COMPUTATIONAL SEMANTICS OF MASS TERMS

Jan Tore Lønning
Department of Mathematics
University of Oslo
P.O. Box 1053, Blindern
0316 Oslo 3, Norway

ABSTRACT

Although the formalisms normally used for describing the semantics of natural languages are far from computationally tractable, it is possible to isolate particular semantic phenomena and interpret them within simpler formal systems. Quantified mass noun phrases is one such part. We describe a simple formal system suitable for the interpretation of quantified mass noun phrases. The main issue of this paper is to develop an algorithm for deciding the validity of sentences in the formal system and hence for deciding the validity of natural language inferences where all the involved noun phrases are quantified mass noun phrases. The decision procedure is based on a tableau calculus.

INTRODUCTION

A formal semantics for a part of a natural language attempts to describe the truth conditions for sentences, or propositions expressed by sentences, in model theoretic terms, and thereby also the relation of valid inferences between sentences. From the point of view of computational linguistics and natural language understanding, it is important whether this relation of entailment can be made computational. In general, the question must be answered in the negative. All proposed formal semantics of, say, English are at least as complex as first order logic and hence at best semi-decidable, which means that if a sentence β is a logical consequence of a set of sentences Σ, then there exists a proof for β from Σ, but no effective way to find such a proof. Several proposals use even more complex logics, like the higher order intensional logic used in Montague grammar, which has no deductive theory at all.

We will not oppose to the view that English incorporates at least the power of first order logic and that even more complex formalisms may be needed to represent the meaning of all aspects of English. But we believe there are two different possible strategies when one is to study one particular semantic phenomenon in natural languages. The first one is to try to interpret the particular phenomenon into a system that attempts to capture all semantic aspects of the natural language. The other strategy is to try to isolate the particular semantic phenomenon one wants to study and to build a semantic interpretation suited for this particular phenomenon. By following the latter strategy it might be possible to find systems simpler than even first order logic that reflect interesting semantic phenomena, and in particular to come up with systems that are computationally tractable.

Quantified mass noun phrases is one such phenomenon that can be easily isolated. The properties particular for the semantics of quantified mass terms have been difficult to capture in extensions of systems already developed for count terms, like first order logic. However, if one isolate the mass terms and tries to interpret only them, it is possible to build a model where their typical properties fall out naturally. We have earlier proposed such a system and shown it to have a decidable logic (Lønning, 1987). We repeat the key points in the two following sections. The main point of this paper is a description of an algorithm for deciding validity of sentences and inferences involving quantified mass terms.

The strategy of isolating parts of a natural language and giving it a semantics that can be computational is of course the strategy underlying all computational uses of semantics. For example, in queries towards data bases one disregards all truly quantified sentences, and use only atomic sentences and pseudo quantified sentences where e.g. *for all* means for all individuals represented in the data base. The system we present here contains genuine quantifiers like *all water* and *much water*, but contain other restrictions compared to full first order logic. In particular the mass quantifiers behave simpler with respect to scope phenomena than count quantifiers.

REPRESENTING QUANTIFIED MASS NOUNS

We will make use of a very simple formal language for representing sentences containing quantified mass nouns, called LM (Lønning, 1987). We refer to the original paper for motivation of the particular chosen format and more examples and repeat only the key points here.

1. A particular LM language consists of a (non-empty) set of basic terms, say: *water*, *gold*,

blue, hot, boil, disappear ..., and a (possibly empty) set of non-logical determiners, say: *much, little, less_than_two_kilos_of...*

2. Common to all LM languages are the unary operator: –, the binary operator: •, the logical determiners: : *all, some*, and the propositional connectives: : ¬, ∧, ∨, →.
3. A term is either basic or of one of the two forms $(t•s)$ and $(-t)$ if t and s are terms.
4. An atomic formula has the form $D(t)(s)$ where D is a determiner and t and s are terms.
5. More complex formulas are built by the propositional connectives in the usual way.

A model for the particular language is a pair consisting of a Boolean algebra $A = <A, +, *, -, 0, 1>$ and an interpretation function [ ], such that

1. $[t]$ is a member of A for all basic terms t,
2. $[D]$ is a set of pairs of members of A for all determiners D.

The interpretation of more complex expressions is then defined as an extension of [ ]:

1. $[-t]= -[t]$, the Boolean complement of $[t]$, $[t•s]=[t]*[s]$, the Boolean product (or meet) of $[t]$ and $[s]$.
2. $[D(t)(s)]$ is *true* provided $([t],[s])\in [D]$, in particular $[All(t)(s)]$ is *true* provided $[t]\leq[s]$, and $[Some(t)(s)]$ is *true* provided $[t]*[s] \neq 0$.
3. The propositional part is classical.

To get an intuition of how the semantics work one can think of [*water*] as the totality of water in the world or in the more limited situation one considers, of [*blue*] as the totality of stuff that is blue and of [*disappear*] as the totality of stuff that disappeared. However, one shall not take this picture too literally since the model works as well for abstract as for concrete mass nouns.

In the formalism, a sentence like (1a) is represented as (1b) and (2a) is represented as (2b) if the negation is read with narrow scope and as (2c) if the negation is read with wide scope.

(1) (a) All hot water is water.
(b) *All(hot•water)(water)*

(2) (a) Much of the water that disappeared was not polluted.
(b) *Much(water•disappeared)(–polluted)*
(c) ¬ *Much(water•disappeared)(polluted)*

Formula (1b) is a valid LM formula. In general, the valid English inferences that become valid if a mass term like *water* is interpreted as *quantities of water* and *all water* is read as *all quantities of water* are also valid under the LM interpretation.

In addition, this approach can explain several phenomena that are not easily explained on other approaches. Roeper (1983) pointed out that paraphrasing *water* as *quantities of water* was problematic when propositional connectives were considered. If some water disappeared and some did not disappear, there will be many quantities that partly disappeared and partly did not disappear. If *disappear* denotes the set of all quantities that wholly disappeared and *did not disappear* denotes the complement set of this set, then all quantities that partly disappeared will be members of the denotation of *did not disappear*. The sum of quantities that are members of the denotation of *did not disappear* will equal all the water there is. Roeper solved this problem by letting the quantities be members of a Boolean algebra and used a non-standard interpretation of the negation. In LM it naturally comes out by the Boolean complement as in (2b) and *water that did not disappear* is represented by *(water•(–disappear))*.

A main feature of the current proposal is that it introduces non-logical quantifiers co-occurring with mass nouns, like *much, little, most,...* in a straightforward way. A sentence like *much water was blue* does not say anything about the number of quantities of blue water, but says something about the totality of blue water, which is the way it is interpreted in LM.

It might seem a little like cheating that the system only introduces interpretations of mass quantifiers with minimal scope with respect to other quantifiers, that it is not possible to interpret one quantifier with scope over another quantifier. In particular, since this is a main reason to why the logic in the sequel becomes simple. However, it is characteristic for mass quantifiers that they get narrow scope.

(3) (a) A boy ate many pizzas.
(b) A girl drank much water.

While it might be possible to get a reading of (3a) which involves more than one boy, i.e., one boy for each pizza, it is not possible to get a reading of (3b) involving more than one girl.

The only determiners that get a fixed, or logical interpretation in LM are : *all* and *some*. For the other determiners we can add various sorts of constraints on the interpretations that will make more of the inferences we make in English valid in the logic. For example, it has been claimed that all natural language determiners are **conservative** (or "live on" their first argument), i.e: $(a,b)\in [D]$ if and only if $(a,b*a)\in [D]$ (Barwise and Cooper, 1981).

Several determiners are **monotone**, either **increasing** or **decreasing**, in one or both arguments, e.g., *much* is monotone increasing in its

second argument: if (a,b) ∈ [*much*] and b ≤ c then (a,c) ∈ [*much*], and *less_than_two_kilos_of* is monotone decreasing in its second argument. Whether an inference like

Much water evaporated.
All that evaporated disappeared.
∴ Hence much water disappeared,

becomes valid in the logic, will depend on whether the denotation of *much* is constrained to be monotone increasing in its second argument or not.

LOGICAL PROPERTIES

We will repeat shortly several of the properties of the logic LM shown in (Lønning, 1987) as a background for the decision algorithm for validity.

A Hilbert style axiomatization was given and it was shown that any set of LM sentences consistent in the logic has a model: the logic is complete and compact.

It was implicitly shown, but not stated, that any model for LM must be a Boolean algebra: let a model be any set A with one unary operation [–], one binary operation [•], and a binary relation [*All*] then the model is a Boolean algebra with [–] the Boolean complement, [•] the Boolean product (meet) operation and [*All*] the ordering ≤ on A.

It was also shown that the logic was complete and compact with respect to the smaller model class: the atomic Boolean algebras, i.e. any consistent set of sentences has a model which is an atomic algebra, and in fact a finite such.

From this, it was shown that LM with no non-logical determiners, let us call it LA, is equivalent to a subset of monadic first-order logic, hence it is decidable. It was also shown that the full LM is decidable. The argument is based on the fact that the number of possible models for a sentence is finite and decidable. This number grows rapidly, however. Already a sentence in LA with n different basic terms have $2^{(2^n)}$ different models, so the argument does not establish a good procedure for checking validity in practice. We will establish procedures that are better (in most cases) in the next section.

Several natural restrictions on determiners, like conservativity and monotonicity can be expressed completely in LM. It is not surprising that this can be done for a fixed LM language given the finiteness of its nature. The more important point is that the properties can be expressed in a uniform way independently of the atomic terms of the language, which in next section will give rise to uniform inference rules.

A DECISION PROCEDURE

We shall establish a procedure for deciding the validity of LM sentences. The procedure is a combination of a normal form procedure and a tableau procedure (see e.g. Smullyan, 1968).

We start with an LM formula φ for which we want to decide whether it is valid, ⊨ φ. This is equivalent to deciding whether ¬φ is satisfiable. One can think of the process as an attempt on building a model for ¬φ. If we succeed, we have shown that φ is not valid and we have found an explicit counterexample; if we fail, we have shown that φ is valid. We assume all propositional connectives in φ to be basic: ¬, ∧, ∨.

1. First we introduce a new unary quantifier *Null* such that *Null*(t) is a formula when t is a term. The meaning is that [*Null*(t)] is true if and only if [t]=0. Then substitute

Null(t•(–s)) for *All*(t)(s)

Null(t•s) for ¬ *Some*(t)(s)

¬ *Null*(t•s) for *Some*(t)(s)

This step is not necessary, but it gives a more convenient notation to work with and fewer instances to consider in the sequel. The substitutions correspond to the fact that *Some* and *Every* can be taken to be unary and the one can be defined from the other, as in the case with count nouns.

2. Then transform φ to conjunctive normal form, that is, a conjunction of disjunctions of literals, where a literal is an atomic or negated atomic formula.

3. Observe that (i) and (ii) are equivalent.

(i) ⊨ $\psi_1 \wedge \psi_2 \wedge \ldots \wedge \psi_n$

(ii) ⊨ ψ_1, ⊨ ψ_2 ... and ⊨ ψ_n,

Hence we can proceed with each conjunct separately.

4. We shall now check ⊨ $\psi_1 \vee \psi_2 \vee \ldots \vee \psi_n$, where each ψ_i has the form *Null*(t), ¬*Null*(t), *D*(t)(s) or ¬*D*(t)(s). Observe that the following two formulas are equivalent, where t+s is shorthand for –((–t)•(–s)).

(i) ¬ *Null*(t) ∨ ¬ *Null*(s)

(ii) ¬ *Null*(t+s)

(This corresponds to the equivalence between ∃xφ ∨ ∃xψ and ∃x(φ ∨ ψ) in first order logic.) Hence contract all the literals of the form ¬*Null*(t) to one.

5. We are then left with a formula of the form

$\neg Null(t) \vee Null(s_1) \vee \ldots \vee Null(s_n) \vee \psi_1 \vee \ldots \vee \psi_m$,

where each ψ_i has the form *D*(u)(v) or ¬*D*(u)(v), for some non-logical *D*. First assume that there are no ψ_i's. Then observe that (i) and (ii) are equivalent.

(i) $\vDash \neg Null(t) \vee Null(s_1) \vee \ldots \vee Null(s_n)$

(ii) $\vDash \neg Null(t) \vee Null(s_1)$ or ... or $\vDash \neg Null(t) \vee Null(s_n)$.

(If there are no s_i's proceed with $\vDash \neg Null(t)$.) This equivalence might need some argument. That (ii) entails (i) is propositional logic. For the other way around, there are two possibilites. Either $\vDash \neg Null(t)$, which yields the equivalence. Otherwise, there exists a model **A** for the language of ϕ where $[t]=0$ and for all other terms s in the language: $[s]=0$ if and only if $All(s)(t)$ is a valid formula. Let ψ be the formula $\neg Null(t) \vee Null(s_1) \vee \ldots \vee Null(s_n)$. Then ψ is valid if and only if ψ is true in **A**. To show this, it is sufficient to show that if there is a model **B** in which ψ is not true then ψ is not true in **A**. If **B** is a model in which ψ is not true, then $[t]=0$ and each $[s_i] \neq 0$ in **B**. Hence $All(s_i)(t)$ cannot be valid and $[s_i] \neq 0$ in **A** for each s_i. Since $[t]=0$ in **A**, ψ cannot be true in **A**. The same type of argument yields that $\neg Null(t) \vee Null(s_i)$ is valid if and only if it is true in **A**. If we write $\mathbf{A} \vDash \eta$ for η is true in **A**, the following equivalence is propositional logic and yields the equivalence above.

(i) $\mathbf{A} \vDash \neg Null(t) \vee Null(s_1) \vee \ldots \vee Null(s_n)$

(ii) $\mathbf{A} \vDash \neg Null(t) \vee Null(s_1)$ or ... or $\mathbf{A} \vDash \neg Null(t) \vee Null(s_n)$.

6. a. We shall describe two different ways for checking $\vDash \neg Null(t) \vee Null(s_i)$. The first one proceeds by a transformation to normal form and may be the easiest one to understand if one is not accustomed to tableau calculus. The second one which uses a tableau approach is more efficient. First observe that (i) and (ii) are equivalent.

(i) $\vDash \neg Null(t) \vee Null(s_i)$

(ii) $\vDash Null((-t) \bullet s_i)$, (i.e. $\vDash All(s_i)(t)$).

The last claim entails the first one since

$Null((-t) \bullet s_i) \rightarrow \neg Null(t) \vee Null(s_i)$

is a valid LM-formula. To see that (ii) entails (i) observe that if $\neg Null(t) \vee Null(s_i)$ is valid, it will in particular be true in the model **A** described in step 5, hence $\vDash All(s_i)(t)$.

To check $\vDash Null((-t) \bullet s_i)$ rewrite the term $(-t) \bullet s_i$ in disjunctive normal form: allow the symbol + and write the term on the form $s_1 + \ldots + s_m$ where each s_i has the form $u_1 \bullet \ldots \bullet u_k$ and each u_j is either an atomic term or on the form $-v$ for an atomic term v. Then $\vDash Null(s_1 + \ldots + s_m)$ if and only if $\vDash Null(s_1)$ and ... and $\vDash Null(s_m)$, and $\vDash Null(u_1 \bullet \ldots \bullet u_k)$ if and only if there is a v such that one u_j equals v and another u_j equals $-v$.

b. The checking of $\vDash \neg Null(t) \vee Null(s_i)$ will be faster using a tableau procedure instead of rewriting to normal form. Note that the following are equivalent:

$\vDash \neg Null(t) \vee Null(s_i)$

$\vDash Null((-t) \bullet s_i)$

$\vDash \neg Null(t + (-s_i))$

There is a close connection between propositional logic and Boolean algebras. To each term in LM, t, there corresponds a formula p_t in pure propositional logic such that $\neg Null(t)$ is valid in LM if and only if the corresponding formula p_t is a tautology: shift each basic LM term t with a corresponding propositional constant p_t, and exchange – with $\neg$, * with $\wedge$, and + with $\vee$. In particular, the following are equivalent:

$\vDash \neg Null(t + (-s_i))$ (in LM)

$\vDash (p_t \vee (\neg p_{si}))$ (in propositional logic)

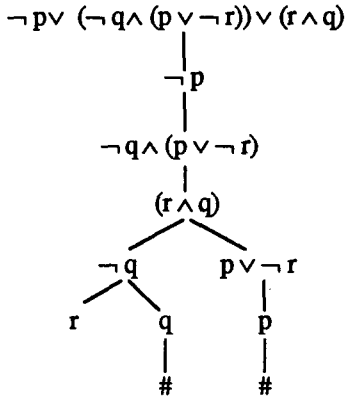
(By the earlier mentioned connection between LM and first order logic this corresponds to the fact that a first order formula $\exists x \varphi$ is valid if and only if φ is a tautology whenever φ is quantifier free.)

Step (6a) above is equivalent to checking this latter formula for validity by transformation to a normal form. Instead it can be checked by a standard tableau procedure (see e.g. Smullyan 1968).

We give a short description of the tableau approach to propositional logic. In order to verify a formula ψ, we try to build a model that falsifies it. To ease the description we assume that ψ is on negation normal form, that is, built up from literals by $\wedge$, $\vee$. The attempt to build a model is done by building a tree for ψ. We start with a root node depicted by ψ and use the following two rules:

1. For each node a depicted by a formula of the form $\gamma \vee \eta$ attach to each leaf below a in the tree constructed so far one successor node b depicted by γ and one successor node c to b depicted by η.
2. For each node a depicted by a formula of the form $\gamma \wedge \eta$ attach to each leaf below a in the tree constructed so far two new leaf nodes one depicted by γ and another one depicted by η.

The tree is complete when all formulas of the forms $\gamma \vee \eta$ and $\gamma \wedge \eta$ are reduced according to the rules above. A branch in a tree is called closed if there is a formula γ such that one node along the branch is depicted by γ and another node along the branch is depicted by $\neg\gamma$. A branch in a complete tree for ψ which is not closed describes a valuation that falsifies ψ. Conversely, if all branches in a complete tree for ψ are closed, ψ is valid. We illustrate with an example:



The sign # indicates that a branch is closed. We have not completed the rightmost branch since it is already closed. Since there is one open branch in the tree, the formula is not valid. The literals along the open branch: ¬p, ¬q, r shows that any valuation V such that $V(p) = T$, $V(q) = T$, $V(r) = \perp$, falsifies ψ.

The strategy in step (6a) above with transformation to normal form corresponds to construction of separate copies for each branch, hence duplicating parts of the tree, while the tableau procedure exploits the possibility of structure sharing.

Returning to our main point, we can observe one additional gain by using the tableau approach. Our goal is to check whether $\models \neg Null(t) \vee Null(s_1)$ or, ... or $\models \neg Null(t) \vee Null(s_n)$, which is equivalent to check whether $(t \vee (\neg s_1))$ or ...,or $(t \vee (\neg s_n))$ is a tautology. The part of the tableau tree that corresponds to t can be constructed (and if possible reduced by removing closed branches) once and for all, and then be reused with all the different s_i's.

7. We now return to step 5 and consider the case where one or more disjuncts have the form $D(u)(v)$ or $\neg D(u)(v)$, for some non-logical D. Then the following are equivalent.

(i) $\models \neg Null(t) \vee Null(s_1) \vee \ldots \vee Null(s_n) \vee \psi_1 \vee \ldots \vee \psi_m$

(ii) $\models \neg Null(t) \vee Null(s_i)$ for some s_i, $1 \leq i \leq n$, or $\models \neg Null(t) \vee \psi_k \vee \psi_j$ for some k and j between 1 and m, where ψ_k has the form $D(a)(b)$ and ψ_j has the form $\neg D(u)(v)$ for the same determiner D.

That (ii) entails (i) is immediate. For the other way around, suppose that (ii) does not hold. We shall then construct a model which falsifies the original formula in (i). Let **A** be the model where only terms provably less than t denote 0 and where a pair $([d],[e])$ is a member of $[D]$ if and only if $\neg D(d)(e)$ is one of the disjuncts ψ_i's. By the construction, **A** will falsify $\neg Null(t)$ and each disjunct of the form $\neg D(d)(e)$. As in step 5 above, **A** will falsify each $Null(s_i)$. It remains to show that **A** falsifies each disjunct of the form $D(a)(b)$. Let γ be one such disjunct, let η_j, $1 \leq j \leq s$, be all the disjuncts of the form $\neg D(d)(e)$ with the same determiner D as in γ and let ε_j be $\neg Null(t) \vee \gamma \vee \eta_j$. Since (ii) does not hold, there exists a model $\mathbf{B}_j$ where ε_j is false, for each ε_j. Then there also exists a model $\mathbf{A}_j$ which equals **A** except possibly for the interpretation of D, and where D gets the same interpretation as in $\mathbf{B}_j$. Hence ε_j is false in $\mathbf{A}_j$. Since there exists such an $\mathbf{A}_j$ for each ε_j, γ cannot be true in **A**.

8. Whether $\models \neg Null(t) \vee D(a)(b) \vee \neg D(u)(v)$ holds, depends on which restrictions are put on D. With no restrictions, any possible counterexample is one where $[t]=0$, $([u],[v]) \in [D]$ while $([a],[b]) \notin [D]$. The only reason we should not be able to construct such a model is that $[a] = [u]$ and $[b] = [v]$ whenever $[t] = 0$. We can hence proceed to check

$$\models \neg Null(t) \vee (Null((-a \bullet u)+(a \bullet -u)) \wedge Null((-b \bullet v)+(b \bullet -v))),$$

according to the same procedures as in step 6 above.

If we have the additional constraint that the determiner in question is conservative, the last rule is changed such that the last conjunct, which above stated that the symmetric difference between b and v was zero, now instead states that the symmetric difference between $a \bullet b$ and $u \bullet v$ is zero.

$$\models \neg Null(t) \vee (Null((-a \bullet u)+(a \bullet -u)) \wedge Null((-(a \bullet b) \bullet v)+((a \bullet b) \bullet -v)))$$

Similarly, if we know that the determiner is upwards monotone in its second argument $D(a)(b) \vee \neg D(u)(v)$ has to be true in any model where $[a]=[u]$ and $[v] \leq [b]$, so the last conjunct will be $Null((-b \bullet v))$ instead of $Null((-b \bullet v)+(b \bullet -v))$. If the determiner is restricted to be both conservative and monotone, the last conjunct shall be $Null((-(a \bullet b) \bullet u \bullet v))$. Similar modifications of the rule can be done for determiners with other forms of monotone behaviour.

GENERALIZED QUANTIFIERS

One main feature of the decision procedure is that it incorporates generalized quantifiers (step 7 and 8). The rules for generalized quantifiers correspond to axioms one will use in an axiomatization of LM. For example, the rule for quantifiers with no additional constraints correspond to the extensionality schemata:

For all terms a, b, u, v:

$$(All(a)(u) \wedge All(u)(a)) \rightarrow (D(a)(b) \rightarrow D(u)(b))$$

$$(All(b)(v) \wedge All(v)(b)) \rightarrow (D(a)(b) \rightarrow D(a)(v))$$

One should remember that we do not try to develop a logic for the strong logical interpretation of determiners like *most*, but a logic for some minimal constraints that interpretations of the determiners should at least satisfy.

Just like there is a meaning preserving translation from LA into first order logic, LM can be translated into first order logic extended with generalized quantifiers. A proof procedure for first order logic, like a tableau or a sequent calculus, can be extended with rules for generalized quantifiers similar to the rules introduced here. If Q is a binary quantifier with no additional constraints on its interpretation then the following are equivalent.

(i) $\models Qx(A(x),B(x)) \vee \neg Qx(C(x),D(x))$

(ii) $\models \forall x(A(x) \leftrightarrow C(x)) \wedge \forall x(B(x) \leftrightarrow D(x))$

So to show that (i) is valid one has to show that (ii) is valid. This can be incorporated into a tableau or sequent calculus for first order logic. If the first order logic is monadic, as the logic we get after translating LM into first order logic is, one can use a similar procedure as the one described here. If the extended first order logic is not monadic, the procedure one gets when rules corresponding to the reduction from (i) to (ii) are included, becomes more complex.

EFFICIENCY

We chose to transform the formula being tested to normal form early in the procedure (step 2). Alternatively to the described algorithm one could think of using a tableau procedure all the way, and not first transform to conjunctive normal form in step 2. In general, transformation to normal form is slower than using a tableau procedure (cf. step 6 above). The reason we made the transformation to normal form was that this was necessary to split the formula in step 5 and step 7. In the procedure one gets by translating LM into first order logic (with generalized quantifiers) and using a tableau procedure from the start, it is not possible to split the tree similarly. If we for simplicity considers a formula with no generalized quantifiers, the pure tableau calculus will not lead to a separate tree for each s_i together with t but to one big tree containing all the s_i's and roughly one copy of t for each s_i. This corresponds to the quantifier rules in a tableau calculus for first order logic: (i) for each formula of the form $\forall x\phi$ introduce one new formula $\phi(a)$ where a is some new term, (ii) for each formula $\exists x\psi$ introduce one new formula $\psi(a)$ for each term a introduced in the tree at a branch to which $\exists x\psi$ belongs. The successful separation in the described algorithm here will also be possible in a proof procedure for monadic first order logic.

The two different procedures will be of the same time complexity in worst cases. In the practical applications we have in mind, the procedure described here will be faster. Typically we want to check whether a formula β follows from $\alpha_1,\ldots,\alpha_n$. This is the same as deciding whether $\neg\alpha_1\vee\cdots\vee\neg\alpha_n\vee\beta$ is valid or not. The transformation to normal form will produce one additional copy for each $\vee$ within an α_i and each $\wedge$ within β. If each α_i and β are LM formulas that represent English sentences, they can each be expected to be relatively short and in particular not contain many $\vee$'s, so the number of copies made will be relatively small. On the other hand, the number of α_i's may be large if they represent the dialogue so far or the agent's knowledge. It is therefore important that each disjunct can be split up as much as possible.

IMPLEMENTATION

The inference algorithm has been implemented in PROLOG. To test it out we have built a small (toy) natural language question-answering-system around it. The program reads in simple sentences and questions from the terminal and answers the questions. It can handle simple statements, like *If some of the hot coffee that did not disappear was black then much gold is valuable* (the fragment in Lønning 1987) and yes/no questions like *Did much water evaporate?* and *Was the old gold that disappeared valuable?* We have written the grammar and translation to LM in the built in DCG rules (Pereira and Warren, 1980).

Statements typed on the terminal are interpreted as facts about the world and stored as simple sentences $\phi_1,\ldots,\phi_n$. When a question like *Did much water evaporate?* is asked, it is parsed and turned into a formula like ψ: *Much*(*water*)(*evaporate*). Then the program proceeds to check the validity of $(\phi_1\wedge\ldots\wedge\phi_n)\rightarrow\psi$. If it is valid, the program answers *yes*, otherwise it checks $(\phi_1\wedge\ldots\wedge\phi_n)\rightarrow\neg\psi$. If this is valid, the answer is *no*, otherwise the program answers that it *does not know*. When a statement is made the program checks whether it is consistent with what the program already knows before it is added to the knowledgebase.

The system is mainly made to test the inference algorithm and is not meant as an application by itself. But it illustrates some general points. It is a system where natural language inferences are made from natural language sentences and not from a fixed database. The system contains a complete treatment of propositional logic and illustrates a sound treatment of negation where failure is treated as *does not know* instead of negation. On the other hand, there is also a price to pay for incorporating full propositional logic. The system can only handle examples of a limited size in reasonable time.

CONCLUSION

We have here presented a computational approach to the semantics of quantified mass noun phrases. We think the semantics ascribed to quantified mass

nouns through a translation into LM is the one that most adequately reflects their particular semantic properties. In addition this semantics can be made computational in a way not possible for other approaches to the semantics of mass terms, like Bunt's (1985) which extends axiomatic set theory, or Link's approach (1983) based on Montague's higher-order intensional logic.

We have modified and adapted a tableau calculus to be used with mass terms and extended it with generalized quantifiers. Although the implementation we have made is of limited applicability, we hope that the algorithm can be used to incorporate quantified mass noun phrases into larger systems treating count terms. In particular, it should be possible to combine the algorithm with other approaches based on a tableau calculus, like the one described by Guenthner, Lehmann and Schönfeld (1986).

REFERENCES

Barwise, J. and R. Cooper: 1981, 'Generalized Quantifiers and Natural Language", *Linguistics and Philosophy*, **4**, 159–219.

Bunt, H.: 1985, *Mass terms and model-theoretic semantics*, Cambridge University Press, Cambridge.

Guenthner, F., H. Lehmann and W. Schönfeld: 1986,'A theory for the representation of knowledge', *IBM J. Res. Develop.* **30**, 39–56.

Link, G.:1983,'The Logical Analysis of Plurals and Mass terms: A Lattice-Theoretical Approach', in Bäuerle et al. (eds.), *Meaning, Use, and Interpretation of Language*, Walter de Gruyter, Berlin.

Lønning, J.T.:1987,'Mass Terms and Quantification', *Linguistics and Philosophy*, **10**, 1–52.

Pereira, F.C.N. and D.H.D. Warren: 1980, 'Definite Clause Grammars for Language Analysis — A Survey of the Formalism and a Comparison with Augmented Transition Networks', *Artificial Intelligence* **13**, 231–278.

Roeper, R.:1983,'Semantics for Mass terms with Quantifiers', *Nous*, **17**, 251–265.

Smullyan, R.M.:1968, *First-Order Logic*, Springer, New York.