Sentential Semantics for Propositional Attitudes

Andrew R. Haas

Department of Computer Science
State University of New York at Albany
Albany, New York, 12222

The sentential theory of propositional attitudes is very attractive to AI workers, but it is difficult to use such a theory to assign semantics to English sentences about attitudes. The problem is that a compositional semantics cannot easily build the logical forms that the theory requires. We present a new notation for a sentential theory, and a unification grammar that builds logical forms in our notation. The grammar is implemented using the standard implementation of definite clause grammars in Prolog.

1 Logical Forms for Propositional Attitudes

The sentential theory of propositional attitudes claims that propositions are sentences of a thought language. It has an obvious appeal to AI workers, since their programs often contain sentences of an artificial language, which are supposed to represent the program's beliefs. These sentences can be true or false, and the program can make inferences from them, so they have two essential properties of beliefs. It is tempting to conclude that they are the program's beliefs, and that human beliefs are also sentences of a thought language. If we extend this to all propositional attitudes, we have a sentential theory of propositional attitudes. In such a theory, an English sentence expresses a proposition, and this proposition is itself a sentence—although in a different language. Other theories of attitudes hold that a proposition is a set of possible worlds, or a situation—something very different from a sentence. Moore and Hendrix (1979), Haas (1986), Perlis (1988), and Konolige (1986) have argued for sentential theories and applied them to artificial intelligence.

Kaplan (1975) proposed an analysis of quantification into the scope of attitudes within a sentential theory, and other authors using sentential theories have offered variations of his idea (Haas 1986; Konolige 1986). Most of these theories present serious difficulties for formal semantics. The problem is that they assign two very different logical forms to a clause: one form when the clause is the object of an attitude verb, and another when it stands alone. This means that the logical form of the clause depends on its context in a complicated way. It is difficult to describe this dependence in a formal grammar. The present paper aims to solve this problem—to present a grammar that assigns logical forms that are correct according to Kaplan's ideas. We also describe a parser that builds the logical forms required by the grammar.

This grammar is a set of definite clauses written in the notation of Pereira and Warren (1980). However, it is not a definite clause grammar for two reasons. First, our grammar cannot be parsed by the top-down left-to-right method used for definite clause grammar (although it can be modified to allow this). Second, we do not allow any of the nonlogical operations of Prolog, such as checking whether a variable is bound or free, negation as failure, and the rest. This means that our grammar is a set of ordinary first-order sentences (in an unusual notation) and its semantics is the ordinary semantics of first-order logic. So the grammar is declarative, in the sense that it defines a language and assigns logical forms without reference to any algorithm for parsing or generation.

If we stick to the declarative semantics, a neglected problem demands our attention. We must choose the bound variables that appear in the logical forms generated by the grammar. Logic grammars that include semantics nearly always ignore this problem, using free variables of the meta-language to represent the bound variables of the logical form. This solution directly violates the declarative semantics of definite clauses, and we therefore reject it. We will see that this problem interacts with the semantics of NP conjunction and of quantification into attitudes. To untangle this knot and handle all three problems in one grammar is the goal of this paper.

Section 1 of this paper will propose logical forms for sentences about propositional attitudes and explain the semantics of the logical forms in terms of certain relations that we take as understood. Section 2 presents a unification grammar for a fragment of English that includes quantifiers, NP conjunction, pronouns, and relative clauses. The grammar combines syntax and semantics and assigns one or more logical forms to each sentence that it generates.

Section 3 extends the grammar to include verbs that describe propositional attitudes. Section 4 describes the implementation and summarizes the results.

1.1 KAPLAN'S ANALYSIS OF *DE RE* BELIEF REPORTS

Noun phrases in the scope of attitude verbs commonly have an ambiguity between *de re* and *de dicto* readings. Consider the example "John believes that Miss America is bald" (Dowty, Wall, and Peters 1981). Under the *de re* reading of "Miss America," this sentence says that John has a belief about a woman who in fact is Miss America, but it doesn't imply that John realizes she is Miss America. A sentential theorist might say that the sentence tells us that John has a belief containing some name that denotes Miss America, but it doesn't tell us what name. The other reading, called *de dicto,* says that John believes that whoever is Miss America is bald. The *de dicto* reading, unlike the *de re,* does not imply that anyone actually is Miss America—it could be true if the Miss America pageant closed down years ago, while John falsely supposes that someone still holds that title.

Kaplan (1975) considered examples like these. He said that an agent may use many names that denote the same entity, but there is a subset of those names that *represent* the entity to the agent (this use of "represent" is different from the common use in AI). If an agent has a *de re* belief about an entity *x*, that belief must be a sentence containing, not just any term that denotes *x*, but a term that represents *x* to the agent. Thus if "person0" is a name that represents Miss America to John, and the thought language sentence "bald(person0)" is one of John's beliefs, then the sentence "John thinks Miss America is bald" is true (under the *de re* reading).

Kaplan said that a name represents an entity to an agent if, first, it denotes that entity; second, it is sufficiently *vivid;* and, finally, there is a causal connection between the entity and the agent's use of the name. A name N is vivid to an agent if that agent has a collection of beliefs that mention N and give a good deal of relevant information about the denotation of N. What is relevant may depend on the agent's interests.

Other authors have accepted the idea of a distinguished subset of names while offering different proposals about how these names are distinguished. I have argued that the distinguished names must provide information that the agent needs to achieve his or her current goals (Haas 1986). Konolige (1986) proposed that for each agent and each entity, the set of distinguished names has exactly one member. In this paper, we adopt Kaplan's term "represent" without necessarily adopting his analysis of the notion. We assume that representation is a relation between an agent, a name, and the entity that the name denotes. If an agent has an attitude toward a thought-language sentence, and that sentence contains a name that represents a certain entity to the agent, then the agent has a *de re* attitude about that entity. Our grammar will build logical forms that are compatible with any sentential theory that includes these assumptions.

One problem about the nature of representation should be mentioned. This concerns the so-called *de se* attitude reports. This term is attributable to Lewis (1979), but the clearest definition is from Böer and Lycan (1986). *De se* attitudes are "attitudes whose content would be formulated by the subject using the equivalent in his or her language of the first-person singular pronoun 'I' " (Böer and Lycan 1986). If John thinks that he is wise, and we understand this as a *de se* attitude, what name represents John to himself? One possibility is that it is his *selfname.* An agent's selfname is a thought-language constant that he standardly uses to denote himself. It was postulated in Haas (1986) in order to solve certain problems about planning to acquire information. To expound and defend this idea would take us far from the problems of compositional semantics that concern us here. We simply mention it as an example of the kinds of theories that are compatible with the logical forms built by our grammar. See also Rapaport (1986) for another AI approach to *de se* attitudes.

1.2 COMPOSITIONAL SEMANTICS AND LOGICAL FORMS

Consider the logical form that Kaplan assigns for the *de re* reading of "John believes that some man loves Mary."

(1)
$$\begin{aligned}&\exists(\mathrm{y},\mathrm{man}(\mathrm{y})\ \&\\ &\quad\exists(\alpha,\mathrm{R}(\alpha,\mathrm{y},\mathrm{john})\ \&\\ &\qquad\mathrm{believe}(\mathrm{john},\ulcorner\mathrm{love}(\alpha,\mathrm{mary})\urcorner)))\end{aligned}$$

The notation is a slight modification of Kaplan's (Kaplan 1975). The predicate letter R denotes representation. The symbol α is a special variable ranging over names. The symbols $\ulcorner$and$\urcorner$ are Quine's quasi-quotes (Quine 1947). If α denotes a name *t*, then the expression "$\ulcorner$love(α,mary)$\urcorner$" will denote the sentence "love(*t*,mary)."

It is hard to see how a compositional semantics can build this representation from the English sentence "John believes some man loves Mary." The difficult part is building the representation for the VP "believes some man loves Mary." By definition, a compositional semantics must build the representation from the representations of the constituents of the VP: the verb "believe" and the embedded clause. Following Cooper's notion of quantifier storage (Cooper 1983), we assume that the representation of the embedded clause has two parts: the wff "love(y,mary)" and an existential quantifier that binds the free variable y. Informally, we can write the quantifier as "some(y,man(y) & S)," where S stands for the scope of the quantifier. Applying this quantifier to the wff "love(y,mary)" gives the sentence "some(y,man(y) & love(y,mary))." In the present paper, the term "quantifier" will usually refer to this kind of object—not to the symbols ∀ and ∃ of first-order logic, nor to the generalized quantifiers of Barwise and Cooper (1981).

In Section 2.2 we present a more precise formulation of our representation of quantifiers.

When the clause "some man loves Mary" forms an utterance by itself, the semantics will apply the quantifier to the wff "love(y,mary)" to get the sentence "some (y,man(y) & love(y,mary))." The problem is that the wff "love(y,mary)" does not appear in Kaplan's representation. In its place is the expression "love(α,mary)," containing a variable that ranges over names, not men. It might be possible to build this expression from the wff "love(y,mary)," but this sounds like a messy operation at best. Similar problems would arise if we chose another quotation device (such as the one in Haas 1986) or another scoping mechanism (as in Pereira and Shieber 1987).

Konolige (1986) proposed a very different notation for quantifying in, one that would abolish the difficulty described here. His proposal depends on an ingenious nonstandard logic. Unfortunately, Konolige's system has two important limitations. First, he forbids a belief operator to appear in the scope of another belief operator. Thus, he rules out beliefs about beliefs, which are common in everyday life. Second, he assumes that each agent assigns to every known entity a unique "id constant." When an agent has a belief about an object *x*, that belief contains the id constant for *x*. Using Kaplan's terminology, Konolige is saying that for any entity *x* and agent *y*, there is a unique α such that $R(\alpha,x,y)$. Kaplan never suggests that representation has this property, and as Moore (1988) pointed out, the claim is hard to believe. Surely an agent can have many names for an entity, some useful for one purpose and some for another. Why should one of them be the unique id constant? We will propose a notation that has the advantages of Konolige's notation without its limitations. Section 1.3 will present the new notation. In Section 1.4, we return to the problem of building logical forms for English sentences.

1.3 A NEW NOTATION FOR QUANTIFYING IN

Our logical forms are sentences in a first-order logic augmented with a quotation operator. We call this language the *target language*. Since the grammar is a set of definite clauses, our notation is like Prolog's. The variables of the target language are u, v, w, x, y, z, etc. Constants, function letters, and atomic wffs are defined in the usual way. If *p* and *q* are wffs, then not(*p*), and(*p*,*q*), and or(*p*,*q*) are wffs. If *p* and *q* are wffs, *x* a variable, and *t* a term, the following are wffs:

(2) some(*x*,*p*,*q*)
(3) all(*x*,*p*,*q*)
(4) unique(*x*,*p*,*q*)
(5) let(*x*,*t*,*p*)

The first wff is true iff *p* and *q* are both true for some value of *x*. The second is true iff *q* is true for all values of *x* that make *p* true. The third is true iff there is exactly one value of *x* that makes *p* true, and *q* is true for that value of *x*. The last wff is true iff *p* is true when the value of *x* is set to the value of *t*. This language should be extended to include the iota operator, forming definite descriptions, since a definite description may often represent an entity to an agent. However, we omit definite descriptions for the time being.

For any expression *e* of the target language, q(*e*) is a constant of the target language. Therefore we have a countable infinity of constants. The intended models of our language are all first-order models in which the domain of discourse includes every expression of the language, and each constant q(*e*) has the expression *e* as its denotation. Models of this kind are somewhat unusual, but they are perfectly consistent with standard definitions of first-order logic, which allow the universe of discourse to be any nonempty set (Enderton 1972). Our language does depart from standard logic in one way. We allow a variable to appear inside a constant—for example, since v is a variable, q(v) is a constant that denotes the variable v. Enderton explicitly forbids this: "no symbol is a finite sequence of other symbols" (p. 68). However, allowing a variable to appear inside a constant is harmless, as long as we are careful about the definition of a free occurrence of a variable. We modify Enderton's definition (p. 75) by changing his first clause, which defines free occurrences of a variable in an atomic wff. We say instead that a variable *v* appears free in variable *w* iff $v = w$; no variable occurs free in any constant; a variable *v* occurs free in the term $f(t_1 \ldots t_n)$ iff it occurs free in one of $t_1 \ldots t_n$; and *v* occurs free in the atomic wff $p(t_1 \ldots t_m)$ iff it occurs free in one of $t_1 \ldots t_m$. Under this definition x does not occur free in the constant q(red(x)), although it does occur free in the wff red(x).

