Title
A new semantic computation while parsing : presupposition and entailment

Author
Weischedel, Ralph M.

Publication Date
1976

Peer reviewed

A New Semantic Computation While Parsing: Presupposition and Entailment*

Ralph M. Weischedel**

TECHNICAL REPORT #76

*The work was carried out while the author was in the Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA 19174. It was partially supported by NSF Grant SOC 72-0546A01.

**Department of Information and Computer Science, University of California, Irvine, CA 92717.

ABSTRACT

This paper treats the computation of presupposition and entailment while parsing. We also discuss certain phenomena that presupposition and entailment exhibit, and the details of their computation via tree transformations. In particular, we show how to organize an augmented transition network (ATN) and how to write lexical entries in order to compute presupposition and entailment.

Presupposition and entailment are a subclass of inferences that seem to be tied to the structure of language. Distinguishing presupposition and entailment from the general class of inferences enables this subclass to be computed while parsing. This subclass of inferences is further distinguished in that they appear to be relatively independent of context.

0. Introduction

Because natural languages exhibit complex interaction of syntax, semantics, and pragmatics, many computational models of natural languge have included certain semantic computations in a syntactic input component. Winograd (1972) incorporated a search for referents of noun phrases to guide parsing of noun phrases. Davies and Isard (1972) demonstrated the computation of semantic representations of sentences, without overtly computing syntactic structures for sentences. These are just a few among several examples.

This paper discusses the computation of a subclass of inferences while parsing. We demonstrate the details of writing an augmented transition network (ATN) parser and its lexicon to compute the presuppositions and entailments of a sentence. Tree transformations are used in the computation.

Section 1 defines presupposition and entailment. Section 2 demonstrates several phenomena exhibited by presupposition and entailment. Sections 2 and 3 show that the computation is more than a simple composition of the pieces of semantic representation stored in registers of the ATN. Sections 3, 4, and 5 detail the computation of presupposition, the computation of entailment, and their computation in compound sentences, respectively. Section 6 suggests a criterion for what semantic information belongs in a lexicon. Section 7 presents our conclusions. Appendix A indicates the syntactic constructions of the system. Appendix B contains some sample terminal sessions.

1. Definitions

Several terms and assumptions should be made clear before proceeding further. We assume that the goal of a syntactic component processing natural language input is to translate from natural language sentences to sentences in an artificial language of semantic representations. Since the language of semantic representations

is artificial, we assume that all such semantic representations are a subset of a context-free language. (For example, legal Algol programs are a subset of a context-free language.) Therefore, we may assume that the semantic representations have a tree representation which specifies well-formed formulas and well-formed subformulas.

Assuming that the semantic representations have a tree representation does not preclude general graph models. It is very reasonable to assume that the semantic representations of individual sentences have a tree structure. However, the representation for the information state of an individual, and therefore the representation for the context of a sentence, may best be represented as a general graph. Thus, our assumption does not eliminate that possibility.

In general, we will use the term "context" to mean previous text, cultural rules, universal laws, etc. which constitute the given set of facts of the participants of a dialog. All phenomena attributable to the effect of context and all computation to model such phenomena are termed "pragmatic" in this paper.

Given those assumptions, we define three binary relations on sentences in a given natural language. We say that a sentence S' may be <u>subformula-derived</u> from a sentence S if there is a tree transformation f and a well-formed subformula p of the semantic representation L of S such that

$$L' = f(p).$$

where L' is a semantic representation of S'.

A sentence S' is an <u>entailment</u> of a sentence S if and only if in every context in which S is true, S' is also true. Entailment differs from material implication but for linguistic reasons is preferable to material implication (see Givon (1973)).

A sentence S' is a <u>presupposition</u> of a sentence S if and only if both S entails S' and also the negation of S entails S'. Therefore, whether S is true or false, S' must be true if S is to have a truth value at all. If a presupposition

is false, the sentence presupposing it is nonsense, for it can be neither true nor false.

As examples, let us consider sentence (1).

1. John began to win when he changed rackets.

This sentence entails (2) and (3) because of the meaning of "begin".

2. John was winning, immediately after he changed rackets.

3. John was not winning, immediately before he changed rackets.

The negation of (1) is (4), which also entails (3). Hence, (3) is a presupposition of (1).

4. John did not begin to win when he changed rackets.

The meaning of "begin" is the source of (1) presupposing (3). However, syntactic constructions also have presuppositions. An example is the temporal subordinate clause of sentence (1). Sentence (1) presupposes (5).

5. John changed rackets.

In general, there are two sources of presuppositions: the meaning of particular words and the use of particular syntactic constructions. The meaning of particular words is also a source of entailments. Henceforth, we will use <u>entailment</u> to refer to entailments which are not also presuppositions.

In brief, the result of this work is that the presuppositions and entailments of a sentence are subformula-derived. That is, presupposition and entailment may be computed by structural means by a syntactic input component.

The term "inference" has been used recently to refer to any conjecture or conclusion from a set of facts. Presupposition and entailment are inferences that seem to be tied to the structure of language, for they arise from the meaning of particular words (semantic structure) and from the use of particular syntactic constructs (syntactic structure).

Of course, not all inferences are presuppositions or entailments. For instance, from (6), one might feel that (7) should be entailed.

6. John saw Jim in the hall, and Mary saw Jim in his office.

7. John and Mary saw Jim in different places.

However, (7) is not an entailment of (6), because by appropriately chosen previous texts, (7) need not be true whenever (6) is. In particular, the previous text might indicate that Jim's office is in the hall.

