BOOLEAN AND NON-BOOLEAN 'AND'

MANFRED KRIFKA

SNS, University of Tübingen
Biesingerstr. 10, 7400 Tübingen
FRG

1. INTRODUCTION

The subject of this article is the semantics of the coordination *and* in English and its equivalents in other natural languages.* In particular, I will be concerned on the one hand with the so-called Boolean conjunction, which is a sentence operator that shares essential properties with conjunction in propositional logic (1a), and on the other hand with what has come to be known as non-Boolean conjunction, which is basically an operator on indiviudal terms (1b):

(1) a. [[John sang] and [Mary danced]].
b. [[*John*] *and* [*Mary*]] *met at the opera.*

One problem is to account for the fact that these operators may conjoin expressions of other types than sentences or individual terms. A second problem is to explain why many languages have only one word for both functions.

The first problem — the occurrence of *and* as an operator on different types — has found several more or less equivalent solutions for the Boolean case, and I will discuss these solutions in **Section 2**. There is, however, no satisfying account for why the non-Boolean conjunction occurs in different types as well. I will discuss the existing approaches in **Section 3**. In **Section 4**, I propose what I consider the solution to this problem. **Section 5** treats plurality as iterated conjunction and contains an analysis of cumulative readings, and **Section 6** is concerned with the conjunction of quantifiers which yields so-called 'branching' quantifiers.

The second problem — why Boolean and non-Boolean conjunction are often expressed by one and the same word — is dealt with in **Section 7**. It is shown that both conjunctions can in fact be reduced to one interpretation.

The article presupposes some knowledge of the use of type theory in the syntax and semantics of natural languages. I will assume e and t as basic types (entities,

László Kálmán and László Pólos (eds.), *Papers from the Second Symposium on Logic and Language*. Akadémiai Kiadó, Budapest, 1990.

truth values). If τ and σ are types, then $(\sigma)\tau$ is the type of functions that map objects of type σ to objects of type τ. I will use the notation $\sigma\tau$ to mean $(\sigma)\tau$ if σ is a simple symbol. The semantic representations of words and expressions whose internal structure does not matter is given in boldface, e.g., the meaning of *sing* is represented by **sing**, and the meaning of *met at the opera* by **met_at_the_opera**.

2. BOOLEAN CONJUNCTION AND ITS GENERALIZATION

As mentioned above, Boolean conjunction does not only occur as a sentence operator of type *ttt* which takes two sentences of type *t* and maps them to another sentence of type *t*. It can take other expressions as well — for example, it maps two intransitive verbs (type *et*) to another intransitive verb, or two terms (type *e*, or type *(et)t*, after lifting to generalized quantifiers) to a term, or two predicate modifiers (type *(et)et*) to another predicate modifier:

(2) a. John sings AND Mary dances. [type *t*]
b. *Mary* sings AND dances. [type *et*]
c. John AND Mary *sing*. [type *e*, after lifting type *(et)t*]
d. *She was wearing a* new AND expensive *dress*. [type *(et)et*]

And there are still other types in which we can find *and*. This calls for a principled explanation as to the types in which it occurs, and which meaning we should assign to it in different types. There are several works which proposed solutions to this, among others von Stechow (1973), Gazdar (1980), Partee and Rooth (1983) and Keenan and Faltz (1985). The basic observation is that when we apply a complex expression conjoined by Boolean conjunction to an argument, it distributes over the argument. For example, when we apply a complex intransitive verb like *sing and dance* to an individual term, we have to apply *sing* and *dance* separately and conjoin the result by *and*. This is shown in (3a); the general case is given in (3b):

(3) a. *Mary sings and dances.* $\leftrightarrow$ *Mary sings and Mary dances.*
b. If α, α' are two expressions of type $(\sigma)\tau$ which can be conjoined by Boolean conjunction $\wedge$, and if β is an expression of type σ, we have:
$[\alpha \wedge \alpha'](\beta) = \alpha(\beta) \wedge \alpha'(\beta)$.

From this rule, the semantics of Boolean conjunction for different types follows naturally. For example, Partee and Rooth (1983) first define the notion of a 'conjoinable type' (here called *t*-conjoinable), and then define the interpretation of Boolean conjunction for *t*-conjoinable types:

(4) Partee and Rooth (1983):
a. Recursive definition of *t*-conjoinable types:
— *t* is a *t*-conjoinable type;
— if τ is a *t*-conjoinable type, then for all σ, $(\sigma)\tau$ is a *t*-conjoinable type.

b. Recursive definition of Boolean conjunction for t-conjoinable types:
— if α, α' are of type t, then $\alpha \wedge \alpha'$ as usual;
— if α, α' are of a t-conjoinable type $(\sigma)\tau$, then $\alpha \wedge \alpha' = \lambda u[\alpha(u) \wedge \alpha'(u)]$ (where u is a variable which does not occur in α, α').

This is a syncategorematic definition of Boolean conjunction. We could as well have defined it directly: If $(\sigma)\tau$ is a t-conjoinable type, u is a variable of type σ and v, v' are variables of type σ, then the Boolean conjunction for expressions of type $(\sigma)\tau$ is $\lambda v' \lambda v \lambda u[v(u) \wedge v'(u)]$. However, I will give the definitions in the more perspicuous syncategorematic format.

Let us look how these definitions work in the case of the examples in (2).

(5) a. *Mary sings and dances.*
$\mathrm{sing} \wedge \mathrm{dance}(\mathrm{Mary}) =$
$= \lambda x[\mathrm{sing}(x) \wedge \mathrm{dance}(x)](\mathrm{Mary}) =$
$= \mathrm{sing}(\mathrm{Mary}) \wedge \mathrm{dance}(\mathrm{Mary})$

b. *John and Mary sing.*
$\lambda P[P(\mathrm{John})] \wedge \lambda P[P(\mathrm{Mary})](\mathrm{sing}) =$
$= \lambda P'[\lambda P[P(\mathrm{John})](P') \wedge \lambda P[P(\mathrm{Mary})](P')](\mathrm{sing}) =$
$= \mathrm{sing}(\mathrm{John}) \wedge \mathrm{sing}(\mathrm{Mary})$

c. *new and expensive dress*
$\mathrm{NEW} \wedge \mathrm{EXPENSIVE}(\mathrm{dress}) =$
$= \lambda P[\mathrm{NEW}(P) \wedge \mathrm{EXPENSIVE}(P)](\mathrm{dress}) =$
$= \mathrm{NEW}(\mathrm{dress}) \wedge \mathrm{EXPENSIVE}(\mathrm{dress}),$

with $\mathrm{NEW} = \lambda P \lambda x[P(x) \wedge \mathrm{new}(x)]$, and
$\mathrm{EXPENSIVE} = \lambda P \lambda x[P(x) \wedge \mathrm{expensive}(x)]$:

$= \lambda x[\mathrm{dress}(x) \wedge \mathrm{new}(x)] \wedge \lambda x[\mathrm{dress}(x) \wedge \mathrm{expensive}(x)] =$
$= \lambda x'[\lambda x[\mathrm{dress}(x) \wedge \mathrm{new}(x)]](x') \wedge \lambda x[\mathrm{dress}(x) \wedge \mathrm{expensive}(x)](x')] =$
$= \lambda x[\mathrm{dress}(x') \wedge \mathrm{new}(x') \wedge \mathrm{expensive}(x')]$

In (5b) we first have to type-lift John and Mary from type e to type $(et)t$ in order to arrive at a t-conjoinable type. (Type-liftings like that are allowed in 'shake & bake semantics' to get the right argument-types or function-types). In (5c) I made the simplifying assumption that the predicate modifiers NEW and EXPENSIVE can be analyzed as intersective, that is, they can be traced back to a conjunction of the predicate they apply to and predicates new and expensive which apply to objects. Note that we have to use the rule for the reduction of Boolean conjunction to lower types twice. The choice of variables P', x' instead of, say, P, x is for the sake of clarity.

3. THE NON-BOOLEAN CONJUNCTION AND ATTEMPTS FOR ITS GENERALIZATION

As already indicated in the introduction, there is another use of *and* which cannot be a case of Boolean conjunction, as it applies to types which are not *t*-conjoinable, namely *e*, and which does not allow for distribution over arguments. For example, the following equivalence does not hold:

(6) *Barbara and Mats wrote an article together.* $\not\leftrightarrow$
Barbara wrote an article together and Mats wrote an article together.

The fact that this use of *and* cannot be traced back to Boolean conjunction was observed by several authors, notably Massey (1976), Link (1983), and Hoeksema (1983). To capture its semantics, these authors propose an operation which maps entities onto a new entity, their 'sum' or 'collection'. In (6), we can assume that *Barbara and Mats* refers to the sum individual consisting of Barbara and Mats, and that the predicate applies to that sum individual.

(7) *Barbara and Mats wrote an article together.*
wrote_an_article_together(Barbara $\oplus$ Mats)

Here, "$\oplus$" denotes a two-place operation in the domain of entities such that whenever we have two entities a, b, $a \oplus b$ is another entity. That is, "$\oplus$" is of type *eee*. There are different ways to spell out the semantics of this operation. One is to think of "$\oplus$" as the join operation of a join semi-lattice (cf. Link, 1983). Then we assume that "$\oplus$" is idempotent ($a \oplus a = a$), symmetric ($a \oplus b = b \oplus a$), and associative ($a \oplus [b \oplus c] = [a \oplus b] \oplus c$). We will stick to these assumptions, although some of them are problematic, as will be discussed in **Section 7**. A plausible model for the domain of entities is a Boolean algebra (possibly without a bottom element), where "$\oplus$" denotes the join operation. So the non-Boolean conjunction turns out to be in a way 'Boolean', after all. From now on, I will call "$\oplus$" the 'join'.

As with Boolean conjunction, there are examples which suggest that non-Boolean conjunction does not only live in type *eee*, but in other types as well. For example, Link (1983) discusses cases where we can assume that it might apply to expressions of type *et*:

(8) a. *John and Mary are* husband and wife.
b. boy and girl *who kissed each other*

Link (1983) therefore introduces a predicate join of type $(et)(et)et$ whose interpretation is derived from the join of entities. If we use "$\oplus$" both for join of entities and predicates, the rule can be given as follows:

(9) If α, α' are predicates (type *et*), then
$\alpha \oplus \alpha' = \lambda u'' \exists u, u'[u'' = u \oplus u' \wedge \alpha(u) \wedge \alpha'(u')]$
(where u, u', u'' are variables of type *e* that do not occur free in α, α').

For the examples above, we get the following analyses:

(10) a. $\lambda x''\exists x, x'[x'' = x \oplus x' \wedge \mathbf{husband}(x) \wedge \mathbf{wife}(x')](\mathbf{John} \oplus \mathbf{Mary}) =$
$= \exists x, x'[\mathbf{John} \oplus \mathbf{Mary} = x \oplus x' \wedge \mathbf{husband}(x) \wedge \mathbf{wife}(x')]$

b. $\lambda P \lambda y[P(y) \wedge \mathbf{kiss\_each\_other}(y)]$
$(\lambda x''\exists x, x'[x'' = x \oplus x' \wedge \mathbf{boy}(x) \wedge \mathbf{girl}(x')]) =$
$= \lambda y \exists x, x'[y = x \oplus x' \wedge \mathbf{boy}(x) \wedge \mathbf{girl}(x') \wedge \mathbf{kiss\_each\_other}(y)]$

There are other types of examples which can be handled by Link's predicate join:

(11) *This* (a) *is* beer and lemonade.
$\lambda x''\exists x, x'[x'' = x \oplus x' \wedge \mathbf{beer}(x) \wedge \mathbf{lemonade}(x')](\mathbf{a})$

(12) The dogs and the roosters *barked and crowed* all night.
$\lambda x''\exists x, x'[x'' = x \oplus x' \wedge \mathbf{barked}(x) \wedge \mathbf{crowed}(x')]$
$(\mathbf{the\_dogs} \oplus \mathbf{the\_roosters})$

(13) *The flag* (a) *is* green and white.
$\lambda x''\exists x, x'[x'' = x \oplus x' \wedge \mathbf{green}(x) \wedge \mathbf{white}(x')](\mathbf{a})$

(14) *Every* student and professor *came to the party.*
$\lambda P' \lambda P \forall y[P'(y) \rightarrow P(y)]$
$(\lambda x''\exists x, x'[x'' = x \oplus x' \wedge \mathbf{student}(x) \wedge \mathbf{professor}(x')])(\mathbf{came}) =$
$= \forall y[\exists x, x'[y = x \oplus x' \wedge \mathbf{student}(x) \wedge \mathbf{professor}(x')] \rightarrow \mathbf{came}(y)]$

The first example is true when a is a mixture of beer and lemonade (cf. Wald, 1977). The second one is true when the dogs barked and the roosters crowed all night. The third one is true when the flag a consists of two parts which are green and white, respectively. The case of the last example is more difficult, as the conjoined predicate is an argument of the quantificational determiner *every*. We get an interpretation which says that every object consisting of a student and a professor came. If we assume the plausible rule of 'divisivity' for predicates like came, which can be formulated as

if $\mathbf{came}(x \oplus x')$ then $\mathbf{came}(x) \wedge \mathbf{came}(x')$,

this amounts to:

$\forall x[\mathbf{student}(x) \rightarrow \mathbf{came}(x)] \wedge \forall x[\mathbf{professor}(x) \rightarrow \mathbf{came}(x)]$

Actually, example (14) can be treated as a case of Boolean conjunction as well (we first have to lift the predicates **student** and **professor** from type *et* to type $((et)t)t$, conjoin them by Boolean conjunction and apply the complex predicate to the determiner; see Keenan and Faltz, 1985). However, we should explain why a natural interpretation needs the lifting of the predicates of type *et* to type $((et)t)t$. In general it is assumed that lifting only occurs when necessary (see

Partee and Rooth, 1983), and so the reading where (14) applies to persons who are both students and professors should be the most natural one, if not the only one. Furthermore, cases with collective predicates, like *every boy and girl kissed each other*, cannot be handled by Boolean conjunction at all.

We have seen that Link's predicate join can explain a range of facts. However, he only assumes a lifted version of join for predicates and gives no general rule for the lifting of the join of entities to other types. There are two attempts which propose such a rule, namely Partee and Rooth (1983) and Hoeksema (1988).

Partee and Rooth try out their recipe for the lifting of Boolean conjunction on Link's join operation. They define a notion of an e-conjoinable type (they call it s-conjoinable) parallel to the notion of a t-conjoinable type as a type ending in e, and generalize the join operation for individuals in a similar fashion as the Boolean conjunction for t-conjoinable types. But much to their dissapointment, they realize that in Montague's type hierarchy the only e-conjoinable type in use is se, the type of individual concepts. (See Krifka, 1986, for a discussion of the join operation for individual concepts.)

Hoeksema also tried to define the join operation for other types than type e. His first attempt (Hoeksema, 1983) is not of much interest for a general lifting operation, as it is essentially restricted to quantifiers (type $(et)t$). In short, he defines a join operation for 'atomic' quantifiers (that is, quantifiers which denote ultrafilters or unions of ultrafilters in every model). The definition amounts roughly to: If Q_1, Q_2 are atomic quantifiers, then $Q_1 \oplus Q_2(A)$ iff there is an x, y, where x is a minimal element of Q_1 and y is a minimal element of Q_2, such that $A(x \oplus y)$.

Hoeksema (1988) is more interesting, for he proposes to type-raise "$\oplus$" according to the general rules of the Lambek calculus and its interpretation in terms of lambda-terms (cf. Lambek, 1958, van Benthem, 1986, 1987). In a natural deduction type version of the Lambek calculus (which slightly differs from the one used by Hoeksema), we can get the following interpretation of the join for quantifiers, type $(et)t$:

(15) Derivation of type

a. $e \Rightarrow e \quad eee \Rightarrow eee$

b. $\underline{e;\ eee \Rightarrow ee \quad e \Rightarrow e \quad 1}$

c. $\underline{e;\ eee;\ e \Rightarrow e \quad et \Rightarrow et}$

d. $\underline{e;\ eee;\ e;\ et \Rightarrow et \quad 2}$

e. $\underline{eee;\ e;\ et \Rightarrow et \quad (et)t \Rightarrow (et)t}$

f. $\underline{eee;\ e;\ et;\ (et)t \Rightarrow t \quad 3}$

g. $\underline{eee;\ et;\ (et)t \Rightarrow et \quad (et)t \Rightarrow (et)t}$

h. $\underline{eee;\ et;\ (et)t;\ (et)t \Rightarrow t}$

i. $\underline{eee;\ (et)t;\ (et)t \Rightarrow (et)t}$ (remove 1)

j. $\underline{eee;\ (et)t \Rightarrow ((et)t)(et)t}$ (remove 2)

k. $eee \Rightarrow ((et)t)((et)t)(et)t$ (remove 3)

Interpretation as Lambda-terms:

a. $x \quad \oplus$

b. $\underline{x \oplus \quad y}$

c. $\underline{x \oplus y \quad P}$

d. $\underline{P(x \oplus y)}$

e. $\underline{\lambda x[P(x \oplus y)] \quad T'}$

f. $\underline{T'(\lambda x[P(x \oplus y)])}$

g. $\underline{\lambda y[T'(\lambda x[P(x \oplus y)])] \quad T}$

h. $\underline{T(\lambda y[T'(\lambda x[P(x \oplus y)])])}$

i. $\underline{\lambda P[T(\lambda y[T'(\lambda x[P(x \oplus y)])])]}$

j. $\underline{\lambda T' \lambda P[T(\lambda y[T'(\lambda x[P(x \oplus y)])])]}$

k. $\lambda T \lambda T' \lambda P[T(\lambda y[T'(\lambda x[P(x \oplus y)])])]$

As an example, consider the analysis of the following sentence:

(16) *Every student and every professor* met.

$\lambda T \lambda T' \lambda P[T(\lambda y[T'(\lambda x[P(x \oplus y)])])]$
$(\lambda P \forall x[\mathbf{student}(x) \rightarrow P(x)])(\lambda P \forall x[\mathbf{professor}(x) \rightarrow P(x)])(\mathbf{met}) =$
$= \forall y[\mathbf{student}(y) \rightarrow \forall x[\mathbf{professor}(x) \rightarrow \mathbf{met}(x \oplus y)]]$

This gives us the right result in this case. However, the Lambek calculus is too restricted; it cannot provide all the lifting rules we need. In general, we have to lift the join operation from type *eee* to a type $(\tau)(\tau)\tau$. This is possible according to the Lambek calculus in the case of $\tau = (et)t$, but not for every type τ. For example, with the possible exception of (21) the following types τ don't allow a lifting from *eee* to $(\tau)(\tau)\tau$:

(17) boy and girl *who kiss each other* $[\tau = et]$

(18) green and white *flag* [$\tau = (et)et$]
(19) John's and Mary's *house* [$\tau = (et)(et)t$]
(20) extremely and moderately *expensive dresses* [$\tau = ((et)et)(et)et$]
(21) *The planes flew* above and below *the clouds* [$\tau = e(et)et$ or $((et)t)(et)et$]
(22) *John and Mary* read and sang *a poem and a song* [$\tau = eet$]
(23) the father and the mother *of John* [$\tau = ee$]

The reason is that these liftings (with the exception of (21)) do not preserve the so-called e-count or t-count. However, derivations in the Lambek calculus necessarily preserve e-count and t-count (cf. van Benthem, 1986, 1987). The e-count is defined as follows: e has an e-count of 1, and the e-count of a type $(\sigma)\tau$ is the e-count of τ minus the e-count of σ. The definition of t-count is parallel. For type eee the e-count is -1 and the t-count is 0. But, to give just two examples, for type $(et)(et)et$ the t-count is -1, and for type $((et)et)((et)et)(et)et$ the e-count is 0.

We conclude that both the suggestions of Partee and Rooth (1983) and of Hoeksema (1988) are not general enough to cover all the types for which we want to lift the join operation for individuals.