As usual in a sentential theory of attitudes, we assume that an agent's beliefs are sentences of thought language stored in the head, and that knowledge consists of a subset of those sentences. Then simple belief is a relation between an agent and a sentence of thought language. To represent *de re* belief reports, we introduce a predicate of three arguments, and we define its extension in terms of simple belief and the notion of representation. If [*p*,*l*,*w*] is a triple in the extension of the predicate "believe," then *p* is the agent who has the belief, *l* is a list of entities $x_1 \ldots x_n$ that the belief is about, and *w* is a wff of the target language. The free variables in *w* will stand for unspecified terms that represent the entities $x_1 \ldots x_n$ to the agent *p*. These free variables are called *dummy variables*. If John believes of Mary that she is a fool, then, using Prolog's notation for lists we write

(6) believe(john,[mary],q(fool(x))).

The constant "mary" is called a *de re argument* of the predicate "believe." The free occurrence of x in fool(x) stands for an unspecified term that represents Mary to John. This means that there is a term *t* that represents Mary to John, and John believes fool(*t*). x is the dummy variable for the *de re* argument "mary." This notation is inspired by Quine (1975), but we give a semantics quite different from Quine's. Note that the symbol "believe" is an ordinary predicate letter, not a special operator. This is a minor technical advantage of the sentential approach: the quotation operator eliminates the need for a variety of special propositional attitude operators.

To define this notation precisely, we must have some way of associating dummy variables with the *de re* arguments. Suppose we have the wff believe(x,[$t_1 \ldots t_n$],q(p)). Let $v_1 \ldots v_n$ be a list of the free variables of p in order of their first occurrence. Then v_i will be the dummy variable for t_i. In other words, the dummy variable for i-th *de re* argument will be the i-th free variable of p. This method of associating dummy variables with *de re* arguments is somewhat arbitrary—another possibility is to include an explicit list of dummy variables. Our choice will make the notation a little more compact.

Then the extension of the predicate "believe" is defined as follows. Let s be an agent, [$x_1 \ldots x_n$] a list of entities from the domain of discourse, and p a wff of the target language. Suppose that p has exactly n free variables, and let $v_1 \ldots v_n$ be the free variables of p in order of their first occurrence. Suppose that $t_1 \ldots t_n$ are closed terms such that t_i represents x_i to s, for i from 1 to n. Suppose the simple belief relation holds between s and the sentence formed by substituting $t_1 \ldots t_n$ for free occurrences $v_1 \ldots v_n$ in p. Then the extension of the predicate "believe" includes the triple containing s, [$x_1 \ldots x_n$], and p.

As an example, suppose the term "personl" represents Mary to John, and John believes "fool(person1)." Then, since substituting "person1" for "x" in "fool(x)" produces the sentence "fool(person1)," it follows that

(7) believe(john,[mary],q(fool(x)))

is true in every intended model where "believe" has the extension defined above.

Consider an example with quantifiers: "John believed a prisoner escaped." The reading with the quantifier inside the attitude is easy:

(8) believe(john,[],q(some(x,prisoner(x),escaped(x)))).

In this case the list of *de re* arguments is empty. For the "quantifying in" reading we have:

(9) some(x,prisoner(x),believe(john,[x],q(escaped(y)))).

This says that for some prisoner x, John believes of x that he escaped. The dummy variable y in the wff escaped(y) stands for an unspecified term that occurs in one of John's beliefs and represents the prisoner x to John.

Let us consider nested beliefs, as in the sentence "John believed Bill believed Mary was wise." Here the *de re/de dicto* ambiguity give rise to three readings. One is a straightforward *de dicto* reading:

(10) believe(john,[],q(believe(bill,[],q(wise(mary))))).

To understand examples involving nested beliefs, it is helpful to write down the sentence that each agent believes. Since this example does not involve quantifying in, it is easy to write down John's belief—we just take the quotation mark off the last argument of "believe":

(11) believe(bill,[],q(wise(mary))).

If this belief of John's is true, then Bill believes

(12) wise(mary).

In the next reading, the name "Mary" is *de dicto* for John, but *de re* for Bill:

(13) believe(john,[],q(believe(bill,[mary],q(wise(x))))).

Here, John is using the constant "mary" to denote Mary, but he does not necessarily think that Bill is using the same constant—he only thinks that some term represents Mary to Bill. The sentence that John believes is

(14) believe(bill,[mary],q(wise(x))).

If John is right, Bill's belief is formed by substituting for the free variable x in "wise(x)" some term that represents Mary to Bill. Suppose this term is "person0," then Bill's belief would be

(15) wise(person0).

Finally, there is a reading in which "Mary" is *de re* for both agents:

(16) believe(john,[mary],q(believe(bill,[x],q(wise(y))))).

Here there is a name that represents Mary to John, and John thinks that there is a name that represents Mary to Bill. Again, John does not necessarily believe that Bill uses the same name that John uses. Suppose "person3" is the term that represents Mary to John, then John's belief would be

(17) believe(bill,[person3],q(wise(y))).

If "person4" is the term that represents Mary to Bill, then Bill's belief would be

(18) wise(person4).

One might expect a fourth reading, in which "Mary" is *de re* for John and *de dicto* for Bill, but our formalism cannot represent such a reading. To see why, let us try to construct a sentence that represents this reading. In our notation a nonempty list of *de re* arguments represents a *de re* belief, while an empty list of *de re* arguments represents a *de dicto* belief. Therefore the desired sentence should have a nonempty list of *de re* arguments for John's belief, and an empty list for Bill's belief. This would give

(19) believe(john,[mary],q(believe(bill,[],q(wise(x)))))

This sentence does not assert that John believes Bill has a *de dicto* belief about Mary. To see this, consider John's belief. If he uses the constant "personl" to denote Mary, the belief is

(20) believe(bill,[],q(wise(x))).

In forming John's belief we do not substitute "person1" for the occurrence of x under the quotation operator—because by our definitions this is not a free occurrence of x. Thus John's belief says that Bill has a belief containing a free variable, which our theory forbids.

It is not clear to me whether the desired reading exists in English, so I am not certain if this property of the notation is a bug or a feature. In either case, other notations for describing attitudes have similar properties. For example, in a modal logic of attitudes we use the scope of quantifiers to represent *de re/de dicto* distinctions. If a quantifier appears in the scope of an attitude operator, we have a *de dicto* reading, and if it appears outside the scope (while binding a variable inside the scope) we get a *de re* reading. In a sentence like "John thinks Bill thinks Mary saw a lion," there are three places to put the existential quantifier: in the scope of Bill's belief operator, in the scope of John's operator but outside Bill's, or outside both. These give the same three readings that our formalism allows. To make "a lion" be *de re* for John and *de dicto* for Bill, we would have to put the quantifier outside the scope of John's belief operator, but inside the scope of Bill's belief operator. Since Bill's belief operator is in the scope of John's, that is impossible.

The same method applies to other attitudes—for example, knowledge. Given a simple knowledge relation, which expresses *de dicto* readings of sentences with "know," one can define the predicate "know," which expresses both *de re* and *de dicto* readings. "Know" will take three arguments just as "believe" does.

Next we consider examples like "John knows who likes Mary," in which "know" takes a wh noun phrase and a sentence containing a gap. The intuition behind our analysis is that John knows who likes Mary if there is a person *s* such that John knows that *s* likes Mary. This is of course a *de re* belief report, and its logical form should be

(21) some(x,person(x),know(john,[x],q(like(y,mary)))).

As an example, suppose the sentence

(22) like(bill,mary)

is one of John's beliefs, and it belongs to the subset of beliefs that constitute his knowledge. If the constant "bill" represents Bill to John, then since substituting "bill" for "y" in "likes(y,mary)" gives the sentence "like(bill,mary)," we have

(23) know(john,[bill],q(like(y,mary)))

and therefore

(24) some(x,person(x),know(john,[x],q(like(y,mary)))).

This proposed analysis of "knowing who" is probably too weak. As a counter example, suppose a night watchman catches a glimpse of a burglar and chases him. Then the night watchman has formed a mental description of the burglar—a description that he might express in English as "the man I just saw sneaking around the building." The burglar might say to himself, "He knows I'm in here." This is a *de re* belief report, so it follows that the night watchman's mental description of the burglar must represent the burglar to the watchman (by our assumption about representation). Yet the night watchman surely would not claim that he knows who is sneaking around the building. It seems that even though the watchman's mental description represents the burglar, it is not strong enough to support the claim that he knows who the burglar is.

It would be easy to extend our notation to allow for a difference between "knowing who" and other cases of quantification into attitudes. It would be much harder to analyze this difference, Böer and Lycan (1986) have argued that when we say someone knows who N is, we always mean that someone knows who N is for some purpose. This purpose is not explicitly mentioned, so it must be understood from the context of the utterance in which the verb "know" appears. Then the predicate that represents "knowing who" must have an extra argument whose value is somehow supplied by context. These ideas look promising, but to represent this use of context in a grammar is a hard problem, and outside the scope of this work.

Next we consider intensional transitive verbs like "want," as in "John wants a Porsche." The intuition behind the analysis is that this sentence is roughly synonymous with "John wishes that he had a Porsche"—under a reading in which "he" refers to John. Then the logical form would be

(25) wish(john,[],q(some(x,porsche(x),have(john,x))))

for a *de dicto* reading, and

(26) some(x,porsche(x),wish(john,[x],q(have(john,y))))

for a *de re* reading. The predicate letter "wish" need not be identical to the one that translates the English verb "wish"—it might only be roughly synonymous. The predicate letter "have" probably is the same one that translates the verb "have"—or rather, one of many predicates that can translate this highly ambiguous verb. For the present purpose let us assume that the predicate "have" represents a sense of the verb "have" that is roughly synonymous with "possess," as in "John has a Porsche." Another sense of "have" is relational, as in "John has a son," and "want" has a corresponding sense, as in "John wants a son." The present paper will not analyze this relational sense.

This grammar will express the meanings of intensional verbs in terms of propositional attitudes. This may not work for all intensional verbs. For example, it is not clear that "the Greeks worshipped Zeus" is equivalent to any statement about propositional attitudes. Montague (1974a) represented intensional verbs more directly, as relations between agents and the intensions of NP's. A similar analysis is possible in our framework, provided we extend the target language to include typed lambda calculus. Suppose the variable p ranges over sets of individuals. Then we could represent the *de dicto* reading of "John wants a Porsche" as

(27) want(john,q(lambda(p,some(x,porsche(x),x $\in$ p)))).

Here the predicate "want" describes a relation between a person and an expression of thought language, but that expression is not a wff. Instead it is a closed term denoting a set of sets of individuals. Certainly this is a natural generalization of a sentential theory of attitudes. If agents can have attitudes toward sentences of thought language, why shouldn't they have attitudes toward other expressions of the same thought language?

1.4 COMPOSITIONAL SEMANTICS AGAIN

We now return to the problem of building logical forms with a compositional semantics. Consider the formula

(28) some(x,prisoner(x),believe(john,[x],q(escaped(y)))).

Following Cooper as before, we assume that the semantic features of the clause "a prisoner escaped" are a wff containing a free variable and an existential quantifier that binds the same variable. In formula (28) the existential quantifier does not bind the variable that appears in the wff "escaped(y)"—it binds another variable instead. Therefore we have the same problem that arose for Kaplan's representation—it is not clear how to build a representation for the belief sentence from the representations of its constituents.

The choice of bound variables is arbitrary, and the choice of dummy variables is equally arbitrary. Thus, there is an obvious solution: let the *de re* arguments and the dummy variables be the same. Thus, the wide scope reading for "John believes a prisoner escaped" is not (28), but

(29) some(x,prisoner(x),believe(john,[x],q(escaped(x)))).