This example illustrates a general principle, that common nouns frequently do not exhibit presupposition and entailment phenomena. Nevertheless, these are phenomena that are widespread in language. Fillmore (1971), Givon (1973), Keenan (1971). Kiparsky and Kiparsky (1970), Karttunen (1970), Karttunen and Peters (1975), Lakoff (1971), and Weischedel (1975) are a few papers filled with examples of verbs, complex noun predicates, complex adjective predicates, quantifiers, adverbial elements, and syntactic constructions that have presuppositions and entailments. Furthermore, they are necessary (but certainly not sufficient) semantic knowledge for understanding natural language. For instance, it is questionable whether one understands sentence (8) or the meaning of "able" if one does not infer (9), which is entailed by (8).

8. John was not able to win.

9. John did not win.

Also, it is questionable whether one understands the meaning of "only", if one does not know that (12) may be concluded from both (10) and (11).

10. Only John won.

11. Not only John won.

12. John won.

Since (11) is the negation of (10), (12) is a presupposition of (10).

It is on these grounds then that we advocate this kind of study; the phenomena of presupposition and entailment demonstrate necessary semantic knowledge. The phenomena are widespread throughout language. The fact that the phenomena are tied to the structure of language enables their computation to be particularly simple.

2. Aspects to be accounted for

By the definition of subformula-derived, presupposition and entailment would seem at first glance to be a trivial computation, for one need only specify the proper subformulas and tree transformations to compute them. However, there are several properties or aspects that we observe in natrual language which make the computation nontrivial. The remainder of this section gives concrete examples of several aspects of presupposition and entailment phenomena. Each such aspect should be accounted for in order to compute these inferences. Sections 3, 4 and 5 detail solutions for these.

2.1 Presupposition

There are several phenomena that make the computation of presuppositions complex. These factors include the ambiguity in internal and external readings for "not", the determination of the tense and time of the presupposition, the effect of syntactic environment, and the relation of various words having affixes to their stem words. The solution to all of these problems is explained in section 3.2. The problems are illustrated here.

2.1.1 Internal and external negation

Negation has at least two readings or interpretations. Internal negation is the "normal" reading and is assumed when testing a sentence for presuppositions. Internal negation denies or negates the assertion of the affirmative form of the sentence and presupposes the same sentences as the affirmative form of the sentence.

External negation, on the other hand, denies at least one of the presuppositions of the affirmative form and of the reading with internal negation. In this sense, it is a correction of false presuppositions of a sentence of another participant in the dialog.

For example, (13a) presupposes (13b); so does (14) under the reading of internal negation. However, in the dialog of (15) it is clear that (13b) is not presupposed by (14).

13. a. Mary left when John came.

    b. John came.

14. Mary did not leave when John came.

15. Herman: Did Mary leave when John came?

    Frank: Mary did <u>not</u> leave when John came, because John never came.

Thus, context affects which reading is appropriate. However, there is a simple solution to this difficulty which does not introduce context-dependent computations; one writes the parser to compute (if need be) each of the multiple readings. The context-dependent computation of selecting among the multiple readings would occur in a pragmatic component (which is designed for all context-dependent computations) of a complete natural language understanding system.

2.1.2 Assigning tense and time

In the remaining example sentences, we adopt the convention that a sentence labelled (a) presupposes a (b) sentence.

A second phenomenon is assigning the proper tense and time to a presupposition. In (16), the tense is available directly in the surface sentence. However, in (17), there is no tense overtly marked.

16. a. John resented the fact that he will not play.

    b. He will not play.

17. John resented his not playing.

In other cases, a particular tense (and time) is crucial to the presupposition. Consider (18) as an illustrative case.

18. a. John stopped beating Mary today.

b. Immediately before today, John had been beating Mary.

How to specify the appropriate tense (or that no tense may be assigned as in (17)) is the problem.

2.1.3 Effect of syntactic environment

The syntactic environment of a word can affect whether the word has a presupposition. For instance, (19a) presupposes (19b); however, (20) has no corresponding presupposition.

19. a. John knows that Henry is coming.

b. Henry is coming.

20. John knows whether or not Henry is coming.

This is a problem in marking the lexical entries.

2.1.4 Stems and affixes

One final factor adding complexity to the elementary examples is to relate stem words having presuppositions to the many words derived from these stem words by affixes. For instance, consider "disappoint", "disappointed", and "disappointment". All of (21a) through (21c) presuppose (21d).

21. a. That John is leaving disappoints Sam.

b. Sam felt disappointed that John is leaving.

c. Sam's disappointment that John is leaving is great.

d. John is leaving.

This, too, is a problem for specifying presuppositions in lexical entries.

Let us now turn our attention to corresponding phenomena for entailments.

2.2 Entailment

Several phenomena to be accounted for are also a problem for instances of presuppositions: assigning tense and time to entailments and relating entailments of words with affixes to the stem words. Other phenomena unique to entailments are their dependence on negation, blocking entailments in certain syntactic environments, and promoting entailments to presupposition in other environments. We consider these phenomena unique to the case of entailments first.

For example sentences involving entailment, we adopt the convention that (a) sentences entail (b) sentences.

2.2.1 Dependence on negation

Whether the entailment should be computed and what the entailment should be, depend on the presence or absence of negation for each of five classes of predicates. (See Karttunen (1970) for the taxonomy and examples.) For instance, (22a) entails (22b), but (23) has no entailment.

22. a. John was not able to leave.

    b. John did not leave.

23. John was able to leave.

We must be able to write negation conditions into the lexical entry for each predicate.

2.2.2 Blocking entailments

Another phenomenon is that all entailments are blocked in yes-no questions. For instance, (24) entails nothing about whether Mary left.

24. Did John prevent Mary from leaving?

This is obviously due to the dependence of entailments on negation. A similar effect occurs if we embed (24) in a "whether" environment such as (25), which is a paraphrase of (24).

25. I ask you whether John prevented Mary from leaving.

Entailments are also blocked in commands.

2.2.3 Promoting entailments to presuppositions

We have just seen how some syntactic environments block entailments. On the other hand, a "wh-some" question ("which", "what", "who", "when", "how", for example) has the effect of promoting an entailment to the status of a presupposition. Both (26a) and its paraphrase as (27) presuppose (26b).

26. a. Who prevented Mary from leaving?

b. Mary did not leave.

27. I ask you who prevented Mary from leaving.

The reason that this occurs is that these "wh-some" questions presuppose that there is someone or something in the shared information having the property described in the question. Any entailment of a presupposition is also a presupposition.

The purpose of this section has been to introduce some specific phenomena which must be accounted for in computing presuppositions and entailments.

2.3 Compound sentences

The problem of computing presuppositions and entailments of compound sentences from the presuppositions and entailments of the constituent sentences forming the compound has been called the projection problem. A solution to the projection problem has evolved over several years. The solution involves only structural means and is summarized in Joshi and Weischedel (1976), as well as a more thorough description appearing in Weischedel (1975). Here we review some of the phenomena of the projection problem, and demonstrate the solutions in section 5.

All predicates except for three special classes exhibit simple behavior. Presuppositions of sentences embedded under these predicates become presuppositions of the compound sentence. Entailments of sentences under these predicates either become entailments of the compound sentence or are blocked depending on whether a chain of entailments exists, one link in the chain per sentential level.

2.3.1 Speech acts and predicates of propositional attitude

Speech acts are the predicates or verbs of saying. Predicates of propositional attitude are verbs such as "think" and "hope". They exhibit similar behavior with regard to the projection problem.

Presuppositions of sentences embedded under these predicates become presuppositions of the compound sentence, but under the speaker's "claims" (for speech acts) or the actor's "beliefs" (for predicates or propositional attitude). For instance, (28a) presupposes (28b), not (28c).

28. a. John said that Mary regretted that she left.

b. John claimed Mary left.

c. Mary left.

An exception to this case are presuppositions that definite noun phrases have referents. When the noun phrase makes transparent reference (that is, all

participants in the communication agree that there is a referent), those presuppositions become presuppositions of the compound sentence without being embedded under the speaker's claims. Opaque reference is the case of the referent existing only in the world of the speaker or actor; presuppositions of definite noun phrases making opaque reference must be embedded under the speaker's claims or the actor's beliefs, as in (28).

Entailments of embedded sentences must also be embedded under the speaker's claims (for speech acts) or the actor's beliefs (for predicates of propositional attitude). However, a further crucial phenomenon is that the speech act or predicate of propositional attitude itself has negation conditions on the embedded entailment becoming an entailment of the compound sentence. For instance, (29a) entails (29b); when part of a compound sentence as in (30a), (30b) is entailed. However, (31) has no such entailment.

29. a. John forced Mary to leave.
    b. Mary left.

30. a. Jack said that John forced Mary to leave.
    b. Jack claimed Mary left.

31. Jack did not say that John forced Mary to leave.

These are the phenomena to account for.

2.3.2 Connectives

Connectives such as "and", "or", and "if...then" offer related phenomena. As in the previous section, presuppositions of embedded sentences become presuppositions of the compound sentence but embedded under a world created by the connective. For instance, presuppositions of "if A then B" (interpreted as material implication) are the presuppositions of A plus all propositions "if A then C" where C is a presupposition of B.

Entailments also exhibit instances similar to the phenomena of the previous section (2.3.1). For instance, the connectives themselves have negation conditions

on having entailments. "A and B" entails A, B and the entailments of each. However, if "A and B" is negated as in "I doubt that A and B", none of the entailments of A or of B become entailments of the whole.

Also, as in section 2.3.1, some connectives require embedding under a predicate to become entailments of the compound sentence. If B entails C, then the entailments of "if A then B", are of the form "if A then C".

This concludes our discussion of specific phenomena to account for. Sections 3, 4, and 5 present solutions to the phenomena cited in 2.1, 2.2 and 2.3, respectively.

3. Generating presuppositions

Our system computes presuppositions and entailments of an input sentence while parsing. We have used the augmented transition network (Woods, 1970 and 1973) or ATN as the language for writing the parser. Of course, the parser refers to a lexicon for detailed information about words. (A description of the system as a whole appears in Joshi and Weischedel (1976).)

The data structure needed for storing presuppositions while parsing is the set. We may see this by the following consideration. In elementary sentences (those not having embedded sentences with presuppositions), the output is to be a set of presuppositions. None of those presuppositions interact with each other in elementary sentences. Thus, for elementary sentences, a set is sufficient to store them while computing.

A solution to the problem of computing the presuppositions of compound sentences was summarized in Joshi and Weischedel (1976). The presuppositions of an embedded sentence become presuppositions of the compound sentence by a simple tree transformati depending on the embedding predicate only. Thus, one can compute the set of presuppositions at any sentential level from the structure of this sentential level and the presuppositions of all sentences at the next lower level.

Therefore, the set is the proper data structure for storing presuppositions. In the system, we have implemented sets as lists.

Presuppositions are added to the set in one of three ways. Those arising from syntactic structures are added to the set as soon as the syntactic structure has been successfully parsed. Those arising from lexical entries are added at the end of parsing the sentence at the current level. When the word is encountered, the lexical entry is added to a set of such lexical entries. At the end of the sentence at this level, each lexical entry in this set is processed by a function NEWBUILD (described in section 3.2) and the tense is added from the tense at this level and the table of tenses (function) associated with the lexical entry. This gives a presupposition for each of the lexical entries in the set. They are then added to the set of presuppositions at the current sentential level.

The third way that presuppositions are added to the set is from the presupposition of an embedded sentence. After parsing the sentence at the current level plus all of its embedded sentences, the projection algorithm (see section 5) is applied to the sets of presuppositions of embedded sentences. The presuppositions yielded by the projection algorithm are then added to the presuppositions at the current level.

Let us now consider how to compute the syntactic examples and the lexical examples of presupposition.

3.1 Generating presuppositions from Syntactic Constructs

There are two key facts to understanding how to compute presuppositions that arise from syntactic constructs. One is that since they arise from syntactic constructs, their occurrence is syntactically marked. Therefore, to compute them we need to construct the ATN graph such that there is a path which corresponds exactly to that syntactic construct; that is, the path is taken if and only if

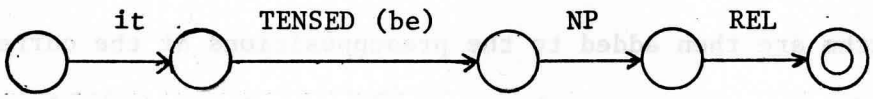
those syntactic clues for that syntactic construct are present. Then, we can associate with the arcs of that path a function to select the proper well-formed sub-formula and apply the proper tree transformation to it.

The second key fact is that when there is ambiguity, when two different syntactic constructs yield the same surface sentence but differ in presuppositions (and the context external to the sentence could determine which is the desired reading), the responsibility of any parser and of our system is <u>not</u> to resolve the ambiguity, but to be able to generate both readings so that the semantic and pragmatic components can choose between them.

With these two facts in mind, let us consider some specific syntactic examples of presuppositions: cleft sentences and definite noun phrases.

3.1.1 Cleft sentences

The cleft sentences, with a noun phrase, not a prepositional phrase extracted, are syntactically marked by the following path, where REL represents relative clause and the double circle indicates final state.



The presupposition of a cleft sentence is that there is some individual (person or thing) having the property mentioned in the relative clause REL. To get the semantic representation of the presupposition, we need to change the binding of the variable representing the noun phrase from that of the noun phrase to that of the constant representing "some individual". The new binding when associated with the semantic representation of the relative clause is the semantic representation of the presupposition.

The value returned by the NP arc gives the relativized variable; the value returned by the relative clause arc gives the semantic representation of the relative clause. Thus, the transformation of extracting the relativized variable of the NP arc, inserting it in the constant structure corresponding to "some individual", and binding this to the semantic representation of the relative clause would occur during the REL arc above. As an action of that arc, the result would be added to the set (list) of presuppositions.

3.1.2 Definite noun phrases

Noun phrases making definite reference have presuppositions. Examples are proper names, possessives, and noun phrases whose determiner is "the".

Let us consider the case of noun phrases whose determiner is "the". A noun phrase such as "the extremely big, red dog" presupposes that "there is an extremely big, red dog". More precisely, that there is an extremely big, red dog in the shared information of the dialog participants. In a function argument notation we might represent this as (1).

1. (binding (*UNTENSED (IN-THE-SHARED-INFO variable)))

In (1), IN-THE-SHARED-INFO is a predicate meaning "in the shared information of the dialog participants"; the "*UNTENSED" indicates that the tense is not known. If "binding" gives the binding of "variable", then we need to fill in "binding" with the semantic representation of "an extremely big, red dog". That semantic representation is just the semantic representation of the noun phrase itself except that the existential quantifier replaces the determiner (quantifier) "the".

Thus, after parsing noun phrases with determiners, the path corresponding to those noun phrases must check to see if the determiner of the head noun is "the". If it is, a presupposition is generated by replacing that determiner by the existential quantifier, and by embedding that semantic representation in the place of "binding"

in (1) and by replacing "variable" in (1) by the variable assigned to the noun phrase as a whole. This presupposition is added to the set (list) of presuppositions and returned among the values of the noun phrase graph.

To summarize, to compute presuppositions arising from syntactic constructs, one should isolate those items that distinguish the construct in the surface syntax. Write the grammar so that a path in the graph is traversed if and only if the construct is present. Then, associate the tree transformation with that path.

3.2 Lexical examples

In this section we outline a solution to generating presuppositions associated with particular words, and in particular a solution to the phenomena described in 2.1. In order to do this, we must give some indication of the organization of the lexicon.

We have used the linguistic string parser (Sager (1967, 1973) and Grisham (1973)) as a pattern for organizing the lexicon, for that parser has a dictionary of 10,000 words and has been very successfully applied. Our parser is implemented using the ATN as a programming language. The linguistic string parser relies upon detailed restrictions placed upon co-occurring constituents in the linguistic strings. A convenient way of implementing such restrictions is via the conditions that one may associate with arcs of an ATN.

The linguistic string parser has each word classified into fine subcategories. One may think of the subcategories as hierarchically arranged and represented by a tre The root of the tree is labelled by the null string. Sons of the root are labelled by N, A, V, for noun, adjective, verb, etc.. These common syntactic categories are further subdivided into classes of words that occur in identical syntactic environments and undergo the same syntactic transformations. A particular subcategory is a path from the root to one of its descendents, and is represented by the concatenation of labels of nodes in the path. Each word may be assigned to more than one subcategory.

This characterization is so precise that one could compute from the subcategory of the word, the syntactic shapes the subject must have, the syntactic shapes the object must have, and syntactic transformations that subject and object may undergo.

Another feature of the subcategories is the association of many words having affixes with their stem word; this is achieved, where possible, by recognizing syntactic transformations from a sentence with the stem word to a paraphrase involving the word with affixes. An example is associating "succeed" as the stem word of all of "success", "successful", and successfulness".

We have argued at some length in Joshi and Weischedel (1976) that the left-hand side encodes necessary aspects of the syntactic environment of the word. By associating the right-hand side of the tree transformation with a lexical subcategory of the word, the pattern inherent in the left-hand side can be matched in the graph because the legal syntactic environments in which a word with that syntactic subcategory may occur are defined by that syntactic subcategory. (An illustration of this for the word "know" appears in Section 3.2.4.)

The root of the tree of the left hand side of the tree transformation is encoded in the lexical subcategories (defined by the linguistic string parser) and the matching occurs by virtue of the graph interpreter following only the paths that give a parse of the sentence.

Tree transformations permute elements, copy elements, add constants, and delete constants. For the resultant tree, we must specify its structure, the specification of constants in it, the variables (registers) holding subtrees, and the "vacant slots" these subtrees should fill.

These requirements lead us to think of the function BUILD of the augmented transition network (Woods (1970, 1973)). However, the restriction that BUILD use only the contents of registers to fill "vacant slots" is too rigid. For instance, (2a) presupposes (2b).

2.a. John stopped running at noon.

b. At a time immediately before noon, John had been running.

Corresponding to "a time immediately before noon" we need to be able to generate a new variable, corresponding to this time, for the logical notation. This cannot be part of the semantic representation of (2a), and is therefore not available as a register in the augmented transition network. Thus, the computation is not merely a composition of the contents of registers. The function we need is similar to BUILD, but the list elements for filling "vacant slots" in the tree to be constructed may be any computable function of the registers.

We may call this new function NEWBUILD. It has one argument, a list (L1, L2, L3...Ln), where L1 is the tree structure and each Li, i>1, is evaluated to fill the slots.

It should be pointed out that our definition of NEWBUILD would allow more complex computations than tree transformations. However, in all cases, the arguments specifying what subtrees to insert are simply register names or else the LISP function giving a symbol for naming a variable in the logical notation. Therefore, the computation is like that of a tree transformation.

The following example clarifies the previous discussion. Associated with the verb "resent" are two presuppositions. It presupposes that its logical subject is human, and that the embedded sentence is true. For instance, (3) presupposes (4) and (5).

3. John resented that he was forced to move.

4. John is human.

5. John was forced to move.

In the lexical entry for "resent" (taking a "that" S complement) we would have a list consisting of two elements. The first could be

6. ((*UNTENSED (HUMAN +)) LSUBJ);

the second could be

7. (+ COMP).

For sentence (3), register LSUBJ, the logical subject, would be "John". Thus, NEWBUILD applied to (6) would yield

(*UNTENSED (HUMAN John))

which could roughly be paraphrased as "John be (unknown tense) human". For sentence (3), COMP, containing the embedded complement sentence, would be

"John was forced to move".

Thus, applying NEWBUILD to (7) would yield (when paraphrased)

"John was forced to move".

This notation assumes that there is a strict convention for naming registers and computing the contents of those registers. The registers are assigned the values regardless of the syntactic form of the sentence; the grammar does the equivalent of inverse syntactic transformations so that the registers are assigned values (semantic representations) as though the sentence were in a normal form.

In the lexicon then, in a lexical entry for a word, we must have a marker, say PRESUPPOSE, and a list. Each member of the list is an argument for NEWBUILD. Thus, there corresponds to each member of the list a presupposition.

Let us now turn to solutions of the problems outlined in section 2.1.

3.2.2 Assigning tense to presuppositions

One problem is the assigning of correct tense to an untensed embedded sentence. The appropriate tense to assign to a (not overtly tensed) presupposition seems to be completely determined by the tense of the surface sentence and the lexical item which has the presupposition. Thus, for each presupposition, one can associate a table or function of the tense of the surface sentence. Each such function has a name; we have chosen the class of names *TENSEi*, where i is a positive integer, as the possible names of such functions.

Let L be the lexical entry to generate a particular presupposition. The presupposition to be generated has a tree representation. Locate the position in the tree where the tense should be placed. In L, place the name of the appropriate tense function *TENSEi* where we want the tense to appear in the complete presupposition. By definition NEWBUILD will leave the atom *TENSEi* at the same position in the tree. If we define a function APPLYTENSE to search a tree for any *TENSEi*, apply that function name to a tense, and insert the result in the tree where *TENSEi* is, then APPLYTENSE (NEWBUILD (L), tense) gives the properly tensed presupposition. ("Tense" is assumed to be the tense at this sentential level.)

3.2.4 Syntactic environment

In 2.1.3 we have seen that the syntactic environment a word occurs in can at times differentiate between two senses of the word, having different presuppositions. At the beginning of 3.2, we have also seen how the lexical subcategories may be used. The syntactic environments a word may occur in and the left-hand side of the tree transformation for a presupposition of that word may be encoded using the syntactic subcategory of the word. By associating the right-hand side of the tree transformation with the word and syntactic subcategory, we have solved the problem of the effect of syntactic environment. Where the syntactic environment differentiates between a sense of the word having a presupposition and one which does not, there are two different syntactic subcategories of the word. One has a presupposition associated; the other does not.

That is, associated with two different syntactic environments (corresponding to two different syntactic subcategories of the word) are two different paths in the ATN graph. After completing the parse at the current sentential level, the path that was taken in parsing determines which of the two syntactic

subcategories of the word was used. Thus, associating presuppositional entries in the lexicon with the word and its syntactic category is a solution to the effect of syntactic environment on computation of presuppositions.

For example, "know" has at least two syntactic subcategories: one corresponding to "that SENTENCE" as complement, and one corresponding to "whether SENTENCE or not" as complement. With the first syntactic subcategory we would associate a lexical entry for presuppositions; with the second, the lexical entry would be empty.

This is analogous to having two senses of "know": $know_1$ and $know_2$. Figure 1a shows an operation that gives the presupposition of $know_1$.

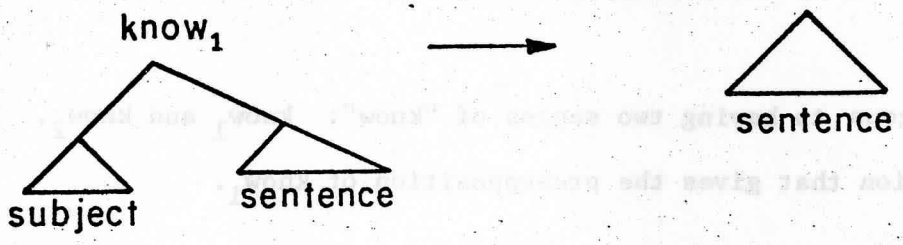
Insert
Figure 1 about
here

We have not used the method exemplified in Figure 1a. Rather, the lexical subcategory of "know" corresponding to "that SENTENCE" corresponds to $know_1$, the root of the left-hand side of the tree transformation. The path in the ATN corresponding to a sentence for this lexical subcategory is given in Figure 1b. In traversing that path, the lexical subcategory of "know" for "that SENTENCE" is matched. Then, the argument for function NEWBUILD can be retrieved from the lexicon to compute the presupposition.

The syntactic subcategories outlined in early work on the linguistic string parser (Anderson, 1970) were appropriate for this kind of analysis.

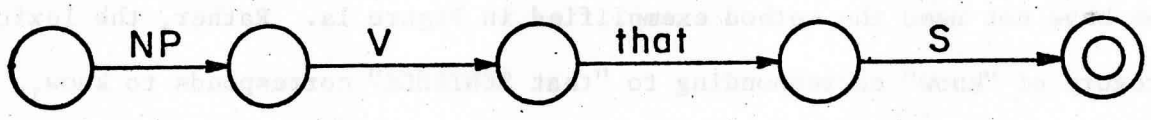
Figure 1

(a)



(This is not a tree transformation.)

(b)



3.2.5 Stem words

The last phenomenon mentioned in 2.1 for computing presuppositions was relating words with affixes to their stem word. For example, "disappoint", "disappointed", and "disappointment" share similar presuppositions. The subcategorization of the linguistic string parser relates their syntactic information. Since we associate the presuppositional information in the lexicon with the word <u>and</u> its subcategory, we have made use of the relationship of the words to their stems inherent in the subcategorization of the linguistic string parser. In many cases, the relation inherent in the subcategorization is so fine that one need only have a pointer to the stem word. For the exceptions, one must include in the lexicon the syntactic information as well as the semantic information.

This concludes the discussion of our solution to computing presuppositions arising from lexical entries.

4. Generating entailments

A data structure permitting the computation of a <u>chain of entailments</u> is needed, because the computation of entailments at a given sentential level in the semantic representation may be affected by embedding predicates and syntactic structures arbitrarily many levels above the current sentential level. This is demonstrated in examples (1) and (2) below. (A (b) sentence is entailed by an (a) sentence.)

1. a. It is false that John prevented Mary from winning.

   b. John did not prevent Mary from winning.

2. a. It is true that John prevented Mary from winning.

   b. John prevented Mary from winning.

   c. Mary did not win.

Here, (2a) entails that Mary did not win. However, from (1a), one cannot conclude anything about whether Mary won.

Entailments depend upon the presence or absence of negation. However, if the embedded sentences of (1a) or (1b) were preposed then the computation of entailments must wait until the entire sentence is traversed. The data structure needed must therefore retain the hierarchical sentential structure and store at each sentential level sufficient information to compute the entailments. This naturally suggests the tree. (Branching occurs in the tree whenever the predicate at that sentential level takes two embedded sentences or whenever a connective occurs at that level.)

At each node of the tree, we need three units of information: whether or not there is negation present in the surface sentence at this sentential level, the surface tense (if any) at this sentential level, and a list of the potential entailed propositions arising from words at this level, as well as the effect of each proposition on negation and tense at lower levels. At the end of the top level sentence, the system applies a function CHAIN of three arguments: the tree described above, an atom indicating whether negation is passed from the higher sentential level, and the tense from a higher sentential level. This function is applied to each node of the tree. It checks the negation requirements of the possible entailments against the negation actually present, and thereby yields the entailments. At any given level, if the negation actually present does not agree with the requirements of any of the possible entailments at this level, the remainder of the subtree is ignored. This corresponds to blocking entailments as in example (1) above.

We shall refer to the tree as a "tree of potential entailments", though this is a misleading term. As indicated above, this is <u>not</u> a binary tree having one level for each sentential level, with two branches for the two possible sets of

entailments depending on negation. Instead of computing a tree of the 2^n possible cases, the tree we compute has labels on the nodes which contain an atom telling whether negation is present or not at this surface sentential level, the tense at this surface sentential level, and the list of entailment information extracted from the lexicon for the words at this sentential level only.

This gives a sketchy outline of the data structure built up, and when it is used to give entailments. However, how the tree is built up has not yet been described. Suppose we are at a given sentential level. One of the entities returned for any further embedded sentences is the tree built up during that embedded sentence. Each of these trees is kept in proper order in a list. (Indeed, the order is very important, for certain entailments involve only one of the two subtrees. This fact must be taken into account in the lexical entries.) For each word at this sentential level possibly having entailments, the lexical information of possible entailments arising from that word is a set (list). As each word which could have entailments is encountered, we store the union of such lexical information found thus far at this level with the set of lexical information of this word. At the end of the current sentential level, each member of this set is replaced by the value of applying function NEWBUILD to that member. The tree corresponding to this level has then as its root the node consisting of an atom telling whether or not negation was present in the surface structure, the surface tense (if any) at this level, and the set after NEWBUILD has been applied to it. The subtrees hanging from the root are the trees corresponding to each embedded sentence.

4.1 The lexical information

We now describe a notation for lexical entries for computing entailments and illustrate it. Fundamentally, we must account for the dependence of the entailment on whether negation is present or absent. As in the case of presuppositions, we associate the entailment information with a word <u>and</u> syntactic subcategory rather

than with the word alone. The entailment information is a set (list) of elements, each corresponding to a possible entailment of the word used according to the subcategory.

For each element, four units of information are necessary. The first unit is whether the entailment requires the presence or absence of negation. We used the atom AFFIRMATIVE to indicate that the entailment arises only if negation is not present. NEGATIVE indicates that the entailment arises if negation is present.

A second essential unit is the logical form of the entailment. This is identical to the form used for presuppositions. Function NEWBUILD substitutes the register values in the logical form at the end of the sentence at this level.

A third element is necessary for correctly computing the chain of entailments; the sign (positive or negative) of the entailed proposition is the third unit. This sign is important to determine whether the negation condition of possible embedded entailments is fulfilled. AFFIRMATIVE signifies that the entailed proposition is positive; NEGATIVE signifies negative; and NIL indicates that the entailed proposition is not an embedded sentence (and therefore, not a link in a chain of further embedded sentences).

The fourth unit is necessary since some predicates have two embedded sentences. "Prove" and conjunctions are examples. The fourth unit specifies whether the left or right hand branch is intended for continuing the chain of entailments.

As an example, consider the entailments of "fail". In the positive, "fail" entails the negation of the embedded sentence, as in (3). In the negative, it entails the embedded sentence, as in (4).

3. a. John failed to leave.

   b. John did not leave.

4. a. John did not fail to leave.

   b. John left.

The list of entailments under "fail" in the lexicon would have two elements. If the embedded sentence were stored in register COMP, the element giving (3b) would be

(AFFIRMATIVE ((NOT (*TENSE5* +)) COMP) NEGATIVE ALL).

The element giving (4b) would be

(NEGATIVE ((*TENSE5* +) COMP) AFFIRMATIVE ALL).

The "ALL" corresponds to the fourth element mentioned above. Since "fail" takes only one embedded sentence, the value of the fourth element is unimportant.

4.2 Solution of difficult factors

In section 2.2, we presented several factors which must be accounted for in the computation of entailments. Some solutions are presented in this section.

The left-hand side of tree transformations is encoded using the syntactic subcategories, just as it was for presuppositions. We associate the right-hand sides of the tree transformation with the word and its syntactic subcategory. Therefore, this solves the difficulties of syntactic environment affecting the computation and of relating words with affixes to their stem words. How to account for the dependence on negation and for the chain of entailments should be clear from the lexical entries and the tree of potential entailments.

The remaining factor mentioned in 2.2 is the promotion of entailments to presuppositions, when those are entailed by a presupposition arising form the meaning of the word. For instance, (5a) presupposes (5b), which entails (5c).

5. a. I know that Mary prevented Joe from leaving.

b. Mary prevented Joe from leaving.

c. Joe did not leave.

One way to solve this is to write under the entailment entry for "know" that its complement sentence (which is a presupposition) is entailed whether in the positive nor negative. This is in addition to the presuppositional entry that the complement is presupposed. We could mark that it is entailed just by marking

the condition on the entailment arising (the first unit of information as referred to in section 4.1) as T, rather than the aformentioned AFFIRMATIVE or NEGATIVE. At the end of the sentence at this level, the program would check the list of entailment entries of this level for such a condition marked T. If it finds one, the subtree of the tree of potential entailments indicated would be immediately computed and added to the set of presuppositions. That subtree could be replaced by the null tree then. This gives a solution to promoting entailments of lexical presuppositions to be presuppositions.

5. Presuppositions and entailments of compound sentences

From the statement of the projection problem in 2.3, it is clear that the computation for compound sentences has been factored into computing these inferences for the constituent embedded sentences and applying a composition rule to those inferences from embedded sentences. The fact that the composition rule is a structural computation is justified in Joshi and Weischedel (1976).

We consider first how the presuppositions and entailments of a constituent sentence are made available to the composition rule, then how the composition rule accounts for the phenomena described in 2.3.

5.1 Computation for embedded sentences

While parsing a compound sentence, the embedded sentences are recognized. Their presuppositions and entailments may be found from the principles in sections 3 and 4 if the embedded sentences are not themselves compound and by recursively applying the principles in section 5 if the embedded sentences are themselves compound.

Using the augmented transition network (ATN) as the means of writing the parser and grammar, we structure the graphs so that a "push" occurs to parse an embedded sentence. This occurs whether the embedded sentence has a form identical to top level sentences as in (1) or it is deformed as in (2).

1. John regretted that Mary left.
2. John forced Mary to leave.

One component of the value returned on popping from an embedded sentence is the set of presuppositions of that sentence; another component of the value is the tree of potential entailments of that embedded sentence. The noun phrase graph is similarly structured and returns a component of presuppositions of the noun phrase and a component of potential entailments.

Just before popping upon completion of a parse, an action of each path having an embedded sentence is to apply the composition rule. We describe the composition rule next.

5.2 Projection rule for presuppositions

From the description in section 2.3, it is clear that the composition rule or projection rule is a case statement, depending on the class of predicate that the embedding verb is. The class of predicate that a verb belongs to would be part of the lexicon.

Further, from section 2.3 the tree transformations for the four cases should be clear, and need not be elaborated. However, one aspect might not be clear: how to deal with the phenomenon of transparent and opaque reference (see section 2.3.1).

We can easily separate those presuppositions that pertain to existence of referents of noun phrases, because they arise from syntactically distinguished strings. Given that one can maintain two subsets of presuppositions, the composition rule must generate two different alternative interpretations. The one corresponding to opaque reference embeds both subsets of presuppositions of embedded sentences in the world of the speaker's claims (for speech acts) or the world of the actor's beliefs (for predicates of propositional attitude). The interpretation corresponding to transparent reference embeds only the subset of presuppositions not pertaining to noun phrases under the speaker's claims or

actor's beliefs; each member of the subset of presuppositions pertaining to noun phrases becomes a presuppositionof the compound sentence without such embedding.

5.3 Projection rule for entailments

The computation of the chain of entailments was covered in section 4. For predicates which are holes, the computation of the chain of entailments is not affected; the tree of potential entailments is computed from the lexical entries at this level plus the subtrees from embedded sentences.

Let us now consider solutions to the special problems mentioned for the other three classes of predicates.

5.3.1 Speech acts and predicates of propositional attitude

In 2.3.1 we saw that entailments of embedded sentences must be embedded under the speaker's claims for speech acts or the actor's beliefs for predicates of propositional attitude. Suppose "speaker" is a register which has the semantic representation of the speaker for speech acts; and "actor" has the semantic representation of the actor, if the predicate is of propositional attitude. Thus, for the subtree under the speech act or predicate or propositional attitude, each potentially entailed proposition is embedded in (3) for speech acts or in (4) for predicates of propositional attitude.

3. (CLAIM speaker proposition)

4. (BELIEVE actor proposition)

A second phenomenon is that the embedded entailments rise to become entailments of the compound sentence depending on whether negation is present or absent. We can encode these negation conditions in the lexicon for each speech act and predicate of propositional attitude with the same four units of information described in section 4.1. In this case the proposition entailed by a speech act or predicate of propositional attitude is the trivial, universally true proposition T.

This accounts for the projection rule for these two classes of predicates.

5.3.2 Projection rule for connectives

By now the projection rule for connectives should be clear, for the phenomena to account for are analogous to those for speech acts. For "if A then B", we embed each entailment in the list or tree structure in (5).

5. (IF a c).

Of course, "a" is a register containing the semantic representation of A, and "c" contains a potentially entailed proposition of B.

There are negation conditions for "and" and "or" just as there were for speech acts and predicates of propositional attitude. The same device described in 5.3.1 for those verbs will work for the connectives as well.

6. A suggestion for organizing the lexicon

We suggest that the test of what should be included in a parser and its associated lexicon or dictionary is whether the phenomenon is independent of nonstructural context or not. If it is independent of nonstructural context, then it may be included. Presupposition and entailment offer an example of semantic information which may appropriately be included in a lexicon.

The definition of entailment requires that in every context in which the original sentence is true, its entailments must also be true. A presupposition, by definition, is entailed by both the original sentence and its negation. Consequently, one might hypothesize that presupposition and entailment could be computed independent of (nonstructural) context. That is, all context not inherent in the structure of the sentence should be unnecessary in order to compute the presuppositions and entailments of the sentence.

We have demonstrated something like that claim in Joshi and Weischedel (1976) and here in sections 3, 4, and 5. We do not claim that the presuppositions or entailments of a sentence do not alter depending on context. That would be as ridiculous as claiming that context does not affect what interpretation or reading

of a sentence is intended. Rather, we claim that the presuppositions and entailments for a given interpretation or reading of a sentence may be computed independent of context.

Our system computes the presuppositions and entailments corresponding to each particular reading as the parser generates the various readings for a sentence. This additional semantic knowledge can serve as further ground upon which general semantic and pragmatic components can ascertain which reading is intended. However, our system does not include general semantic and pragmatic components. (Other limitations of the system appear in Joshi and Weischedel (1976).)

Similar principles have been implicit in the linguistic string parser, which has a lexicon of 9,000 words (Sager (1973), Grishman (1973), Fitzpatrick and Sager (1974)). The linguistic string parser is based on a theory of grammar by Harris (1970). The empirical basis of his model of grammar is not acceptability itself, but rather whether hypothesized syntactic transformations preserve the acceptability or unacceptability of sentences. This theory of grammar has been studied formally in Joshi, Kosaraju, and Yamada (1972), and its relation to transformational grammar has been considered in Joshi (1973).

To summarize, we suggest that the test of what should be included in a parser and its associated lexicon is whether the phenomenon is independent of nonstructural context. Presupposition and entailment are examples of semantic phenomena which, according to the test, may be included in a lexicon. Similar principles have been inplicit in the extensive project of the linguistic string parser.

7. Conclusion

This paper demonstrates how a parser, written as an augmented transition network, and its associated lexicon may compute the presuppositions and entailments of a system. Our system achieves this by computing the presuppositions and entailments

of each reading or interpretation of a sentence as the parser generates the various readings. The computation is structural in nature, using tree transformations. Presupposition and entailment are examples of semantic information that may appropriately be included in a parser.

A discussion of presupposition and entailment as a subclass of inferences appears in Joshi and Weischedel (1976). (See Schank et. al (1975), Wilkes (1975), and Charniak (1973) for a discussion of the general class of inferences.) The computation of presupposition and entailment demonstrates the importance of sorting inferences into subclasses, for presupposition and entailment offer a particularly simple computation (via tree transformations), which is not possible for the general class of inferences. This is an example of the importance of distinguishing various sources of knowledge for natural language processing.

REFERENCES

Anderson, B., "Transformationally Based English Strings and Their Word Subclasses". String Program Reports No. 7. New York: Linguistics String Project, New York University, 1970.

Charniak, Eugene, "Jack and Janet in Search of a Theory of Knowledge". In Proceedings of the Third International Joint Conference on Artificial Intelligence. Palo Alto, CA: Stanford Research Institute, 1973.

Davies, D.J.M and S.D. Isard, "Utterances as Programs", presented at the Seventh International Machine Intelligence Workshop, