1(C))$
$> \mathcal{S}_1$
$B \quad \forall v_n((\mathcal{S}_1(B)(v_n)) \leftrightarrow (queen(v_n(B)) \quad \wedge n > 1)) \quad v_3(B) = jul \quad people(v_2(B)) \quad kindto(v_2(B))(v_3(B))$

[v(1):wilhelmina,v(2):people,v(1):classifier(queen),v(3):juliana,v(2):people] $\Longrightarrow$ This list of types is consistent.

The representation of 2. is more complex than the representation of 1. and thus explains why 2. takes more time to read than 1. In contrast, what needs to be added to 1. in 3. (which involves a switch to a new state, differing in the store for 'juliana') is more complex than what needs to be added to 2. in 4.; the representation of 2. already includes the introduction of all other queens, so no new state for 'juliana' is necessary and the new proper name can simply refer to one of those queens, instead of introducing one itself. This explains why reading the last sentence of 4. takes less time than reading the last sentence of 3. Also, these representations can be considered as accounting for the fact that subjects easier continue a text like 2. with a sentence about a co-subordinate class-member than a text like 1.; for the representation of 2. expresses, unlike that of 1., that these other class members have indeed been introduced into the model.

The last two examples, on the following page, show that a text like 4. may be continued by other proper names referring back to a shadow individual, as long as they have a subtype of the demonstrative, like 'beatrix' in example 5.; once they have a type not being such a subtype, they are analysed as being an introducing term however, like 'bernhard' (the prince consort of Queen Juliana) in example 6.

5.[wilhelmina(1),stond-ver-van,het(2),volk,.,deze(1),koningin,was-formeel,.,juliana(3),was-vriendelijk-voor,het(2),volk,.,beatrix(4),lijkt-op,haar(3)]

$\lambda A\ \lambda B\ \exists C(diff(A)(v_1)(C) \wedge v_1(C) = wilhelmina \wedge people(v_2(C)) \wedge distantfrom(v_2(C))(v_1(C)) \wedge queen(v_1(C)) \wedge formal(v_1(C)) \wedge diff(C)(\mathcal{S}_1)(B) \wedge \forall v_n((\mathcal{S}_1(B)(v_n)) \leftrightarrow (queen(v_n(B)) \wedge n > 1)) \wedge v_3(B) = juliana \wedge people(v_2(B)) \wedge kindto(v_2(B))(v_3(B)) \wedge v_4(B) = beatrix \wedge resemble(v_3(B))(v_4(B)))$

A
$> v_1$
$C \quad v_1(C) = wilhelmina \quad people(v_2(C)) \quad distantfrom(v_2(C))(v_1(C)) \quad queen(v_1(C)) \quad formal(v_1(C))$
$> \mathcal{S}_1$
$B \quad \forall v_n((\mathcal{S}_1(B)(v_n)) \leftrightarrow (queen(v_n(B)) \quad \wedge n > 1)) \quad v_3(B) = jul \quad people(v_2(B)) \quad kindto(v_2(B))(v_3(B))$
$v_4(B) = beatrix \quad resemble(v_3(B))(v_4(B))$

[v(1):wilhelmina,v(2):people,v(1):classifier(queen),v(3):juliana,v(2):people,v(4):beatrix,v(3):female]
$\Longrightarrow$ This list of types is consistent.

6.[wilhelmina(1),stond-ver-van,het(2),volk,.,deze(1),koningin,was-formeel,.,juliana(3),was-vriendelijk-voor,het(2),volk,.,bernhard(4),was-ook-vriendelijk-voor,het(2),volk]

$\lambda A\ \lambda B\ \exists C(diff(A)(v_1)(C) \wedge v_1(C) = wilhelmina \wedge people(v_2(C)) \wedge distantfrom(v_2(C))(v_1(C)) \wedge \exists D(queen(v_1(C)) \wedge formal(v_1(C)) \wedge diff(C)(\mathcal{S}_1)(D) \wedge \forall v_n((\mathcal{S}_1(D)(v_n)) \leftrightarrow (queen(v_n(D)) \wedge n > 1)) \wedge v_3(D) = juliana \wedge people(v_2(D)) \wedge kindto(v_2(D))(v_3(D)) \wedge diff(D)(v_4)(B) \wedge v_4(B) = bernhard \wedge kindto(v_2(D))(v_4(B))))$

A
$> v_1$
$C \quad v_1(C) = wilhelmina \quad people(v_2(C)) \quad distantfrom(v_2(C))(v_1(C)) \quad queen(v_1(C)) \quad formal(v_1(C))$
$> \mathcal{S}_1$
$D \quad \forall v_n((\mathcal{S}_1(D)(v_n)) \leftrightarrow (queen(v_n(D)) \quad \wedge n > 1)) \quad v_3(D) = jul \quad people(v_2(D)) \quad kindto(v_2(D))(v_3(D))$
$> v_4$
$B \quad v_4(B) = bernhard \quad kindto(v_2(D))(v_4(B))$

[v(1):wilhelmina,v(2):people,v(1):classifier(queen),v(3):juliana,v(2):people,v(4):bernhard] $\Longrightarrow$
This list of types is consistent.

References

[1] Carpenter, B., and G. Penn: 'ALE: The Attribute Logic Engine, User's Guide', Carnegie Mellon University 1994.

[2] Groenendijk, J, and M. Stokhof: 'Dynamic Predicate Logic', in *Linguistics and Philosophy* 14 (1991): 39-100.

[3] Hustinx, L., F. Maes, C. Schouten, W. Vonk: 'Discourse Functions of NP-anaphora; Linguistic and text-analytic form/function analysis of NP-anaphora'. Dandelion Deliverable R1.4.1, C.E.C.

[4] Hustinx, L., W. Vonk: 'Discourse Functions of NP-anaphora; Psycholinguistic experimentation on the qualificational function of demonstrative noun anaphora'. Dandelion Deliverable R1.4.2a, C.E.C.

[5] Kamp, H.: 'A Theory of Truth and Semantic Representation', in J. Groenendijk, Th. Janssen, M. Stokhof (eds.): *Formal Methods in the Study of Language, Part I*, Math. Center, Amsterdam, 227-322.

[6] Muskens, R.: Anaphora and the Logic of Change. In J. van Eijck (ed.) *Logics in AI, Proceedings of JELIA '90, Lecture Notes in AI nr.478*, Springer-Verlag, Berlin.

[7] Pereira, F. and S. Shieber: *Prolog and Natural-Language Analysis*, Center for the Study of Language and Information; Lecture Notes Number 10, Chicago University Press, 1987.

[8] Carl Pollard and Ivan A. Sag: *Head-driven Phrase Structure Grammar*, Center for the Study of Language and Information, Chicago University Press, 1994.

[9] Verschuur, L.: Dynamic Semantics in HPSG. In *Proceedings of the International Workshop on Computational Semantics (IWCS)*, Tilburg, December 1994. Tilburg: ITK.