However, there is an obvious generalization of Link's procedure for the construction of a predicate join out of a join operation for entities to a lifting operation for other types as well. If we look at the case of join of predicates, the following formula suggests itself:

(24) If α, α' are of type $(\sigma)t$ and β, β' are of type σ, and both α and α' and β and β' can be conjoined by non-Boolean conjunction, then we have:
$\alpha(\beta) \wedge \alpha'(\beta') \rightarrow \alpha \oplus \alpha'(\beta \oplus \beta')$.

For example,

(25) a. *John sings and Mary dances* → *John and Mary sing and dance.*
b. *John sings and Mary sings* → *John and Mary sing.*
c. *John sings and John dances* → *John sings and dances.*

Note that we cannot replace "→" in (24) by "↔", as the "←" direction does not hold in general (cf. *John and Mary met*). The generalization (24) suggests the following definition for a generalized join operation, modeled according to Partee and Rooth's treatment of generalized Boolean conjunction:

(26) a. Recursive definition of e-conjoinable types:
— e is an e-conjoinable type;
— if σ is an e-conjoinable type, then $(\sigma)t$ is an e-conjoinable type, or more general:
— if $\sigma_1, \ldots, \sigma_n$ are e-conjoinable types, then $(\sigma_1)\ldots(\sigma_n)t$ is an e-conjoinable type.

b. Recursive definition of non-Boolean conjunction:
— if α, α' are of type e, then $\alpha \oplus \alpha'$ as above;

— if α, α' are of an *e*-conjoinable type $(\sigma)t$, then
$\alpha \oplus \alpha' = \lambda u'' \exists u, u'[u'' = u \oplus u' \wedge \alpha(u) \wedge \alpha'(u')]$,
where u, u', u'' are variables of type σ not occurring free in α, α', or more general:

— if α, α' are of an *e*-conjoinable type $(\sigma_1)\ldots(\sigma_n)t$, then
$\alpha \oplus \alpha' =$
$= \lambda x_1 \ldots x_n \exists y_1, z_1, \ldots y_n, z_n$
$[y_1 \oplus z_1 = x_1 \wedge \ldots \wedge y_n \oplus z_n =$
$x_n \wedge \alpha(y_1)\ldots(y_n) \wedge \alpha'(z_1)\ldots(z_n)]$,
where x_1, y_1, z_1 are variables of type $\sigma_1, \ldots$, and x_n, y_n, z_n are variables of type σ_n, all not occurring free in α, α'.

This obviously gives us Link's predicate join for the case when α, α' are of type *et*. Let us look whether it yields the right result for other types as well — for example, for type $(et)et$:

(27) green and white *(flag)* [type $(et)et$]
$\mathbf{GREEN} \oplus \mathbf{WHITE} =$
$= \lambda P'' \lambda x'' \exists P, P', x, x'[P'' =$
$= P \oplus P' \wedge x'' = x \oplus x' \wedge \mathbf{GREEN}(P)(x) \wedge \mathbf{WHITE}(P')(x')]$

We must apply the rule for generalized conjunction a second time. Using different variables of type *e* for the sake of perspicuity, $P \oplus P'$ can be spelled out as $\lambda y'' \exists y, y'[y'' = y \oplus y' \wedge P(y) \wedge P'(y')]$:

$\lambda P'' \lambda x'' \exists P, P', x, x'$
$[P'' = \lambda y'' \exists y, y'[y'' = y \oplus y' \wedge P(y) \wedge P'(y')] \wedge x'' = x \oplus x' \wedge$
$\wedge \mathbf{GREEN}(P)(x) \wedge \mathbf{WHITE}(P')(x')] =$
$= \lambda P'' \lambda x'' \exists P, P', x, x'$
$[\forall y''[P''(y'') \leftrightarrow \exists y, y'[y'' = y \oplus y' \wedge P(y) \wedge P'(y')]] \wedge x'' = x \oplus x' \wedge$
$\wedge \mathbf{GREEN}(P)(x) \wedge \mathbf{WHITE}(P')(x')]$

We assume that **GREEN** and **WHITE** are intersective predicate modifiers and render **GREEN** as $\lambda P \lambda x[P(x) \wedge \mathbf{green}(x)]$, and similarly for **WHITE**:

$\lambda P'' \lambda x'' \exists P, P', x, x'$
$[\forall y''[P''(y'') \leftrightarrow \exists y, y'[y'' = y \oplus y' \wedge P(y) \wedge P'(y')]] \wedge$
$\wedge x'' = x \oplus x' \wedge P(x) \wedge \mathbf{green}(x) \wedge P'(x') \wedge \mathbf{white}(x')]$

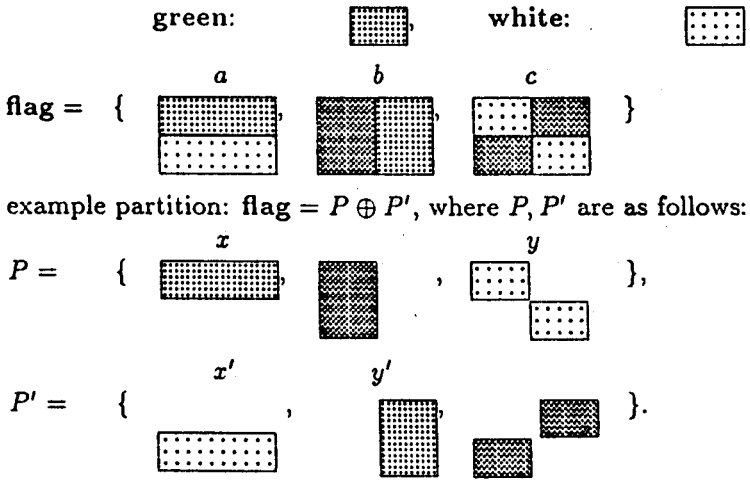
To see whether this is an adequate representation, let us consider the treatment of a sentence like the following one:

(28) *This* (a) *is a green and white flag.*
$\lambda P'' \lambda x'' \exists P, P', x, x'$
$[\forall y''[P''(y'') \leftrightarrow \exists y, y'[y'' = y \oplus y' \wedge P(y) \wedge P'(y')] \wedge x'' = x \oplus x' \wedge$
$\wedge P(x) \wedge \mathbf{green}(x) \wedge P'(x') \wedge \mathbf{white}(x')](\mathbf{flag})(\mathrm{a}) =$

$$= \exists P, P', x, x'$$
$$[\forall y''[\mathbf{flag}(y'') \leftrightarrow \exists y, y'[y'' = y \oplus y' \wedge P(y) \wedge P'(y')]] \wedge$$
$$\wedge \mathrm{a} = x \oplus x' \wedge P(x) \wedge \mathbf{green}(x) \wedge P'(x') \wedge \mathbf{white}(x')]$$

This says that a is a green and white flag if the predicate flag can be partitioned into two predicates P, P' and a can be partitioned into two entities x, x' such that P applies to x, P' applies to x', x is green, and x' is white. Let us look at a simple model to see whether this analysis is correct. We assume that flag has an extension of three flags as shown below. Then P and P' form a partition of flag of the type we are looking for.

(29)



As indicated, we find x and x' in the extension of P and P' such that P applies to x, P' applies to x', x and x' make up a, and x is green, x' is white.

However, there is a problem, as (28) claims that P, P' are such that whenever we have an entity y in P and an entity y' in P', then their join must also be a flag. For a count noun predicate like flag, this can be true just in case it has only one entity in its extension. With more entities in the extension, we do not get proper representations. For example, in our model the join of what I have indicated as y, y' in the example should be a flag as well — which obviously is not true.

So a very simple extension of Link's predicate join to other types leads to wrong results, and we have to look for something better.

4. THE PROPER GENERALIZATION OF NON-BOOLEAN CONJUNCTION

In this section, I am going to propose a generalization of conjuction which gets rid of the unwelcome consequences of the simple extension of Link's predicate join. Also, its definition will turn out to be simpler than (26), the generalized

join operation. (26) is complicated because it uses both Boolean and non-Boolean conjunction to generalize non-Boolean conjunction, and because the join of an n-place relation ($n \geq 2$) cannot be traced back to the join of an $(n-1)$-place relation, but has to be reduced 'at once'.

A central idea of my proposal is that we have to generalize two things simultaneously, namely a conjunction operation and an inclusion relation. Another central idea is that we define both the conjunction operation and the inclusion relation for the basic types e and t, and have only one generalization formula for conjunction and inclusion, regardless of which type we generalize to.

We start with the recursive definition of a general inclusion relation, "$\sqsubseteq$". It is the same as that which was proposed in van Benthem (1986, 1987) to specify constraints for type changes:

(30) Recursive definition of a generalized inclusion relation "$\sqsubseteq$":
- if α, α' are of type e, then $\alpha \sqsubseteq \alpha'$ iff $\alpha = \alpha'$,
- if α, α' are of type t, then $\alpha \sqsubseteq \alpha'$ iff $\alpha \rightarrow \alpha'$ (that is, the truth value of α is less or equal than the truth value of α'),
- if α, α' are of type $(\sigma)\tau$, then $\alpha \sqsubseteq \alpha'$ iff for all β of type σ, $\alpha(\beta) \sqsubseteq \alpha'(\beta)$

When α, α' are of type et, $\alpha \sqsubseteq \alpha'$ amounts to set inclusion, $\alpha \subseteq \alpha'$. — Now we can give a definition for a generalized conjunction operation "$\sqcup$":

(31) Recursive (partial) definition of a generalized conjunction $\sqcup$:
- if α, α' are of type e, then $\alpha \sqcup \alpha' = \alpha \oplus \alpha'$
- if α, α' are of type t, then $\alpha \sqcup \alpha' = \alpha \wedge \alpha'$
- if α, α' are of type $(\sigma)\tau$ and β, β' are of type σ, then $\alpha(\beta) \sqcup \alpha'(\beta') \sqsubseteq \alpha \sqcup \alpha'(\beta \sqcup \beta')$

Note that the last part is a generalization of the rule for "$\oplus$" for predicates in (24), $\alpha(\beta) \wedge \alpha'(\beta') \rightarrow \alpha \oplus \alpha'(\beta \oplus \beta')$.

That last part is obviously not a proper *definition* of a generalized conjunction, as we claim "$\sqsubseteq$" instead of "=". So it only gives a constraint, or an approximation, for the relation between "$\sqcup$" and "$\sqsubseteq$". This is an essential part of the treatment to be developed here; it captures the fact that, in the generalization for predicates, we cannot replace "$\rightarrow$" by "$\leftrightarrow$" (see the discussion of (24)).

It might seem that such an approximation of the meaning of a complex expression by the meaning of its part is not a valid meaning rule at all. However, I think there are reasons to assume that recursive meaning rules typically have the form of approximations, and that definitions are nothing but the limiting case. This can be seen in the following example. The usual formulation of the meaning rule for the composition of an (intersective) adjective with a noun yields a predicate which applies to entities to which both the adjective and the noun applies; for example, we can assume that $\mathbf{cold\_water} = \lambda x[\mathbf{cold}(x) \wedge \mathbf{water}(x)]$. However, in idiomatic uses this does not hold. For example, a *cold war* is neither cold,

nor is it a war. But, of course, something which is cold and which is a war, say Napoleon's invasion of Russia in 1812, would have to count as a cold war as well. Therefore it is safer to specify recursive semantic rules as approximations. In our case, this would yield, e.g., $\lambda x[\mathbf{cold}(x) \wedge \mathbf{war}(x)] \sqsubseteq \mathbf{cold\_war}$. This allows for the fact that the extension of a complex expression is larger than that, and comprises, for instance, an idiomatic part which cannot be specified recursively. In the case that we have only an approximation, like $\lambda x[\mathbf{cold}(x) \wedge \mathbf{water}(x)] \sqsubseteq \mathbf{cold\_water}$, as a meaning rule, we are forced to use that rule and end up with the same as if we had assumed equality.

Note that with (31) we do not need the notion of a t-conjoinable or an e-conjoinable type, as the definitions of inclusion and conjunction fit for every type. This results from the definition of inclusion and conjuction for both basic types.

Can we derive from (31) a general rule which tells us how to interpret conjunction for an arbitrary type? Again, we cannot expect a proper definition for it, but only an approximation. Let us start with the conjunction of t-based types, that is, types ending in t (these types, together with the basic type e, are the only relevant ones anyhow). Let us assume that we want to conjoin two expressions α, α' of type $(\sigma)t$, and let us assume that u'' is a variable of type σ. Given (31), we can assume that whenever we have $\alpha \sqcup \alpha'(u'')$ and u'' can be partitioned into u, u' such that $u'' = u \sqcup u'$, we have $\alpha(u) \sqcup \alpha'(u') \sqsubseteq \alpha \sqcup \alpha'(u'')$. Therefore we can assume that $\lambda u'' \exists u, u'[u'' = u \sqcup u' \wedge \alpha(u) \wedge \alpha'(u')] \sqsubseteq \alpha \sqcup \alpha'$. In the general case, for relations with n arguments, we have the following:

(32) If α, α' are of type $(\sigma_1)\ldots(\sigma_n)t$, u, u' and u'' are variables of type σ_1 and $u_2 \ldots u_n$ are variables of type $\sigma_2 \ldots \sigma_n$, all not occurring free in α, α', then
$\lambda u'' \lambda u_2 \ldots \lambda u_n \exists u, u'[u'' = u \sqcup u' \wedge [\alpha(u) \sqcup \alpha'(u')](u_2)\ldots(u_n)] \sqsubseteq \alpha \sqcup \alpha'$.

This says that we can approximate $\alpha \sqcup \alpha'$ by the lambda-expression on the left side. What this amounts to can be best seen with an example. For expressions of type et, we get the following analysis:

(33) *sing and dance* [type et]
$\lambda x'' \exists x, x'[x'' = x \sqcup x' \wedge [\mathbf{sing}(x) \sqcup \mathbf{dance}(x')]] \sqsubseteq \mathbf{sing} \sqcup \mathbf{dance} =$
$= \forall x''[\exists x, x'[x'' = x \oplus x' \wedge \mathbf{sing}(x) \wedge \mathbf{dance}(x')] \rightarrow [\mathbf{sing} \sqcup \mathbf{dance}](x'')]$

That is, whenever the lambda-expression applies to some entity x'', it applies to $\mathbf{sing} \sqcup \mathbf{dance}$ as well, but not necessarily vice versa. This means that (31) specifies only sufficient, but not necessary truth conditions for conjoined expressions. However, this is presumably the only rule which tells us how the meaning of the complex expression *sing and dance* can be related to the parts, *sing*, *and* and *dance*. Therefore we can assume some pragmatic strengthening of "$\rightarrow$" to "$\leftrightarrow$", such that whenever an entity is in the extension of $\mathbf{sing} \sqcup \mathbf{dance}$, it is such that it can be partitioned into two parts, one of which sings and one of which dances.

Before we try out (32) on other cases, I will present a way to generalize this rule for arbitrary types — up to now, it holds for t-based types only. There is a way

to generalize existential quantification to other types. Existential quantification expresses the maximal value of a sentence for a range of variable assignments; for example, $\exists x \Phi$ is the maximal value of Φ with respect to all assignments of x. Perhaps the clearest way to introduce a general maximalization is with the help of an operator **sup** which takes a variable of arbitrary type and a sentence, and yields the maximal value of the variable for which the sentence is true:

(34) If u is a variable and Φ is an expression of type t, then
$\mathbf{sup}(u, \Phi) = \alpha$ iff $\Phi[\alpha/u]$ and for all α' such that $\Phi[\alpha'/u]$, $\alpha' \sqsubseteq \alpha$.

Here, $\Phi[\alpha/u]$ is like Φ, but with all free occurrences of u replaced by α. With **sup**, we can formulate a constraint for conjunction which applies to e-based types as well:

(35) If α, α' are of type $(\sigma_1)\ldots(\sigma_n)\tau$, if u, u' and u'' are variables of type σ_1, $u_2 \ldots u_n$ are variables of type $\sigma_2 \ldots \sigma_n$, and v is a variable of type τ, none of which occur free in α, α', then
$\lambda u'' \lambda u_2 \ldots \lambda u_n$
$[\mathbf{sup}(v, \exists u, u'[u'' = u \sqcup u' \wedge v = [\alpha(u) \sqcup \alpha'(u')](u_2)\ldots(u_n)])] \sqsubseteq$
$\sqsubseteq \alpha \sqcup \alpha'$.

In the case of t-based types, this reduces to the definition (32), as v is a variable of type t and equals 1 (truth) if the sentence in the second argument place of **sup** is true. In the case of e-based types, we also get a plausible result. To see this, consider a case where two expressions of type ee are conjoined (for example, functional nouns like *(the) father (of)* can be analyzed as such expressions):

(36) *father and mother*
$\lambda x''[\mathbf{sup}(x''', \exists x, x'[x'' = x \sqcup x' \wedge x''' = [\mathbf{father}(x) \sqcup \mathbf{mother}(x')]])] \sqsubseteq$
$\sqsubseteq \mathbf{father} \sqcup \mathbf{mother}$

John and Mary's father and mother
$\lambda x''[\mathbf{sup}(x''', \exists x, x'[x'' = x \sqcup x \wedge x''' = [\mathbf{father}(x) \sqcup \mathbf{mother}(x')]])]$
$(\mathbf{John} \sqcup \mathbf{Mary}) =$
$= \mathbf{sup}(x''', \exists x, x'[\mathbf{John} \sqcup \mathbf{Mary} = x \sqcup x' \wedge x''' = [\mathbf{father}(x) \sqcup \mathbf{mother}(x')]])$

If we take "$\sqcup$" as a symmetric relation, then this denotes the parents of John and Mary if they are brother and sister, and is undefined otherwise. If we take "$\sqcup$" to be asymmetric (cf. **Section 7**), it denotes the join of John's father and Mary's mother.

As e-based types play a marginal role, we will assume the constraint specified in (32) for the remainder of this article.

Now let us look at the treatment of the example which caused difficulties in the last section:

(37) *green and white* [type $(et)et$]
$\lambda P''\lambda x\exists P, P'[P'' = P \sqcup P \wedge [\mathbf{GREEN}(P) \sqcup \mathbf{WHITE}(P')](x)] \sqsubseteq$
$\sqsubseteq \mathbf{GREEN} \sqcup \mathbf{WHITE}$

With $\lambda y''\exists y, y'[y'' = y \sqcup y' \wedge P(y) \wedge P'(y')] \sqsubseteq P \sqcup P'$, we can derive from that the following formula:

$\lambda P''\lambda x\exists P, P'$
$[\lambda y''\exists y, y'$
$[y'' = y \sqcup y' \wedge [P(y) \sqcup P'(y')]] \sqsubseteq P'' \wedge$
$\wedge [\mathbf{GREEN}(P) \sqcup \mathbf{WHITE}(P')](x)] \sqsubseteq \mathbf{GREEN} \sqcup \mathbf{WHITE}$

which is tantamount to:

$\lambda P''\lambda x\exists P, P'$
$[[\forall y, y'[P(y) \wedge P'(y')] \rightarrow P''(y \oplus y')] \wedge$
$\wedge [\mathbf{GREEN}(P) \sqcup \mathbf{WHITE}(P')](x)] \sqsubseteq$
$\sqsubseteq \mathbf{GREEN} \sqcup \mathbf{WHITE}$

In order to determine the truth conditions of $[\mathbf{GREEN}(P) \sqcup \mathbf{WHITE}(P')](x)$, we have to use the approximation rule

$\lambda z''\exists z, z'[z'' = z \sqcup z' \wedge [\mathbf{GREEN}(P)(z) \sqcup \mathbf{WHITE}(P')(z')]] \sqsubseteq$
$\sqsubseteq \mathbf{GREEN}(P) \sqcup \mathbf{WHITE}(P')$

That is, we have to replace $[\mathbf{GREEN}(P) \sqcup \mathbf{WHITE}(P')]$ by that lambda-expression and arrive at

$\lambda P''\lambda x\exists P, P'$
$[\forall y, y'[[[P(y) \wedge P'(y')] \rightarrow P''(y \oplus y')] \rightarrow P''(y')] \wedge$
$\wedge \lambda z''\exists z, z'[z'' = z \sqcup z' \wedge [\mathbf{GREEN}(P)(z) \sqcup \mathbf{WHITE}(P')(z')]](x)] \sqsubseteq$
$\sqsubseteq \mathbf{GREEN} \sqcup \mathbf{WHITE}$

which is tantamount to:

$\lambda P''\lambda x\exists P, P'$
$[\forall y, y'$
$[[[P(y) \wedge P'(y')] \rightarrow P''(y \oplus y')] \rightarrow P''(y'')] \wedge$
$\wedge \exists z, z'[x = z \oplus z' \wedge \mathbf{GREEN}(P)(z) \wedge \mathbf{WHITE}(P')(z')]] \sqsubseteq$
$\sqsubseteq \mathbf{GREEN} \sqcup \mathbf{WHITE}$

Finally, we can simplify this formula in some respects for the sake of readability and spell out **GREEN** as $\lambda P\lambda x[P(x) \wedge \mathbf{green}(x)]$, and similarly for **WHITE**:

$\lambda P''\lambda x\exists P, P', z, z'$
$[\forall y, y'[[P(y) \wedge P'(y')] \rightarrow P''(y \oplus y')] \wedge x = z \oplus z' \wedge$
$\wedge P(z) \wedge \mathbf{green}(z) \wedge P'(z') \wedge \mathbf{white}(z')] \sqsubseteq$
$\sqsubseteq \mathbf{GREEN} \sqcup \mathbf{WHITE}$

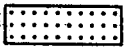
Let us look now at the treatment of a sentence like the following one:

(38) *That* (a) *is a green and white flag*
[GREEN ⊔ WHITE](**flag**)(**a**)

The only rule we have to trace back the truth conditions of **GREEN ⊔ WHITE** to is the one developed in (37). So we have to assume:

$$\lambda P''\lambda x\exists P, P', z, z'$$
$$[\forall y, y'[P(y) \wedge P'(y') \rightarrow P''(y \oplus y')] \wedge x = z \oplus z' \wedge$$
$$\wedge P(z) \wedge \mathbf{green}(z) \wedge P'(z') \wedge \mathbf{white}(z')](\mathbf{flag})(\mathbf{a}) =$$
$$= \exists P, P', z, z'$$
$$[\forall y, y'[P(y) \wedge P'(y') \rightarrow \mathbf{flag}(y \oplus y')] \wedge \mathbf{a} = z \oplus z' \wedge$$
$$P(z) \wedge \mathbf{green}(z) \wedge P'(z') \wedge \mathbf{white}(z')]$$

The basic difference from our first attempt to formalize the sentence in (28) is that now we assume only "→" instead of "↔". Therefore P and P' need not 'exhaust' the whole extension of the predicate **flag**. It is possible that P and P' only apply to one element each. Then we get a proper representation of our example, as in the following choice of P and P':

$P = \{$  $\}$, $P' = \{$  $\}$

Thus, the generalization of conjunction proposed here seems to work better than a direct generalization of Link's definition for predicate join.

Before we will look into the conjunction of expressions of other types, especially quantifiers, I will sketch one extension of the approach developed here to the semantics of pluralization. It is interesting in its own right, and we will have to make use of it later.

5. INTERLUDE: PLURALIZATION AS ITERATED CONJUNCTION

It is obvious that conjunction and plurality are closely related concepts; for example, if I have *an apple and an apple*, then I have *apples*. In an intuitive way pluralization is iterated conjunction:

(39) **apples** = **apple** ⊔ **apple** ⊔ ...

More formally, we can define pluralization for arbitrary types as follows:

(40) Let α be an expression of an arbitrary type, then α^p (the plural expression corresponding to α) can be defined as the smallest β such that
— $\alpha \sqsubseteq \beta$;

— for all β', if $\beta' \sqsubseteq \beta$, then $\alpha \sqcup \beta' \sqsubseteq \beta$.

I assume here that $\alpha \sqsubseteq \alpha^P$, that is, that plural expressions include their singular versions (see Krifka, 1989, for an argument to that effect in the case of nominal predicates). It is obvious that iteration of pluralization does not change the meaning, that is, we have $\alpha^{PP} = \alpha^P$.

As an example, we can prove that the predicate

$$A = \lambda x'' \exists x, x'[x'' = x \sqcup x' \wedge \mathbf{apple}(x) \wedge \mathbf{apple}(x')]$$

is included in apples, as we have $A \sqsubseteq \mathbf{apple} \sqcup \mathbf{apple}$ and $\mathbf{apple} \sqcup \mathbf{apple} \sqsubseteq \mathbf{apple}^P$. The notion of plurality developed here is compatible with Link (1983).

Now plurality makes sense not only with nominal predicates, but with verbal predicates as well. For example, we can assume that the predicate sing originally applies only to single persons who sing. However, we can derive a plural predicate $\mathbf{sing}^P$ from that. It is this predicate which is applied in cases like the following one:

(41) *John and Mary sing.*
$\mathbf{sing}^P(\mathbf{John}' \sqcup \mathbf{Mary}') = \mathbf{sing}^P(\mathbf{John}) \wedge \mathbf{sing}^P(\mathbf{Mary}) =$
$= \mathbf{sing}(\mathbf{John}) \wedge \mathbf{sing}(\mathbf{Mary})$

One use of such plural predicates is that they allow us to give a straightforward semantics for so-called 'cumulative readings' which does not need stipulations like those proposed by Scha (1981). Scha's original example and his treatment goes as follows (card is interpreted as the cardinality function on sets):

(42) *600 Dutch firms have 5000 American computers.*
$\mathbf{card}(\lambda x[\mathbf{Dutch\_firm}(x) \wedge \exists y[\mathbf{Am_comp}(y) \wedge \mathbf{have}(x, y)]]) = 600$
$\wedge\, \mathbf{card}(\lambda x[\mathbf{Am_comp}(x) \wedge \exists y[\mathbf{Dutch\_firm}(y) \wedge \mathbf{have}(y, x)]]) =$
$= 5000$

The sentence should be true if the number of Dutch firms which have American computers is 600, and the number of American computers owned by Dutch firms is 5000. The problem is to derive this rather complicated representation in a natural way. Furthermore, Scha's representation does not cover, without additional stipulations, cases with collective individual owning relations, for example, cases where one computer is owned by several firms or vice versa.

The representation of plurality developed here provides us with a natural way to treat cumulative readings. We can assume that NPs which contain number words can be analyzed as indefinite NPs, with the number word being something like an adjective — for example, a predicate modifier based on a measure function (see Link, 1987, Krifka, 1986, for this analysis). Here, we simply assume that we have a measure function N for entities, and that *600 Dutch firms* is analyzed on the basis of a predicate $\lambda x[\mathbf{Dutch\_firm}^P(x) \wedge \mathrm{N}(x) = 600]$ (see Krifka, 1986, for a more detailed analysis). This predicate applies to entities which consist of 600

Dutch firms. If we want to analyze NPs uniformly as generalized quantifiers, the English NP *600 Dutch firms* is represented by the existential quantifier

$$\lambda P \exists x[P(x) \wedge \textbf{Dutch\_firm}^{P}(x) \wedge \textbf{N}(x) = \textbf{600}].$$

The predicate *have* is represented by the plural version $\textbf{have}^{P}$. Then we get the following representation:

(43) *600 Dutch firms have 5000 American computers.*

$$\begin{aligned}&\lambda P \exists x[P(x) \wedge \textbf{Dutch\_firm}^{P}(x) \wedge \textbf{N}(x) = \textbf{600}] \\ &\quad (\lambda x[\lambda P \exists x[P(x) \wedge \textbf{Am\_comp}^{P}(x) \wedge \textbf{N}(x) = \textbf{5000}] \\ &\qquad (\lambda y[\textbf{have}^{P}(x, y)])) = \\ &= \exists x, y \\ &\quad [\textbf{Dutch\_firm}^{P}(x) \wedge \textbf{N}(x) = \textbf{600} \wedge \\ &\qquad \wedge \textbf{Am\_comp}(y) \wedge \textbf{N}(y) = \textbf{5000} \wedge \textbf{have}^{P}(x, y)]\end{aligned}$$

This is true in case there is an x which consists of 600 Dutch firms and a y which consists of 5000 American computers and x has y in the 'plural' way. The interpretation of $\textbf{have}^{P}$, in turn, depends on the individual owning relations. Look at a simple example: Assume $\textbf{have}(a, x)$ and $\textbf{have}(b, y)$ and $\textbf{have}(b \oplus c, z)$ (which represents that b and c have z collectively), then we have $\textbf{have}^{P}(a \sqcup b \sqcup c, x \sqcup y \sqcup z)$.

Of course, this representation still allows for the fact that in cases where a sentence like (43) is true, a sentence like *three Dutch firms have fifty American computers* is true as well. However, we can invoke pragmatic rules of maximalization of information which force the speaker to choose as high a number as possible. In the example at hand, a sentence like *n Dutch firms have m American computers* implies sentences *n' Dutch firms have m' American computers*, where $n' \leq n$ and $m' \leq m$. If we assume the conversational maxime of Quantity (cf. Grice, 1967), which roughly claims that a speaker should say as much as he truthfully can (modulo some other maximes), then a hearer can implicate that the speaker used maximal numbers n, m in uttering *n Dutch firms have m American computers.*

6. QUANTIFIER CONJUNCTION

A particularly interesting test case for our rule for generalized conjunction are quantifiers. Conjoined quantifiers were the paradigm case for the use of so-called 'branching quantifiers' in natural language. Here I will show that an adequate reading for these sentences falls out from the assumptions we made so far.

We start with an easy example which should exemplify general conjunction for expressions of type $(et)t$:

(44) *a boy and a girl* [type $(et)t$]

$$\lambda P'' \exists P, P'[P'' = P \sqcup P' \wedge [\textbf{a\_boy}(P) \sqcup \textbf{a\_girl}(P')] \sqsubseteq \textbf{a\_boy} \sqcup \textbf{a\_girl}$$

with $\textbf{a\_boy} = \lambda P \exists x[\textbf{boy}'(x) \wedge P(x)]$ (and similarly for **a_girl**), with

$$\lambda x''\exists x, x'[x'' = x \sqcup x' \wedge P(x) \wedge P'(x)] \sqsubseteq P \sqcup P'$$

and some simplifications, this equals

$$\lambda P''\exists P, P' \\ [\forall x, x'[P(x) \wedge P'(x') \rightarrow P''(x \oplus x')] \wedge \\ \wedge \exists x[\mathbf{boy}(x) \wedge P(x)] \wedge \exists x[\mathbf{girl}(x) \wedge P'(x')]] \sqsubseteq \mathbf{a\_boy} \sqcup \mathbf{a\_girl}$$

To see how this works, we apply this representation to a reciprocal predicate like the representation of *kiss each other*. Reciprocal predicates, in general, are one-place predicates which are derived from two-place relations. In general, we can assume that any two-place relation α has a reciprocal version α^r:

(45) If α is a two-place relation of type $(\tau)(\tau)t$, then its reciprocal version α^r is a predicate of type $(\tau)t$ defined as follows:
- For all β, β' of type τ, if $\alpha(\beta, \beta')$ and $\alpha(\beta', \beta)$ and $\beta \neq \beta'$, then $\alpha^r(\beta \sqcup \beta')$;
- For all α' of type $(\tau)t$, if for all β, β' of type τ, if $\alpha(\beta, \beta')$ and $\alpha(\beta', \beta)$ and $\beta \neq \beta'$, then $\alpha'(\beta \sqcup \beta')$, then $\alpha^r \sqsubseteq \alpha'$.

Normally, the derivation of a reciprocal form is marked by reciprocal pronouns like *each other*, *one another* and the like, but this is not necessarily the case — for example, the reciprocal form of *connected with* is *connected*, and a possible reciprocal form of *kiss* is *kiss*.

Now let us look at an example:

(46) *A boy and a girl kissed each other.*
$[\mathbf{a\_boy} \sqcup \mathbf{a\_girl}](\mathbf{kiss}^r)$

The only meaning rule we have for that leads us to the following assumption:

$$\lambda P''\exists P, P' \\ [\forall x, x'[P(x) \wedge P'(x') \rightarrow P''(x \sqcup x')] \wedge \\ \wedge \exists x[\mathbf{boy}(x) \wedge P(x)] \wedge \exists x[\mathbf{girl}(x) \wedge P'(x')]](\mathbf{kiss}^r) = \\ = \exists P, P' \\ [\forall x, x'[P(x) \wedge P'(x') \rightarrow \mathbf{kiss}^r(x \sqcup x')] \wedge \exists x[\mathbf{boy}(x) \wedge P(x)] \wedge \\ \wedge \exists x[\mathbf{girl}(x) \wedge P'(x')]]$$

This gives the right truth conditions: It says that there are two sets, P and P', such that every x in P and every x' in P' kiss each other, and that P contains a boy and P' contains a girl.

Now let us look at the conjunction of quantifiers which, unlike in the case considered so far, yield true branching quantifier structures (cf. Barwise, 1979, Westerståhl, 1987). I give two typical examples (from Barwise, 1979):

(47) *Most linguists and most philosophers agree with each other about branching quantifiers.*

(48) ***Most of the circles and most of the stars are all connected by lines.***

Let us concentrate on example (48). In the preferred reading, this sentence is true in **Figure 2**, but false in **Figure 1**.

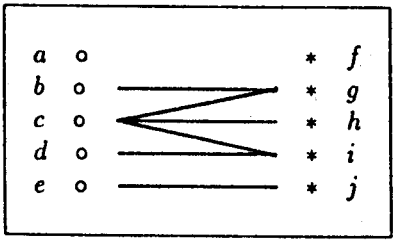


Figure 1

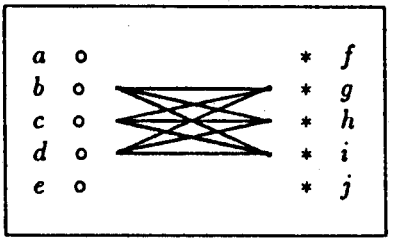


Figure 2

There are different ways to render the prominent reading of (48). The original one is to have two quantifiers which are not in each other's scope:

(49) $\left.\begin{array}{l}\textbf{most\_circles x}\\ \textbf{most\_stars y}\end{array}\right\}$ $\textbf{connected}(x,y)$

This reflects the fact that the prominent reading of (48) is neither captured with *most circles* having scope over *most stars*, nor the other way round. It has been shown by Barwise (1979) and others that we get a linearized representation if we allow for second-order quantification. If we interpret quantifiers as predicates of type $(et)t$ and assume that the two quantifiers Q_1, Q_2 are upward monotone, then we have the following equivalence, where "×" denotes Cartesian product (cf. Westerståhl, 1987):

(50) Branching quantifiers for upward monotone quantifiers Q_1, Q_2:

$$\left.\begin{array}{l}Q_1(A)\\ Q_2(B)\end{array}\right\} R \leftrightarrow \exists(P \subseteq A)\exists(P' \subseteq B)[Q_1(A)(P) \wedge Q_2(B)(P') \wedge P \times P' \subseteq R]$$

Let us assume the ordinary General Quantifier analysis in which a quantifier like *most stars* is represented by **most(star)**, the set of sets which contain more than half of the stars (that is, $\textbf{most} = \lambda P'\lambda P[\textbf{card}(P \cap P') > \frac{1}{2}\textbf{card}(P')]$). Then we get the following representation for (48):

(51) $\exists(P \subseteq \textbf{star})\exists(P' \subseteq \textbf{circle})$
$[\textbf{most}(\textbf{star})(P) \wedge \textbf{most}(\textbf{circle})(P') \wedge$
$\wedge \forall x, x'[P(x) \wedge P'(x') \rightarrow \textbf{connected}(x,y)]]$

That is, there is a set P of stars which makes up more than half of the stars, and a set P' of circles which makes up more than half of the circles, such that every element in P is connected to every element in P' (and as **connected** is symmetric, this holds vice versa as well).

Now the interesting thing is that we get a similar representation if we employ our rule for generalized conjunction. So there is no longer any need to stipulate a rule like (50). To do so, we would have to assume that *connected* is represented by the reciprocal form of connected as used in (51).

(52) *Most circles and most stars are connected.*
$\lambda P''\exists P, P'[$
$\forall x, x'[P(x) \wedge P'(x') \rightarrow P''(x \sqcup x')] \wedge$
$\wedge$ **most(circle)**$(P) \wedge$
most(star)$(P')] \sqsubseteq$
$\sqsubseteq$ [**most(circle)** $\sqcup$ **most(star)**](**connected**^{r}) =
$= \exists P, P'$
$[\forall x, x'[P(x) \wedge P'(x') \rightarrow$ **connected**$^r(x \sqcup x')] \wedge$
$\wedge$ **most(circle)**$(P) \wedge$ **most(star)**$(P')]$

This amounts to the same as the representation (51).

However, (50) only specifies the interpretation for upward monotone quantifiers. We get the wrong truth conditions for quantifiers which are either downward monotone, like *less than four stars*, or quantifiers which specify an exact number or range and therefore are neither increasing nor decreasing, such as *exactly four stars* or *between two and four stars*. To see this, look at the representations we would assign according to our rules to the following examples (here I use <3 and $=2$ as representations of the determiners $\lambda P'\lambda P[\text{card}(P \wedge P') < 3]$ and $\lambda P'\lambda P[\text{card}(P \cap P') = 2]$):

(53) *Less than three circles and less than three stars are connected.*
$\exists P, P'[\forall x, x'[P(x) \wedge P'(x') \rightarrow$ **connected**$^r(x \sqcup x')] \wedge$
$\wedge <3$**(circle)**$(P) \wedge <3$**(star)**$(P')]$

(54) *Exactly two circles and exactly two stars are connected.*
$\exists P, P'[\forall x, x'[P(x) \wedge P'(x') \rightarrow$ **connected**$^r(x \sqcup x')] \wedge$
$\wedge =2$**(circle)**$(P) \wedge =2$**(star)**$(P')]$

According to the representation we derive by our rules, these sentences would be true in Figure 2 sinc, e.g., $P = \{b, c\}$ and $P' = \{g, h\}$ are relevant values for P and P'.

Of course, Barwise's rule (51) does not yield the correct results in these cases either. Therefore, Barwise (1979) proposed different representations for upward and downward monotone quantifiers. In addition, van Benthem proposed a rule for 'exact' quantifiers like *exactly two*. According to Westerståhl (1987), we have the following truth conditions in these cases:

(55) Branching quantifiers for downward monotone quantifiers Q_1, Q_2:
$\left.\begin{matrix} Q_1(A) \\ Q_2(B) \end{matrix}\right\} R \leftrightarrow$

$$\leftrightarrow \exists(P \subseteq A)\exists(P' \subseteq B)$$
$$[Q_1(A)(P) \wedge Q_2(B)(P') \wedge [R \cap (A \times B')] \subseteq P \times P']$$

(56) Branching quantifiers for quantifiers Q_1, Q_2 of the type *exactly n*:

$$\left.\begin{matrix} Q_1(A) \\ Q_2(B) \end{matrix}\right\} R \leftrightarrow$$
$$\leftrightarrow \exists(P \subseteq A)\exists(P' \subseteq B)$$
$$[Q_1(A)(P) \wedge Q_2(B)(P') \wedge [R \cap (A \times B')] = P \times P']$$

Westerståhl himself developed a uniform representation; it partitions a quantifier into an upward monotone and a downward monotone part which are then handled separately. Of course, this defines the semantics of branching quantifiers only for a limited set of quantifiers, namely for so-called continuous quantifiers. But Westerståhl argues that branching quantification only occurs with continuous quantifiers. Furthermore, Barwise has already claimed that acceptable cases of quantifier conjunctions always have quantifiers of the same monotonicity type, so according to that we don't have to assume special rules for the conjunction of quantifiers of mixed nature.

Is there a natural way in the framework developed here to cover the conjunction of quantifiers which are not upward monotone? I think there is, and I will outline it in the rest of this section.

We have to assume that the choice of P, P' is not completely arbitrary, but that we have to choose 'maximal' sets for which it holds that every pair of elements is related in the required way. For example, in **Figure 2**, we have $P = \{b, c, d\}$ and $P' = \{g, h, i\}$ as such maximal sets, and in **Figure 1**, we have $P = \{b\}$ and $P' = \{g\}$, or $P = \{c\}$ and $P' = \{g, h, i\}$, or $P = \{d\}$ and $P' = \{i\}$, or $P = \{e\}$ and $P' = \{j\}$ as such maximal sets. If we choose these maximal sets, then (53) and (54) are false with respect to **Figure 2**.

We have to spell out the maximality condition in the formal representations, and, if possible, give independent motivation for it. To start with the second point, the independent motivation, note that the condition

$$\forall x, x'[P(x) \wedge P'(x') \rightarrow \mathbf{connected}^r(x \sqcup x')],$$

which is a shorter form of

$$\lambda x''\exists x, x'[x'' = x \sqcup x' \wedge P(x) \wedge P'(x')] \subseteq \mathbf{connected}^r,$$

is derived from

$$\mathbf{connected}^r = P \sqcup P' \text{ and } \lambda x''\exists x, x'[x'' = x \sqcup x' \wedge P(x) \wedge P'(x)] \subseteq P \sqcup P'.$$

We have seen in Section 3 that "$\subseteq$" cannot generally be interpreted as "=", as this yields the wrong truth conditions. However, we can still assume that there is a pragmatic rule which forces us to choose P and P' in such a way that it comes as close to "=" as possible — and this can be done by choosing maximal P, P'.

This maximalization can be rendered technically with the help of a family of three-place relations $\mathbf{MAX}_\tau$ of type $(\tau)(\tau)(\tau)t$. $\mathbf{MAX}_\tau$ can be specified as follows:

(57) If $\alpha, \alpha', \alpha''$ are expressions of type τ, then $\mathbf{MAX}_\tau(\alpha, \alpha', \alpha'')$ is true iff
— $\alpha \sqcup \alpha' \sqsubseteq \alpha''$;
— for every β, β' of type τ, if $\alpha \sqsubseteq \beta$ and $\alpha' \sqsubseteq \beta'$ and $\beta \sqcup \beta' \sqsubseteq \alpha''$, then $\beta = \alpha$ and $\beta' = \alpha'$.

With the help of this relation, we can give the following, more specific approximation for generalized conjunction:

(58) If α, α' are of type $\tau = (\sigma_1)\dots(\sigma_n)t$, u, u' and u'' are variables of type σ_1 and $u_2 \dots u_n$ are variables of type $\sigma_2 \dots \sigma_n$, all not occurring free in α, α', then
$\lambda u'' \lambda u_2 \dots \lambda u_n \exists u, u'$
$[\mathbf{MAX}_\tau(u, u', u'') \wedge [\alpha(u) \sqcup \alpha'(u')](u_2)\dots(u_n)]] \sqsubseteq$
$\sqsubseteq \alpha \sqcup \alpha'$.

In the case of (53), we get the following interpretation with this new rule:

(59) *Less than three circles and less than three stars are connected.*
$\exists P, P'[\mathbf{MAX}_{et}(P, P', \mathbf{connected}^r) \wedge <3(\mathbf{circle})(P) \wedge <3(\mathbf{star})(P')]$

This amounts to the claim that there are maximal sets P, P' such that the elements of P and the elements of P' are all connected with each other, and that P contains less than three circles and P' contains less than three stars. This is false in Figure 2 (as $\{b, c, d\}$ and $\{g, h, i\}$ are the only possible instantiations for P and P', and then we have e.g. $\neg <3(\mathbf{circle})(P)$), but true in Figure 1 (for example, with $\{c\}$ and $\{g, h, i\}$ as two instantiations of P and P').

However, this reconstruction still does not represent the truth conditions for non-increasing branching quantifiers given by Barwise and van Benthem, as (59) comes out as true in the following situation as well, and similarly *exactly one circle and exactly one star is connected* (in these cases, $\{e\}$ and $\{j\}$ would be a proper instantiation of P and P'):

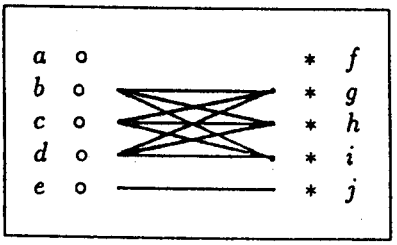


Figure 3

It is unclear whether this corresponds to a natural interpretation of (53) — as intuitions about branching quantifiers, especially with decreasing ones, are quite shaky in general. However, we can show that we can expect readings of these examples which correspond to Barwise's and van Benthem's interpretations. We have to make two assumptions for this: First, that reciprocal predicates can be interpreted as semantically plural, and second, that NPs like *less than three circles* can be interpreted as indefinite NPs.

Before I go into details, I want to point out that these two assumptions are by no means *ad hoc*. First, we have seen in Section 5 that we get a simple treatment of cumulative readings by assuming that an NP like *600 Dutch firms* is an indefinite NP based on a predicate which applies to entities consisting of 600 Dutch firms. It is a natural extension of this treatment to assume that an NP like *less than three circles* may apply to entities which are circles and which count less than three circles (say, two circles). That this analysis is a plausible one, is also shown by the fact that we can point to, say, a heap of apples and say: *That's less than twenty apples*. It can be argued that *less than twenty apples* is a predicate in this case which applies to the object pointed to.

Second, it is well-known that the rule for reciprocals we assumed so far is too strong, as natural-language reciprocals have 'weaker' interpretations (cf. Langendoen, 1979). Two natural examples are (60a,b):

(60) a. *The ten children took one another by the hand.*
b. *The prisoners released each other.*

For example, (60a) can be true if the children took one another by the hand in pairs. We can get such weak interpretations of reciprocals if we assume that the reciprocal predicate is taken as semantically plural. For example, if there are ten children $x_1, \ldots x_5, y_1 \ldots y_5$ such that we have $\mathbf{took\_by\_the\_hand}^{\cup}(x_i \oplus y_i)$, then, according to the rule for plural predicates and associativity of "$\oplus$", we also have $\mathbf{took\_by\_the\_hand}^{\cup p}(x_i \oplus \ldots \oplus y_5)$.

These two assumptions accepted, we can get Barwise's interpretation for decreasing quantifiers by assuming that we have an indefinite interpretation of the quantifiers and a plural interpretation of the reciprocal predicate. If we interpret *less than three* as

$$<3_N = \lambda P' \lambda P \exists x[P(x) \wedge P'(x) \wedge \mathrm{N}(x) < 3],$$

we get the following interpretation:

(61) *Less than three circles and less than three stars are connected.*
$\exists P, P'$
$[\mathrm{MAX}_{et}(P, P', \mathbf{connected}^{\cup p}) \wedge$
$\wedge <3_N(\mathbf{circle}^p)(P) \wedge <3_N(\mathbf{star}^p)(P')]$

This is false in figures 1, 2 and 3, as it should be, according to the rule of Barwise.

Note that we can use a similar representation for increasing quantifiers, like *more than three circles and more than three stars are connected.* If we interpret NPs like *more than three circles* as

$$\lambda P' \lambda P \exists x[P(x) \wedge P'(x) \wedge \mathrm{N}(x) > 3],$$

and *connected* as a plural reciprocal predicate, then this example would come out true with respect to figures 1 and 3, and false with respect to Figure 2.

However, these analyses still do not cover van Benthem's interpretation for 'exact' quantifiers. If we interpret *exactly four circles* as

$$\lambda P' \lambda P \exists x[P(x) \wedge P'(x) \wedge \mathrm{N}(x) = 4],$$

then a sentence like *exactly four circles and exactly four stars are connected* would come out true in figures 1 and 3. Van Benthem's rule, on the other hand, requires that there is a connection between every element in $\{b, c, d, e\}$ and every element in $\{g, h, i, k\}$, so that it cannot be true these cases.

We can get van Benthem's interpretation as well, if we assume that *be connected* is interpreted in the strict sense, as a non-plural predicate $\mathbf{connected}^r$, and that *exactly* is treated as a focusing operator in the sense of Jacobs (1983) and Rooth (1985). These operators have to be analyzed in some form of alternative semantics (see von Stechow, 1988, for an overview). Complications and variations aside, the focusing operator *exactly* can be analyzed as expressing the fact that the item in its focus is the only item of a range of alternatives for which the sentence is true and is maximally informative. For example, a sentence like

(62) *John has exactly three children.*

expresses the fact that 3 is the most informative value of n for which the sentence pattern *John has n children* is true. In the case of

(63) *Exactly three circles and exactly three stars are connected.*

I assume that the first occurrence of *exactly* focuses on the first occurrence of *three*, and the second occurrence of *exactly* focuses on the second occurrence of *three*. So what (63) says is that $n = 3$ and $m = 3$ are the maximally informative values for which the sentence pattern *n circles and m stars are connected* is true. When we interpreted *connected* strictly, that is, as $\mathbf{connected}^r$, then this is true only if P, P' in

$$\mathrm{MAX}_{et}(P, P', \mathbf{connected}^r) \wedge = \mathrm{m}(\mathbf{circle})(P) \wedge = \mathrm{n}(\mathbf{star})(P)$$

are unique, that is, there is only one maximal partition of $\mathbf{connected}^r$ into P, P'. If P, P' are not unique, then n and m could not be maximally informative. Thus we get van Benthem's truth conditions.

To conclude this section, I wish to emphasize that sentences with branching quantifiers, as outlined here, are not treated as a uniform phenomenon. With a

sentence like *DET circles and DET stars are connected* we get a whole variety of readings, depending on whether *connected* is interpreted as singular (connected^{r}) or plural (connected^{rp}), and wether the DET's are interpreted as adjectival number words, as quantifiers, or as focusing operators. These factors may contribute to the fact that the truth conditions of sentences with branching quantifiers are notoriously hard to specify.

7. OUTLOOK

In this final section, I will discuss possible modifications of the join operation and the relation between Boolean and non-Boolean conjunction.

An obvious generalization of the conjunction is to assume operations of arbitrary arity, instead of two-place operations. For example, we have a three-place operation in the following case:

(64) *The flag is red, white, and green.*

Of course, as soon as we assume arbitrary arity for the basic operations $\oplus$ and $\wedge$, we generate operations of arbitrary arity for the corresponding operations in other types as well.

If we assume associativity of the basic operations "$\oplus$" and "$\wedge$", the extension to arbitrary arity is straightforward. However, it is not clear whether we really should think of "$\oplus$" as being associative (cf. Link, 1984b, Landman, 1988, Hoeksema, 1988, Lønning, 1989). Some relevant examples (the brackets are intended to represent the relevant readings):

(65) a. *Napoleon and (Wellington and Blücher) fought against each other.*
b. ?*((Mary and John) and (Lisa and Stefan)) and ((Ann and Bill) and (Steffi and Boris)) played against each other in the mixed-double semi-finals.*

The intended bracketing can be indicated by the intonational structure; roughly, bracket in the sentence corresponds to an optional pause. However, it is unclear whether we should posit bracketings of more than one level; Lønning (1989), who proposed example (65b) as a case of two levels, is not sure about its status. But this restriction may well be a performance restriction, especially due to the marginal possibilities of marking the brackets.

I will not go into the solutions proposed for non-associativity here, but simply want to point out that we have to assume it for non-Boolean conjunction of other types as well. Some examples:

(66) a. *The French and (the English and the Prussian) soldiers fought against each other.*
b. *The (red and white) and (yellow and green) flags were put together.*

Therefore we have to assume that the non-associativity of non-Boolean conjunction for entities has to be passed to the non-Boolean conjunction for expresssions of other types.

Furthermore, it might be the case that non-Boolean conjunction is not symmetric (cf. Link, 1984):

(67) a. *John and Mary are husband and wife.*
b. *John and Mary are 28 and 31 years old (respectively).*
c. *The flag is red and white.*

In these examples, we cannot switch the order of the conjoined elements *salva veritate*. In (67a,b), which have two conjunctions each, the order of the conjuncts have to match (this is well-known for the *respectively*-construction, which has been cited as a case where natural language is not context-free). (67c) evokes some spatial order, either from top to bottom or from left to right. This is similar to natural uses of Boolean conjunctions, which, in the case of non-stative sentences, evoke a temporal order (cf. *Mary married John and got pregnant* vs. *Mary got pregnant and married John*).

Again, I will not go into the solutions which can be proposed for the non-symmetry of conjunction. One attractive possibility is, I think, to assume that conjunction is basically symmetric, but subject to other semantic modules for which the order of utterances is decisive. One could think of different levels of semantic representations which interact with each other, similar to the assumption of different tiers in phonology. What is important here is that any solution which is proposed for non-Boolean conjunctions of entities should be generalized for non-Boolean conjunction of expressions of other types.

Another point which should be stressed here is that the generalization of conjunction developed in this article offers a uniform notion of a conjuction which captures both the Boolean and non-Boolean variety. We have seen that Boolean conjunction is basically an operation on type t, and non-Boolean conjunction is an operation on type e. However, to derive an interesting generalization for non-Boolean conjunction we had to make use of Boolean conjunction as well. In particular, the notion of conjoinable types for Boolean conjunction and for non-Boolean conjunction coincide, if we exclude the basic types. This suggests that Boolean and non-Boolean conjunctions can be traced back to one uniform notion of conjunction. This also would explain why languages typically use one word, as for example English *and*, for both types.

Indeed, with (31) we already have proposed such a uniform and generalized notion of conjunction, as this definition applies to general Boolean conjunction as well. To see this, look at the following examples:

(68) a. *new and expensive* [type *et*]
$\lambda x'' \exists x, x' [x'' = x \sqcup x' \wedge \mathbf{new}(x) \wedge \mathbf{expensive}(x')]$

b. *green and white* [type *et*]
$\lambda x \exists x, x'[x'' = x \sqcup x' \wedge \mathbf{green}(x) \wedge \mathbf{white}(x')]$

In (68a), we get Boolean conjunction if we assume $x = x'$. We can assume that this is the default case for the interpretation of x and x'. In (68b), this default interpretation cannot be carried through, because **green** and **white** are contradictory. Therefore we have to assume that $x \neq x'$, and arrive at non-Boolean conjunction.

Thus, Boolean conjunction ends up as a particularly simple case of non-Boolean conjunction.

NOTE

* I had the chance to discuss the ideas developed here with several colleagues — including Barbara Partee during a stay in Amherst in May 1988, Godehard Link, Sebastian Löbner and Jeff Pelletier at a workshop in Tübingen in June 1989, Fritz Hamm, and Zuzana Dobes. Thanks to them all.

REFERENCES

Barwise, J. 1979. 'On Branching Quantifiers in English'. *Journal of Philosophical Logic* 8, 47–80.

van Benthem, J. 1986. 'The Semantics of Variety in Categorial Grammar'. In: W. Buszkowski *et al*, *Categorial Grammar*. John Benjamins, Amsterdam.

van Benthem, J. 1987. 'Categorial Grammar and Type Theory'. ITLI Prepublication Series 87-07. To appear in *Linguistics and Philosophy*.

Gazdar, G. 1980. 'A Cross-Categorial Semantics for Coordination'. *Linguistics and Philosophy* 3, 407–409.

Gillon, B. 1987. 'The Readings of Plural Noun Phrases in English'. *Linguistics and Philosophy* 10, 199–220.

Grice, P. 1967. *Logic and Conversation*. Unpublished ms. of the William James Lectures, Harvard University. Partly published in P. Cole and J. Morgan, eds., *Syntax and Semantics 3: Speech Acts*. Academic Press, New York, 1976. pp. 41–58.

Hintikka, J. 1974. 'Quantifiers vs. Quantification Theory'. *Linguistic Inquiry* 5, 153–177.

Hoeksema, J. 1983. 'Plurality and Conjuction'. In: Alice ter Meulen, ed., *Studies in Model-Theoretic Semantics*. Foris, Dordrecht.

Hoeksema, J. 1988. 'The Semantics of Non-Boolean *and*'. *Journal of Semantics* 6, 19–40.

Jacobs, J. 1983. *Fokus und Skalen. Zur Syntax und Semantik von Gradpartikeln im Deutschen*. Niemeyer, Tübingen.

Keenan, E. and L.M. Faltz. 1985. *Boolean Semantics for Natural Language.* Reidel, Dordrecht.

Krifka, M. 1986. *Nominalreferenz und Zeitkonstitution. Zur Semantik von Massentermen, Pluraltermen und Aspektklassen.* Dissertation, Universität München. To be published by Wilhelm Fink, München, 1989.

Landman, F. 1988. 'Groups, Plural Individuals and Intentionality'. In: J. Groenendijk, M. Stokhof and F. Veltman, eds., *Proceedings of the Sixth Amsterdam Colloquium.* ITLI, University of Amsterdam. pp. 197–217.

Langendoen, T.D. 1978. 'The Logic of Reciprocity'. *Linguistic Inquiry* 9, 177–197.

Link, G. 1983. 'The Logical Analysis of Plurals and Mass Terms: A Lattice-Theoretical Approach'. In: R. Bäuerle, Chr. Schwarze, A. von Stechow, eds., *Meaning, Use and Interpretation of Language.* De Gruyter, Berlin and New York. pp. 303–323.

Link, G. 1984. 'Hydras: On the Logic of Relative Clause Constructions with Multiple Heads'. In: F. Landman and F. Veltman, eds., *Varieties of Formal Semantics.* Foris, Dordrecht. pp. 245–257.

Link, G. 1984b. 'Plurals'. To appear in A. von Stechow and D. Wunderlich, eds., *Handbuch der Semantik.* Athenäum, Kronberg.

Lønning, J,T. 1989. 'Some Aspects of the Logic of Plural Noun Phrases'. COSMOS-Report 11, Department of Mathematics, University of Oslo.

Massey, G.J. 1976. 'Tom, Dick, and Harry, and all the king's men'. *American Philosophical Quarterly* 13, 89–107.

Partee, B.H. and M. Rooth. 1983. 'Generalized Conjunction and Type Ambiguity'. In: R. Bäuerle, Chr. Schwarze, A. von Stechow, eds, *Meaning, Use and Interpretation of Language.* De Gruyter, Berlin and New York. pp. 361–383.

Rooth, M. *Association with Focus.* Ph.D. Dissertation, University of Massachusetts at Amherst.

Scha, R.J.H. 1981. 'Distributive, Collective, and Cumulative Quantification'. In: J.A. Groenendijk, T.M. Jansen, M. Stokof, eds., *Formal Methods in the Study of Language.* Mathematical Centre Tracts 135, Amsterdam. Part 2, pp. 483–512.

von Stechow, A. 1974. 'η-λ kontextfreie Sprachen: Ein Beitrag zur natürlichen formalen Semantik'. *Linguistische Berichte* 34, 1–33.

von Stechow, A. 1988. 'Focusing and Backgrounding Operators'. Arbeitspapier 6, Fachgruppe Sprachwissenschaft, Universität Konstanz.

Wald, J.D. 1977. *Stuff and Words: A Semantic and Linguistic Analysis of Non-Singular Reference.* Ph.D. dissertation, Brandeis University.

Westerståhl, D. 1987. 'Branching Generalized Quantifiers and Natural Language'. In: Peter Gärdenfors, ed., *Generalized Quantifiers. Linguistic and Logical Approaches.* Reidel, Dordrecht. pp. 269–298.