

# Replicating Quantified Noun Phrases in Database Semantics

Roland Hausser

Universität Erlangen-Nürnberg  
Abteilung Computerlinguistik (CLUE)  
rrh@linguistik.uni-erlangen.de

## Abstract

Predicate calculus treats determiner-noun sequences like *the man*, *every man*, or *several men* as ‘quantified noun phrases.’ This analysis in terms of quantifiers, variables, and connectives creates a major structural difference compared to the handling of proper names. The modeling of natural language communication in database semantic (DBS), in contrast, treats the functor-argument structure as primary, regardless of whether an argument is of the sign type symbol (determiner-noun sequence), name, or indexical (pronoun). In this paper, the meanings carried by different determiners are reanalyzed as controlling the internal matching between nominal symbols and individuals, or sets of individuals, at the level of context.

## 1 Introduction

Before turning to quantified noun phrases in predicate calculus, let us review the reconstruction of propositional calculus in DBS. It is based on treating propositional constants as feature structures with the following attributes:

### 1.1 FEATURE STRUCTURE OF A PROPOSITIONAL CONSTANT

attribute 1: name
attribute 2: proposition number
attribute 3: name of next
attribute 4: connective to next
attribute 5: name of previous
attribute 6: connective to previous

In this way, the relations of a propositional constant to previous and next in the context of a formula are integrated into the constant’s feature structure. This is in contradistinction to the graphical characterization of relations in logical notation and has the advantage that the constants of a formula may be stored in accordance with the indexing and retrieval principles of a database, e.g., alphabetically.

The feature structures are built automatically by parsing formulas of propositional calculus in conjunctive normal form (CNF),<sup>1</sup> using a semantically interpreted LA-grammar called *LA-pci*.<sup>2</sup> For example, the formula  $\mathbf{p} \vee \mathbf{q} \vee \mathbf{r} \ \& \ \tilde{\mathbf{p}} \vee \mathbf{s} \vee \mathbf{q}$  translates equivalently into the following set of feature structures called *proplets*:

### 1.2 REPRESENTATION OF ' $\mathbf{p} \vee \mathbf{q} \vee \mathbf{r} \ \& \ \tilde{\mathbf{p}} \vee \mathbf{s} \vee \mathbf{q}$ ' IN DBS

p	q	r	$\tilde{\mathbf{p}}$	s	q
1	2	3	4	5	6
q	r	$\tilde{\mathbf{p}}$	s	q	s
∨	∨	∧	∨	∨	∨
∨	p	q	r	$\tilde{\mathbf{p}}$	∨
∨	∨	∨	∧	∨	∨

During storage in the data structure of a word bank (cf. Hausser 2001), the completed proplets are added at the end of their respective token lines (ordering of a previously unordered set):

### 1.3 STORING ' $\mathbf{p} \vee \mathbf{q} \vee \mathbf{r} \ \& \ \tilde{\mathbf{p}} \vee \mathbf{s} \vee \mathbf{q}$ ' IN A WORD BANK

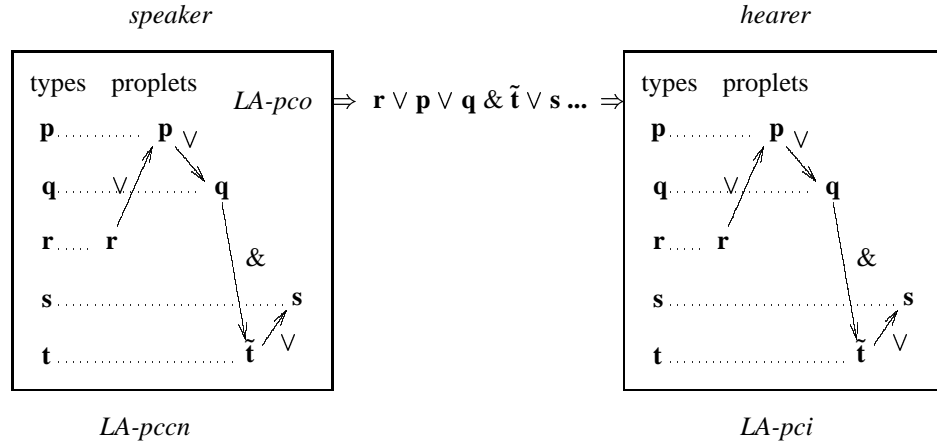
<i>types</i>	<i>proplets</i>	
p	p	$\tilde{\mathbf{p}}$
q	1	4
r	q	s
∨	∨	∨
∨	r	r
∨	∧	∧
∨	q	q
∨	2	6
∨	r	∨
∨	∨	s
∨	p	∨
∨	∨	∨
∨	r	r
∨	3	∧
∨	$\tilde{\mathbf{p}}$	∧
∨	∧	q
∨	q	∨
∨	∨	∨