Here the variable x serves two purposes—it is a *de re* argument, and also a dummy variable. When it occurs as a *de re* argument, it is bound by the quantifier in the usual way. When it occurs as a dummy variable, it is definitely not bound by the quantifier. In fact the dummy variable is a mention of the variable x, not a use, because it occurs under a quotation mark.

Formula (29) may be a little confusing, since the same variable appears twice with very different semantics. This formula has a major advantage over formula (28), however—it contains the wff "escaped(x)" and a quantifier that binds the free variable of that wff. Since these are precisely the semantic features of the clause "a prisoner escaped," it is fairly easy to build the logical form (29) from the sentence "John believed a prisoner escaped."

We can describe this technique as a convention governing the logical forms that our grammar assigns to English phrases. In any wff of the form believe(x,[$t_1 \ldots t_n$], q(p)), the nth *de re* argument is equal to its own dummy variable. Then the nth *de re* argument t_n is equal to the nth free variable of p. In other words, the list [$t_1 \ldots t_n$] is just a list of the free variables of p in order of occurrence. The same convention holds for all predicates that represent attitudes.

Finally, note that the convention holds only for the logical forms that the grammar assigns to sentences. Once the grammar has built a logical form, inference procedures can freely violate the convention. For example, consider the logical form of the sentence "Every man believes that Mary loves him":

(30) all(x,man(x),believe(x,[x],q(love(mary,x)))).

From this sentence and the premise man(bill) we can infer

(31) believe(bill,[bill],q(love(mary,x)))

by substituting for a universal variable as usual. The occurrence of the variable under the quotation mark is naturally unaffected, because it is not a free occurrence of x.

1.5 SELF-REFERENCE AND PARADOX

Other writers (cited above) have already expounded and defended sentential theories of attitudes. This paper takes a sentential theory as a starting point, and aims to solve certain problems about the semantics of attitude reports in such a theory. However, one problem about sentential theories deserves discussion. The results of Montague (1974b) have been widely interpreted as proof that sentential theories of attitudes are inconsistent and therefore useless. Montague did indeed show that certain sentential theories of knowledge produce self-reference paradoxes, and are therefore inconsistent. However, he did not show that these were the only possible sentential theories. Recently des Rivières and Levesque (1986) have constructed sentential theories without self-reference and proved them consistent. Thus they showed that while Montague's theorem was true, its significance had been misunderstood. Perlis (1988) has shown that if we introduce self-reference into a modal theory, it too can become inconsistent. In short, there is no special connection between sentential theories and paradoxes of self-reference. A sentential theory may or may not include self-reference; a modal theory may or may not include self-reference; and in either case, self-reference can lead to paradoxes.

Kripke (1975) has shown that even the most commonplace utterances can create self-reference if they occur in unusual circumstances. Therefore the problem is not to avoid self-reference, but to understand it. The problem for advocates of sentential theories is to find a sentential analysis of the self-reference paradoxes that is, if not wholly satisfactory, at least as good as nonsentential analyses. For the purposes of AI, a successful analysis must avoid paradoxical conclusions, without sacrificing axioms or rules of inference that have proved useful in AI programs.

One idea is that ordinary human intuitions about self-reference are inconsistent. To most people, it appears that the sentence "This statement is false" must be both true and false, yet it cannot be both. The only error in the formal analyses is that having derived a contradiction, they allow us to derive any conclusion whatever. This happens because standard logic allows no inconsistent theories except trivial ones, containing every sentence of the language. Therefore we need a new kind of logic to describe the inconsistent intuitions of the ordinary speaker. Priest (1989) attempted this—he constructed an inconsistent but nontrivial theory of truth using a paraconsistent logic. Priest's theory includes the T-scheme, written in our notation as

(32) $P \leftrightarrow \text{true}(q(P))$.

P is a meta-variable ranging over sentences of the language. Tarski (1936) proposed this scheme as capturing an essential intuition about truth. Unfortunately, the rule of modus ponens is invalid in Priest's system, which means that most of the standard AI reasoning methods are invalid. Priest considers various remedies for this problem.

Another approach is to look for a consistent theory of self-reference. Such a theory will probably disagree with speakers' intuitions for paradoxical examples like "This statement is false." Yet these examples are rare in practice, so a natural language program using a consistent theory of self-reference might agree with speakers' intuitions in the vast majority of cases. Kripke (1975) proposed such a theory, based on a new definition of truth in a model—an alternative to Tarski's definition. Kripke's definition allows truth-value gaps: some sentences are neither true nor false. Suppose P is a sentence; then the sentence true(q(P)) is true iff P is true, and false iff P is false. Therefore if P is neither true nor false, true(q(P)) also has no truth value. In other respects, Kripke's definition of truth resembles Tarski's—it assigns the same truth values to sentences that do not contain the predicate "true," and it never assigns two different truth values to one sentence. Suppose that a model of this kind contains a sentence that says "I am not true." Formally, suppose the constant c denotes the sentence $\neg$ true(c). What truth value can such a sentence have under Kripke's definition? Just as in standard logic, $\neg$ true(c) is true iff true(c) is false. True(c) in turn is false iff c is false. Since c is the sentence $\neg$ true(c), we have shown that c is true iff c is false. Since no sentence has two truth values, it follows that c has no truth value.

Once again, problems arise because the system is too weak. If P is a sentence with no truth value, then the sentence $P \vee \neg P$ has no truth value, even though it is a tautology of first-order logic. One remedy for this appears in the system of Perlis (1985). Perlis considers a first-order model M containing a predicate "true," whose extension is the set of sentences that are true in M by Kripke's definition. He accepts as theorems all sentences that are Tarski-true in every model of this kind. Thus Perlis's system uses two notions of truth: P is a theorem only if P is Tarski-true, but true(q(P)) is a theorem only if P is Kripke-true. Suppose we have $P \leftrightarrow \neg \text{true}(q(P))$; then Perlis's system allows us to prove both P and $\neg$ true(q(P)). This certainly violates the intuitions of ordinary speakers, but such violations seem to be the inevitable price of a consistent theory of self-reference. Perlis devised a proof system for such models, using standard first-order proof and an axiom schema GK for the predicate "true." Perlis proved that if L is any consistent set of first-order sentences that does not mention the predicate "true," then the union of L and GK has a model M in which the extension of "true" is the set of sentences that are Kripke-true in M. Perlis's system has one important advantage over Kripke's: since the formalism is just a standard first-order theory, we can use all the familiar first-order inference rules. In this respect, Perlis's system is better suited to the needs of AI than either Kripke's or Priest's. However, it still excludes some inferences that are standard in everyday reasoning. For example, we have $\text{true}(q(P)) \rightarrow P$ for every P, but $P \rightarrow \text{true}(q(P))$ is not a theorem for certain sentences P—in particular, sentences that are self-referential and paradoxical.

An adequate account of self-reference must deal not only with the Liar, but also with paradoxes arising from propositional attitudes—for example, the Knower Paradox (Montague and Kaplan 1974), and Thomason's paradox about belief (Thomason 1980). Perlis (1988) has considered the treatment of attitudes within his system, and Asher and Kamp (1986) have treated both paradoxes using ideas akin to Kripke's (their treatment is not sentential, but they claim that it could be extended to a sentential treatment).

Let us briefly consider the treatment of the Knower paradox within Perlis's system. To simplify the treatment, we will assume that knowledge is true belief. If we are working in Perlis's system, this naturally means that knowledge is Kripke-true belief. We write "the agent knows that P" as $\text{true}(q(P)) \wedge \text{believe}(q(P))$. The paradox arises from a sentence R that says "The agent knows $\neg$R." Formally,

(33) $R \leftrightarrow (\text{true}(q(\neg R)) \wedge \text{believe}(q(\neg R)))$.

Since $\text{true}(q(\neg R)) \rightarrow \neg R$ is a theorem of Perlis's system, (33) implies $\neg$R. Now suppose that the agent believes (33); then with modest powers of inference the agent can conclude $\neg$R, so we have believe(q$\neg$R). Combining this with (33) gives

(34) $R \leftrightarrow \text{true}(q(\neg R))$,

which at once implies that $\neg$R is not Kripke-true. It follows that although $\neg$R is a theorem of the system, and the agent believes it, the agent does not know it—because it is not Kripke-true, and only a sentence that is Kripke-true can be known. The Knower paradox arises if we insist that the agent does know $\neg$R. This example brings out a counter-intuitive property of Perlis's system: a sentence may follow directly from Perlis's axioms, yet he refuses to call it true, or to allow that any agent can know it. Strange though this appears, it is a natural consequence of the use of two definitions of truth in a single theory.

Belief is different from knowledge because it need not be true. This makes it surprising that Thomason's paradox involves only the notion of belief, not knowledge or truth. In fact the paradox arises exactly because Thomason's agent thinks that all his beliefs are true. This is stated as

(35) $\alpha(<\alpha(<\varphi>) \rightarrow \varphi>)$

(Thomason 1980). The notation is as follows: φ is a variable ranging over all formulas of the language, $< \varphi >$ is a constant denoting (the Gödel number of) φ, and $\alpha(< \varphi >)$ means that the agent believes φ. This axiom says that for every formula φ, the agent believes

(36) $\alpha(< \varphi >) \rightarrow \varphi$.

This sentence says that if the agent believes φ, φ must be true. Since φ ranges over all sentences of the language, the agent is claiming that his beliefs are infallible. This leads the agent into a paradox similar to the Knower, and his beliefs are therefore inconsistent. Asher and Kamp showed that one can avoid this conclusion by denying (35) in certain cases where ϕ is a self-referential sentence. Another alternative is to dismiss (35) completely. It is doubtful that human beings consider their own beliefs infallible, and Perlis (1986) has argued that a rational agent may well believe that some of his or her beliefs are false.

We have looked at three sentential analyses of the self-reference paradoxes, and each one sacrifices some principle that seems useful for reasoning in an AI program. The alternative is an analysis in which propositions are not sentences. Thomason (1986) considers such analyses and finds that they have no clear advantage over the sentential approaches. The unpleasant truth is that paradoxes of self-reference create equally serious problems for all known theories of attitudes. It follows that they provide no evidence against the sentential theories.

2 The Basic Grammar

2.1 Notation

The rules of our grammar are definite clauses, and we use the notation of definite clause grammar (Pereira and Warren 1980). This notation is now standard among computer scientists who study natural language and is explained in a textbook by Pereira and Shieber (1987). Its advantages are that it is well defined and easy to learn, because it is a notational variant of standard first-order logic. Also, it is often straightforward to parse with grammars written in this notation (although there can be no general parsing method for the notation, since it has Turing machine power). DCG notation lacks some useful devices found in linguistic formalisms like GPSG—there are no default feature values or general feature agreement principles (Gazdar et al. 1985). On the other hand, the declarative semantics of the DCG notation is quite clear—unlike the semantics of GPSG (Fisher 1989).

The grammar is a set of meta-language sentences describing a correspondence between English words and sentences of the target language. Therefore, we must define a notation for talking about the target language in the meta-language. Our choice is a notation similar to that of Haas (1986). If f is a symbol of the target language, 'f is a symbol of the meta-language. Suppose f is a constant or a variable, taking no arguments. Then 'f denotes f. Thus 'john is a meta-language constant that denotes a target-language constant, while 'x is a meta-language constant that denotes a target-language variable. Suppose f is a functor of the target language and takes n arguments. Then 'f is a meta-language function letter, and it denotes the function that maps n expressions of the target language $e_1 \ldots e_n$ to the target-language expression f($e_1 \ldots e_n$). Thus 'not is a meta-language function letter, and it denotes the function that maps a target language wff to its negation. In the same way, 'or is a meta-language function letter, and it denotes the function that maps two target-language wffs to their disjunction.

