SENSE LOGIC:
APPLICATION TO NATURAL LANGUAGE PROCESSING

by

CLARK ALAN SEXTON

B. A., Fort Hays State University, 1983

A MASTER'S THESIS

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computing and Information Sciences

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1989

Approved by:

W. A. Hankley

Major Professor

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENTS

I would like to thank my major professor, Dr. William Hankley, for his time and encouragement throughout the development of this thesis. I would also like to thank my committee members, Dr. David Schmidt and Dr. Maria Zamfir. I would also like to thank my family for their love and support. I would especially like to thank my wife, Teri, for her patience, love, and understanding throughout my graduate program. Finally, I would like to thank Dr. Stephen Tramel for his guidance and for first presenting Sommers' work to me.

1 Background

1.1 Introduction

In the early 60's, Fred Sommers introduced a logic for "sense relations". This sense-logic has as its domain meaningful/nonsensical sentences. Sense-logic is to be distinguished from conventional logic which has a domain of true/false sentences (propositions). Sommers' work has been applied in category theory, but has not yet received attention in computer science fields. The purpose of this paper is to (1) give a brief exposition of Sommers' sense-logic, (2) show the relationship it has to computer science work done on natural language processing (especially attribute grammars and John Sowa's conceptual graphs), and (3) briefly describe an implementation of some of Sommers' ideas.

Chapter 1 provides the background information which is necessary for those unfamiliar with the concepts utilized in the exposition of Sommers' Sense-Logic. If the reader is already acquainted with this background material, then the reader may wish to skip Chapter 1 and go directly to Chapter 2. Chapter 2 includes an explication of the central points of Sommers' Sense-Logic (as well as the key terms and notation). Chapter 3 shows how Sommers' Sense-Logic can be utilized in natural language processing, and therefore contains the main thesis of this paper. Chapter 4 covers

some theoretical issues related to Sommers' Sense-Logic as well as an implementation of the basic concepts of Sense-Logic. Chapter 5 summarizes the paper.

1.2 Relation of Sense-logic to Truth-logic

Sommers' sense-logic was presented in a series of articles (most notably "The Ordinary Language Tree" [Sommers, 1959], "Types and Ontology" [Sommers, 1963], and "Predicability" [Sommers, 1965]) during the years 1959 through 1965. The purpose of these articles was to show that much of natural language which was thought not to be governed by formal rules in fact was so governed. Sommers' sense-logic can be used to help determine the sense values of sentences (i.e., whether they are meaningful or nonsensical).

Sommers' sense-logic is analogous to ordinary truth-valued logic in that it offers a formal structure (viz., a set of rules) which can be used to derive "new" information (in the form of previously unrecognized implications) from a given body of information. It should be noted that, just as in truth-valued logic, it is necessary to supply some information (i.e., assumptions) before the rules of the logic can be utilized. Further, the results from the application of the rules are only as veracious as the information that was supplied. In other words, one cannot get "something for nothing", and the "garbage in, garbage out" law applies.

1.3 The Concept of "Nonsense" and Different Levels of Nonsense

Nonsense is something that does not make sense. Our discussion is, of course, restricted to nonsense that occurs in the use (misuse) of language. Since the primary function of language is to convey meaning, linguistic nonsense is often called "meaningless". There are five levels of linguistic meaninglessness (these are adapted from Sommers and Sowa), and they are:

(1) Gibberish:

Example: Opnjge drfnaqt zbwix

(2) Grammatical:

Example: A am I number prime.

(3) Categoreal:

Example: I am a prime number.

(4) Logical:

Example: I am prime minister and so is Joe.

(5) Empirical/Pragmatic

Example: I am prime minister.

These levels of meaninglessness are ranked in order of priority. There are rules for each level, and the rules for a given level are only applicable if the rules at all lower levels have been satisfied. Sense-logic deals with the type of nonsense at level (3), examples of which are commonly called "category mistakes".

1.4 The Denial/Negation Distinction and the Law of Excluded Middle

Since some of Sommers' definitions require an understanding of the denial/ negation distinction and its use in resolving some apparent problems with the Law of Excluded Middle, we will need to briefly cover these issues. The Law of Excluded Middle states that for any proposition, P, either it or its negation must be true (i.e., P V ~P is a tautology). However, it at least appears that some propositions violate the Law of Excluded Middle, since it seems that they and their negation are both false. If this is in fact the case it would have dire consequences for the foundations of logic, since the Law of Excluded Middle would have to be rejected or some reason found for classifying those propositions which violate it as special in some way. Consider the following examples that are typical of those supposed to violate the Law:

(a) The square root of 3 is red.

(b) The square root of 3 isn't red.

(c) 'Abbey Road' (an album by the Beatles) is happy.

(d) 'Abbey Road' (an album by the Beatles) is unhappy.

(e) George likes asparagus.

(f) George doesn't like asparagus.

(g) Socrates was in awe of Wilt Chamberlain.

(h) Socrates was not particulary impressed with Wilt

Chamberlain.

At first glance it appears that either (a) or (b) above must be true according to the Law of Excluded Middle, and likewise for the pairs (c) - (d), (e) - (f), and (g) - (h). However, the Law only applies to a proposition and its NEGATION, and we shall see that, in one way or another, each of the above pairs fails to meet this qualification. The first two pairs, (a) - (b) and (c) - (d), fail since none of the sentences involved is meaningful, and thus the question of truth or falsehood cannot even be asked. I included both of these pairs as examples because the second pair also has the problem of appearing to have another option, since being happy or being unhappy are often not viewed as exhausting the possible states (i.e., someone might be tempted to think that perhaps 'Abbey Road' is neither happy or unhappy, but rather in an emotional state "between" the two). This is obviously not possible, since 'Abbey Road' is not the kind of entity that can have an emotional state at all. We can, however, find a third option for the pair (e) - (f), since it is possible that George has no preference for asparagus one way or the other. Thus, while (e) and (f) are both propositions, (f) is not the negation of (e), and so they may both be false without contradicting the Law of Excluded Middle. The last pair of sentences, (g) - (h), are also both propositions, and although they may suffer the same defect as the pair (e) - (f),

they are included to illustrate the concept of a presupposition. Both (g) and (h) presuppose that Socrates knew of Wilt Chamberlain (which is highly unlikely), and so even if they were exhaustive alternatives for Socrates opinion of Wilt, they could still both be false without contradicting the Law of Excluded Middle. Although this may be intuitively clear, here is a somewhat more formal definition of presupposition: "S presupposes T" =df= "Neither S nor its denial are true unless T is true". Since this definition refers to the denial of a statement, we need to know what the denial of a statement is. Actually, we have already been using this idea in explaining the pair (e) - (f). Let us consider the following three sentences:

(1) Ralph is happy.

(2) Ralph is unhappy.

(3) It is not the case that Ralph is happy.

The first of these sentences affirms the predicate, 'happy', of Ralph, (i.e., it claims that the predicate, 'happy', applies to Ralph). The second sentence denies this by claiming that a predicate, 'unhappy', applies to Ralph. The second sentence constitutes a denial of the first in that the predicate of (2) is a negative version of the predicate of (1), while the subject is the same in both. For a given claim, C, whose subject is S and whose predicate is P, the denial of C, denoted by C', has the same subject, S, but

has -P (read 'not P') or some semantically equivalent version of -P as its predicate. The notion of denial (as well as affirmation) really only directly applies to predicates, but it is derivatively applied to whole statements. On the other hand, negation (and assertion) apply to whole statements, and (3), not (2), is the negation of (1) above. Since (2) is not the negation of (1), they can both be false without contradicting the Law of Excluded Middle. However, either (1) or (3) must be true according to the Law. One other crucial difference between (2) and (3) is that (2) is false if Ralph does not exist, but (3) is made trivially true. The reason for this is that (2) claims two things: first, that there is a Ralph and second, that he is unhappy, while (3) merely states that the claims that there is a Ralph and that he is happy are not both true.

2 Sommers' Sense-Logic

2.1 Definition of Key Terms and Notation

2.1.1 "Expression", U and N Relations

Having covered the denial/negation distinction, we can now proceed to examine the symbols and definitions Sommers uses in presenting his sense-logic. In truth-valued logic, a set of symbols has been introduced to aid in performing the calculations, and Sommers introduces a set of symbols for the same purpose. In addition to this set of symbols, Sommers presents a diagrammatical notation to illustrate his sense relational concepts. In explaining these concepts, it will prove useful to examine their corresponding diagrams. The form of diagrammatical notation that Sommers uses to illustrate his concepts is the tree notation. I assume the reader is familiar with the tree structure, and so I will merely demonstrate how it is used for the particular concepts as they are introduced.

Sommers' sense-logic involves the sense-relations of expressions, and so Sommers stipulates what he means by an expression: "'an expression' will be a word or phrase which can occur as the subject or the predicate of a well-formed subject-predicate sentence" [Sommers, 1959]. With this definition of "expression", Sommers then presents two sense relations that may obtain between any two expressions. The first of these is the U-relation:

U(X,Y) =df= X and Y (two expressions) can be used together meaningfully in a subject-predicate sentence.

The second relation is the N-relation:

N(X,Y) =df= ~ U(X,Y) (i.e., X and Y cannot be used together meaningfully in a subject-predicate sentence, and doing so results in a "nonsense" sentence).

Examples of U-related expression-pairs are (man,tall), (argument,interesting), and (Kareem Abdul-Jabbar,short). Note that the U-relation does not imply that sentences involving the expression-pairs are true. The sentence "Kareem Abdul-Jabbar is short" is meaningful, but false. Examples of N-related expression-pairs are (computer program,bald), (chair,false), and (angry,logic). The U and N relations are not dependent on the logical form of the sentence involving the expression-pairs. It does not matter whether the sentence is universal or existential, includes negatives, conjunctions, disjunctions, conditionals, etc., the expression-pair involved will be U in all possible permutations or it will be N in all possible permutations. We can see that the U relation is symmetric ((X)(Y)(U(X,Y) --> U(Y,X))), reflexive ((X)U(X,X)), but not transitive.

There are four general tree structures to represent the possible sense relation between two terms T1 and T2. These four possible structures are given in Figure 1.

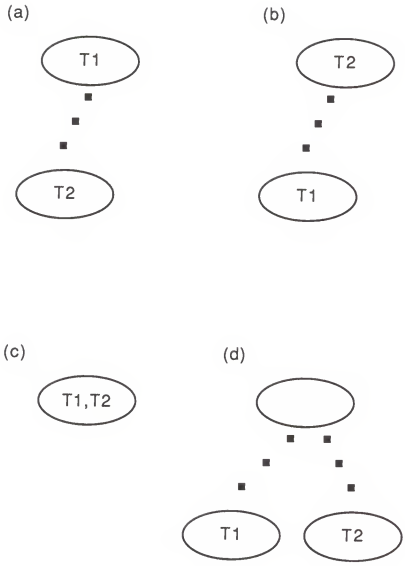

Figure 1: The Four General Tree Structures

The first three of these (a, b, and c) all represent two terms that are U related, while (d) represents the general form of two terms that are N related. The circles represent nodes in the tree, and the dots are used to indicate that there may be many nodes between the nodes in the diagrams. An informal statement of the U relation with regard to its representation in the tree structure is this: two terms are U related if and only if (1) one of the terms is "higher" than the other (forms (a) and (b) above) or (2) the terms are at the same node ((c) above). We shall examine each of the four general forms in detail as well as specific examples of each. Let us look at (a) first. The first thing to note about (a) is that T1 is higher in the tree than T2, and that there is a path from T1 to T2 such that one need only go down to arrive at T2 from T1 (i.e., at no node is it necessary to go up; all steps are in a downward direction). In Figure 2, the trees all have the same general structure as (a). (a.1) is the simplest, since it only contains the two U related terms with no nodes in between them in the tree structure. (a.2) is slightly more complicated, and it illustrates that two terms that are U related need not be contiguous in the tree diagram representing their relation to each other. It also shows that the lower term does not have to be to the left of the higher term. In fact, for our purposes, there is no significant difference between the left and right children for the parent node.

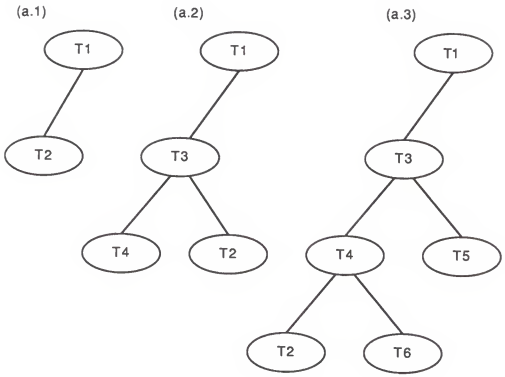

Figure 2: Three Examples with
T2 'Lower' than T1

For example, T4 and T2 are both U related to T3 (as well as T1) in (a.2). (a.3) is presented to show that the U relation is not transitive. In (a.3), all the terms are U related to T1 and T3, but T2 and T4 are N related to T5 and T6. If the U relation was transitive, then, since U(T2,T1) and U(T1,T6), it would follow that U(T2,T6), which is not the case. A specific example will amplify this point. Let A = 'interesting', B = 'argument', C = 'color'. It obviously does not follow that U(B,C) simply because U(B,A) and U(A,C). Moreover, since expressions like 'interesting' and 'is talked about' are U related to every expression (try to think of an expression for which it does not make sense to say that it is interesting), if the U relation was transitive, then every expression would be U related to every other expression.

2.1.2 "Predicable of", "Spanned by" Relations

Seeing the need for a transitive sense relation, Sommers constructed the "predicable of" relation. This relation requires a preliminary definition of the "spanned by" relation. "A predicate will be said to span a thing if it is predicated of it truly or falsely but not absurdly" [Sommers, 1963]. The spanned by relation is sometimes indicated by the use of the absolute value symbols. Thus, $|F|x$ =df= $Fx \vee F'x$ (where F' indicates the denial of F). The "spanned by" relation holds between a term and a thing, but what is desired is a relation that holds between terms. Thus, Sommers defines the "predicable of" relation in terms of the spanning relation:

F <-- G ("F is predicable of G")

=df= $(x)(|G|x \dashrightarrow |F|x)$.

F is predicable of G if and only if F spans whatever G spans. Note: I will use P for the predicable-of relation, and thus P(F, G) will mean "F is predicable of G". The "predicable of" relation is transitive and reflexive, but neither symmetric nor asymmetric. The U relation cited earlier can be defined in terms of the predicability relation:

U(F,G) =df= P(F, G) V P(G, F).

Hence, two terms, F and G, can be conjoined in a significant sentence if and only if F is predicable of G or G is predicable of F.

Let us now examine the predicable of relation in terms of the tree structure. In (a), P(T1,T2) (T1 is predicable of T2), but ~ P(T2,T1) (it is not the case that T2 is predicable of T1), while in (b), P(T2,T1) and ~ P(T1,T2), and in (c), both P(T1,T2) and P(T2,T1) (T1 and T2 are co-predicable). Finally, in (d), ~ P(T1,T2) and ~ P(T2,T1) (neither term is predicable of the other). Thus the tree structure provides an easy method of identifying the P relations of terms: we simply look to see whether a term is "higher" than (in the previously mentioned sense of "higher") or at the same node as another term. The tree structure also illustrates the transitivity of the P relation. For example, in (a.2), T1 is higher than T3, which is itself higher than T2, and so we can see that T1 is higher than T2 in the diagram, and so T1 must be predicable of T2.

2.1.3 Definition of "Category", "Type"

The spanning relation mentioned earlier can be used to define both the category and type concepts. A category is defined as a set of individuals which a given term spans, and so a category is relative to a given term. If we put this concept in terms of expressions rather than individuals, it turns out that, for a given expression, its category is the set of expressions to which it is U related. In Figure 3, the category of T2 has been encircled.

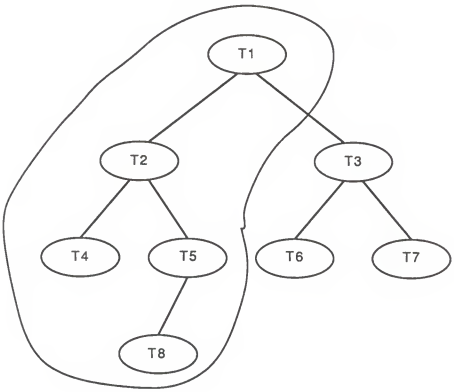

Figure 3: Example of a Category

The "is the same type as" relation is defined as follows:

a is the same type as b =df= (F)(|F|a <--> |F|b).

The members of a type are spanned by all the same predicates. In the tree structure, a type corresponds to a single node. For example, in Figure 4, T4 and T5 are members of the same type.

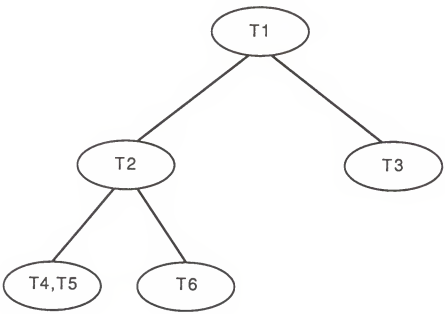

Figure 4: Example of a Type

2.2 Two Key Criteria: Type Difference and Individuality

Besides these definitions, two criteria are of central importance. The first is Sommers' criterion for type difference:

> "Two things are of different types if and only if there are two predicates P and Q such that it makes sense to predicate P of the first thing but not of the second, and it makes sense to predicate Q of the second thing but not of the first" [Sommers, 1965].

In Figure 5, we can see that T2 and T3 are of different types, since P may be meaningfully predicated of T2 but not of T3, and Q may be meaningfully predicated of T3 but not of T2.

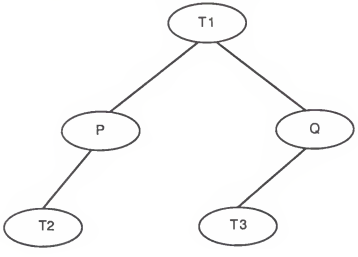

Figure 5: Example of Type Difference

The second is a criterion for individuality:

> "An entity x is an individual if and only if every pair of predicates P and Q that is true of x is such that either P is predicable of Q or Q is predicable of P" [Sommers, 1965].

This gives us the three possible tree structures illustrated in Figure 6.

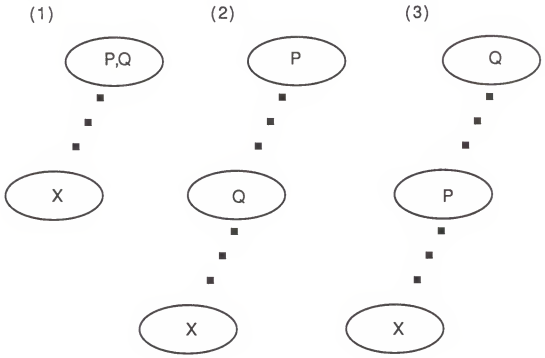

Figure 6: The Three Possible Tree Structures for an Individual, X

What cannot be the case, if x is an individual, is a tree of either of the forms shown in Figure 7, where P and Q are N related.

> "Entities that do not satisfy this criterion are 'category composites', and it is category composites, not individuals, that can give the misleading impression that there are innumerable counterexamples that undermine the plausibility of" [Greenberg, 1972] Sommers' rules.

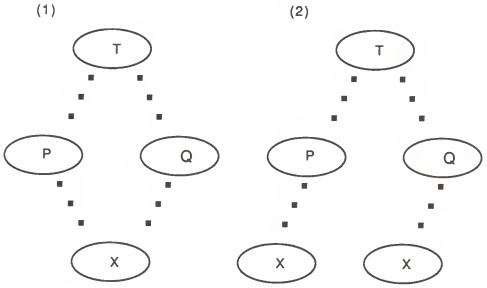

Figure 7: Two Impossible Tree Structures for an Individual, X

Sommers provides the following excellent example of a category composite:

> "Suppose someone defines a red earache to be what a man has when he has a red ear and a pain in that ear. By this definition, a red earache is an entity. But clearly, the entity so defined is not an individual. Moreover the statement 'Some aches are red' is still a category mistake, since there is no individual that instantiates the statement, nor could there be any such individual. It is nevertheless true that the (artificially composite) entity we have defined is literally red and literally an ache. For it is a pain cum red ear. But nothing about the category status of pains and ears has changed merely because we have a new 'entity'." [Sommers, 1965]

2.3 Two Rules: The Tree Rule and the Rule for Enforcing Ambiguity

Given this formal symbolism and the corresponding definitions of the terms presented, Sommers offers two rules governing sense relations. The first rule, known as the "Tree Rule", governs the relations of four terms:

- (U(A,B) & U(B,C) & U(A,D) & N(A,C) & N(B,D)).

To put this rule in english, it is not the case that there are four terms A, B, C, and D such that A and B are U related, B and C are U related, A and D are U related, A and C are N related, and yet B and D are N related. The proof for this rule is rather straightforward:

(U(A,B) & U(B,C) & N(A,C)) =>

(P(B,A) & P(B,C) & -P(A,B) & -P(C,B))

(U(A,B) & U(A,D) & N(B,D)) =>

(P(A,B) & P(A,D) & -P(B,A) & -P(D,A))

Thus:

- (U(A,B) & U(B,C) & U(A,D) & N(A,C) & N(B,D))

We can also demonstrate the validity of the tree rule by means of the tree diagrams. The first premise of the proof given above corresponds to the tree in Figure 8. The second premise of the proof is represented by the tree in Figure 9. The basic incon-

sistency here is that in the first tree B is higher than A, whereas in the second tree A is higher than B, and these cannot both be the case.

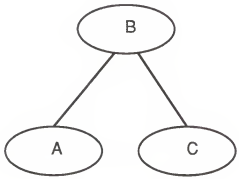

Figure 8: Tree Structure for Premise 1

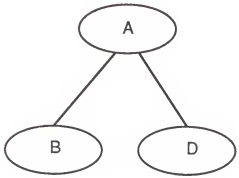

Figure 9: Tree Structure for Premise 2

The second rule, known as the "Rule for Enforcing Ambiguity", governs the relations of predicates and individuals:

$$-(\exists F)(\exists G)(\exists x)(\exists y)(\exists z)$$
$$(|Fx \;\&\; |Fy \;\&\; -|Fz \;\&\; |Gz \;\&\; |Gy \;\&\; -|Gx).$$

As Sommers claims:

> "If a, b, and c are any three things and P and Q are predicates such that it makes sense to predicate P of a and of b but not of c, and it makes sense to predicate Q of b and of c but not a, then P must be equivocal over a and b or Q must be equivocal over b and c. Conversely if P and Q are univocal predicates, then there can be no three things a, b, and c, such that P applies to a and to b but not to c while Q applies to b and to c but not to a" [Sommers, 1965].

What the Rule for Enforcing Ambiguity rules out is any tree of the form shown in Figure 10, if P and Q are univocal predicates, and a, b, and c are true individuals.

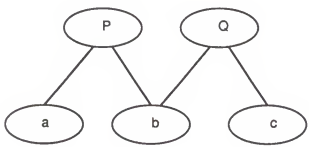

Figure 10: An Impossible Tree Structure

3. The Relationship of Sommers' Work to Computer Science

3.1 Introduction

In this section I shall show how Sommers' sense-logic rules may be used to help solve problems in the area of semantic-directed parsing. In order to do this, I must first define semantic-directed parsing and clarify its uses so that I can indicate the role Sommers' rules might play in this area. I shall also need to examine the tools currently used for semantic-directed parsing: attribute grammars and Sowa's conceptual graphs. Attribute grammars provide us with a notation for adding semantic constraints to a grammar, but they do not offer any semantic model, whereas Sowa's conceptual graphs give us an example of a semantic model, but do not explicitly deal with the syntactic elements of parsing. To facilitate the discussion of these areas, I shall present a number of sentences which will be used as a sort of running example throughout our examination of the topics listed above. The sentences are given below:

(1) a cat is an animal.

(2) the man is angry.

(3) the desk is angry.

(4) tigers last an hour.

I shall refer to these sentences at numerous points in this chapter and use them to illustrate the concepts and techniques of Sowa's conceptual graphs, attribute grammars, and Sommers' sense-logic rules, as well as the way in which these three areas can be integrated. However, before we can adequately deal with these issues we must spend a little time covering some background material on parsing in general.

3.2 Parsing

To parse a sentence is to delineate the syntactical relationship of each part of the sentence. Parsing is done via a grammar which defines the syntax of the language containing the sentence. There are different levels of sophistication of parsers. Some parsers rely entirely on syntactic information such as the grammatical categories of noun, verb, etc. in order to determine the correctness of the sentence under consideration. More complex parsers include semantic information that is used to either guide the parser or at least to reject otherwise syntactically correct sentences as incorrect. A purely syntax-directed parser cannot determine whether a given sentence makes sense or not, merely whether it is grammatically correct. There are, of course, degrees to which semantic information is used, from the fairly straightforward application of data type information utilized in Pascal compilers to the more complex use of the meanings of words in various attempts at natural language processing. A semantic-directed parser uses the meanings of the words already encountered in a sentence to determine the appropriate syntactic structure to attempt to apply to the remainder of the sentence.

Let us look at an example grammar for the sentences listed in the introduction of this chapter. The grammar is adapted from one presented in Sowa's book, Conceptual Structures: Information

Processing in Mind and Machine and although it will only handle a subset of English, it will suit our purpose. The grammar is listed in Figure 11.

```
sentence --> noun_phrase verb_phrase period
noun_phrase --> determiner noun_phrase2
            --> noun_phrase2
noun_phrase2 --> adjective noun_phrase2
             --> noun
verb_phrase --> tranverb noun_phrase
            --> intranverb
            --> linkverb adjective
            --> tobeverb ingtranverb noun_phrase
            --> tobeverb ingintranverb
            --> tobeverb noun_phrase
determiner --> 'the' | 'a' | 'an'
tobeverb --> 'is' | 'are' | 'was' | 'were'
linkverb --> 'is' | 'are'
ingtranverb --> 'eating'
tranverb --> 'lasts' | 'last' | 'eats' | 'eat'
           | 'runs' | 'run' | 'thinks'
           | 'think' | 'takes' | 'take'
ingintranverb --> 'eating' | 'running'
intranverb --> 'runs' | 'run' | 'thinks'
             | 'think' | 'dies' | 'die'
noun --> 'cat' | 'cats' | 'dog' | 'dogs' | 'tiger'
       | 'tigers' | 'lion' | 'lions' | 'animal'
       | 'animals' | 'man' | 'men' | 'woman'
       | 'women' | 'flower' | 'flowers' | 'desk'
       | 'desks' | 'chair' | 'chairs' | 'car'
       | 'cars' | 'robot' | 'robots' | 'hour'
       | 'hours' | 'game' | 'games' | 'speech'
       | 'speeches'
adjective --> 'angry' | 'tall' | 'hopeful' | 'red'
            | 'hungry' | 'weak' | 'dead' | 'long'
period --> '.'
```

Figure 11: Sample Context Free Grammar

This grammar may be viewed as having two parts: the phrase structure grammar and the word grammar. The phrase structure grammar

consists of the rules specifying the relative positions the various syntactic categories (e.g., noun_phrase, verb_phrase) may have in a sentence. The word grammar specifies the words that fall under a given category (e.g., 'cat' is a noun). I shall only examine the phrase structure grammar rules, since the word grammar rules are self-explanatory. The first rule of the grammar states that a sentence consists of a noun phrase followed by a verb phrase followed by a period, and so in order to successfully parse a given sentence, we must satisfy the conditions of the subgoals noun_phrase, verb_phrase, and period. The rule for noun_phrase states that a noun phrase consists of a determiner followed by a noun_phrase2 or just a noun_phrase2 (effectively making the determiner optional). Subsequently, a noun_phrase2 consists of either an adjective followed by another noun_phrase2 or just a noun (this allows there to be any number of adjectives preceding the noun in the noun phrase). The rule for the verb_phrase is more complicated, since it has more options. The first option says that a verb_phrase may consist of a tranverb (transitive verb) followed by a noun_phrase. The second option allows a verb_phrase to consist of an intranverb (intransitive verb) alone. The third option provides for the case of a linkverb (linking verb) followed by an adjective. The fourth option states that a verb_phrase may consist of a tobeverb (a form of the verb followed by a noun_phrase. The fifth option allows a verb_phrase to consist of a tobeverb followed by an ingintranverb

(an intransitive verb ending in 'ing'). The sixth, and final, option provides for the case of a tobeverb followed by a noun_phrase. These are all of the phrase structure grammar rules, and the remaining word grammar rules simply specify the categories that the words in the vocabulary belong in.

In order to better understand how the grammar works, let us parse an example sentence with the grammar. To parse sentence (1), "a cat is an animal", we would start with the 'start' (or 'goal') symbol, 'sentence'. Since every sentence in this grammar must have a noun_phrase first, then we must meet the conditions of the rule for a noun_phrase. The first option of the noun_phrase requires a determiner (either 'the', 'a', or 'an'), and since our example sentence begins with 'a', it matches 'a' as a determiner and meets the first requirement of the noun_phrase. The determiner must be followed by a noun_phrase2, and one of the options for noun_phrase2 is just a noun, which is matched by the word 'cat'. Thus, the noun_phrase has been successfully parsed, and we continue to the verb_phrase. The last option of the verb_phrase is a tobeverb (a form of the verb 'to be') followed by a noun_phrase. The tobeverb is matched by the word, 'is', and the noun_phrase is matched by the words 'an' and 'animal' much like the words 'a' and 'cat' matched the initial noun_phrase. Lastly, the period is matched by '.', and since this is the end of our sample sentence, we have successfully

parsed it. A parse tree for sentence (1) is given in Figure 12.

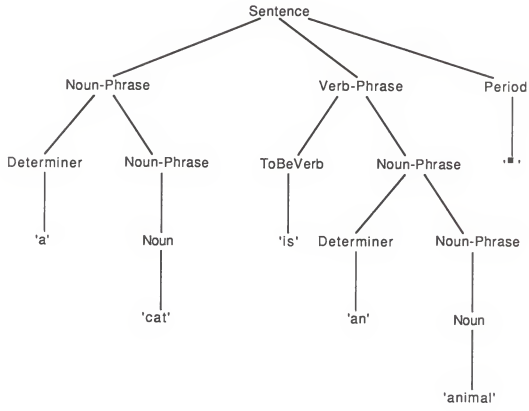

Figure 12: Parse Tree for Sentence 1

As mentioned previously, two of the tools currently used in semantic-directed parsing are attribute grammars and Sowa's conceptual graphs. A brief examination of each is necessary in order to indicate the role Sommers' sense-logic rules could have in semantic-directed parsing.

3.3 Relation of Sommers' Work to Sowa's Conceptual Graphs

3.3.1 Sowa's Conceptual Graphs

The second tool used in semantic-directed parsing mentioned above is John Sowa's conceptual graphs. Sowa's graphs are used to represent semantic information in much the same way as the predicate logic notation. The primary difference is the graphical representation allowed via the 'display form' of the graphs, which uses boxes, circles, and arrows to illustrate concepts and their relations. For example, conceptual graphs for each of the sample sentences given earlier are presented in Figure 13.

(1) a cat is an animal.

(2) the man is angry.

(3) the desk is angry.

(4) tigers last an hour.

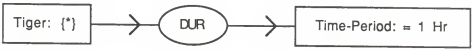

Figure 13: Sample Conceptual Graphs

It should be possible to utilize Sommers' sense-logic rules to supplement Sowa's conceptual graphs. In section 3.4 of his book, Conceptual Structures: Information Processing in Mind and Machine , Sowa defines 'canonical' graphs as 'the meaningful graphs that represent real or possible situations in the external world' [Sowa, 1984]. These are, then, the graphs which 'make sense', and that is

just another way of saying that their concepts (expressions) are category-correct. Sowa claims that we have developed, through experience, a host of canonical graphs, and this does indeed seem to be the case. However, these graphs need to be checked for type consistency. Specifically, a single graph may be category-correct within itself, but it may contain expressions which are used in inconsistent ways in other graphs. Thus, the set of graphs needs to be checked for type consistency. Further, we do not need to depend entirely on experience, since we can now check to determine which graphs are canonical by using Sommers' sense-logic rules. Given the type information for the expressions in the graphs, we can assertain whether any category mistakes have been made.

3.3.2 A Comparison of Sommers' and Sowa's Type Hierarchies

In this section I shall compare the type hierarchy developed through Sommers' sense-logic rules to the one Sowa presents in connection with his conceptual graphs. To facilitate this comparison, I will present a set of concepts and illustrate how both Sowa and Sommers would represent them in their respective type hierarchies. The set of concepts is listed below:

> agent, event, entity, attribute, animate, angel, machine, animal, robot, vertebrate, mammal, carnivore, herbivore, omnivore, wild-animal, feline, canine, dog, cat, tiger, jaguar, lion, red, physical-object, plant, angry, man, interesting

The relations between these concepts is represented in a type lattice such as Sowa might use in Figure 14. The relations between the same set of concepts is represented in a tree structure such as Sommers might use in Figure 15.

We can see from this relatively small set of concepts the differences between the two approaches. I have selected terms primarily from Sowa's list of type labels given in Appendix B of his book, Conceptual Structures: Information Processing in Mind and Machine, so that there would be less dispute over the form of the

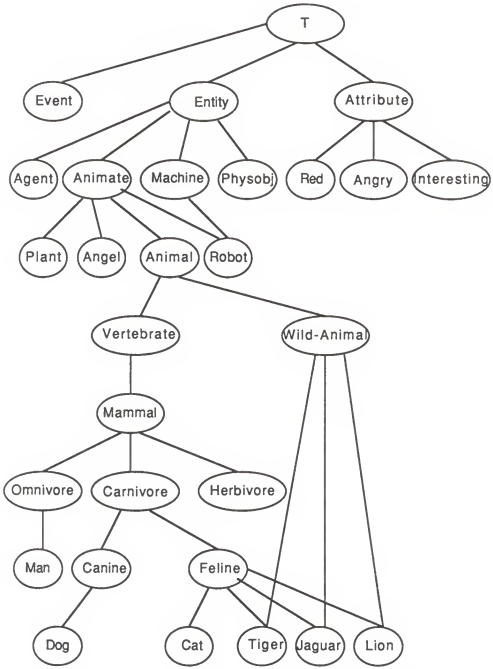

Figure 14: Example Type Hierarchy (Sowa)

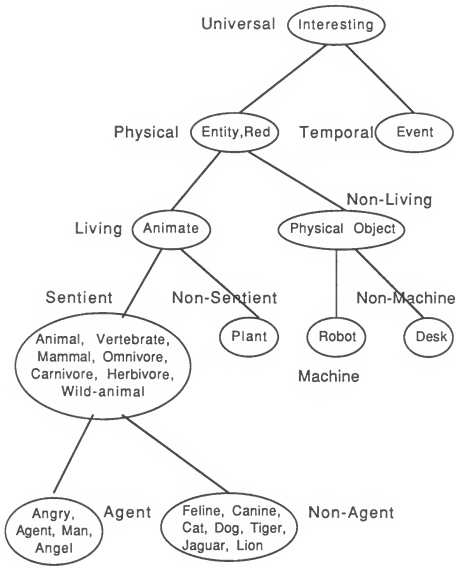

Figure 15: Example Type Hierarchy (Sommers)

lattice. In fact, for most of the concepts, Sowa explicitly states their relation to at least one other concept, and although he does not collect all of the concepts into a single lattice as I have done, it is relatively straightforward to do so, given his claims about them. He could quibble about the placement of such terms as 'plant', 'agent', 'angry', 'omnivore', 'herbivore', 'canine', and 'man', but I believe their positions in the lattice are faithful to Sowa's general approach. The concepts 'vertebrate', 'carnivore', 'cat, 'dog', 'feline', 'wild-animal', 'jaguar', 'lion', and 'tiger' are all taken from pages 81 and 82 of Sowa's text, and their placements in the lattice seem dictated from what Sowa says regarding them on those pages. Given these two diagrams, we can see that Sowa's approach requires a distinct node for each concept, whereas Sommers' method requires that all concepts that are in the same type (using Sommers definition of 'type') be located at the same node. There is obviously, then, a difference in the information represented in the two diagrams. In Sowa's diagram, the nodes represent classes of entities (actions, attributes, etc.) that share a set of characteristics. These characteristics must include such factual information (i.e., knowledge about the world) as that tigers have stripes and lions do not, or at least some information along these lines, since we could not justify placing them at different nodes unless there was some difference. This information cannot be limited to the analytic information supplied from the

meaning of the term, since it is not true by definition that tigers are wild animals, and in fact, it is not even universally true. To contrast this with Sommers' approach, Sommers' diagram only represents the information contained in the senses of the concepts, and so no information about the real world is represented, but only the analytical truths based on the definitions of the terms. Moreover, Sommers' diagram does not even represent all of the information that can be derived from the definitions of the terms, since it only indicates the senses of the concepts and not their meanings (recall that 'short' and 'tall' have the same sense but not the same meaning). Thus, there is a crucial distinction between the type information represented in Sommers' diagram and the knowledge of the real world represented in Sowa's diagram.

3.4 Relation of Sommers' Work to Attribute Grammars

3.4.1 Attribute Grammars

Typically, an attribute grammar is developed as an extension of an existing grammar. The existing grammar defines the syntax, and thus the grammatical categories of the language in question. An attribute grammar includes the original grammar as well as a set of attributes describing the grammatical entities in the language. Thus, we might include the attribute 'noun' for that part of a sentence which was parsed and found to be in that syntactic category. This attribute could then be used later to aid in parsing the remainder of the sentence. It is in this fashion that attribute grammars allow us to include some semantic and contextual information in the parsing of sentences. For a discussion of attribute grammars, the reader may wish to read Compiler Design Theory by Lewis, Rosenkrantz, and Stearns [Lewis, 1976]. For a specific example of an attribute grammar, I have included the following C-Prolog program (see Figures 16 and 17) which implements an attribute grammar for the context-free grammar introduced earlier. The attributes are now attached to the non-terminals and passed up the parse tree. For example, in the production for 'sentence', the type of the noun_phrase is derived from the productions related to 'noun_phrase' and passed up via the attribute, Type1.

```
sentence(sentence(N,V,P)) -->
    noun_phrase(N,Num,Type1),
    verb_phrase(V,Num,Type2),
    period(P).

noun_phrase(np(det(D,Num),N),Num,Type) -->
    determiner(det(D,Num)),
    noun_phrase2(N,Num,Type).

noun_phrase(N,Num,Type) -->
    noun_phrase2(N,Num,Type).

noun_phrase2(np2(adj(A,Type1),N),Num,Type) -->
    adjective(adj(A,Type1)),
    noun_phrase2(N,Num,Type).

noun_phrase2(N,Num,Type) -->
    noun(N,Num,Type).

verb_phrase(vp(V,N),Num,Type1) -->
    tranverb(V,Num,Type1,Type2),
    noun_phrase(N,Num1,Type3).

verb_phrase(V,Num,Type) -->
    intranverb(V,Num,Type).

verb_phrase(vp(V,adj(A,Type)),Num,Type) -->
    linkverb(V,Num,Type1,Type2),
    adjective(adj(A,Type)).

verb_phrase(vp(V1,V2,N),Num,Type3) -->
    tobeverb(V1,Num,Type1,Type2),
    ingtranverb(V2,Num,Type3,Type4),
    noun_phrase(N,Num1,Type5).

verb_phrase(vp(V1,V2),Num,Type) -->
    tobeverb(V1,Num,Type1,Type2),
    ingintranverb(V2,Num,Type).

verb_phrase(vp(V,N),Num,Type) -->
    tobeverb(V,Num,Type1,Type2),
    noun_phrase(N,Num,Type).

period(period(.)) --> [46].
```

Figure 16: Attribute grammar - Phrase Structure

```
determiner(det(the,_)) --> [the].
determiner(det(a,singular)) --> [a].
determiner(det(an,singular)) --> [an].

tobeverb(verb(is,singular,universal,universal),
        singular,universal,universal) --> [is].
tobeverb(verb(are,plural,universal,universal),
        plural,universal,universal) --> [are].
tobeverb(verb(was,singular,universal,universal),
        singular,universal,universal) --> [was].
tobeverb(verb(were,plural,universal,universal),
        plural,universal,universal) --> [were].

linkverb(verb(is,singular,universal,universal),
        singular,universal,universal) --> [is].
linkverb(verb(are,plural,universal,universal),
        plural,universal,universal) --> [are].

ingtranverb(verb(eating,Num,living,physical),
        Num,living,physical) --> [eating].

tranverb(verb(lasts,singular,temporal,temporal),
        singular,temporal,temporal) --> [lasts].
tranverb(verb(last,plural,temporal,temporal),
        plural,temporal,temporal) --> [last].
tranverb(verb(eats,singular,living,physical),
        singular,living,physical) --> [eats].
tranverb(verb(eat,plural,living,physical),
        plural,living,physical) --> [eat].
tranverb(verb(eats,singular,machine,physical),
tranverb(verb(eats,singular,machine,physical),
        singular,machine,physical) --> [eats].
tranverb(verb(eat,plural,machine,physical),
        plural,machine,physical) --> [eat].
tranverb(verb(runs,singular,sentient,temporal),
        singular,sentient,temporal) --> [runs].
tranverb(verb(run,plural,sentient,temporal),
        plural,sentient,temporal) --> [run].
tranverb(verb(thinks,singular,agent,universal),
        singular,agent,universal) --> [thinks].
tranverb(verb(think,plural,agent,universal),
        plural,agent,universal) --> [think].
tranverb(verb(takes,singular,sentient,physical),
        singular,sentient,physical) --> [takes].
tranverb(verb(take,plural,sentient,physical),
        plural,sentient,physical) --> [take].
```

```
ingintranverb(verb(eating,Num,sentient),
        Num,sentient) --> [eating].
ingintranverb(verb(running,Num,sentient),
        Num,sentient) --> [running].

intranverb(verb(runs,singular,sentient),
        singular,sentient) --> [runs].
intranverb(verb(run,plural,sentient),
        plural,sentient) --> [run].
intranverb(verb(thinks,singular,agent),
        singular,agent) --> [thinks].
intranverb(verb(think,plural,agent),
        plural,agent) --> [think].
intranverb(verb(dies,singular,living),
        singular,living) --> [dies].
intranverb(verb(die,plural,living),
        plural,living) --> [die].
intranverb(verb(dies,singular,machine),
        singular,machine) --> [dies].
intranverb(verb(die,plural,machine),
        plural,machine) --> [die].

noun(noun(cat,singular,nonagent),singular,nonagent)
    --> [cat].
noun(noun(cats,plural,nonagent),plural,nonagent)
    --> [cats].
noun(noun(dog,singular,nonagent),singular,nonagent)
    --> [dog].
noun(noun(dogs,plural,nonagent),plural,nonagent)
    --> [dogs].
noun(noun(tiger,singular,nonagent),singular,nonagent)
    --> [tiger].
noun(noun(tigers,plural,nonagent),plural,nonagent)
    --> [tigers].
noun(noun(lion,singular,nonagent),singular,nonagent)
    --> [lion].
noun(noun(lions,plural,nonagent),plural,nonagent)
    --> [lions].
noun(noun(animal,singular,sentient),singular,sentient)
    --> [animal].
noun(noun(animals,plural,sentient),plural,sentient)
    --> [animals].
noun(noun(man,singular,agent),singular,agent)
    --> [man].
noun(noun(men,plural,agent),plural,agent)
    --> [men].
```

```
noun(noun(flower,singular,nonsentient),singular,nonsentient)
    --> [flower].
noun(noun(flowers,plural,nonsentient),plural,nonsentient)
    --> [flowers].
noun(noun(desk,singular,nonmachine),singular,nonmachine)
    --> [desk].
noun(noun(desks,plural,nonmachine),plural,nonmachine)
    --> [desks].
noun(noun(chair,singular,nonmachine),singular,nonmachine)
    --> [chair].
noun(noun(chairs,plural,nonmachine),plural,nonmachine)
    --> [chairs].
noun(noun(car,singular,machine),singular,machine)
    --> [car].
noun(noun(cars,plural,machine),plural,machine)
    --> [cars].
noun(noun(robot,singular,machine),singular,machine)
    --> [robot].
noun(noun(robots,plural,machine),plural,machine)
    --> [robots].
noun(noun(hour,singular,temporal),singular,temporal)
    --> [hour].
noun(noun(hours,plural,temporal),plural,temporal)
    --> [hours].
noun(noun(game,singular,temporal),singular,temporal)
    --> [game].
noun(noun(games,plural,temporal),plural,temporal)
    --> [games].
noun(noun(speech,singular,temporal),singular,temporal)
    --> [speech].
noun(noun(speeches,plural,temporal),plural,temporal)
    --> [speeches].

adjective(adj(angry,agent)) --> [angry].
adjective(adj(tall,physical)) --> [tall].
adjective(adj(hopeful,agent)) --> [hopeful].
adjective(adj(red,physical)) --> [red].
adjective(adj(hungry,sentient)) --> [hungry].
adjective(adj(weak,sentient)) --> [weak].
adjective(adj(dead,living)) --> [dead].
adjective(adj(dead,machine)) --> [dead].
adjective(adj(long,temporal)) --> [long].
adjective(adj(long,physical)) --> [long].
```

Figure 17: Attribute grammar - Vocabulary

The program presented in Figures 16 and 17 utilizes a Definite Clause Grammar (DCG), and since the use of a DCG in C-Prolog is not well-documented, I shall briefly go over the notation and the basic concepts involved. The first thing to note regarding the notation is the use of '-->'. In C-Prolog, '-->' is a system operator specifically designed for use in DCGs. It has essentially the same meaning in C-Prolog as it has in a DCG in general. For example, in Figure 16 we have:

```
sentence(sentence(N,V,P)) -->
    noun_phrase(N,Num,Type1),
    verb_phrase(V,Num,Type2),
    period(P).
```

and the '-->' in this rule means that a sentence consists of a noun phrase followed by a verb phrase followed by a period. The Head of the rule (the part on the left-hand side of the '-->') must be a non-terminal in the grammar, and the Body (the part on the right-hand side) is a sequence of terminals and/or non-terminals. One of the hidden apsects of the DCG in C-Prolog are the extra parameters that are not listed in the parameter list of the Head, but are listed in the initial call to the rule. For example, there is only one parameter shown in the rule for sentence (i.e., the functor, 'sentence(N,V,P)'), but when the rule for sentence is called, we must pass both the list of words (tokens) and the list that will be

left after the sentence has been parsed and the appropriate tokens have been consumed (removed from the list). In the case of parsing a single sentence, the remainder should be the empty list ('[]'), and so a call to sentence might look like: 'sentence(S, Wordlist, [])'. The last aspect of the use of DCGs in C-Prolog that I shall discuss is the consumption of tokens, which is done by a fact such as:

```
determiner(det(a,singular)) --> [a].
```

The brackets ('[', ']') are system operators in C-Prolog, and they indicate that the token between them is to be removed from the input list and that the remaining list (minus the token) is to be returned. So, when a call to determiner is made, if the first token on the input list matches the token in the brackets, then the call succeeds and the token is consumed. For a more complete account of the use of a DCG in C-Prolog, the reader is instructed to examine "Use of Definite Clause Grammars", a Master's Report by Weilin Chen [Chen, 1987] and The Art of Prolog: Advanced Programming Techniques by Leon Sterling and Ehud Shapiro [Sterling, 1986].

Figure 18 illustrates the results of parsing sentences (1) and (2), using the attribute grammar presented in Figures 16 and 17.

```
a cat is an animal.

sentence(
  np(det(a,singular),
     noun(cat,singular,nonagent)),
  vp(verb(is,singular,universal,universal),
     np(det(an,singular),
        noun(animal,singular,sentient))),
  period(.))

the man is angry.

sentence(
  np(det(the,singular),
     noun(man,singular,agent)),
  vp(verb(is,singular,universal,universal),
     adj(angry,agent)),
  period(.))
```

Figure 18: Functor results of parsing sentences (1) and (2) (with attached attributes)

Figure 18 and Figures 19 and 20 are equivalent ways of representing the parsed sentences. Note that the key difference in Figure 19 from the parse tree presented earlier (Figure 12) is the inclusion of the attributes pertaining to the number (singular or plural) and type (e.g., nonagent, universal, agent). It should also be noted that the Functor results and the parse trees are merely two equivalent ways of representing the same information.

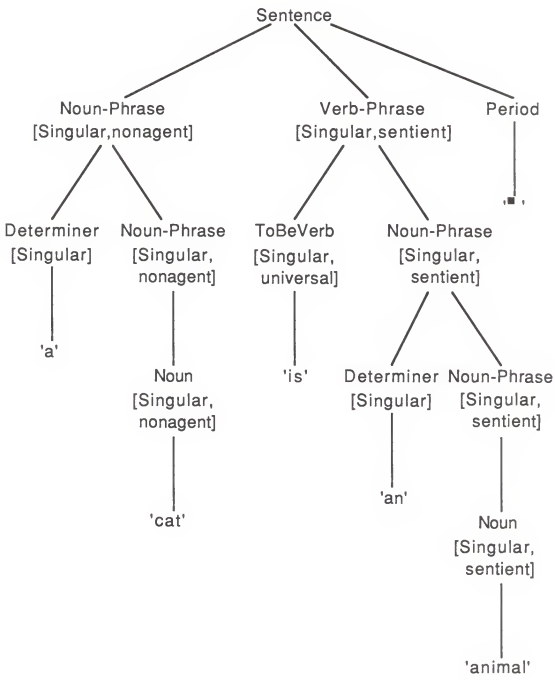

Figure 19: Parse Tree for Sentence 1
(with attached attributes)

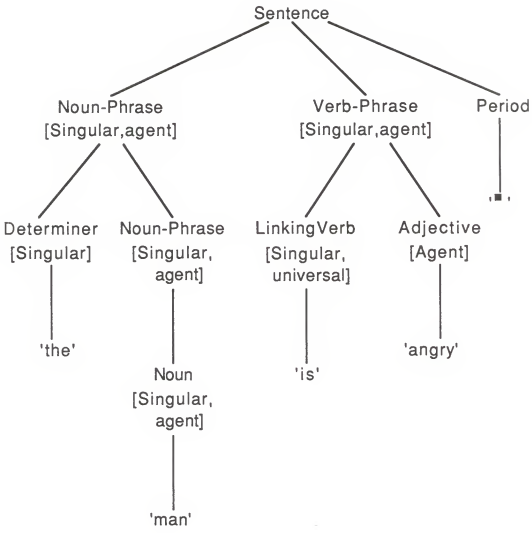

Figure 20: Parse Tree for Sentence 2
(with attached attributes)

3.4.2 The Use of Sommers' Sense-Logic Rules in Attribute Grammars

With this background information, it should now be possible to show how Sommers' sense-logic rules could be used to aid in the solution of problems in semantic-directed parsing. We have seen that the type of an expression is a partial specification of its meaning in that it tells us what other expressions that make sense with that expression. We are thus able to include type constraints as (at least one of) the semantic constraints utilized in an attribute grammar. The type of an expression is associated with it as an attribute and checked for correctness. It is necessary to have a dictionary of terms and their type information which are looked up when parsing a sentence. If one attempt at a parse leads to a type conflict with two of the expressions in the sentence, then a different approach is attempted until a successful parse for the sentence is found, or all approaches have failed.

For a specific example of an attribute grammar with type checking, I have included a C-Prolog program (see Figures 21 and 22) which implements an attribute grammar with type checking for the context-free grammar introduced earlier. The types of the non_terminals are now checked for consistency. For example, in the production for 'sentence', the type of the noun_phrase which is derived from the productions related to 'noun_phrase' and passed up

```
sentence(sentence(N,V,P)) -->
     noun_phrase(N,Num,Type1),
     verb_phrase(V,Num,Type2),
     period(P),
     {rel(Type1,Type2)}.

noun_phrase(np(det(D,Num),N),Num,Type) -->
     determiner(det(D,Num)),
     noun_phrase2(N,Num,Type).
noun_phrase(N,Num,Type) -->
     noun_phrase2(N,Num,Type).

noun_phrase2(np2(adj(A,Type1),N),Num,Type) -->
     adjective(adj(A,Type1)),
     noun_phrase2(N,Num,Type),
     {rel(Type,Type1)}.
noun_phrase2(N,Num,Type) -->
     noun(N,Num,Type).

verb_phrase(vp(V,N),Num,Type1) -->
     tranverb(V,Num,Type1,Type2),
     noun_phrase(N,Num1,Type3),
     {rel(Type3,Type2)}.
verb_phrase(V,Num,Type) -->
     intranverb(V,Num,Type).
verb_phrase(vp(V,adj(A,Type)),Num,Type) -->
     linkverb(V,Num,Type1,Type2),
     adjective(adj(A,Type)),
     {rel(Type,Type2)}.
verb_phrase(vp(V1,V2,N),Num,Type3) -->
     tobeverb(V1,Num,Type1,Type2),
     ingtranverb(V2,Num,Type3,Type4),
     noun_phrase(N,Num1,Type5),
     {rel(Type5,Type4)}.
verb_phrase(vp(V1,V2),Num,Type) -->
     tobeverb(V1,Num,Type1,Type2),
     ingintranverb(V2,Num,Type).
verb_phrase(vp(V,N),Num,Type) -->
     tobeverb(V,Num,Type1,Type2),
     noun_phrase(N,Num,Type).

period(period(.)) --> [46].
```

Figure 21: Attribute grammar - Phrase Structure (with type checking)

```
% Type Hierarchy

% The rel predicate checks the relation between two types.
% The rel predicate succeeds if the 2nd parameter is
%   predicable of the 1st parameter.
rel(T,T).
rel(T,universal).
rel(living,physical).
rel(nonliving,physical).
rel(sentient,physical).
rel(nonsentient,physical).
rel(machine,physical).
rel(nonmachine,physical).
rel(agent,physical).
rel(nonagent,physical).
rel(machine,nonliving).
rel(nonmachine,nonliving).
rel(sentient,living).
rel(nonsentient,living).
rel(agent,living).
rel(nonagent,living).
rel(agent,sentient).
rel(nonagent,sentient).
```

Figure 22: Prolog Implementation of the Type Hierarchy

via the attribute, Type1, is checked for consistency with the type of the verb_phrase (which is passed up via the attribute, Type2). In this case, the consistency check amounts to determining whether something of Type2 is predicable of something of Type1 (i.e., whether things of Type2 are at the same node or 'higher than' things of Type1). If this is not the case, then all other possible approaches are attempted, and if none succeed, then the parse is not successful. What has been added to the previous attribute grammar is a check for nonsensical sentences. Without type checking, a nonsensical, but grammatically correct sentence, would

succeed, and now it will not.

The Prolog Implementation of the Type Hierarchy given in Figure 22 requires a little explanation. The set of Prolog facts in Figure 22 comprise one way of representing part of the information contained in the type hierarchy presented previously in Figure 15. The part that these Prolog facts represent is the relation between the type labels that were attached to the nodes of the tree in Figure 15. The rest of the information (i.e., that a given word belongs at a particular node) is contained in the vocabulary section of the program, where, for example, the word 'cat' is indicated as having type 'nonagent'. The type label, 'nonagent' is shown (in Figure 22) as being 'lower' (in terms of the tree structure) than the type labels: 'universal', 'physical', 'living', and 'sentient'. This means that things of these types are all predicable of things of type 'nonagent'.

It should prove instructive to examine an attempt to parse a sentence containing a type conflict. Sentence (3), 'the desk is angry' will serve as an example of such a sentence, since desks are not the type of thing that can meaningfully be said to be angry. We would begin by parsing the noun phrase 'the desk', and this would give us 'nonmachine' as the type of the noun phrase (Type1) and 'singular' as the number of the noun phrase. Then we would parse the verb phrase 'is angry' and we would get 'agent' as the

type of the verb phrase (Type2) and 'singular' as the number of the verb phrase. We would even be successful in parsing the period, and if we had no type checking, the sentence as a whole would parse successfully. However, when we performed the semantic action included within the brackets (i.e., rel(Type1,Type2)), we would fail, since things of type 'agent' (Type2) are not predicable of things of type 'nonmachine' (Type1). This is indicated by the absence of 'rel(nonmachine,agent)' in the list of facts in the type hierarchy. So we can see how the type checking works, and that it adds a new aspect to our grammar.

Since both attribute grammars and Sowa's conceptual graphs can be used in semantic-directed parsing, and since Sommers' sense-logic rules can aid in the use of both of them, it seems resonable to attempt to combine all three elements into one unified approach to semantic-directed parsing. Attribute grammars would supply the syntax and the notation for including semantic constraints, and Sommers' rules would offer us the ability to include type checking as one of the semantic constraints. After a sentence was parsed, it could be represented in a conceptual graph which would be known to be canonical, since the expressions in the sentence were previously checked for type correctness.

3.5 Summary of the Role of Sense-Logic Rules in Natural Language Processing

Beyond their application to semantic-directed parsing, Sommers' sense-logic rules may prove quite beneficial in the analysis of natural language. Sommers provides cogent arguments to support his claims, and I see no way to rebut them. One of the more significant aspects of his theory is its potential application to natural language processing. Since Sommers gives a formal symbolism, there should be little difficulty in implementing his ideas in some computer application. Sommers' analysis of the order of priority of the rules governing natural language gives an indication of the significance and the place in an approach to natural language processing that his sense rules should have. There are five levels of rules, corresponding to the five levels of meaninglessness listed earlier, and they are ranked according to their priority. The ranking is important, since a rule can neither be satisfied or fail to be satisfied at a given level unless the rules at all previous levels have been satisfied. The five levels are: (1) Word-Construction (Lexicographic) Rules, (2) Grammatical Rules, (3) Type (or Sense) Rules, (4) Consistency Rules (Logic), and (5) Rules governing pragmatical appropriateness (e.g., empirical truth). Given the fact that computer scientists tend to be more familiar with these five levels as they are applied to artificial

languages (i.e., programming languages), I have provided a comparison between the parts of a compiler with these five levels:

Programming Language	Natural Language
(1) Lexical	(1) Word-Construction Rules
(2) Context-Free Grammar	(2) Grammatical Rules
(3) Type-Checking	(3) Type (or Sense) Rules
(4) Assertion-Checking	(4) Consistency Rules (Logic)
(5) Execution	(5) Empirical truth

The rules at levels (1), (2), and (4) have received a great deal of attention in attempts at natural language processing, and some work has been done even on implementing rules for level (5), but very little attention has been paid to level (3). There are computer applications that can enforce grammatical rules, and there is an abundant supply of logic applications, but the gap between these formal systems and anything approximating the meanings of terms is quite wide. Many computer scientists, as well as those in other fields, believe that it is impossible to formalize the meanings of terms. They contend that meaning is too imprecise and too abstract to submit to formal rules. They are quick to point out that there is a big difference between the mere manipulation of symbols according to grammatical and logical rules and a true understanding of a natural language. I contend that the introduction of Sommers' sense-rules is the first step in bridging the gap and making up the

difference. With these rules it should be possible to construct computer applications that can determine the sense of a term if not its meaning (the meaning of a term is more specific than its sense, since, for example, "short" and "tall" have the same sense but different meanings). I am currently unaware of any computer applications other than my own that are explicitly utilizing Sommers' rules, but further research in that area needs to be undertaken.

4. Issues Involved in Generating a Type Hierarchy

4.1 Some Theoretical Issues Involving Sommers' Sense-Logic

For n terms, there are n squared unique pairs of terms. Let {t1, ..., tm} represent the set of terms. There are n pairs of terms whose values (i.e., U, N, P) can be determined without any information about the relations between the terms. These n pairs are those in which the 1st term and the 2nd term in the pair are the same (i.e., all pairs (ti, ti)). There are n pairs since there are n terms. The value for all these pairs is P(ti, ti), which is to say that all terms are predicable of themselves. Thus there are n squared minus n pairs of terms whose values are initially unknown. For any two terms, the possible combinations of values are given below:

(1) P(ti, tj) & P(tj, ti) =df= the two terms are co-predicable, which means that they occupy the same node in the tree

(2) P(ti, tj) & ~ P(tj, ti) =df= ti is predicable of tj, but tj is not predicable of ti, which means that ti occupies a higher node in the tree than tj

(3) ~ P(ti, tj) & P(tj, ti) =df= ti is not predicable of tj, but tj is predicable of ti, which means that ti occupies a lower node in the tree than tj

(4) N(ti, tj) & N(tj, ti) =df= the two terms are not meaningfully related, which means that they are on different branches of the tree

An important question is whether the given information w.r.t. the sense relations of the n terms is sufficient to completely determine the sense relations for all of the n squared minus n pairs of terms. We need a general procedure that will determine, for any set of n terms and any given information w.r.t. the sense relations of the n terms, whether the given information is sufficient to completely determine the sense relations for all n squared minus n terms. I believe the only way to determine this for all possible cases is through the use of an algorithm that derives all the information possible from the given information and then checks this derived information for completeness. I have constructed such an algorithm and implemented it in the version of C - Prolog available on the Vax at Kansas State University. The algorithm is given below:

```
Assert all the given information
(i.e., add it to the dynamic database)
Repeat
  more <-- false
  {derive all the information possible from the current
   amount of information. If 'new' (previously unknown)
   information is derived, then set more to true and
   assert any new information}
Until not (more)
Check the final set of information for completeness
```

As can be seen, the algorithm is not very complex at the general level. One important part of the algorithm is that after every single derivation of information, the information derived is checked to see whether is it new. For the implementation in Prolog, the commented part of the algorithm consists of two sets of clauses which derive information first from isolated pairs of terms, and then from combinations of three terms. I do not wish to give a detailed account of the implementation at this point, but rather just to give the reader an idea of how one of the major theoretical issues was dealt with. More will be said about the implementation later.

4.2 Ten Calculations Involving the Sense Relations of Expressions

The rules Sommers discovered supply the means for performing various calculations involving the sense relations of expressions. The following is a list of some of the calculations that can be performed using Sommers' rules and criteria:

(1) Construct a tree representing the sense "locations" of terms (i.e., the sense relations they have to one another).

(2) Determine whether a term is used univocally.

(3) Determine whether the sense relations (e.g., U and N) given for a set of terms are consistent (Can they be placed on a tree?).

(4) Evaluate "doubtful" sentences (i.e., determine whether a sentence whose sense-value was previously unknown is U or N in sense-value).

(5) Name new concepts. "We have empty locations on the tree for which we have no words... The construction of a tree opens up the possibility of numbering locations and giving names to these concepts."

(6) Determine whether terms are of the same category (the tree will show this).

(7) Determine whether terms are of the same type (the tree will show this).

(8) Determine whether a term is a member of the set of terms constituting a single language (the tree will show this).

(9) Determine whether a set of terms "belong to" the same language (the tree will show this).

(10) Determine whether a term is predicable of another (the tree will show this).

4.3 An Implementation of Sommers' Sense Logic Rules

The implementation I developed is wriProlog, since it seems to be the programming language best suited for the expression of Sommers' sense logic rules. Essentially, I decided to write a program to construct a tree representing the sense locations of terms (see #1 above in the list of ten calculations involving the sense relations of expressions). In order to do this, the program must be given (1) a list of terms and (2) a set of known relations involving those terms. The program is set up to read these from separate files (the files were kept separate to make it easier to assert the known relations and thereby add them to the program's database). After the known relations are read in, the program derives all of the information it can from those relations using Sommers' sense logic rules. For example, if one term is predicable of another term, the program will assert (i.e., add to the database) that the two terms are U-related and that they are not N-related. After all such implicit information is added to the database, the program attempts to construct a tree illustrating the sense relations of the terms. If not enough initial information is given, the program will be unable to construct a tree for the terms and will indicate this. The particulars of running the program are explained in the user's manual. The program I have implemented is merely a prototype for an eventual system that would perform all of

the ten calculations listed above.

5. Summary

I have attempted to give the briefest account of Sommers' type theory sufficient to convey its importance. I provided an analogy with ordinary truth-valued logic to explain its status, and I summarized the central terms and their definitions. Further, the relevant criteria and rules were detailed as well as a list of some of the calculations that could be performed with them. Finally, I outlined the relevance of Sommers' work to computer science, especially to work done in natural language processing and briefly described an implementation of some of Sommers' ideas.

REFERENCES

[CHEN, 1987] Chen, Weilin T. "Use of Definite Clause Grammars", Master's Report, Kansas State University, Manhattan, KS, 1987

[GREENBERG, 1972] Greenberg, Robert S. "Individuals and the Theory of Predication", J Phil, Vol. 69, pages 345 - 348, August 17 1972

[LEWIS, 1976] Lewis II, Philip M., Rosenkrantz, Daniel J., Stearns, Richard E. Compiler Design Theory, Addison-Wesley, 1976

[SOMMERS, 1959] Sommers, Fred. "The Ordinary Language Tree", Mind, Vol. 68, pages 160 - 185, April 1959

[SOMMERS, 1963] Sommers, Fred. "Types and Ontology", Phil Rev, Vol. 72, pages 327 - 363, July 1963

[SOMMERS, 1965] Sommers, Fred. "Predicability", Phil in Am, ed. Max Black, Cornell 1965

[SOWA, 1984] Sowa, John F. Conceptual Structures: Information Processing in Mind and Machine, Addison-Wesley, 1984

[STERLING, 1986] Sterling, Leon, Shapiro, Ehud., The Art of Prolog: Advanced Programming Techniques, MIT Press, 1986

SELECTED BIBLIOGRAPHY

WORKS BY SOMMERS:

(1) "The Passing of Privileged Uniqueness", J Phil, Vol. 49, pages 392 - 396, May 1952

(2) "The Ordinary Language Tree", Mind, Vol. 68, pages 160 - 185, April 1959

(3) "Types and Ontology", Phil Rev, Vol. 72, pages 327 - 363, July 1963

(4) "Meaning Relations and the Analytic", J Phil, Vol. 60, pages 524 - 533, August 1963

(5) "Truth-functional Counterfactuals", Analysis, Vol. 24, pages 120 - 126, January 1964

(6) "A Program For Coherence", Phil Rev, Vol. 73, pages 522 - 527, October 1964

(7) "A Reply to Mr. Odegard's 'On Closing The Truth-value Gap'", Analysis, Vol. 25, pages 66 - 68, January 1965

(8) "Predicability", Phil in Am, ed. Max Black, Cornell 1965

(9) "Why Is There Something And Not Nothing?", Analysis, Vol. 26, pages 177 - 181, June 1966

(10) "What We Can Say About God", Judaism, Vol. 15, pages 1966

(11) "On a Fregean Dogma", Problems in the Phil. of Math, Amsterdam 1967

(12) "Do We Need Identity?", J Phil, Vol. 66, pages 499 - 504, August 7 1969

(13) "On Concepts of Truth in Natural Languages", Rev Metaph, Vol. 23, pages 259 - 286, December 1969

(14) "The Calculus of Terms", Mind, Vol. 79, pages 1 - 39, January 1970

(15) "Confirmation and the Natural Subject", Phil Forum (Boston), Vol. 2, pages 245 - 250, Winter 1970 - 1971

(16) "Structural Ontology", Philosophia (Israel), Vol. 1, pages 21 - 42, January 1971

(17) "Distribution Matters", Mind, Vol. 84, pages 27 - 46, January 1975

(18) "Are There Atomic Propositions?", Midwest Stud Phil, Vol. 6, pages 59 - 68, 1981

(19) The Logic of Natural Language, Oxford Clarendon Pr, 1982

(20) "Linguistic Grammar and Logical Grammar", in How Many Questions, Cauman, Leigh S (ed), pages 180 - 194, Indianapolis Hackett 1983

RELATED WORKS:

(1) Nelson, John O. "On Sommers' Reinstatement Of Russell's Ontological Program", Phil Rev, Vol. 73, pages 517 - 521, October 1964

(2) Odegard, Douglas. "On Closing the Truth-value Gap", Analysis, Vol. 25, pages 10 - 12, October 1964

(3) Khatchadourian, Haig. "Vagueness, Meaning, and Absurdity", Amer Phil Quart, Vol. 2, pages 119 - 129, April 1965

(4) Nelson, John O. "An Examination of Sommer's Truth-Functional Counter-Factuals", Theoria, Vol. 31, pages 61 - 63, 1965

(5) De Sousa, Ronald Bon. "The Tree of English Bears Bitter Fruit", J Phil, Vol. 63, pages 37 - 46, January 1966

(6) Odegard, Douglas. "Absurdity and Types", Mind, Vol. 75, pages 97 - 113, January 1966

(7) Reinhardt, L R. "Dualism and Categories", Proc Aris Soc, Vol. 66, pages 71 - 92, 1966

(8) Massie, David. "Sommers' Tree Theory, A Reply to De Sousa", J Phil, Vol. 64, pages 185 - 193, March 30 1967

(9) Swanson, J W. "Denial in First Order Logic", Analysis, Vol. 27, pages 171 - 173, April 1967

(10) Guerry, Herbert. "Sommers' Ontological Proof", Analysis, Vol. 28, pages 60 - 61, December 1967

(11) Haack, Susan. "Equivocality, A Discussion of Sommers' Views", Analysis, Vol. 28, pages 159 - 165, April 1968

(12) Chandler, Hugh S. "Persons and Predicability", Austl J Phil, Vol. 46, pages 112 - 116, August 1968

(13) Van Straaten, R. "Sommers' Rule and Equivocality", Analysis, Vol. 29, pages 58 - 61, December 1968

(14) Martin, Robert L. "Drange on Type Crossings", Phil Phenomenol Res, Vol. 30, pages 126 - 135, S 1969

(15) Martin, Robert L. "Sommers on Denial and Negation", Nous, Vol. 3, pages 219 - 226, May 1969

(16) Passell, Dan. "On Sommers' Logic of Sense and Nonsense", Mind, Vol. 78, pages 132 - 133, January 1969

(17) Elgood, A G. "Sommers' Rules of Sense", Phil Quart, Vol. 20, pages 166 - 169, April 1970

(18) Chandra, Suresh. "Strawson On M-Predicates", Visva Bharati J Phil, Vol. 7, pages 67 - 76, August 1970

(19) Erwin, Edward. "The Concept of Meaninglessness", Baltimore Johns Hopkins Pr, 1970

(20) Englebretsen, George. "Elgood on Sommers' Rules of Sense", Phil Quart, Vol. 21, pages 71 - 73, January 1971

(21) Van Straaten, R. "Sommers on Strawson's and Descartes' Ontology", Mind, Vol. 80, pages 148 - 149, January 1971

(22) Richmond, Samuel A. "Sommers on Predicability", J Phil, Vol. 68, pages 138 - 142, March 11 1971

(23) Englebretsen, George. "The Incompatibality of God's Existence and Omnipotence", Sophia, Vol. 10, pages 28 - 31, April 1971

(24) Altham, J E. "Ambiguity and Predication", Mind, Vol. 80, pages 253 - 257, April 1971

(25) Cornman, James W. "Materialism and Sensations", New Haven Yale Univ Pr, 1971

(26) Englebretsen, George. "Sommers' Theory and the Paradox of Confirmation", Phil Sci, Vol. 38, pages 438 - 441, September 1971

(27) Englebretsen, George. "On the Nature of Sommers' Rule", Mind, Vol. 80, pages 608 - 611, October 1971

(28) Brody, B A. "Sommers on Predicability", Phil Stud, Vol. 23, pages 138 - 140, February 1972

(29) Englebretsen, George. "Vacuosity", Mind, Vol. 81, pages 273 - 275, April 1972

(31) Englebretsen, George. "On Van Straaten's Modification of Sommers' Rule", Phil Stud, Vol. 23, pages 216 - 219, April 1972

(32) Englebretsen, George. "Sommers on Empty Domains and Existence", Notre Dame J Form Log, Vol. 13, pages 350 - 358, July 1972

(33) Englebretsen, George. "True Sentences and True Propositions", Mind, Vol. 81, pages 451 - 452, July 1972

(34) Sayward, Charles and Voss, Stephen H. "Absurdity and Spanning", Philosophia (Israel), Vol. 2, pages 227 - 238, July 1972

(35) Suzman, J. "The Ordinary Language Lattice", Mind, Vol. 81, pages 434 - 436, July 1972

(36) Swiggart, P. "The Limits of Statement Denial", Mind, Vol. 81, pages 437 - 442, July 1972

(37) Greenberg, Robert S. "Individuals and the Theory of Predication", J Phil, Vol. 69, pages 345 - 348, August 17 1972

(38) Kasher, Asa. "On Sommers' Concept of Natural Syntax", Phil Stud (Ireland), Vol. 20, pages 139 - 143, 1972

(39) Englebretsen, George. "Persons and Predicates", Phil Stud, Vol. 23, pages 393 - 399, December 1972

(40) Englebretsen, George. "A Revised Category Mistake Argument", Phil Stud, Vol. 23, pages 421 - 423, December 1972

(41) Englebretsen, George. "Armstrong on Disembodied Minds", Dialogue (Canada), Vol. 11, pages 576 - 579, December 1972

(42) Englebretsen, George. "Meinong on Existence", Man World, Vol. 6, pages 80 - 82, February 1973

(43) Watson, David. "On 'Types and Ontology'", Second Order, Vol. 2, pages 73 - 81, July 1973

(44) Englebretsen, George. "Erwin on the Category Mistake Argument", Second Order, Vol. 3, pages 47 - 53, January 1974

(45) Englebretsen, George. "More on Disembodied Minds", Phil Papers, Vol. 3, pages 48 - 50, May 1974

(46) Fjeld, Jon. "Sommers' Ontological Programme", Phil Stud, Vol. 25, pages 411 - 416, August 1974

(47) Englebretsen, George. "Brody on Sommers", Phil Stud, Vol. 26, pages 149 - 150, October 1974

(48) Englebretsen, George. "Sommers on the Predicate Exists", Phil Stud, Vol. 26, pages 419 - 423, December 1974

(49) Englebretsen, George. "Sommers' Proof That Something Exists", Notre Dame J Form Log, Vol. 16, pages 298 - 300, April 1975

(50) Englebretsen, George. "Speaking of Persons", Halifax, Dalhousie Univ Pr, 1975

(51) Englebretsen, George. "Sommers' Theory and Natural Theology", Int J Phil Relig, Vol. 6, pages 111 - 116, Summer 1975

(52) Richmond, Samuel. "A Possible Empirical Violation of Sommers' Rule For Enforcing Ambiguity", Phil Stud, Vol. 28, pages 363 - 366, November 1975

(53) Cogan, Robert. "A Criticism of Sommers' Language Tree", Notre Dame J Form Log, Vol. 17, pages 308 - 310, April 1976

(54) Sayward, Charles. "A Defense of Sommers' "Types and Ontology"", Phil Stud, Vol. 29, pages 343 - 347, May 1976

(55) Rosenthal, David M. "Possibility, Existence, and an Ontological Argument", Phil Stud, Vol. 30, pages 185 - 192, September 1976

(56) Englebretsen, George. "Sommers' Tree Theory and Possible Things", Phil Stud (Ireland), Vol. 24, pages 131 - 139, 1976

(57) Shearson, W A. "Speaking of Philosophy: A Reply to Paul Churchland", Dialogue (Canada), Vol. 16, pages 502 - 506, September 1977

(58) Griffin, Nicholas. "Do We Need Predication?", Dialogue (Canada), Vol. 16, pages 653 - 663, December 1977

(59) Osherson, Daniel N. "Three Conditions on Conceptual Naturalness", Cognition, Vol. 6, pages 263 - 289, December 1978

(60) Bryant, John. "Negation, Denial and Possibility", Critica, Vol. 11, pages 111 - 122, April 1979

(61) Englebretsen, George. "Noncategorical Syllogisms in the Analytics", Notre Dame J Form Log, Vol. 21, pages 602 - 608, July 1980

(62) Englebretsen, George. "Bryant on Sommers (Theory of Ontology and Logic)", Critica, Vol. 12, pages 81 - 86, December 1980

(63) Englebretsen, George. "Denotation and Reference", Phil Stud (Ireland), Vol. 27, pages 229 - 236, 1980

(64) Sayward, Charles. "The Tree Theory and Isomorphism", Analysis, Vol. 41, pages 6 - 11, January 1981

(65) Englebretsen, George. Logical Negation, Atlantic Highlands Humanities Pr, 1981

(66) Englebretsen, George. Three Logicians, Atlantic Highlands Humanities Pr, 1981

(67) Englebretsen, George. "A Further Note on a Proof By Sommers", Log Anal, Vol. 24, pages 271 - 272, June 1981

(68) Barr, A., Feigenbaum, E. A.. The Handbook of Artificial Intelligence, Kaufman, 1981

(69) Englebretsen, George. "Do We Need Relative Identity?", Notre Dame J Form Log, Vol. 23, pages 91 - 93, January 1982

(70) Englebretsen, George. "Reference, Anaphora and Singular Quantity", Dialogos, Vol. 18, pages 67 - 72, April 1983

(71) Rich, Elaine. Artificial Intelligence, McGraw-Hill, 1983

(72) Sowa, John. Conceptual Structures: Information Processing in Mind and Machine, Addison-Wesley, 1984

SENSE-LOGIC:
APPLICATION TO NATURAL LANGUAGE PROCESSING

by

CLARK ALAN SEXTON

B. A., Phil., Fort Hays State University, 1983

AN ABSTRACT OF A MASTER'S THESIS

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computing and Information Sciences

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1989

ABSTRACT

In the early 60's, Fred Sommers introduced a logic for "sense relations". This sense-logic has as its domain meaningful/nonsensical sentences. Sense-logic is to be distinguished from conventional logic which has a domain of true/false sentences (propositions). Sommers' work has been applied in category theory, but has not yet received attention in computer science fields. The purpose of this paper is to (1) give a brief exposition of Sommers' sense-logic, (2) show the relationship it has to computer science work done on natural language processing (especially attribute grammars and John Sowa's conceptual graphs), and (3) briefly describe an implementation of some of Sommers' ideas.