<sup>1</sup>Cf. Hopcroft & Ullman 1979, p. 325. CNF consists of constants with or without negation, connected by '&' (and) and '∨' (or). The negation of, for example,  $\mathbf{p}$  is written as  $\tilde{\mathbf{p}}$ , called *external negation*, and paraphrased as *It is not the case that p*. CNF formulas are written without parentheses whereby, for example,  $\mathbf{p} \vee \mathbf{q} \vee \mathbf{r} \ \& \ \mathbf{s} \vee \mathbf{t} \vee \mathbf{u}$  is treated as equivalent to  $(\mathbf{p} \vee \mathbf{q} \vee \mathbf{r}) \ \& \ (\mathbf{s} \vee \mathbf{t} \vee \mathbf{u})$ .

<sup>2</sup>*LA-pci* stands for LA-propositional\_calculus\_interpretation and is defined in Hausser 2002b.



## 1.4 PROPOSITIONAL CALCULUS AS A COMMUNICATION LANGUAGE



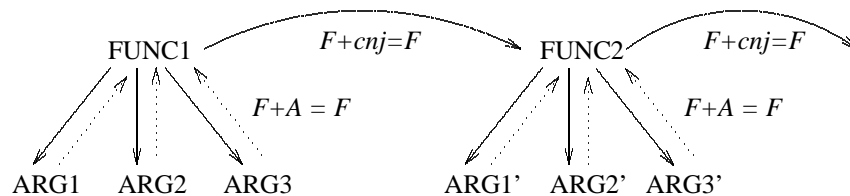
During production, the navigation powered by *LA-pccn* serves as the speaker's conceptualization. The conceptualization is mapped into language by (i) copying the proplets traversed into a sequence of proplets and (ii) mapping the sequence into an equivalent CNF formula. Procedure (ii) is based on the *LA-pco*.<sup>4</sup>

The hearer parses the incoming surfaces using *LA-pci*. The semantic interpretation of *LA-pci* results in a set of proplets which are stored in the hearer's database. The interaction of *LA-pccn*, *LA-pco*, and *LA-pci* ensures that the content mapped by the speaker into CNF formulas is reconstructed by the hearer into an equivalent content, resulting in successful communication.

## 2 From propositional constants to building propositions

The first step of upscaling from propositional to predicate calculus consists in (i) disassembling propositional constants into complex structures consisting of a functor and a characteristic number of arguments and (ii) replacing the concatenation between propositional constants by one between the functors. This more differentiated data structure supports the following kind of navigation, illustrated here with three place functors:<sup>5</sup>

### 2.1 EXTRAPROPOSITIONAL cnj-NAVIGATION



<sup>4</sup>*LA-pco* stands for LA-propositional\_calculus\_output and is defined in Hausser 2002b.

<sup>5</sup>This navigation is powered by *LA-MOTOR*, defined in Hausser 2001.

Like the propositional constants illustrated in 1.2, the functors and arguments are defined as feature structures (proplets). Furthermore, proplets for functors like *sleep* (one-place), *see* (two-place), or *give* (three-place), and for arguments like *John* (name) or *man* (noun) are stored alphabetically at the end of their respective token line, similar to 1.3. For example, if **p** equals *John loves Mary* and **q** equals *Mary loves John*, then representations of **p & q** in logical notation and DBS format, respectively, would be as follows:

## 2.2 REPRESENTATIONS OF *John loves Mary* and *Mary loves John*

old logic: <sup>6</sup>	<b>love (John, Mary) &amp; love (Mary, John)</b>																							
DBS:	<i>types</i>	<i>proplets</i>																						
	<table style="border-collapse: collapse; width: 100%;"> <tr><td style="padding: 2px;">arg: John</td></tr> <tr><td style="padding: 2px;">mspr: name</td></tr> <tr><td style="padding: 2px;">func:</td></tr> <tr><td style="padding: 2px;">id:</td></tr> <tr><td style="padding: 2px;">prn:</td></tr> </table>	arg: John	mspr: name	func:	id:	prn:	<table style="border-collapse: collapse; width: 100%;"> <tr><td style="padding: 2px;">arg1: John</td></tr> <tr><td style="padding: 2px;">mspr: name</td></tr> <tr><td style="padding: 2px;">func: love</td></tr> <tr><td style="padding: 2px;">id: 1</td></tr> <tr><td style="padding: 2px;">prn: 1</td></tr> </table>	arg1: John	mspr: name	func: love	id: 1	prn: 1	<table style="border-collapse: collapse; width: 100%;"> <tr><td style="padding: 2px;">arg2: John</td></tr> <tr><td style="padding: 2px;">mspr: name</td></tr> <tr><td style="padding: 2px;">func: love</td></tr> <tr><td style="padding: 2px;">id: 1</td></tr> <tr><td style="padding: 2px;">prn: 2</td></tr> </table>	arg2: John	mspr: name	func: love	id: 1	prn: 2						
arg: John																								
mspr: name																								
func:																								
id:																								
prn:																								
arg1: John																								
mspr: name																								
func: love																								
id: 1																								
prn: 1																								
arg2: John																								
mspr: name																								
func: love																								
id: 1																								
prn: 2																								
	<table style="border-collapse: collapse; width: 100%;"> <tr><td style="padding: 2px;">func: love</td></tr> <tr><td style="padding: 2px;">mspr: 2pl</td></tr> <tr><td style="padding: 2px;">arg1:</td></tr> <tr><td style="padding: 2px;">arg2:</td></tr> <tr><td style="padding: 2px;">ctp:</td></tr> <tr><td style="padding: 2px;">ctn:</td></tr> <tr><td style="padding: 2px;">prn:</td></tr> </table>	func: love	mspr: 2pl	arg1:	arg2:	ctp:	ctn:	prn:	<table style="border-collapse: collapse; width: 100%;"> <tr><td style="padding: 2px;">func: love</td></tr> <tr><td style="padding: 2px;">mspr: 2pl</td></tr> <tr><td style="padding: 2px;">arg1: John</td></tr> <tr><td style="padding: 2px;">arg2: Mary</td></tr> <tr><td style="padding: 2px;">ctp:</td></tr> <tr><td style="padding: 2px;">ctn: 1 &amp; 2</td></tr> <tr><td style="padding: 2px;">prn: 1</td></tr> </table>	func: love	mspr: 2pl	arg1: John	arg2: Mary	ctp:	ctn: 1 & 2	prn: 1	<table style="border-collapse: collapse; width: 100%;"> <tr><td style="padding: 2px;">func: love</td></tr> <tr><td style="padding: 2px;">mspr: 2pl</td></tr> <tr><td style="padding: 2px;">arg1: Mary</td></tr> <tr><td style="padding: 2px;">arg2: John</td></tr> <tr><td style="padding: 2px;">ctp: 1 &amp; 2</td></tr> <tr><td style="padding: 2px;">ctn: 2 &amp; 3</td></tr> <tr><td style="padding: 2px;">prn: 2</td></tr> </table>	func: love	mspr: 2pl	arg1: Mary	arg2: John	ctp: 1 & 2	ctn: 2 & 3	prn: 2
func: love																								
mspr: 2pl																								
arg1:																								
arg2:																								
ctp:																								
ctn:																								
prn:																								
func: love																								
mspr: 2pl																								
arg1: John																								
arg2: Mary																								
ctp:																								
ctn: 1 & 2																								
prn: 1																								
func: love																								
mspr: 2pl																								
arg1: Mary																								
arg2: John																								
ctp: 1 & 2																								
ctn: 2 & 3																								
prn: 2																								
	<table style="border-collapse: collapse; width: 100%;"> <tr><td style="padding: 2px;">arg: Mary</td></tr> <tr><td style="padding: 2px;">mspr: name</td></tr> <tr><td style="padding: 2px;">func:</td></tr> <tr><td style="padding: 2px;">id:</td></tr> <tr><td style="padding: 2px;">prn:</td></tr> </table>	arg: Mary	mspr: name	func:	id:	prn:	<table style="border-collapse: collapse; width: 100%;"> <tr><td style="padding: 2px;">arg2: Mary</td></tr> <tr><td style="padding: 2px;">mspr: name</td></tr> <tr><td style="padding: 2px;">func: love</td></tr> <tr><td style="padding: 2px;">id: 2</td></tr> <tr><td style="padding: 2px;">prn: 1</td></tr> </table>	arg2: Mary	mspr: name	func: love	id: 2	prn: 1	<table style="border-collapse: collapse; width: 100%;"> <tr><td style="padding: 2px;">arg1: Mary</td></tr> <tr><td style="padding: 2px;">mspr: name</td></tr> <tr><td style="padding: 2px;">func: love</td></tr> <tr><td style="padding: 2px;">id: 2</td></tr> <tr><td style="padding: 2px;">prn: 2</td></tr> </table>	arg1: Mary	mspr: name	func: love	id: 2	prn: 2						
arg: Mary																								
mspr: name																								
func:																								
id:																								
prn:																								
arg2: Mary																								
mspr: name																								
func: love																								
id: 2																								
prn: 1																								
arg1: Mary																								
mspr: name																								
func: love																								
id: 2																								
prn: 2																								

In this particular example, logical notation and DBS format have in common that both are based on the principle of functor-argument structure. However, while the notation in old logic is dependent on graphical means (linear order), the equivalent recoding in DBS is not.

The DBS data structure resembles that of propositional calculus (cf. 1.3) in that it consists of feature structures for types and for tokens. The feature structures of the types have only two attributes with non-NIL values. The first is called **arg** in nominal and **func** in verbal proplets, and contains the name of the proplet, here *John*, *love*, and *Mary*, respectively. The second attribute is called **mspr** (morphosyntactic properties)<sup>7</sup> and characterizes properties which are common to all

<sup>6</sup>Our use of the term ‘old logic’ is in concord with I. Bochenski 1961, p. 12, C.

<sup>7</sup>As explained in Hausser 1999/2001, p. 62, propositions may be regarded as part of the world, as part of language, or abstractly in terms of logic. The attribute **mspr** has its terminological origin in language, but functions equivalently in the other two interpretations of propositions.

corresponding tokens, such as the sign type<sup>8</sup> in arguments and modifiers, or the number of places in functors.

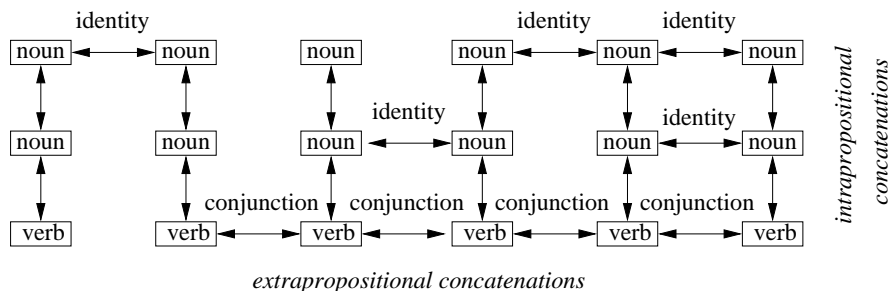
In the feature structures of the tokens (proplets), most attributes have non-NIL values. These values have been copied into the feature structure during syntactic-semantic interpretation.

Proplets belonging to the same proposition have a common proposition number (prn), here 1 and 2, respectively. Arguments specify their functor and functors specify their arguments, and similarly for modifiers and modified (cf. 4.1).

In addition, there are two kinds of extrapositional relations: identity between arguments (nouns), represented by the attribute *id* and conjunctions between functors (verbs), represented by the attributes *ctn* (conjunction to next) and *ctp* (conjunction to previous).

Compared to propositional calculus, the new intrapositional relations between functors and arguments, and the additional extrapositional relation of identity between arguments result in a much wider variety of possible continuations. These may be presented schematically as the following ‘railroad system,’ whereby only elementary nouns and two-place verbs are assumed for the sake of simplicity:

### 2.3 SCHEMATIC RAILROAD SYSTEM OF PREDICATE CALCULUS



Extrapositional relations are shown as horizontal double arrows, while intrapositional relations are shown as vertical double arrows. Each proposition may have up to three extrapositional relations, one based on verbal conjunction and two based on nominal identity (and accordingly for one- and three-place verbs).

The functors (nouns) and arguments (verbs) forming propositions are distributed all over the data structure. In line with the coding technique of DBS, all intra- and extrapositional concatenations are established solely in terms of attributes, i.e., proplet name, proposition number, identity, and conjunction.

## 3 Quantified noun phrases versus proper names

Old logic uses different syntactic structures depending on whether an argument is a quantified noun phrase or a proper name. This is illustrated below with the comparison of 3.1 and 3.2:

<sup>8</sup>The main sign types are symbols, indexicals, and names. See Hausser 1999/2001, Chapter 6.

### 3.1 ALTERNATIVE ANALYSES OF John walks

old logic:        **walk(John)**

DBS:

<i>types</i>	<i>proplets</i>
arg: John	arg: John
mspr: name	mspr: name
func:	func: walk
id:	id:
prn:	prn: 23
func: walk	func: walk
mspr: 1pl	mspr: 1pl
arg1:	arg1: John
ctp:	ctp:
ctn:	ctn:
prn:	prn: 23

### 3.2 ALTERNATIVE ANALYSES OF Every man walks

old logic:         $\forall x [\mathbf{man}(x) \rightarrow \mathbf{walk}(x)]$

DBS:

<i>types</i>	<i>proplets</i>
arg: man	arg: man
mspr: noun	mspr: noun, every
func:	func: walk
id:	id:
prn:	prn: 23
func: walk	func: walk
mspr: 1pl	mspr: 1pl
arg1:	arg1: man
ctp:	ctp:
ctn:	ctn:
prn:	prn: 23

In the case of quantified noun phrases, here **every man**, old logic treats the principle of functor-argument structure as subservient to the structure introduced by the quantifier. The use of two sub-propositions and a connective creates an artificial need to express identity, handled in terms of variables bound by the quantifier. Also, this analysis violates the principle of surface compositionality.

DBS, in contrast, treats the principle of functor-argument structure as primary, regardless of whether an argument is a name or a quantified noun phrase. Also, old logic's syntactic distinction between universal and existential quantification, i.e.,  $\forall x [\mathbf{f}(x) \rightarrow \mathbf{g}(x)]$  versus  $\exists x [\mathbf{f}(x) \ \& \ \mathbf{g}(x)]$ , reduces in DBS to different values in the **mspr** attribute of the noun phrase in question, e.g., **all** or **every** vs. **a(n)** or **some**.

Furthermore, DBS handles coreference uniformly by means of the *id* attributes. This method is suitable for all kinds of arguments, i.e., names, quantified noun phrases, and pronouns, both within propositions and between propositions. Old logic, in contrast, employs three different methods for expressing coreference: (i) equality of names, (ii) variables bound by a quantifier, and (iii) the connective  $=$ .

## 4 Comparing the ontologies of old logic and DBS

A famous attempt to extend the logical quantifiers to natural language is Russell's (1905) analysis of **the** in combination with a singular noun. Consider the following example, which is presented with alternative analyses in old logic and DBS:

### 4.1 ALTERNATIVE ANALYSES OF The present King of France is bald

old logic:  $\exists x \forall y [[\text{present\_King\_of\_France}(y) \leftrightarrow y=x] \ \& \ \text{bald}(x)]^9$

DBS:	<i>types</i>	<i>proplets</i>
	[func: bald mspr: 1pl arg1: ctp: ctn: prn:]	[func: bald mspr: 1pl arg1: King_of_France ctp: ctn: prn: 23]
	[arg: King_of_France mspr: noun func: modr: id: prn:]	[arg: noun mspr: noun, sg, the func: bald modr: present id: prn: 23]
	[modr: bald mspr: pos modd: id: prn:]	[modr: bald mspr: pos modd: King_of_France id: prn: 23]

The complicated structure of old logic's formula has no syntactic counterpart in the linguistic surface. In other words, the difference between the semantic representation of a singular noun combined with the determiner **the** and that of a corresponding name, e.g., **bald(Charles)**, is even greater than the difference between 3.1 and 3.2. Needless to say, such a major structural difference between the semantic representations of expressions which are syntactically equivalent except for the *kind* of argument used is unjustifiable from the linguistic viewpoint of a surface compositional analysis.

<sup>9</sup>See also Montague 1974, p. 261, T2.

The quantified noun phrases of old logic are also known for frequently (i) creating artificial scope ambiguities and (ii) failing to express representations intuitively required. As an example of the first kind consider

Three boys kissed four girls.

Presupposing a representation of numbers using quantifiers (which is not pretty on any account), this sentence has been claimed to be ambiguous. On one reading there is a large group of boys and a large group of girls, and of all the boys three managed to kiss a total of four girls each. On the other reading a group of three boys is simultaneously engaged in kissing a group of four girls. The number of readings may be multiplied by distinguishing for each pair of boys whether they kissed the same girl or not, depending on the order of the quantifiers in the semantic representation,

The second kind of problem is illustrated by the sentences

Every man who loves a woman is happy.

Every man who loves a woman loses her.

and the following attempt to represent their dominant reading in predicate calculus:

$$\forall x [[\text{man}(x) \ \& \ \exists y [\text{woman}(y) \ \& \ \text{love}(x, y)]] \rightarrow [\text{happy}(x)]]$$

$$\forall x [[\text{man}(x) \ \& \ \exists y [\text{woman}(y) \ \& \ \text{love}(x, y)]] \rightarrow [\text{lose}(x, y)]]$$

The parallel formulas are intended to reproduce the subordinate structure of relative clauses. This does not work for the second formula, however, because the scope of the quantifier  $\exists y$  does not reach the  $y$  in  $\text{lose}(x, y)$ . An alternative formula with the quantifier  $\exists y$  in fronted position (as in prenex normal form) would solve the scope problem, but cannot be derived compositionally by translating the words of the sentence into suitable lambda expressions, as required by Montague grammar.

In the DBS analysis, these problems do not arise. One reason is that the functor-argument structure is treated as primary in DBS, such that the differences between arguments, i.e., between different kinds of quantified noun phrases and proper names, are minimized. The other reason is that old logic and DBS are based on different ontological assumptions.

Old logic defines meaning in terms of truth conditions which hold between formulas and set-theoretic models. The definitions are formulated in a metalanguage the words of which are restricted to clearly defined set-theoretic concepts. The truth conditions for the universal quantifier, for example, are defined as follows:

If ' $\phi$ ' is a sentence, then ' $\forall x \phi^{M, g}$ ' is true relative to a model  $M$  and a variable assignment  $g$  iff it holds for all alternative variable assignments  $g'$  that ' $\phi^{M, g'}$ ' is true relative to  $M$ .

Thus, if  $\phi$  equals  $[\text{man}(x) \rightarrow \text{walk}(x)]$ , then  $\forall x \phi^{M, g}$  equals  $\forall x [\text{man}(x) \rightarrow \text{walk}(x)]^{M, g}$ . The latter is true relative to a model  $M$ , iff  $[\text{man}(x) \rightarrow \text{walk}(x)]$  is true for all possible values of  $x$  in  $M$ .

The truth conditions for the existential quantifier are similar. The only difference is that truth relative to a model must hold for *at least one* variable assignment  $g$ , rather than *for all*. This metalanguage-based approach to characterizing the existential and the universal quantifier may be illustrated schematically as follows:

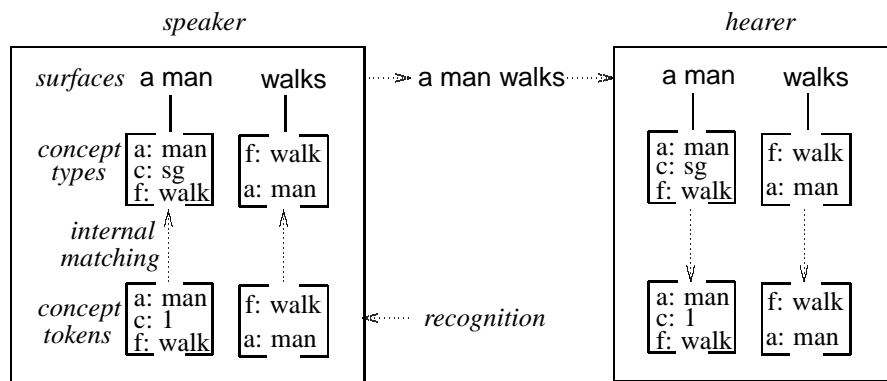


## 5 Interpreting different determiners

In DBS, quantifiers are treated at the level of language as part of feature structures representing nominals (cf. 3.2, 4.1). Their difference of interpretation is focused on the *reference* of determiner-noun sequences (quantified noun phrases) relative to the context of use. Thus the truth conditions of old logic are reinterpreted as pragmatic rules for controlling the procedure of internal matching.

Consider for example a cognitive agent observing a man walking. This content is coded into a language sign (speaker mode). Another cognitive agent interprets the sign by constructing an equivalent content at the level of context (hearer mode):

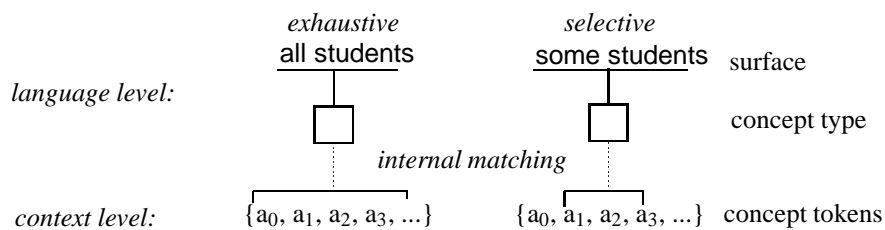
### 5.1 COMMUNICATING WITH A DETERMINER-NOUN SEQUENCE



The attributes a, c, and f stand for argument, cardinality, and functor, respectively. The sentence *a man walks* is true if the speaker's observation is correct and the coding into language has been working properly. The communication of this content is successful, if the hearer's decoding results in a data structure which is equivalent to the corresponding part of the speaker's data structure.

The matching between referents at the level of context (concept tokens) and literal meanings at the level of language (concept types) depends on the structures at both levels. At the level of context, referents may be single individuals or sets of individuals. At the level of language, sets of individuals may be referred to either exhaustively by means of *all/every*, or selectively by means of *some/several*.

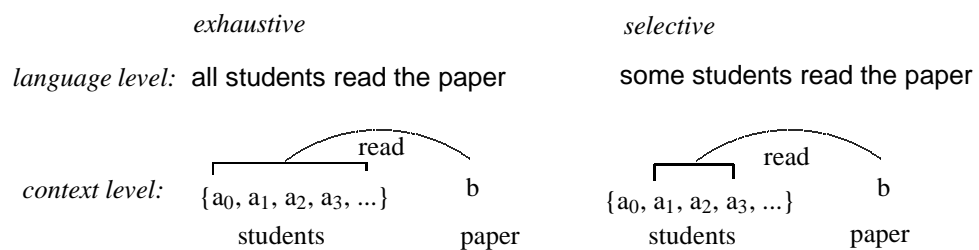
### 5.2 EXHAUSTIVE VERSUS SELECTIVE REFERENCE TO SETS



For simplicity, the levels of context and language meaning are represented as concept tokens and concept types, respectively. These concepts are assumed to be values in the feature structures as represented in 5.1. Programming the exhaustive vs. selective matching relations in question requires ‘vertically’ oriented concepts which are quite different from those underlying  $\forall \mathbf{x} \mathbf{f}(\mathbf{x})$  and  $\exists \mathbf{x} \mathbf{f}(\mathbf{x})$ .

The distinction between exhaustive and selective reference affects the nature of relations which hold between sets of individuals, or between a set and an individual. Consider the following schematic characterization:

### 5.3 RELATIONS BETWEEN A SET AND AN INDIVIDUAL



Another distinction at the level of context is the one between definite and indefinite referents. A definite referent is assumed to be known by the partner of discourse, as in the current president, or has been previously introduced. The distinctions between sets of different size, individuals, and known versus unknown referents are reflected in English as follows:

### 5.4 TYPES OF REFERENTS AND THEIR COUNTERPARTS IN ENGLISH

*sets containing three or more elements:*

exhaustive	all students, every student, students
selective	some students, several students
exhaustive, definite	the students, all the students
selective, definite	some of the students, several of the students

*sets containing one element:*

	one student
definite	the one student

*sets containing two elements:*

	two students
definite	the two students

*etc.*

*individuals:*

	a student
definite	the student

The above table has been purposely restricted to reference with the sign type *symbol*. The analysis of reference with the sign types *name* and *indexical* (pronouns) is analogous, but requires a separate study.<sup>10</sup>

## 6 Replicating scope ambiguity

A standard intuition in old logic are scope ambiguities based on alternative orders of existential and universal quantifiers. Consider the following example:

### 6.1 SCOPE AMBIGUITY OF Every man loves a woman

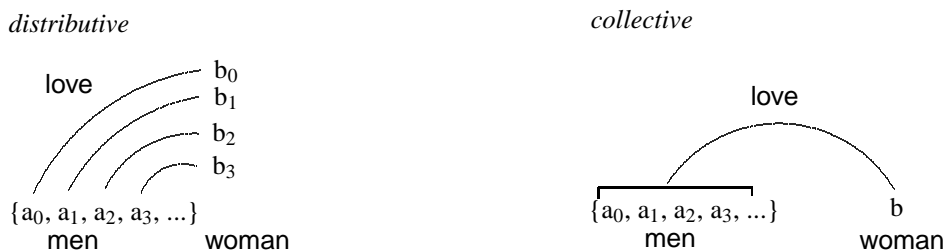
$$\forall x \exists y [\text{man}(x) \ \& \ [\text{woman}(y) \rightarrow \text{love}(x, y)]]$$

$$\exists y \forall x [\text{man}(x) \ \& \ [\text{woman}(y) \rightarrow \text{love}(x, y)]]$$

The standard interpretation of the upper formula is a reading according to which Jack, Bill, and Harry each have their own woman, as in a proper suburb, while the lower formula represents a reading according to which Jack, Bill, and Harry all love one and the same woman, Mary, as in a house of ill repute.

In database semantics, the source of these intuitions may be traced to *distributive* versus *collective* interpretations of relations. At the level of context, this distinction may be presented schematically as follows:

### 6.2 DISTRIBUTIVE VERSUS COLLECTIVE RELATION INTERPRETATION



The nature of some relations is such that they take an argument which may be singular or plural while being restricted semantically to a distributive interpretation of it. Consider the following examples:

### 6.3 RELATIONS RESTRICTED TO DISTRIBUTIVE INTERPRETATION

- Mary closed her eyes (singular)
- The girls closed their eyes (nominative, exhaustive)
- Some girls closed their eyes (nominative, selective)
- John dealt the cards (oblique, exhaustive)
- John sliced some of the apples (oblique, selective)

<sup>10</sup>See Hausser 1999/2001, Chapter 6.

Other relations require a plural argument and are restricted to a collective interpretation of it. Consider the following examples:

#### 6.4 RELATIONS RESTRICTED TO COLLECTIVE INTERPRETATION

\*John mixed the nut (singular)  
The girls formed a circle (nominative, exhaustive)  
Some girls formed a circle (nominative, selective)  
John mixed all the ingredients (oblique, exhaustive)  
John mixed some of the ingredients (oblique, selective)

Most relations, however, may interpret their plural arguments either distributively or collectively, as illustrated by *love* in 6.2. Furthermore, it seems to be a characteristic property of at least some natural languages, e.g., English, French, and German, that the distinction between distributive versus collective interpretations of plural arguments has no reflection in the surface.

Consequently, the alleged scope ambiguity arising with relations which may be interpreted either distributively or collectively are handled in DBS as an underspecification concerning the matching conditions between language meanings and contextual referents. Restricted relations like those of 6.3 and 6.4 are best treated by inferences at the level of context instead of scope ambiguities (cf. 6.1).<sup>11</sup>

## 7 Conclusion

The reconstructions of propositional calculus outlined in the Introduction and of predicate calculus presented in the remainder of this paper are compatible, but different. Propositional calculus has been reconstructed in a ‘horizontal’ manner, based on the extrapositional relation of conjunction between propositions (using on the connectives  $\vee$  and  $\&$  only). Predicate calculus has been reconstructed here in a ‘vertical’ manner, concentrating on the matching relation between language meanings and their contextual counterparts.

The treatment of propositional calculus has been imported into the DBS analysis of propositions by coding extrapositional relations of conjunction into the functor (Section 2). The reconstruction of predicate calculus maintains the functor-argument structure of propositions regardless of whether an argument is a determiner-noun sequence (quantified noun phrase), a name, or a pronoun. As a consequence, the analysis is strictly surface compositional (Section 3). This is made possible in large part by DBS using an ontology different from logic (Section 4).

The information contributed by a determiner is analyzed in DBS as controlling the matching relation between a determiner-noun sequence at the language level

---

<sup>11</sup>This analysis will eventually be extended to examples like the ‘donkey sentence’ (cf. Geach 1969, p. 155, (38), Kamp 1993). This requires more work, however, because the examples involve relative clauses and indexicals (pronouns). Cf. Hausser 1999/2001, p. 457 and pp. 110 f., respectively

and an individual or set of individuals at the level of context. The function of quantifiers binding variables in logic is assigned in DBS to the attributes asserting identity between arguments (Section 5). The scope ambiguity based on alternate orderings of quantifiers in logic is explained alternatively in terms of a distribute versus collective interpretation of relations (Section 6).

The LA-grammars for parsing simple sentences of natural language into feature structures stored in the word bank (interpretation in the hearer mode), for navigation through the concatenated propositions of the word bank (conceptualization), and for mapping sequences of proplets into corresponding surfaces of natural language (production in the speaker mode) are basically those defined in Hausser 2001. A formal integration of the present analysis of determiner-noun sequences into this fragment affects primarily internal matching, i.e. the mapping between structures containing concept types at the level language and corresponding structures containing concept tokens at the level of context.

This involves programming the lexical selection of the determiners in the speaker mode, and the construction of set-theoretic concepts and relations based on determiners in the hearer mode. The rules in question are formulated as subclauses of the Fourth Principle of Pragmatics (PoP-4).<sup>12</sup>

## References

- Bochenski, I. (1961) *A History of Formal Logic*, University of Notre Dame Press.
- Davidson, D. & J. Hintikka (eds.) (1969) *Words and Objections, Essays on the Work of W.V.Quine*, Reidel, Dordrecht.
- Geach, P.T. (1969) "Quine's Syntactical Insights," in Davidson & Hintikka (eds.).
- Hausser, R. (1999/2001) *Foundations of Computational Linguistics, Human-Computer Communication in Natural Language*, Berlin, New York: Springer-Verlag.
- Hausser, R. (2001) "Database Semantics for Natural Language," *Artificial Intelligence*, Vol. 130.1:27–74, Elsevier, Dordrecht.
- Hausser, R. (2002a) "A Hypothesis on the Origin of the Sign Types," in A. Gelbukh (ed.) *Computational Linguistics and Intelligent Text Processing*, Lecture Notes in Computer Science, Berlin-New York: Springer-Verlag.
- Hausser, R. (2002b) "Reconstructing Predicate Calculus in Database Semantics," to appear.
- Hopcroft, J.E. & Ullman, J.D. (1979) *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading, Mass.
- Kamp, J.A.W. & U. Reyle (1993) *From Discourse to Logic*, Kluwer, Dordrecht.
- Montague, R. (1974) *Formal Philosophy*, Yale U. Press, New Haven.
- Russell, B. (1905) "On Denoting," *Mind* 14:479-493.

---

<sup>12</sup>Cf. Hausser 1999/2001, p. 105. For slightly modernized versions of PoP-1 – PoP-7 see Hausser 2002a.