Given these denotations, it is easy to see that if p(a,b) is an atomic sentence in the target language, then 'p('a,'b) is a term in the meta-language, and it denotes the wff p(a,b) in the target language. Suppose that Wff1 and Wff2 are meta-language variables ranging over wffs of the target language. Then 'or(Wff1,Wff2) is a meta-language term, and since the variables Wff1 and Wff2 range over all wffs of the target language, the value of 'or(Wff1,Wff2) ranges over all disjunctions in the target language. These ideas about the relation between meta-language and target language are not new or difficult, but it is worth the time to explain them, because some influential papers about semantics in unification grammar have confused the target language and meta-language (see Section 2.4). For the sake of legibility, we omit the quotation marks—so when or(Wff1,Wff2) appears in a rule of the grammar, it is an abbreviation for 'or(Wff1,Wff2).

2.2 Representing Quantifiers

Noun phrases in the grammar contribute to logical form in two ways, and therefore they have two semantic features. The first feature is a variable, which becomes a logical argument of a verb. This produces a wff, in which the variable appears free. The second feature is a quantifier that binds the variable. By applying the quantifier to the wff, we eliminate free occurrences of that particular variable. After applying all the quantifiers, we have a wff without free variables—a sentence. This is the logical form of an utterance.

In Montague's system (Montague 1974a), the logical form of an NP is an expression denoting a quantifier. This kind of analysis is impossible in our system, because the target language is first-order. It contains no expressions that denote quantifiers. Therefore the representation of an NP cannot be an expression of the target language. Instead of using Montague's approach, we associate with every quantifier a function that maps wffs to wffs. For the NP "every man," we have a function that maps any wff Wff1 to the wff

(37) all(V,man(V),Wff1)

where V is a variable of the target language. Notice that if we took Montague's representation for the quantified NP, applied it to the lambda expression lambda(V,Wff1), and then simplified, we would get an alphabetic variant of (37).

We will call this function the application function for the quantified NP.

To represent application functions in a unification grammar, we use a device from Pereira and Warren (1980). We assign to each NP an infinite set of readings—one for each ordered pair in the extension of the application function. The first and second elements of the ordered pair are semantic features of the NP, and the bound variable of the quantifier is a third feature. For the NP "every man" we have

(38) np(V,Wff1,all(V,man(V),Wff1)) → [every man].

This says that for any variable V and wff Wff1, the string "every man" is an NP, and if it binds the variable V, then the pair [Wff1,all(V,man(V),Wff1)] is in the extension of its application function. It follows that the application function maps Wff1 to the wff all(V,man(V),Wff1). When other rules fix the values of the variables V and Wff1, the result of the mapping will be fixed as well. A more complex example is

(39) np(V,Wff1,and(some(V,man(V),Wff1),some(V, woman(V),Wff1))) → [a man and a woman].

Here the application function's output includes two copies of the input.

It is important to consider the declarative semantics of these rules. Each one states that a certain NP has an infinite set of possible readings, because there are infinitely many wffs in the target language. Thus we might say that the NP in isolation is infinitely ambiguous. This "ambiguity" is purely formal, however; in any actual utterance the value of the variable Wff1 will be supplied by other rules, so that in the context of an utterance the ambiguity is resolved. In the same way, the VP "liked Mary" is ambiguous in person and number—but in the context of the utterance "John liked Mary," its person and number are unambiguous.

In one respect the declarative semantics of these rules is not quite right. The variable V is supposed to range over variables of the target language, and the variable Wff1 is supposed to range over wffs of the target language. Yet we have not defined a type system to express these range restrictions. However, such a type system could be added, for example, using the methods of Walther (1987). In fact, the type hierarchy would be a tree, which allows us to use a simplified version of Walther's methods. For brevity's sake we will not develop a type system in this paper. Except for this omission, the declarative semantics of the above rules is quite clear.

Typed variables have mnemonic value even if we do not use a typed logic. Therefore we adopt the following conventions. The meta-language variables V, V0, V1 ... range over target language variables. Wff, Wff1, Wff2 ... range over target language wffs. Q, Q1, Q2 ... range over quantifiers. QL, QL1, QL2 ... range over lists of quantifiers. When a wff forms the range restriction of a quantifier, we will sometimes use the variables Range, Range1 ... for that wff.

2.3 SCOPING AND QUANTIFIER STORAGE

Given a means of describing quantifiers, we must consider the order of application. Cooper (1983) has shown how to allow for different orders of application by adding to NPs, VPs, and sentences an extra semantic feature called the *quantifier store*. The store is a list of quantifiers that bind the free variables in the logical form of the phrase. The grammar removes quantifiers from the store and applies them nondeterministically to produce different logical forms, corresponding to different orders of application. If a sentence has a logical form *p* and a quantifier store *l*, then every free variable in *p* must be bound by a quantifier in *l*—otherwise the final logical form would contain free variables.

Our treatment of quantifier storage is different from Cooper's in two ways. First, Cooper's grammar maps phrases to model-theoretic denotations, not logical forms. This sounds like a bigger difference than it is. The basic technique is to put quantifiers in a store, and use some kind of marker to link the stored quantifiers to the argument positions they must bind. Whether we work with the logical forms or with their denotations, much the same problems arise in applying this technique.

A second difference is that in Cooper's grammar, each NP has two readings—one in which the NP's quantifier is in the store, and one in which it is not. The first reading leads to wide-scope readings of the sentence, while the second leads to narrow-scope readings. In our grammar only the first kind of reading for an NP exists—that is, the quantifier of an NP is always in the store. We generate both wide- and narrow-scope readings by applying the quantifiers from the store in different orders.

We represent a quantifier as a pair p(Wff1,Wff2), where the application function of the quantifier maps Wff1 to Wff2. We represent a quantifier store as a list of such pairs. The predicate apply_quants(QL1,Wff1,QL2,Wff2) means that QL1 is a list of quantifiers, Wff1 is a wff, Wff2 is the result of applying some of the quantifiers in QL1 to Wff1, and QL2 contains the remaining quantifiers. The first axiom for the predicate says that if we apply none of the quantifiers, then QL2 = QL1 and Wff2 = Wff1:

(40) apply_quants(QL,Wff,QL,Wff).

The second axiom uses the predicate choose(L1,X,L2), which means that X is a member of list L1, and L2 is formed by deleting one occurrence of X from L1.

```
(41)
    apply_quants(QL1,Wff1,QL3,Wff3)
    :- choose(QL1,p(Wff1,Wff2),QL2),
       apply_quants(QL2,Wff2,QL3,Wff3).
```

Consider the first literal on the right side of this rule. It says that p(Wff1,Wff2) is a member of QL1, and deleting p(Wff1,Wff2) from QL1 leaves QL2. By definition, if the pair p(Wff1,Wff2) is in the extension of the application function for a certain quantifier, the application function maps Wff1 to Wff2. The second literal says that applying a subset of the remaining quantifiers QL2 to Wff2 gives a new wff Wff3 and a list QL3 of remaining quantifiers. Then applying a subset of QL1 to Wff1 gives Wff3 with remaining quantifiers QL3.

Suppose that QL1 is

(42) [p(Wff1,all(V1,man(V1),Wff1)),p(Wff2,some(V2, woman(V2),Wff2))].

Then solutions for the goal

(43) :- apply_quants(QL1,loves(V1,V2),QL3,Wff3)

include

(44)
```
Wff3 = all(V1,man(V1),
           some(V2,woman(V2),loves(V1,V2)))
QL3 = []
```

and also

(45)
```
Wff3 = some(V2,woman(V2),
            all(V1,man(V1),loves(V1,V2)))
QL3 = [].
```

There are also solutions in which some quantifiers remain in the store:

(46)
```
Wff3 = all(V1,man(V1),loves(V1,V2))
QL3 = [p(Wff2,some(V2,woman(V2),Wff2))].
```

These solutions will be used to build wide-scope readings for propositional attitude reports.

2.4 THE PROBLEM OF ASSIGNING DISTINCT VARIABLES TO QUANTIFIERS

The rules we have given so far do not tell us which target language variables the quantifiers bind. These rules contain meta-language variables that range over target language variables, rather than meta-language constants that denote particular variables of the target language. In choosing the bound variables it is sometimes crucial to assign distinct variables to different quantifiers. The logical form of "Some man loves every woman" can be

(47) some(x,man(x),all(y,woman(y),loves(x,y)))

but it cannot be

(48) some(y,man(y),all(y,woman(y),loves(y,y))).

This reading is wrong because the inner quantifier captures the variables that are supposed to be bound by the outer quantifier. To be more precise: the outer quantifier binds the variable y, but not all occurrences of y in the scope of the outer quantifier are bound by the outer quantifier. Some of them are bound instead by the inner quantifier. In this situation, we say that the inner quantifier *shadows* the outer one. We require that no quantifier ever shadows another in any logical form built by the grammar. This requirement will not prevent us from finding logical forms for English sentences, because any first-order sentence is logically equivalent to a sentence without shadowing.

The same problem arises in cases of quantification into the scope of attitudes. Consider the sentence "John thinks some man loves every woman," and suppose that "some man" has wide scope and "every woman" has narrow scope. The logical form can be

(49)
```
some(x,man(x),
  thinks(john,[x],q(all(y,woman(y),loves(x,y)))))
```

but it cannot be

(50)
```
some(y,man(y),
  thinks(john,[y],q(all(y,woman(y),loves(y,y))))).
```

In this formula, the inner quantifier captures a variable that is supposed to be a dummy variable. In this case also, we say that the inner quantifier shadows the outer one.

Pereira and Warren (1980) prevented shadowing by using Prolog variables to represent variables of the object language. Thus, their translation for "Some man loves every woman" is

(51) exists(Y) : (man(Y) & all(X) : (woman(X) ⇒ loves(Y,X)))

where X and Y are Prolog variables. This works, but it violates the declarative semantics of Prolog. According to that semantics every variable in an answer is universally quantified. Thus if Prolog returns (51) as a description of the logical form of a sentence, this means that for all values of X and Y the expression (51) denotes a possible logical form for that sentence. This means that if v is a variable of the object language, then

(52) exists(v) : (man(v) & all(v) : (woman(v) ⇒ loves(v,v)))

is a possible translation, which is clearly false. Thus, according to the declarative interpretation, Pereira and Warren's grammar does not express the requirement that no quantifier can shadow another quantifier. Pereira and Shieber (1987) pointed out this problem and said that while formally incorrect the technique was "unlikely to cause problems." Yet on p. 101 they describe the structures built by their grammar as "unintuitive" and even "bizarre." This confirms the conventional wisdom: violating the declarative semantics makes logic programs hard to understand. Therefore, let us look for a solution that is formally correct.

Warren (1983) suggested one possible solution. We can use a global counter to keep track of all the variables used in the logical form of a sentence, and assign a new variable to every quantifier. Then no two quantifiers would bind the same variable, and certainly no quantifier would shadow another. This solution would make it easier to implement our treatment of *de re* attitude reports, but it would also create serious problems in the treatment of NP conjunction and disjunction (see Section 2.5). Therefore we consider another possibility.

Let us rewrite the definition of "apply_quants," adding the requirement that each quantifier binds a variable that is not bound in the scope of that quantifier. For each integer N, let v(N) be a variable of the target language. If N is not equal to M, then v(M) and v(N) are distinct variables. We represent the integers using the constant 0 and the function "s" for "successor" in the usual way. The predicate highest_bound_var(Wff1,N) means that N is the largest number such that v(N) is bound in Wff1. To define this predicate, we need one axiom for each quantifier, connective, and predicate letter of the target language. These axioms are obvious and are therefore omitted.

We also need the predicate binds(Wff1,V), which means that the outermost quantifier of Wff1 binds the variable V. To define this predicate we need an axiom for each quantifier and connective. Typical axioms are:

```
(53) binds(all(V,Wff1,Wff2),V).
(54) binds(and(Wff1,Wff2),V) :- binds(Wff1,V).
```

The second axiom applies to complex quantifiers arising from conjoined NPs. In this case there are two branches, but each branch binds the same variable (the rules for NP conjunction ensure that this is so). Therefore, we recursively check the first branch to find the bound variable.

Given these predicates, we can rewrite the second axiom for "apply_quants":

```
(55)
     apply_quants(QL1,Wff1,QL3,Wff3)
     :- choose(QL1,p(Wff1,Wff2),QL2),
        highest_bound_var(Wff1,N),
        binds(Wff2,v(s(N))),
        apply_quants(QL2,Wff2,QL3,Wff3).
```

Wff1 is the scope of the quantifier, and v(N) is the highest bound variable of Wff1. The new quantifier binds the variable v(s(N)), which is different from every bound variable in the scope Wff1. Therefore, the new quantifier is not shadowed by any lower quantifier.

As an example, suppose that QL1 is

```
(56) [p(Wff2,all(V2,woman(V2),Wff2))].
```

Then solutions for the goal

```
(57) :- apply_quants(QL1,loves(V1,V2),QL3,Wff3)
```

include

```
(58)
     Wff3 = all(v(1),woman(v(1)),loves(v(1),V2)))
     QL3 = [].
```

(We have reverted to standard notation for integers.) Suppose that QL1 is

```
(59) [p(Wff1,some(V1,man(V1),Wff1)),p(Wff2,all
     (V2,woman(V2), Wff2) )].
```

Then solutions for the goal

```
(60)
     :- apply_quants(QL1,loves(V1,V2),QL3,Wff3).
```

include

```
(61)
     Wff 3 = some(v(2),man(v(2),
          all(v(1),woman(v(1),loves(v(2),v(1))))
     QL3 = [].
```

The inner quantifier binds the variable v(1), and the outer quantifier binds the variable v(2). This notation for variables is very hard to read, so in the rest of the paper we will use the constants x, y, and z to represent variables of the target language.

2.5 Rules for Noun Phrases

The following grammar is very similar to the work of Pereira and Shieber (1987, Sections 4.1 and 4.2). There are two major differences, however, First, the treatment of quantifiers and scoping uses a version of Cooper's quantifier storage, instead of the "quantifier tree" of Pereira and Shieber. Second, Pereira and Shieber started with a semantics using lambda calculus, which they "encoded" in Prolog. In the present grammar, unification semantics stands on its own—it is not a way of encoding some other formalism.

Formula numbering uses the following conventions. The rules of the grammar are numbered (R1), (R2), etc. Entries in the lexicon are numbered (L1), (L2), etc. Formulas built in the course of a derivation get numbers without a prefix. Groups of related rules are marked by lower case letters: (L1a), (L1b), and so forth.

Every noun phrase has a quantifier store as one of its semantic features. If the NP is a gap, the store is empty; if the NP is not a gap, the first element of the store is the quantifier generated by the NP (in the present grammar, the quantifier store of an NP has at most one quantifier). We represent the quantifier store as a difference list, using the infix operator "-". Thus if L2 is a tail of L1, L1-L2 is the list difference of L1 and L2: the list formed by removing L2 from the end of L1. Therefore a noun phrase has the form np(V,QL1-QL2,Fx-Fy,VL). V is the bound variable of the NP. QL1-QL2 is the quantifier store of the NP. We describe wh-movement using the standard gap-threading technique (Pereira and Shieber 1987), and Fx-Fy is the filler list. Finally, VL is a list of target-language variables representing NPs that are available for reference by a pronoun, which we will call the pronoun reference list.

Consider an NP consisting of a determiner and a head noun: "every man," "no woman," and so forth. The head noun supplies the range restriction of the NP's quantifier, and the determiner builds the quantifier given the range restriction. The bound variable of the NP is a feature of both the determiner and the head noun. Then the following rule generates NPs consisting of a determiner and a head noun:

(R1)
np(V,[Q| QL]-QL,Fx-Fx,VL)
→ det(V,Wff1,Q),n(V,Wff1).

The quantifier list [Q | QL] – QL = [Q] contains the quantifier for the NP. We have the following rules for common nouns:

(R2a) n(V1,pizza(V1)) → [pizza]
(R2b) n(V1,man(V1)) → [man]
(R2c) n(V1,woman(V1)) → [woman].

Recall that p(Wff1,Wff2) is a quantifier that maps Wff1 to Wff2. Then for determiners we have

(R3a) det(V2,Range,p(Wff1,some(V2,Range,Wff1))) → [a]
(R3b) det(V2,Range,p(Wff1,all(V2,Range,Wff1))) → [every]
(R3c) det(V2,Range,p(Wff1,unique(V2,Range, Wff1))) → [the]
(R3d) det(V2,Range,p(Wff1,not(some(V2,Range, Wff1)))) → [no].

Then we get

(62) np(V,[p(Wff1,all(V,man(V),Wff1))| QL]-QL,Fx-Fx,L) → [every man]
(63) np(V,[p(Wff1,not(some(V,woman(V),Wff1)))| QL]-QL,Fx-Fx,L) → [no woman].

Thus "every man" is an NP that binds the variable V and maps Wff1 to all(V,man(V),Wff1).

Following Moore (1988), we interpret the proper name "John" as equivalent to the definite description "the one named "John." "

(R4)
np(V,[p(Wff,unique(V,name(V,C),Wff))| QL]-QL,Fx-Fx,VL)
→ [Terminal], { proper_noun(Terminal,C) }.

The wff proper_noun(X,Y) means that X is a proper noun and Y is its logical form. Our lexicon includes the axioms

(L1a) proper_noun(john,john)
(L1b) proper_noun(mary,mary)
(L1c) proper_noun(bill,bill).

These axioms use the constant "john" to denote both a terminal symbol of the grammar and a constant of the target language—a convenient abuse of notation. Using (L1a) we get

(64) np(V,[p(Wff1,unique(V,name(V,john),Wff1))| QL]-QL,Fx-Fx,VL) → [john].

That is, "john" is an NP that binds the variable V and maps Wff1 to the wff unique(V,name(V,john),Wff1).

Pronouns use the "let" quantifier. We have

(R5)
np(V2,[p(V2,Wff1,let(V2,V,Wff1))| QL]-QL,Fx-Fx,VL)
→ [he], {member(V,VL)}.

If V is a variable chosen from the pronoun reference list VL, then "he" is an NP that binds the variable V2 and maps Wff1 to let(V2,V,Wff1). Thus, the pronoun refers back to a noun phrase whose bound variable is V. Later, we will see the rules that put variables into the list VL. As an example, we have

(65) np(V2,[p(Wff1,let(V2,V1,Wff1))| QL]-QL,Fx-Fx,[V1]) → [he].

The "let" quantifier in pronouns looks redundant, but it is useful because it makes the semantics of NPs uniform—every NP (except gaps) has a quantifier. This is helpful in describing conjoined NPs. Suppose that NP1 binds variable V and maps Wff1 to Wff2. Suppose NP2 also binds variable V and maps the same Wff1 to Wff3. Then the conjunction of NP1 and NP2 binds V and maps Wff1 to and(Wff2,Wff3):

(R6) np(V,[p(Wff1,and(Wff2,Wff3))| QL1]-QL3,Fx-Fx,VL)
→ np(V,[p(Wff1,Wff2)| QL1]-QL2,Fz-Fz,VL),
[and],
np(V,[p(Wff1,Wff3)| QL2]-QL3,Fy-Fy,VL).

As an example we have

(66) np(V,[p(Wff1,all(V,man(V),Wff1))| QL1]-QL1,Fx-Fx,L) → [every man]
(67) np(V,[p(Wff1,all(V,woman(V),Wff1))| QL2]-QL2,Fx-Fx,L) → [every woman]
(68)
np(V,
[p(Wff1,and(all(V,man(V),Wff1),all(V,woman(V),Wff1))) | QL1]-QL1,
Fx-Fx,VL)
→[every man and every woman].

That is, "every man and every woman" is an NP that binds variable V and maps Wff1 to

(69) and(all(V,man(V),Wff1),all(V,woman(V),Wff1)).

We also have

(70) np(V,[p(Wff1,let(V,V1,Wff1))| QL1]-QL1,Fx-Fx,[V1]) → [he]
(71) np(V,[p(Wff1,unique(V,name(V,john),Wff1))| QL2]-QL2,Fx-Fx,VL) → [john]
(72) np(V,[p(Wff1,and(let(V,V1,Wff1),unique(V,name(V,john),Wff1)))| QL1]-QL1,
Fx-Fx,[V1])
→ [he and john].

That is, "he and John" is an NP that binds V and maps Wff1 to

(73) and(let(V,V1,Wff1),unique(V,name(V,john),Wff1)

where V1 is chosen from the pronoun reference list. Thus the conjunction rule works for pronouns and proper nouns exactly as it does for NPs with determiners. There is a similar rule for disjunction of NPs.

In conjoining two NPs we combine their quantifiers, which are the first elements of their quantifier stores. We must also collect the remaining elements of both quantifier stores. The above rule achieves this result by concatenating difference lists in the usual way: if QL1-QL2 is the tail of the first NP's quantifier list, and QL2-QL3 is the tail of the second NP's quantifier list, then QL1-QL3 is the concatenation of the tails. In the present grammar both tails are empty, because the quantifier store of an NP contains at most one quantifier, but in a more general grammar the tails might contain quantifiers—for example, quantifiers from prepositional phrases modifying the NP. Thus the Montague-style semantics for NP conjunction and disjunction requires an extension of standard Cooper storage. When the quantifier store of an NP contains several quantifiers, we must be able to identify the one that represents the NP itself (as opposed to quantifiers that arise from attached PPs, for example). We must then be able to remove this quantifier from the store, build a new quantifier, and put the new quantifier back into the store.

Rule (R6) requires that the two NPs being conjoined should have quantifiers that bind the same variable. Suppose we had chosen the bound variables in the logical forms by using a global counter to ensure that no two quantifiers ever bind the same variable (as suggested in Section 2.4). Then (R6) could never apply. Thus our treatment of NP conjunction forces us to choose the bound variables after the quantifiers from conjoined NPs have been combined into a single quantifier, as described in Section 2.4. This choice in turn creates difficulties in implementing our treatment of *de re* attitude reports, as we will see in Section 3.1.

In this grammar, a conjunction of quantified NPs produces a logical form in which the two quantifiers are in separate wffs, and these wffs are joined by the connective *and*. Thus, neither quantifier is in the scope of the other. This gives the desired reading for a sentence such as "John has no house and no car":

(74) and(not(some(x,house(x),has(john,x))),not(some (x,car(x), has(john,x)))).

However, consider the sentence "John met a farmer and his wife" and suppose the pronoun "his" refers to "a farmer." Under our analysis, the quantifier from "a farmer" cannot bind a variable in the range restriction of the other quantifier—because its scope does not include the other quantifier. Thus, the Montagovian analysis of NP conjunction is certainly correct in some cases, but it cannot be the whole story.

2.6 VERB PHRASE AND SENTENCE RULES

Our grammar includes two kinds of transitive verbs: ordinary verbs like "eat" and "buy," and propositional attitude verbs like "want" and "seek." Only verbs of the second kind have *de dicto* readings. There is a *de dicto* reading for "John wants a Ferrari," which does not imply that there is any particular Ferrari he wants. There is no such reading for "John bought a Ferrari." To build a *de dicto* reading, a verb like "want" must have access to the quantifier of its direct object. Verbs like "buy" do not need this access. This leads to a problem that has been well known since Montague. The two kinds of verbs, although very different in their semantics, seem to be identical in their syntax. We would like to avoid duplication in our syntax by writing a single rule for VPs with transitive verbs. This rule must allow for both kinds of semantics.

Montague's solution was to build a general semantic representation, which handles both cases. When the verb is "eat" or "buy," one uses a meaning postulate to simplify the representation. Our solution is similar: we allow every transitive verb to have access to the quantifier of its direct object, and then assert that some verbs don't actually use the quantifier. However, our solution improves on Montague and Cooper by avoiding the simplification step. Instead, we build a simple representation in the first place.

A verb has one feature, the subcategorization frame, which determines what arguments it will accept and what logical form it builds. The rule for verbs says that if a terminal symbol has a subcategorization frame Subcat, then it is a verb:

(R7) v(Subcat) → [Terminal], { has_subcat(Terminal, Subcat) }.

A subcategorization frame for a transitive verb has the form

(75) trans(V1,V2,QL1,QL2,Wff1).

V1 is a variable representing the subject, and V2 is a variable representing the object. QL1 is the quantifier store of the object. QL2 is a list of quantifiers remaining after the verb has built its logical form. For an ordinary transitive verb, QL2 equals QL1. Wff1 is the logical form of the verb.

In the case of ordinary transitive verbs, we would like to assert once and for all that QL1 = QL2. Therefore, we write

(L2)
has_subcat(Terminal,trans(V1,V2,QL1,QL1,Wff))
:- ordinary_trans(Terminal,V1,V2,Wff).

This axiom says that for an ordinary transitive verb, the two lists of quantifiers are equal, and the values of the other features are fixed by the predicate "ordinary_trans." We have

(L3a) ordinary_trans(saw,V1,V2,saw(V1,V2))
(L3b) ordinary_trans(ate,V1,V2,ate(V1,V2)).

From (R7), (L2), and (L3a) we get

(76) v(trans(V1,V2,QL1,QL1,saw(V1,V2))) → [saw].

The features of a verb phrase are a variable (representing the subject), a wff (the logical form of the VP), a quantifier store, a list of fillers, and a pronoun reference list. The rule for a verb phrase with a transitive verb is

```
(R8)
      vp(V1,Wff1,QL2,Fx-Fy,L)
      → v(trans(V1,V2,QL1,QL2,Wff1)),
        np(V2,QL1,Fx-Fy,L).
```

If the verb is an ordinary transitive verb, then QL1 = QL2, so the quantifier store of the VP is equal to the quantifier store of the direct object. From (R1), (R3a), and (R2b) we have

```
(77) np(V,[p(Wff1,some(V,man(V),Wff1))| QL]-QL,Fx-
     Fx,L) → [a man].
```

Resolving (76) and (77) against the right side of R8 gives

```
(78)
     vp(V1,saw(V1,V2),[p(Wff1,some(V2,man(V2),
     Wff1))| QL]-QL,Fx-Fx,L)
     → [saw a man].
```

The quantifier store contains the quantifier of the NP "a man."

A sentence has four features: a wff, a quantifier store, a list of fillers, and a pronoun reference list. The rule for a declarative sentence is

```
(R9)
      s(Wff2,QL4,Fx-Fz,L) →
      np(V,QL1-QL2,Fx-Fy,L),
      vp(V,Wff1,QL2-QL3,Fy-Fz,L),
      {apply_quants(QL1-QL3,Wff1,QL4,Wff2)}.
```

The variable V represents the subject, so it becomes the first argument of the VP. QL1-QL3 is the concatenation of the quantifier stores from the subject and the VP. "Apply_quants" will apply some of these quantifiers to the logical form of the VP to produce the logical form Wff2 of the sentence. The list QL4 of remaining quantifiers becomes the quantifier store of the sentence. We have

```
(79)
     np(V1,[p(Wff0,all(V1,woman(V1),Wff0))| QL1]-
     QL1,Fx-Fx,L)
     → [every woman].
```

From (R9), (79), and (78), we get

```
(80)
     s(Wff2,QL4,Fx-Fx,L)
     → [every woman saw a man],
       {apply_quants([p(Wff0,all(V1,woman(V1),Wff0)),
               p(Wff1,some(V2,man(V2),Wff1))| QL3]-
               QL3,
               saw(V1,V2),
               QL4,
               Wff2)
       }.
```

The "apply_quants" subgoal has several solutions. Choosing the one in which "every woman" outscopes "a man," we get

```
(81) s(all(x,woman(x),some(y,man(y),saw(x,y))),QL3-
     QL3,Fx-Fx,L) → [every woman saw a man].
```

The derivation is not yet complete, because "s" is not the start symbol of our grammar. Instead we use a special symbol "start," which never appears on the right side of a rule. Thus, the start symbol derives only top-level sentences—it cannot derive an embedded sentence. This is useful because top-level sentences have a unique semantic property: their logical forms must not contain free variables. It might seem that one can eliminate free variables simply by applying all the quantifiers in the store. Hobbs and Shieber (1987) pointed out that this is not so—it is essential to apply the quantifiers in a proper order. Consider the sentence "every man knows a woman who loves him," with "him" referring to the subject. The subject quantifier binds a variable that occurs free in the range restriction of the object quantifier, so one must apply the object quantifier first in order to eliminate all free variables.

Therefore our grammar includes a filter that eliminates readings of top-level sentences containing free variables. Let free_vars(Wff1,L) mean that L is a list of the free variables of Wff1 in order of first occurrence. We omit the easy definition of this predicate. The rule for top-level sentences is:

```
(R10) start(Wff1) → s(Wff1,QL-QL,Fx-Fx,[]),
                   {free_vars(Wff1,[])}.
```

The goal free_vars(Wff1,[]) filters out readings with free variables. The above rule allows us to complete the derivation for "every woman saw a man":

```
(82) start(all(x,woman(x),some(y,man(y),saw(x,y))))
     → [every woman saw a man].
```

Having treated sentences, we can now consider gaps and relative clauses. The rule for gaps follows Pereira and Schieber (1987):

```
(R11) np(V,QL-QL,[gap(V)| Fx]-Fx,VL) → [].
```

This rule removes the marker gap(V) from the filler list, and makes the associated variable V the variable of the empty NP. The list difference QL-QL is the empty list, so the quantifier store of the gap is empty.

The rule that generates NPs with relative clauses is

```
(R12)
       np(V,[Q| QL]-QL,Fx-Fx,L)
       → det(V,and(Range1,Range2),Q),
         n(V,Range1),
         [that],
         s(Range2,QL1-QL1,[gap(V)]-[],[]).
```

The relative clause is a sentence containing a gap, and the logical form of the gap is the variable V—the same variable that the quantifier binds. The logical form of the relative clause becomes part of the range restriction of the quantifier. We have

(83) s(some(x,pizza(x),ate(V,x)),QL-QL,[gap(V)| Fx]-Fx,[]) → [ate a pizza].

The derivation of this sentence is much like the one for "every woman saw a man" above, except that in place of the subject "every woman" we have a gap as a subject. The rule for gaps above ensures that the variable of the gap is V, the only variable in the filler list, and its quantifier store is empty. Therefore, V appears as the first argument of the predicate "ate." Continuing the derivation we get

(84) np(V,[p(Wff1,some(V,and(man(V),some(x,pizza(x),ate(V,x))),Wff1))| QL]-QL,Fx-Fx,[])
→[a man that ate a pizza].

The string "a man that ate a pizza" is an NP that binds V and maps Wff1 to the wff

(85) some(V,and(man(V),some(x,pizza(x),ate(V,x))), Wff1).

Notice that in the rule for NPs with relative clauses, the quantifier store of the relative clause is empty. This means that no quantifier can be raised out of a relative clause. Thus there is no scope ambiguity in "I saw a man that loves every woman." According to Cooper (1979), this is correct. The restriction is easy to state because in our grammar, quantifier raising is combined with syntax and semantics in a single set of rules. It would be harder to state the same facts in a grammar like Pereira and Shieber's (1987), because quantifier raising there operates on a separate representation called a *quantifier tree*. This tree leaves out syntactic information that is needed for determining scopes—for example, the difference between a relative clause and a prepositional phrase.

3 Propositional Attitudes in the Grammar

3.1 Attitude Verbs Taking Clauses

The following rule introduces verbs such as "believe" and "know," which take clauses as their objects.

(R13)
```
      vp(V1,Wff1,QL1,Fx,L)
      → v(takes_s(V1,Wff2,Wff1)),
        s(Wff2,QL1,Fx,[V1 | L]).
```

The verb takes the logical form Wff2 of the object clause and the subject variable V1, and builds the wff Wff1 representing the VP. This rule also adds the subject variable to the pronoun reference list of the object clause. For the verb "thought," we have the following subcategorization frame:

(L4)
```
      has_subcat(thought,
            takes_s(V1,Wff1,
               thought(V1,Vars1,q(Wff1))))
      :- free_vars(Wff1,Vars1).
```

The subject variable becomes the first argument of the predicate "thought." The logical form of the object clause is Wff1, and it appears under a quotation mark as the third argument of "thought." The second argument of "thought" is the *de re* argument list, and the predicate "free_vars" ensures that the *de re* argument list is a list of the free variables in Wff1, as required by our convention. From the rule (R8) for verbs and (L4), we get

(86)
```
      v(takes_s(V1,Wff2,thought(V1,Vars1,q(Wff2))))
      → [thought], { free_vars(Wff2,Vars1) }.
```

The "free_vars" subgoal has not been solved—it has been postponed. Indeed it must be postponed, because as long as its first argument is a variable, it has an infinite number of solutions—one for each wff of our language.

Consider the example "John thought Mary ate a pizza." We consider two readings. "A pizza" is understood *de dicto* in both readings, but "Mary" is *de re* in one reading and *de dicto* in the other. The ambiguity arises from the embedded sentence, because the predicate "apply_quants" can either apply the quantifiers or leave them in the store. If it applies the quantifiers from "Mary" and "a pizza" in their surface order, we get

(87)
```
      s(unique(y,name(y,mary),some(x,pizza(x),ate
      (y,x)),QL-QL, Fx-Fx,L)
      → [mary ate a pizza].
```

From (R13), (86), and (87) we get

(88)
```
      vp(V1,
        thought(V1,Vars1,
            q(unique(y,name(y,mary),some(x,pizza
      (x),ate(y,x))))),
        QL-QL,Fx-Fx,L)
      → [thought Mary ate a pizza],
         {free_vars(unique(y,name(y,mary),some(x,piz-
      za(x), ate(y,x))),Vars1) }.
```

Since the first argument of "free_vars" is now a ground term, we can solve the "free_vars" subgoal, getting Vars1 = []. Then we have

(89)
```
      vp(V1,
        thought(V1,[],q(unique(y,name(y,mary),some(x,
      pizza(x),ate(y,x))))),
        QL-QL,Fx-Fx,L)
      → [thought Mary ate a pizza].
```

If we combine this VP with the subject "John" we get a sentence whose logical form is

```
(90)
    unique(z,name(z,john),
        thought(z,[],q(unique(y,name(y,mary),some
    (x,pizza(x),ate(y,x)))))).
```

Now for the reading in which "mary" is *de re*. Once again, consider the embedded sentence "Mary ate a pizza." Suppose that the predicate "apply_quants" applies the quantifier from "a pizza" and leaves the one from "Mary" in the store. We get

```
(91) s(some(x,pizza(x),ate(V1,x)),
        [p(Wff1,unique(V1,name(V1,mary),Wff1))|
        QL]-QL,
        Fx-Fx,
        L)
     → [Mary ate a pizza].
```

V1 is the bound variable of the NP "Mary." The predicate "apply_quants" will choose a value for V1 when it applies the quantifier. From (R13), (86), and (91), we get

```
(92)
    vp(V2,
      thought(V2,Vars1,q(some(x,pizza(x),ate(V1,
      x))))),
      [p(Wff1,unique(V1,name(V1,mary),Wff1))|
      QL]-QL,
      Fx-Fx,
      L)
    → [thought Mary ate a pizza],
       { free_vars(some(x,pizza(x),ate(V1,x)),Vars1) }.
```

In this case, the first argument of "free_vars" contains the meta-language variable V1. Then the "free_vars" subgoal has an infinity of solutions—one in which V1 = x and there are no free variables, and an infinite number in which V1 = y, for some y not equal to x, and the list of free variables is [y]. Therefore, it is necessary to postpone the "free_vars" subgoal once more. The standard technique for parsing DCGs does not allow for this postponing of subgoals, and this will create a problem for our implementation.

This problem would be greatly simplified if we had chosen to assign a different variable to every quantifier by using a global counter. The DCG parser would work from left to right and assign a target-language variable to each NP as soon as it parsed that NP. In the above example, "Mary" and "a pizza" would both have their variables assigned by the time we reached the right end of the VP. Then we could handle the "free_vars" subgoals by rewriting the grammar as follows: remove the "free_vars" subgoals from the lexical entries for the attitude verbs, and place a "free_vars" subgoal at the right end of each VP rule that introduces an attitude verb ((R13), (R15), and (R8)). This would ensure that when the parser attempted to solve the "free_vars" subgoal, its first argument would be a ground term. However, this solution would make it impossible to use the rule (R6) for NP conjunction (see Section 2.5). If we pick one solution for the problem of choosing bound variables, we have problems with NP conjunction; if we pick the other solution we get problems in implementing our analysis of *de re* attitude reports. This is the kind of difficulty that we cannot even notice, let alone solve, until we write formal grammars that cover a reasonable variety of phenomena.

Continuing our derivation, we combine the VP with the subject "John," apply the quantifier from "Mary," and get

```
(93) s(unique(z,name(z,john),
            unique(y,name(y,mary),
               thought(z,Vars1,q(some(x,pizza(x),ate
            (y,x)))))),
            QL-QL,Fx-Fx,L)
        → [John thought Mary ate a pizza],
          { free_vars(some(x,pizza(x),ate(y,x)),Vars1) }.
```

Now the first argument of "free_vars" is a ground term, because applying the quantifier that arose from "Mary" includes choosing the target language variable that the quantifier binds. The "free_vars" subgoal now has only one solution, Vars1 = [y]. Then the logical form of the sentence is

```
(94) unique(z,name(z,john),
            unique(y,name(y,mary),
               thought(z,[y],q(some(x,pizza(x),
                                 ate(y,x)))))).
```

This means that there is a term T that represents Mary to John, and John believes the sentence

```
(95) some(x,pizza(x),ate(T,x)).
```

For the sentence "John thought he saw Mary," our limited treatment of pronouns allows only one reading, in which "he" refers to John. Using (R9), we get the following reading for the embedded clause:

```
(96)
    s(let(y,V1,unique(x,name(x,mary),saw(y,x)),QL-
    QL,Fx-Fx,[V1])
    → [he saw Mary].
```

The pronoun "he" gives rise to a "let" quantifier, which binds the variable y to V1, the first member of the pronoun reference list. From (R13), (86), and (96) we get

```
(97)
    vp(V2,thought(V2,Vars1,q(let(y,V2,unique(x,name
    (x,mary), saw(y,x)),
      QL-QL,Fx-Fx,[])
    → [thought he saw Mary],
      {free_vars(let(y,V2,unique(x,name(x,mary),saw
      (y,x))),Vars1) }.
```

The VP rule (R13) unifies the subject variable V2 with the first element of the pronoun reference list of the embedded clause, so the "let" quantifier now binds the variable y to the subject variable. Once again, we postpone the "free_vars" goal until its first argument is a ground term. Combining this VP with the subject "John" gives

```
(98)
      s(unique(z,name(z,john),
           thought(z,Vars1,q(let(y,z,unique(x,name(x,mary),
                                                    saw(y,x)))))),
         QL-QL,Fx-Fx,VL)
      → [John thought he saw Mary],
           {free_vars(let(y,z,unique(x,name(x,mary),saw(y,x))),
                    Vars1) }.
```

The first argument of "free_vars" is now a ground term, and solving the "free_vars" subgoal gives Vars1 = [z]. The logical form of the sentence is

```
(99)
      unique(z,name(z,john),
           thought(z,[z],q(let(y,z,
                           unique(x,name(x,mary),
                                    saw(y,x)))))).
```

The dummy variable z stands for a term T that represents John to himself. Then John's belief looks like this:

```
(100)
let(y,T,
    unique(x,name(x,mary),
         saw(y,x))).
```

If John simplifies this belief, he will infer

```
(101) unique(x,name(x,mary),saw(T,x)).
```

3.2 ATTITUDE VERBS TAKING A CLAUSE WITH A GAP

We proposed the following logical form for "John knows who Mary likes":

```
(102) some(x,person(x),know(john,[x],q(like(mary,x)))).
```

The grammar will generate a similar logical form, except for the translations of the proper nouns. The existential quantifier comes from the word "who." The rules for "who" and "what" are

```
(R14a)  wh(V1,[p(Wff1,some(V1,and(person(V1),
        Wff1)))| QL]-QL) → [who]
(R14b)  wh(V1,[p(Wff1,some(V1,and(thing(V1),
        Wff1)))| QL]-QL) → [what].
```

The semantic features of a wh word are a variable, and a list containing a quantifier that binds that variable.

The following rule builds VPs in which the verb takes a wh word and a clause as its objects:

```
(R15)
      vp(V1,Wff3,QL1,Fx-Fx,L)
      → v(takes_wh(V1,Wff1,Wff2)),
        wh(V,QL0),
        s(Wff1,QL1,[gap(V)]-[],[V1 | L]),
        { apply_quants(QL0,Wff2,QL-QL,Wff3)}.
```

The embedded S contains a gap, and the variable of that gap is the one bound by the quantifier from the wh word. The main verb takes the subject variable and the logical form of the embedded S and builds a wff Wff2. The rule finally calls "apply_quants" to apply the quantifier from the wh word to Wff2. "Apply_quants" can apply any subset of the quantifiers in its first argument, but the rule requires the output list of quantifiers to be empty, and this guarantees that the quantifier from the wh word will actually be applied. The resulting wff becomes the logical form of the VP.

The rule requires a verb whose subcategorization frame has the form takes_wh(V1,Wff1,Wff2). "Know" is such a verb:

```
(L5)
      has_subcat(knows,
           takes_wh(V1,Wff1,know(V1,Vars1,q(Wff1))))
      :- free_vars(Wff1,Vars1).
```

Combining this clause with the rule (R7) for verbs gives

```
(103)
      v(takes_wh(V1,Wff1,know(V1,Vars1,q(Wff1))))
      → [knows], { free_vars(Wff1,Vars1) }.
```

Consider the example "John knows who Mary likes," and suppose "Mary" is understood *de dicto*. The embedded S has the following reading:

```
(104)
      s(unique(x,name(x,mary),like(x,V1)),QL-QL,
      [gap(V1) |Fx]-Fx,L)
      → [Mary likes].
```

The object of "likes" is a gap, so the variable V1 from the filler list becomes the second argument of the predicate "like." Resolving (103), (R14a), and (104) against the right side of (R15) gives

```
(105)
      vp(V2,Wff3,QL-QL,Fx-Fx,L)
      → [knows who Mary likes],
         { apply_quants([p(Wff1,some(V1,person(V1),
                                         Wff1))
                     | QL]-QL,
           know(V2,Vars1,q(unique(x,name(x,mary),
                                         like(x,V1))),
           QL-QL,
           Wff3),
        {free_vars(unique(x,name(x,mary),like(x,V1)),
        Vars1) }.
```

Solving the "apply_quants" subgoal gives

```
(106)
Wff3 = some(y,person(y),
          know(V2,Vars1,q(unique(x,name(x,mary),
          like (x,y))))).
```

Solving the "free_vars" subgoal gives Vars1 = [y], and we then have

```
(107)
Wff3 = some(y,person(y),
        know(V2,[y],q(unique(x,name(x,mary),
        like(x,y))))).
```

Therefore

```
(108)
    vp(V2,
        some (y,person(y),
        know(V2,[y],q(unique(x,name(x,mary),like
        (x,y))))),
      QL- QL,Fx-Fx,L)
    → [knows who Mary likes].
```

This VP combines with the subject "John" in the usual way to give a sentence whose logical form is

```
(109)
    unique(z,name(z,john),
        some(y,person(y),
          know(z,[y],q(unique(x,name(x,mary),
                                 like(x,y)))))).
```

3.3 ATTITUDE VERBS TAKING A NOUN PHRASE

Finally, we consider an example with "want." This verb is semantically very different from most transitive verbs, but syntactically it is an ordinary transitive verb, introduced by the rule already given:

```
(R8)
    vp(V1,Wff1,QL2,Fx-Fy,L)
    → v(trans(V1,V2,QL1,QL2,Wff1)),
      np(V2,QL1,Fx-Fy,L).
```

The difference between "want" and other transitive verbs is in its subcategorization frame:

```
(L6)
    has_subcat(wants,
           trans(V1,V2,QL1,QL2,wish(V1,Vars1,q
           (Wff1))))
    :- apply_quants(QL1,have(V1,V2),QL2,Wff1),
       free_vars(Wff1,Vars1).
```

Resolving this rule against the verb rule (R7) gives the following rule for the verb "wants":

```
(110)
    v(trans(V1,V2,QL1,QL2,wish(V1,Vars1,q(Wff1))))
    → [wants],
      { apply_quants(QL1,have(V1,V2),QL2,Wff1),
      free_vars(Wff1,Vars1)
      }.
```

The quantifier list QL1 contains the quantifier from the object NP. The predicate "apply_quants" may or may not apply this quantifier to the wff have(V1,V2), and this nondeterminism gives rise to a *de re/de dicto* ambiguity. If "apply_quants" does not apply the object quantifier, then QL2 = QL1, so the object quantifier is passed up for later application. Otherwise, QL2 is the empty list. As usual, the predicate "free_vars" ensures that the *de re* arguments obey our convention.

Consider the VP "wants a Porsche." The object "a Porsche" has the following interpretation:

```
(111)
    np(V2,[p,(Wff0,some(V2,porsche(V2),Wff0))| QL]-
    QL,Fx-Fx,VL)
    → [a porsche].
```

Resolving (110) and (111) against the left side of (R7) gives

```
(112)
    vp(V1,wish(V1,Vars1,q(Wff1)),QL2,Fx-Fx,VL)
    → [wants a porsche],
    {apply_quants([p(Wff0,some(V2,porsche(V2),
                 Wff0))| QL]-QL,
                 have(V1,V2),
                 QL2,
                 Wff1),
      free_vars(Wff1,Vars1)
      }.
```

One solution of the "apply_quants" subgoal is

```
(113)
    Wff1 = some(x,porsche(x),have(V1,x))
    QL2 = QL0-QL0.
```

Given this solution, the logical form of the VP is

```
(114) wish(V1,Vars1,q(some(x,porsche(x),have(V1,x))))
```

where V1 is the subject variable and the "free_vars" subgoal has been postponed. We can combine this VP with the subject "John" to get a sentence whose logical form is

```
(115)
    unique(y,name(y,john),
        wish(y,Vars1,q(some(x,porsche(x),have
        (y,x))))).
```

Solving the "free_vars" subgoal will then give Vars1 = [y], so the final logical form is

```
(116)
    unique(y,name(y,john),
        wish(y,[y],q(some(x,porsche(x),have(y,x))))).
```

This means that there is a term T that represents John to himself, and the sentence that John wishes to be true is

```
(117) some(x,porsche(x),have(T,x)).
```

This is a *de dicto* reading—there is not any particular Porsche that John wants.

The other solution for the "apply_quants" subgoal is

```
(118)
    Wff1 = have(V1,V2)
    QL2 = [p(Wff0,some(V2,porsche(V2),Wff0))|
    QL]-QL.
```

In this case, the logical form of the VP is

(119) wish(V1,Vars1,q(have(V1,V2)))

and its quantifier store is equal to QL2. Combining this VP with the subject "John" and applying the quantifiers gives a sentence whose logical form is

```
(120)
      unique(y,name(y,john),
           some(x,porsche(x),
              wish(y,Vars1,q(have(y,x))))).
```

Solving the "free_vars" subgoal gives Vars1 = [y,x] so the final logical form is

```
(121)
      unique(y,name(y,john),
           some(x,porsche(x),
           wish (y,[y,x],q(have(y,x))))).
```

This means that there exist terms T1 and T2 such that T1 represents John to himself, T2 represents some Porsche to John, and the sentence John wishes to be true is

(122) have(T1,T2)

This is a *de re* reading, in which John wants some particular Porsche.

The rules for verbs that take clauses as complements did not need to call "apply_quants," because the rules that build the clauses will call "apply_quants" and so create the desired ambiguity. In Cooper's grammar, all NPs have the option of applying their quantifiers, and so there is no need for verbs like "want" to apply quantifiers—they can rely on the rule that built the verb's object, just as other intensional verbs do. This is a minor advantage of Cooper's grammar.

4 Implementation and Conclusions

4.1 Implementation

The implementation uses the standard Prolog facility for parsing definite clause grammars. This facility translates the grammar into a top-down, left-to-right parser. This order of parsing leads to problems with the predicates "apply_quants" and "free_vars." We cannot run "free_vars" until its first argument is a ground term—otherwise we might get an infinite number of solutions. In our exposition, we solved this problem by delaying the execution of "free_vars." The standard DCG parser has no built-in facility for such delaying. As usual in such situations, there are two options: rewrite the predicates so that the existing interpreter works efficiently, or define a more general interpreter that allows the desired order of execution. The second approach is more desirable in the long run, because it achieves a central goal of logic programming: to use logical sentences that express our understanding of the problem in the clearest way. However, defining new interpreters is hard. The present implementation takes the low road—that is, the author rewrote the predicates so that the standard parser becomes efficient. In particular, the rule for top-level clauses calls a Prolog predicate that finds all *de re* argument lists in the final logical form and calls "free_vars" for each one.

There is a similar problem about the predicate "apply_quants" in the rule for "want." Since the parser works left to right, the quantifier from the object of "want" is not available when the logical form for the verb is being constructed. This means that the first argument of "apply_quants" is a free variable—so it has an infinite number of solutions. Here the implementation takes advantage of Prolog's "call" predicate, which allows us to delay the solution of a subgoal. The "apply_quants" subgoal is an extra feature of the verb "want" (in the case of an ordinary transitive verb, this feature is set to the empty list of goals). The rule for VPs with transitive verbs uses the "call" predicate to solve the subgoal—after the object of the verb has been parsed. At this point the first argument is properly instantiated and the call produces a finite set of solutions.

The grammar given above contains the rule NP $\rightarrow$ NP [and] NP, which is left recursive and cannot be parsed by the standard DCG parser. The implementation avoids this problem by adding a flag that indicates whether an NP is conjunctive. This gives the rule

(123) NP(+conj) $\rightarrow$ NP(−conj) [and] NP(Conj),

which is not left recursive—it assigns a right-branching structure to all conjunctions of NPs. These are the only differences between the grammar presented here and the Prolog code. The implementation was easy to write and modify, and it supports the claim that Prolog allows us to turn formal definitions into running programs with a minimum of effort.

4.2 Conclusions and Future Work

This paper has presented a new notation for a sentential theory of attitudes, which unlike most existing notations makes it possible to give a compositional semantics for attitude reports. Our notation distinguishes between the *de re* arguments of an attitude operator and the dummy variables, which stand for unspecified terms that represent the values of the *de re* arguments. The choice of dummy variables is quite arbitrary—just as the choice of bound variables in first-order logic is arbitrary. This allows us to impose a convention, which says that in fact the dummy variables are equal to the *de re* arguments. Given this convention, the logical form of a clause is the same whether it stands alone or appears as the argument of an attitude verb.

This is a simple proposal, and it would be easy to write and implement a grammar that applies the proposal to a few examples. The real question is whether the proposal is robust—whether it can function in a grammar that covers a variety of phenomena. We chose definite clauses and a first-order object language as our semantic formalism. We found a nonobvious interaction between our proposal for *de re* attitude reports, and two other problems about quantification: the choice of bound variables in a logical form, and the conjunction and disjunction of quantified NPs. We considered two possibilities for choosing the bound variables: assigning a different variable to every NP using a global counter, or requiring each quantifier to bind a variable that is not bound by any quantifier within its scope. The first approach makes it impossible to use our rules for NP conjunction and disjunction, while the second creates implementation problems for the *de re* argument lists. We resolved the dilemma by picking the second approach, and then rewriting the grammar to solve the implementation problems. Thus we have shown that the proposal for *de re* attitude reports is not just a plausible notion—it can be made to work in a grammar that is not trivial.

The grammar handles three kinds of attitude constructions: an attitude verb taking a clause as its object ("John thought he saw Mary"), an attitude verb taking a clause with a gap ("John knows who Mary likes"), and an attitude verb taking a noun phrase as its object ("John wants a Porsche"). The grammar includes *de re/de dicto* ambiguities, conjunction of NPs, and a very limited treatment of pronouns.

Another contribution of our work is that it seems to be the first unification grammar that builds logical forms, and at the same time respects the declarative semantics of the notation. We explicitly choose the bound variables of the logical form, instead of using meta-language variables. We also explain the semantics of our representation for quantified NPs: each NP has an infinite set of readings, one for each ordered pair in the extension of the application function. Several authors treat unification grammars with semantics as poor relations of Montague grammar. Pereira and Shieber (1987) propose to "encode" Montague's ideas in unification grammar, while Moore (1989) fears that building logical forms with unification grammar is "unprincipled feature hacking." We claim that these problems arise not from shortcomings of unification grammar, but from failure to take unification grammar seriously—which means respecting its declarative semantics.

The most obvious line for future work is to extend the grammar. It would be fairly easy to include complex wh noun phrases, such as "whose cat" or "how many children" (Cooper's grammar handled these). A more difficult problem arises when a gap is the object of an intensional verb—as in "John knows what Mary wants." The grammar can generate this sentence, but it assigns only a *de re* reading:

```
unique(x,name(x,john),
     some(y,thing(y),
        know(x,[y],q(unique(z,name(z,mary),
           wish(z,[z,y],q(have(z,y)))))))).
```

This is the only reading because the gap has an empty quantifier store—there is no quantifier available to be applied to the wff "have(z,y)." Yet there are examples in which such sentences do have *de dicto* readings. For example, consider "What John wants is a Porsche." Surely this sentence has a *de dicto* reading—yet the object of "want" is a gap, not a quantified NP. Cooper discusses this problem, but his grammars could not handle it, and neither can ours.

Hirst and Fawcett (1986) have argued that the ambiguities in attitude reports are more complex than the familiar distinction between *de re* and *de dicto* readings. They claim that the sentence "Nadia wants a dog like Ross's" has a reading in which Nadia doesn't want a particular dog (so the quantifier ∃ is inside the scope of the attitude operator), but the description "dog like Ross's" is the speaker's, not Nadia's (so the description is outside the scope of the attitude operator). This reading is certainly different from the usual *de re* and *de dicto* readings, in which either the whole logical form of the NP is under the attitude, or none of it is. To represent it, we must be able to separate the logical form of the NP "a dog like Ross's" into two parts (the quantifier ∃ and its range restriction), and we must be able to move the range restriction out of the scope of the attitude without moving the quantifier. This will mean that we cannot use the same mechanism for both quantifier scope ambiguities and the ambiguities that arise from attitudes. These extensions appear feasible, but they amount to a major change in the formalism.

Another possibility for future work is to incorporate the existing implementation into a question-answering program. This requires finding a way to reason about propositional attitudes efficiently without assuming unique standard designators—which means a substantial generalization of the work of Konolige. It also requires us to make much stronger claims about the properties of 'representation' than we have made in this paper. If these problems can be solved, it should be possible to build a natural language question-answering program that can describe the extent of its own knowledge—answering questions like "Do you know the salary of every employee?"

References

Asher, Nicholas M. and Kamp, John A. W. (1986). "The knower's paradox and representational theories of attitudes." In *Theoretical Aspects of Reasoning about Knowledge,* edited by Joseph Halpern, 131–147. Morgan Kaufmann.

Barwise, Jon and Cooper, Robin. (1981). "Generalized quantifiers and natural language." *Linguistics and Philosophy,* 4: 159–219.

Boer, Steven E. and Lycan, William G. (1986). *Knowing Who.* MIT Press.

Cooper, Robin. (1983). *Quantification and Syntactic Theory.* D. Reidel.

Cooper, Robin. (1979). "Variable binding and relative clauses." In *Formal Semantics and Pragmatics for Natural Languages,* edited by F. Gventhner and S. J. Schmidt, 131–169. D. Reidel.

des Rivieres, J., and Levesque, H. 1986. "The Consistency of Syntactical Treatments of Knowledge." in Joseph Y. Halpern, ed., *Theoretical Aspects of Reasoning about Knowledge.* Los Altos, CA, Morgan Kaufmann: 115–130.

Enderton, Herbert. (1972). *A Mathematical Introduction to Logic.* Academic Press.

Fisher, Anthony J. (1989). "Practical parsing of GPSG's." *Computational Linguistics,* 15: 139–148.

Gazdar, Gerald; Klein, Ewan; Pullum, Geoffrey; and Sag, Ivan. (1985). *Generalized Phrase Structure Grammar.* Harvard University Press.

Haas, Andrew R. (1986). "A syntactic theory of belief and action." *Artificial Intelligence,* 28: 245–292.

Hobbs, Jerry R. and Shieber, Stuart M. (1987). "An algorithm for generating quantifier scopings." *Computational Linguistics,* 13: 47–63.

Hirst, Graeme and Fawcett, Brenda. (1986). "The detection and representation of ambiguities of intension and description." In *Proceedings of the 24th Annual Meeting of the Association for Computational Linguistics,* 192–199.

Kaplan, David. (1975). "Quantifying in." In *The Logic of Grammar,* edited by Davidson, Donald and Harman, Gilbert, 112–144. Dickinson.

Konolige, Kurt. (1986). *A Deduction Model of Belief.* Pitman.

Kripke, Saul. "Outline of a Theory of Truth." *Journal of Philosophy,* 72: 690–715.

Lewis, David. (1979). "Attitudes *de dicto* and *de se.*" *Philosophical Review,* 88: 513–543.

Montague, Richard. (1974a). "The proper treatment of quantification in ordinary english." In *Formal Philosophy: Selected Papers of Richard Montague,* edited by Thomason, Richmond H., 247–270. Yale University Press.

Montague, Richard. (1974b). "Syntactical treatments of modality, with corollaries on reflexion principles and finite axiomatizability." In *Formal Philosophy: Selected Papers of Richard Montague,* edited by Thomason, Richmond H., 286–302. Yale University Press.

Montague, Richard and Kaplan, David. (1974). "A paradox regained." In *Formal Philosophy: Selected Papers of Richard Montague,* edited by Thomason, Richmond H., 271–301. Yale University Press.

Moore, Robert C. (1989). "Unification-based semantic interpretation." In *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics,* 33–41.

Moore, Robert C. (1988). "Propositional attitudes and russellian propositions." Report No. CSLI-88-119, Center for the Study of Language and Information, Menlo Park, CA.

Moore, Robert C. and Hendrix, Gary. (1979). "Computational models of belief and semantics of belief sentences." Technical report 187, SRI International, Menlo Park, CA.

Moran, Douglas. (1988). "Quantifier scoping in the SRI core language engine." In *Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics,* 33–40.

Pereira, Fernando C. N. and Shieber, Stuart M. (1987). *Prolog and Natural-Language Analysis.* Center for the Study of Language and Information, Stanford, CA.

Pereira, Fernando C. N. and Warren, David H. D. (1980). "Definite clause grammars for language analysis—a survey of the formalism and a comparison with augmented transition networks." *Artificial Intelligence,* 13: 231–278.

Perlis, Donald. (1988). "Languages with self reference II: knowledge, belief, and modality." *Artificial Intelligence,* 34: 179–212.

Perlis, Donald. (1986). "On the consistency of commonsense reasoning." *Computational Intelligence,* 2: 180–190.

Perlis, Donald. (1985). "Languages with self-reference I: foundations." *Artificial Intelligence,* 25: 301–322.

Rapaport, William J. (1986). "Logical Foundations for Belief Representation." *Cognitive Science,* 10: 371–42.

Quine, Willard van Orman. (1975). "Quantifiers and propositional attitudes." In *The Logic of Grammar,* edited by Davidson, Donald and Harman, Gilbert. Dickinson.

Quine, Willard van Orman. (1947). *Mathematical Logic.* Harvard University Press.

Tarski, Alfred. (1936). "Der Wahrheitsbegriff in den Formalisierten Sprachen. *Studia Philosophia,* 1: 261–405.

Thomason, Richmond H. (1986). "Paradoxes and semantic representation." In *Theoretical Aspects of Reasoning about Knowledge,* edited by Joseph Halpern, 225–239. Morgan Kaufmann.

Thomason, Richmond H. (1980). "A note on syntactical treatments of modality." *Synthese,* 44(3): 391–395.

Walther, Christoph. (1987). *A Many-Sorted Calculus for Resolution and Paramodulation.* Pitman.

Warren, David S. (1983). "Using lambda-calculus to represent meanings in logic grammars." In *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics,* 51–56.