Logical Form Equivalence: the Case of Referring Expressions Generation

Kees van Deemter
ITRI
University of Brighton
Brighton BN2 4GJ
United Kingdom
Kees.van.Deemter@itri.brighton.ac.uk

Magnús M. Halldórsson
Computer Science Dept.
University of Iceland,
Taeknigardur, 107 Reykjavik, Iceland
and Iceland Genomics Corp., Reykjavik
mmh@hi.is

Abstract

We examine the principle of co-extensivity which underlies current algorithms for the generation of referring expressions, and investigate to what extent the principle allows these algorithms to be generalized. The discussion focusses on the generation of complex Boolean descriptions and sentence aggregation.

1 Logic in GRE

A key question regarding the foundations of Natural Language Generation (NLG) is the problem of *logical form equivalence* (Appelt 1987). The problem goes as follows. NLG systems take semantic expressions as input, usually formulated in some logical language. These expressions are governed by rules determining which of them count as 'equivalent'. If two expressions are equivalent then, ideally, the NLG program should verbalize them in the same ways. (Being equivalent, the generator would not be warranted in distinguishing between the two.) The question is: what is the *proper* relation of equivalence? Appelt argued that classical logical equivalence (i.e., having the same truth conditions) is not a good candidate. For example, $p \rightarrow q$ is logically equivalent with $\neg q \rightarrow \neg p$, yet – so the argument goes – an NLG system should word the two formulas differently. Shieber (1993) suggested that some more sophisticated notion of equivalence is needed, which would count fewer semantic expressions as equivalent.[1] In the present paper, a different response to the problem is explored, which keeps the notion of equivalence classical and prevents the generator from distinguishing between inputs that are logically equivalent (i.e., inputs that have the same truth conditions). *Pragmatic constraints* determine which of all the logically equivalent semantic expressions is put into words by the NLG program. Whereas this programme, which might be called 'logic-oriented' generation, would constitute a fairly radical departure from current practice if applied to all of NLG (Krahmer & van Deemter (forthcoming); Power 2000 for related work), the main aim of the present paper is modest: to show that logic-oriented generation is standard practice in connection with the generation of *referring expressions* (GRE). More specifically, we show the *semantics* of current GRE algorithms to be guided by a surprisingly simple principle of *co-extensivity*, while their pragmatics is guided by Gricean Brevity.

Our game plan is as follows. In section 2, we illustrate the collaboration between Brevity and co-extensivity, focussing on 'simple' referring expressions, which intersect atomic properties (e.g., 'dog' and 'black'). Section 3 proceeds by showing how other algorithms use the principle to legitimize the creation of more elaborate structures involving, for example, complex Boolean combinations (e.g., the union of some properties, each of which is the intersection of some atomic prop-

[1] See also van Deemter (1990) where, on identical grounds, a variant of Carnap-style intensional isomorphism was proposed as an alternative notion of equivalence.

erties). This part of the paper will borrow from van Deemter (2001), which focusses on computational aspects of GRE. Section 4 asks how the principle of co-extensivity may be generalized beyond GRE and questions its validity.

2 Intersective reference to sets of domain objects

The Knowledge Base (KB) forming the input to the generator will often designate objects using the jargon of computerized databases, which is not always meaningful for the reader/hearer. This is true, for example, for an artificial name (i.e., a database key) like '$\#Jones083$' when a person's proper name is not uniquely distinguishing; it is also true for objects (e.g., furniture, trees, atomic particles) for which no proper names are in common usage. In all such cases, the NLG program has to 'invent' a description that enables the hearer to identify the *target object*. The program transforms the original semantic structure in the KB into some other structure.

Let us examine simple references first. Assume that the information used for interpreting a description is stored in a KB representing what properties are true of each given object. In addition to these properties, whose extensions are shared between speaker and hearer, there are other properties, which are being conveyed from speaker to hearer. For example, the speaker may say *'The white poodle is pregnant'*, to convey the new information that the referent of 'the white poodle' is pregnant. GRE 'sees' the first, shared KB only. We will restrict attention to the problem of determining the semantic content of a description, leaving linguistic realization aside. (Cf. Stone and Webber 1998, Krahmer and Theune 1999, which interleave linguistic realization and generation.) Accordingly, 'Generation of Referring Expressions' (GRE) will refer specifically to content determination. We will call a GRE algorithm *complete* if it is successful whenever an individuating description exists. Most GRE algorithms are limited to individual target objects (for an exception, Stone 2000), but we will present ones that refer to *sets* of objects (Van Deemter 2000); reference to an individual r will equal reference to the singleton set $\{r\}$.

2.1 The Incremental Algorithm

Dale and Reiter (1995) proposed an algorithm that takes a shared KB as its input and delivers a set of properties which jointly identify the target. Descriptions produced by the algorithm fullfill the criterion of *co-extensivity*. According to this principle, a description is semantically correct if it has the target as its referent (i.e., its extension). The authors observed that a semantically correct description can still be unnatural, but that naturalness is not always easy to achieve. In particular, the problem of finding a ('Full Brevity') description that contains the minimum number of properties is computationally *intractable*, and human speakers often produce non-minimal descriptions. Accordingly, they proposed an algorithm that *approximates* Full Brevity, while being of only linear complexity. The algorithm produces a finite set L of properties $P_1, ..., P_n$ such that the intersection of their denotations $[[P_1]] \cap ... \cap [[P_n]]$ equals the target set S, causing L to be a 'distinguishing description' of S. The properties in L are selected one by one, and there is no backtracking, which is why the algorithm is called Incremental. As a result, some of the properties in L may be logically superfluous.

For simplicity, we will focus here on *properties*, without separating them into Attributes and Values (see also Reiter and Dale 2000, section 5.4.5). Accordingly, reflecting the fact that not all properties are equally 'preferred', they are ordered linearly in a list $\mathbb{P}$, with more preferred ones preceding less preferred ones. We also simplify by not taking the special treatment of head nouns into account. Suppose S is the target set, and C is the set of elements from which S is to be selected.[2] The algorithm iterates through $\mathbb{P}$; for each property, it checks whether it would rule out at least one member of C that has not already been ruled out; if so, the property is added to L. Members that are ruled out are removed from C. The process of expanding L and contracting C continues until $C = S$; if and when this condition is met, L is a distinguishing set of properties.

[2] We have chosen a formulation in which C is a superset of S, rather than a 'contrast set', from whose elements those of S must be distinguished (Dale and Reiter 1995). The difference is purely presentational.

$L := \emptyset$ { L is initialized to the empty set }

For each $P_i \in \mathbb{P}$ do

If $S \subseteq [[P_i]]$ & $C \not\subseteq [[P_i]]$ {P_i removes distractors from C but keeps all elements of S}

Then do

$L := L \cup \{P_i\}$ {Property P_i is added to L}

$C := C \cap [[P_i]]$ {All elements outside $[[P_i]]$ are removed from C}

If $C = S$ *then Return* L { `Success` }

Return `Failure` { All properties in $\mathbb{P}$ have been tested, yet $C \neq S$ }

This algorithm, D&R$_{\text{Plur}}$, constructs better and better approximations of the target set S. Assuming (cf. Dale and Reiter 1995) that the tests in the body of the loop take some constant amount of time, the worst-case running time is in the order of n_a (i.e., $O(n_a)$) where n_a is the total number of properties.

3 Reference using Boolean descriptions

Based on co-extensivity, the algorithms discussed construct an intersective Boolean expression (i.e., an expression of the form $P_1 \cap ... \cap P_n$, where $P_1, .., P_n$ are atomic) that has the target set as its extension. But, intersection is not the only operation on sets. Consider a KB whose domain is a set of dogs (a, b, c, d, e) and whose only Attributes are TYPE and COLOUR:

TYPE: dog $\{a, b, c, d, e\}$, poodle $\{a, b\}$
COLOUR: black $\{a, b, c\}$, white $\{d, e\}$

In this situation, D&R$_{Plur}$ does not allow us to individuate any of the dogs. In fact, however, the KB should enable one to refer to dog *c*, since it is the only black dog that is *not* a poodle:

$$\{c\} \;=\; black \cap \overline{poodle}$$

A similar gap exists where *disjunctions* might be used. For example, D&R$_{Plur}$ does not make the set of dogs that are either white or poodles referable, whereas it *is* referable in English, e.g., 'The white dogs and the poodles'.

Presently, we will investigate how GRE can take negation and disjunction into account. Section 3.1 will ask how GRE algorithms can achieve Full Boolean Completeness; section 3.2, which follows Van Deemter (2001), adds Brevity as a requirement. Boolean descriptions do the same thing that intersective descriptions do, except in a more dramatic way: they 'create' even more structure. As a result, the problem of optimizing these structures with respect to constraints such as Brevity becomes harder as well.

As a first step, we show how one can tell which targets are identifiable given a set of properties and set intersection. We calculate, for each element d in the domain, the 'Satellite set' of d, that is, the intersection of the extensions of all the properties that are true of d. Taking all extensions from our example,

$$\begin{array}{l} Satellites(a) = \\ dog \cap poodle \cap black = \{a, b\} \\ Satellites(b) = \\ dog \cap poodle \cap black = \{a, b\} \\ Satellites(c) = \\ dog \cap black = \{a, b, c\} \\ Satellites(d) = \\ dog \cap white = \{d, e\} \\ Satellites(e) = \\ dog \cap white = \{d, e\} \end{array}$$

If two objects occur in the same Satellite set then no intersective description can separate them: any description true of one must be true of the other. It follows, for example, that no object in the domain is uniquely identifiable, since none of them occurs in a Satellite set that is a singleton.

3.1 Boolean descriptions (i): generate and simplify

Boolean completeness is fairly easy to achieve *until* further constraints are added. Suppose the task is to construct a description of a target set S, given a set $\mathbb{P}$ of atomic properties, without any further constraints. We will discuss an algorithm that starts by calculating a generalized type of Satellite sets, based on all atomic properties and their *negations*.

Construction of Satellite sets:

$\mathbb{P}_{neg} := \mathbb{P} \cup \{\overline{P_i} : P_i \in \mathbb{P}\}$

For each $d \in S$ do

$S_d := \{A : A \in \mathbb{P}_{neg} : d \in [[A]]\}$
$Satellites(d) = \bigcap_{A \in S_d} ([[A]])$

First, the algorithm adds to $\mathbb{P}$ the properties whose extensions are the complements of those in $\mathbb{P}$. Then it calculates, for each element d in S, $Satellites(d)$ by lining up all the properties in $\mathbb{P}_{neg}$ that are true of d, then taking the intersection of their extensions. Satellite sets may be exploited for the construction of descriptions by forming the union of a number of expressions, each of which is the intersection of the elements of S_d (for some $d \in S$).

Description By Satellite sets (DBS**)**:

`Description` $:= \{S_d : d \in S\}$

`Meaning` $:= \bigcup_{d \in S}(Satellites(d))$

If `Meaning` = S
then Return `Description`
else `Fail`

(Note that `Description` is returned instead of `Meaning`, since the latter is just the set S.) `Description` is a set of sets of sets of domain objects. As is made explicit in `Meaning`, this third-order set is interpreted as a union of intersections. A `Description` is successful if it evaluates to the target set S; otherwise the algorithm returns `Fail`. If `Fail` is returned, *no Boolean description of S is possible*:

> **Full Boolean Completeness:** For any set S, S is obtainable by a sequence of boolean operations on the properties in $\mathbb{P}$ if and only if $\bigcup_{d \in S}(Satellites(d))$ equals S.
>
> **Proof:** The implication from right to left is obvious. For the reverse direction, suppose $S \neq \bigcup_{d \in S}(Satellites(d))$. Then for some $e \in S$, Satellites(e) contains an element e' that is not in S. But $e, e' \in$ Satellites(e) implies that every set in $\mathbb{P}$ must either contain both of e and e', or neither. It follows that S, which contains only one of e, e', cannot be obtained by a combination of Boolean operations on the sets in $\mathbb{P}$.

DBS is computationally cheap: it has a worst-case running time of $O(n.p)$, where n is the number of objects in S, and p the number of atomic properties. Rather than searching among all the possible unions of some large set of sets, a set $S = \{s_1, .., s_n\}$ is described as the union of n Satellites sets, each of which equals the intersection of those (at most $2n$) sets in $\mathbb{P}_{neg}$ that contain s_i. Descriptions can make use of the Satellite sets computed for earlier descriptions, causing a further reduction of time. Satellites sets can even be calculated *off-line*, for all the elements in the domain, before the need for specific referring expressions has arisen.[3]

Unfortunately, the descriptions produced by DBS tend to be far from brief:

$S_c = \{dog, black, \overline{poodle}, \overline{white}\}.$
$Satellites(c) = \{c\}$
$S_d = \{dog, white, \overline{poodle}, \overline{black}\}.$
$Satellites(d) = \{d, e\}$
$S_e = \{dog, white, \overline{poodle}, \overline{black}\}.$
$Satellites(e) = \{d, e\}$

To describe the target set $S = \{c, d, e\}$, for example, the algorithm generates the `Description` $\{S_c, S_d, S_e\}$. Consequently, the boolean expression generated is

$(dog \cap black \cap \overline{poodle} \cap \overline{white}) \cup$
$(dog \cap white \cap \overline{poodle} \cap \overline{black}) \cup$
$(dog \cap white \cap \overline{poodle} \cap \overline{black}).$

But of course, a much shorter description, $\overline{poodle}$, would have sufficed. What are the prospects for simplifying the output of DBS? Unfortunately, perfect simplification (i.e., Full Brevity) is incompatible with computational tractibility. Suppose brevity of descriptions is defined as follows: d_1 is less brief than d_2 if *either* d_1 contains only atomic properties while d_2 contains non-atomic properties as well, *or* d_1 contains more Boolean operators than d_2. Then the intractability of Full Brevity for intersections of atomic properties logically implies that of the new algorithm:

> **Proof:** Suppose an algorithm, BOOL, produced a maximally brief Boolean description whenever one exists. Then whenever a target set S can be described as an intersection of *atomic* properties, BOOL(S) would be a maximally brief intersection of atomic properties, and this is inconsistent with the intractability of Full Brevity for intersections of atomic properties.

[3] Compare Bateman (1999), where a KB is compiled into a format that brings out the commonalities between objects before the content of a referring expression is determined.

This negative result gives rise to the question whether Full Brevity may be *approximated*, perhaps in the spirit of Reiter (1990)'s 'Local Brevity' algorithm which takes a given intersective description and tests whether any set of properties in it may be replaced by one other property. Unfortunately, however, simplification is much harder in the Boolean setting. Suppose, for example, one wanted to use the Quine-McCluskey algorithm (McCluskey 1965), known from its applications to electronic circuits, to reduce the number of Boolean operators in the description. This would go only a small part of the way, since Quine-McCluskey assumes logical independence of all the properties involved. Arbitrarily complex information about the *extensions* of properties can affect the simplification task, and this reintroduces the spectre of computationally intractability.[4] Moreover, the 'generate and simplify' approach has other disadvantages in addition. In particular, the division into two phases, the first of which generates an unwieldy description while the second simplifies it, makes it psychologically unrealistic, at least as a model for speaking. Also, unlike the Incremental algorithm, it treats all properties alike, regardless of their degree of preferedness. For these reasons, it is worthwhile to look for an alternative approach, which takes the Incremental algorithm as its point of departure. This does not mean that DBS is useless: we suggest that it is used for determining whether a Boolean description exists; if not, the program returns `Fail`; if a Boolean description is possible, the computationally more expensive algorithm of the following section is called.

3.2 Boolean descriptions (ii): extending the Incremental algorithm

In this section, we will explore how the Incremental algorithm may be generalized to take all Boolean combinations into account. Given that the Incremental algorithm deals with intersections between sets, Full Boolean Completeness can be achieved by the addition of *set difference*. Set difference may be added to D&R$_{\text{Plur}}$ as follows. First we add negations to the list of atomic properties (much like the earlier DBS algorithm). Then D&R$_{\text{Plur}}$ runs a number of times: first, in phase 1, the algorithm is performed using all positive and negative literals; if this algorithm ends before $C = S$, phase 2 is entered in which further distractors are removed from C using negations of intersections of two literals, and so on, until either $C = S$ (`Success`) or all combinations have been tried (`Failure`). Observe that the negation of an intersection comes down to set union, because of De Morgan's Law: $\overline{P_1 \cap ... \cap P_n} = \overline{P_1} \cup ... \cup \overline{P_n}$. Thus, phase 2 of the algorithm deals with disjunctions of length 2, phase 3 deals with disjunctions of length 3, *etcetera*.

A schematic presentation will be useful, in which $P_{+/-}$ stands for any positive or negative literal. The *length* of a property will equal the number of literals occurring in it. We will say that a D&R phase *uses* a set of properties X if it loops through the properties in X (i.e., X takes the place of $\mathbb{P}$ in the original D&R$_{\text{Plur}}$).

D&R$_{Boolean}$:

1. Perform a D&R phase using *all properties of the form* $P_{+/-}$;
if this phase is successful then stop, otherwise go to phase (2).

2. Based on the Values of L and C coming out of phase (1),
perform a D&R phase using *all properties of the form* $\overline{P_{+/-} \cap P_{+/-}}$;
if this phase is successful then stop, otherwise go to phase (3).

3. Based on the Values of L and C coming out of phase (2),
perform a D&R phase using *all properties of the form* $\overline{P_{+/-} \cap P_{+/-} \cap P_{+/-}}$;
if this phase is successful then stop, otherwise go to phase (4).

Etcetera.

One can require without loss of generality that no property, considered at any phase, may have

[4] For example, the automatic simplificator at `http://logik.phl.univie.ac.at/chris/qmo-uk.html.05` can only reduce our description to $dog \cap \overline{poodle}$ if it 'knows' that being black, in this KB, is tantamount to not being white. To reduce even further, the program needs to know that all elements in the domain are dogs. In more complex cases, equalities between complex intersections and/or unions can be relevant.

different occurrences of the same atom.[5] Since, therefore, at phase n, there is room for properties of length n, the maximal number of phases equals the total number of atomic properties in the language.

Note that D&R$_{Boolean}$ is incremental in two different ways: within a phase, and from one phase to the next. The latter guarantees that shorter disjunctions are favoured over longer ones. Once a property has been selected, it will not be abandoned even if properties selected during later phases make it superfluous. As a result, one may generate descriptions like $white \cap (\overline{\overline{cat} \cap \overline{dog}})$ (i.e., 'white (cats and dogs)') when $\overline{\overline{cat} \cap \overline{dog}}$ (i.e., 'cats and dogs') would have sufficed. The empirical correctness of this type of incrementality is debatable, but repairs can be made if needed.[6] Unfortunately, however, the algorithm is not tractable as it stands. To estimate its running time as a function of the number of properties (n_a) in the KB and the number of properties used in the description (n_l), note that the maximal number of properties to be considered equals

$$\sum_{i=1}^{n_l} 2 \left(\begin{array}{c} n_a \\ i \end{array} \right) = \sum_{i=1}^{n_l} 2 \frac{n_a!}{i!(n_a - i)!}$$

(The factor of 2 derives from inspecting both each atom and its negation.) If $n_l << n_a$ then this is in the order of $n_a^{n_l}$.[7] To avoid intractability, the algorithm can be pruned. No matter where this is done, the result is polynomial. By cutting off after phase (1), for example, we generate negations of atomic properties only, producing such descriptions as 'the black dog that is *not* a poodle', while disregarding more complex descriptions. As a result, Boolean completeness is lost, but only for references to non-singleton sets.[8] The number of properties to be considered by this simpler algorithm equals $(n_a)^2 + 2n_a - 1$. If one wanted to produce more complex descriptions like $\overline{\overline{white \cap dog} \cap \overline{poodle}}$ ('the white dogs and the poodles'), the algorithm might be cut off one phase later, leading to a worst-case running time of $O(n_a^3)$.

[5] For example, it is useless to consider the property $\overline{P_1 \cap P_2 \cap \overline{P_1}}$, which must be true of any element in the domain, or the property $\overline{P_1 \cap P_2 \cap P_1}$, which is equivalent to the earlier-considered property $\overline{P_1 \cap P_2}$.

[6] E.g., phases might run separately before running in combination: first phase 1, then 2, 1&2, 3, 1&3, 2&3, 1&2&3, etc. (Suggested by Richard Power.)

[7] Compare an analogous argument in Dale and Reiter (1995, section 3.1.1).

[8] If $P_1(A \cup B) \cap P_2$ individuates the individual x then either $P_1 \cap A \cap P_2$ or $P_1 \cap B \cap P_2$ does. Where singletons are concerned, set union does not add descriptive power.

4 Discussion

Hybrid algorithms, which make use of elements of both algorithms, are possible. In particular, the idea of incrementality can be injected into the *generate and simplify* algorithm of section 3.1, firstly, at the level of the construction of Satellite sets (i.e., by letting S_d take into account only those properties from $\mathbb{P}_{neg}$ that are necessary for singling out d) and, secondly, where the union of the $Satellites$ is formed in DBS (i.e., by taking only those $Satellites$ into account that change the resulting `Meaning`). Instead of offering any details on this, we choose to discuss a more general problem relating to the problem of *Logical Form Equivalence* that was noted in section 1.

GRE algorithms exploit a principle of coextensivity for determining what are semantically correct ways of referring to an entity. Thus, consistent with the idea of logic-oriented generation, the structure of the description is not *prejudged* by the syntactic form of the input to the generator (i.e., by the fact that the input contains an individual constant rather than a description). As a result, GRE can 'create' substantial amounts of new semantic structure containing, for example, any number of Boolean operators. In section 1, it was suggested that the processes of structure transformation used in GRE might have wider applicability. The present section questions the validity of coextensivity as a general principle, first for GRE (section 4.1), then for sentence aggregation (section 4.2).

4.1 Descriptions in intensional contexts

The principle of co-extensivity is not valid in *intensional* contexts. For example, consider

> **(a)** John knows that [the red button] is dangerous
> **(b)** John knows that [the rocket launching button] is dangerous.

(a) and (b) have different truth conditions even if speaker and hearer share the information that *the red button* is *the rocket launching button*. In other words, the two descriptions are not interchangeable, even if reader and hearer know them to be coextensive; what would be necessary is for *John* to know that they are coextensive. Extending current GRE algorithms to the generation of referring expressions in intensional contexts is likely to be a difficult enterprise.

Failiure of substitutivity in intensional contexts is, of course, a well-known problem, for which various solutions are available on the theoretical market (e.g., Montague 1973, Barwise and Perry 1983). But one has to wonder whether co-extensivity is *ever* really sufficient. Consider extensional truncations of (a) and (b), such as may be generated from an input I(1) (where the semantic predicate 'dangerous' is abbreviated as D and a is a constant referring to the button):

I(1) $D(a)$
(a′) [The red button] is dangerous
(b′) [The rocket launching button] is dangerous

Suppose (a) and (b) are semantically interchangeable (e.g., when said to someone who knows the colours and functions of all objects in the domain), so a choice between them can only be motivated by an appeal to pragmatic principles. Even then, it is difficult to accept that the *same* choice must be made regardless whether the input to the generator is I(1), I(2) or I(3): (Here $rl(x)$ says that x is for launching rockets; ι is the Russellian description operator.)

I(2) $[(\iota x)(red(x)\&button(x))] : D(x)$
I(3) $[(\iota x)(rl(x)\&button(x))] : D(x).$

Co-extensivity, after all, does not allow the generator to distinguish between I(1), I(2) and I(3), because these three have the same extension! Perhaps a weakened version of co-extensivity is needed which allows the generator to *add* new structure (e.g., when the input is I(1)), but not to *destroy* existing structure (e.g., when the input is I(2) or I(3)). It is, however, unclear what the theoretical justification for such a limitation of co-extensivity might be.

Note that these problems become more dramatic as GRE is able to 'invent' more structure (e.g., elaborate Boolean structure, as discussed in section 3). Crucially, we have to assume that, in an ideal generator, there are many other pragmatic constraints than Brevity. One description can be chosen over another, for example, because it fullfills some additional communicative goal (Dale and Reiter 1995, section 2.4; also Stone and Webber 1998). Depending on the communicative goal, for example, (b) might be chosen over (a) because the properties that identify the button *also* explain why it is dangerous. Brevity will then have to be interpreted as 'Brevity provided all the other constraints are fullfilled'.

4.2 Logic in sentence aggregation

GRE algorithms are sometimes presented as if the principles underlying them were unrelated to those underlying other components of an NLG system.[9] This is especially true for the logic-based structure transformations on which this paper has focused. In what follows, however, we will suggest that analogous transformations motivate some of the key operations in sentence aggregation (Reiter and Dale 2000, p.133-144). To exemplify, (and limiting the discussion to *distributive* readings only) the choice between the (a) and (b) variants of (1)-(3) involves a decision as to whether information is expressed in one or more sentences:

1a. John is eating; Mary is eating; Carlos is eating.
1b. John, Mary and Carlos are eating.
2a. John is eating; John is drinking; John is taking a rest
2b. John is eating and drinking and taking a rest.
3a. If John is eating then Mary is eating; If Bill is eating then Mary is eating.
3b. If either John or Bill is eating then Mary is eating.

Writing $Eat^*(S)$ for $\forall x \in S(Eat(x))$ (Kamp and Reyle 1993), the linguistic equivalence of (1a) and (1b) rests on the logical equivalence

[9] But see Bateman (2000), where GRE and aggregation are linked.

1′. $(Eat(j)\&Eat(m)\&Eat(c)) \Leftrightarrow Eat^*(\{j,m,c\})$

Analogous to uses of Brevity in GRE, a preference for (1b) over (1a) might be motivated by a preference for a semantic structure with fewer logical operations. Examples (2)-(3) are not dissimilar to what we see in (1). For example, the following logical equivalences support the linguistic equivalence of (2a)/(2b) and (3a)/(3b):

2′. $(Fa\&Ga\&H(a)) \Leftrightarrow (\lambda x : Fx\&Gx\&Hx)(a)$

3′. $((p_1 \rightarrow q)\&(p_2 \rightarrow q)) \Leftrightarrow ((p_1 \vee p_2) \rightarrow q)$

In (2′), three properties, F, G and H, are aggregated into $\lambda x : Fx\&Gx\&Hx$ (i.e., to have each of the three properties F, G and H). In (3′), two antecedents p_1 and p_2 are aggregated into $p_1 \vee p_2$.[10] As before, a generator might prefer the (b) versions because they are structurally simpler than the logically equivalent (a) versions. In sentence aggregation, however, co-extensivity is not enough. For example, we expect 'Eat(j)' to be worded differently from 'Eat(m)', even if both propositions are true and consequently have the same extension. Unlike GRE, therefore, aggregation requires at least *logical* equivalence.[11]

5 Acknowledgment

Thanks are due to Emiel Krahmer for discussion and comments.

6 References

Appelt 1987. D.E. Appelt. Bidirectional Grammars and the Design of Natural Language Generation systems. In *Theoretical Issues in Natural Language Processing*-3, p.185-191. New Mexico State University, Las Cruces.

Barwise and Perry 1983. J. Barwise and J. Perry. Situations and Attitudes. MIT Press.

Bateman 1999. J.A. Bateman. Using Aggregation for Selecting Content when Generating Referring Expressions. In Procs. ACL-99, Univ. Maryland.

[10] Note the disjunction, which would be difficult to get if the transformation was performed at a syntactic level.

[11] In some (e.g., epistemic) contexts, even logical equivalence is not enough. This mirrors the problems with co-extensivity that were noted in connection with GRE.

Dale 1992. R. Dale. *Generating Referring Expressions: Constructing Descriptions in a Domain of Objects and Processes.* MIT Press, Cambridge.

Dale and Reiter 1995. R. Dale and E. Reiter. Computational Interpretations of the Gricean Maximes in the Generation of Referring Expressions. *Cognitive Science* **18**: 233-263.

Grice 1975. P. Grice. Logic and Conversation. In P. Cole and J. Morgan (Eds.), "Syntax and Semantics: Vol 3, Speech Acts": 43-58. New York, Academic Press.

Kamp and Reyle 1993. *From Discourse to Logic.* Kluwer Academic Publishers, Dordrecht.

Krahmer and Theune 1999. E. Krahmer and M. Theune. Generating Descriptions in Context. In R. Kibble and K. van Deemter (Eds.), Procs. of ws. Generation of Nominal Expressions, ESSLLI'99.

McCluskey 1965. McCluskey, Jr., E.J. Introduction to the Theory of Switching. New York. McGraw-Hill.

Montague 1973. R. Montague. The Proper treatment of Quantification in Ordinary English. In R.H.Thomason (ed.) *Formal Philosophy*. Yale University Press, New Haven.

Power 2000. R.J.D. Power. Planning texts by constraint satisfaction. Procs of the 18th Int. Conf. on Computational Linguistics (COLING-2000), Saarbruecken, pp. 642-648.

Reiter 1990. E. Reiter. The Computational Complexity of Avoiding Conversational Implicatures. In Proc. ACL-1990, Pittsburgh.

Reiter and Dale 2000. E. Reiter and R. Dale. *Building Natural language Generation Systems.* Cambridge University Press, Cambridge, UK.

Shieber 1993. S. Shieber. The Problem of Logical-Form Equivalence. Squib in *Computational Linguistics* 19, 1.

Stone 2000. M. Stone. On Identifying Sets. In Procs. of INLG-2000, Mitzpe Ramon.

Stone and Webber 1998. M. Stone and B. Webber. Textual Economy through Close Coupling of Syntax and Semantics. In Procs. of INLG-1998, p.178-187.

van Deemter 1990. Structured Meanings in Computational Linguistics. In Procs. of COLING-1990, Helsinki.

van Deemter 2000. K. van Deemter. Generating Vague Descriptions. In Procs. of INLG-2000, Mitzpe Ramon.

van Deemter 2001. Generating Referring Expressions: Beyond the Incremental Algorithm. In Procs. of Int. Workshop on Computational Semantics (IWCS-4), Tilburg.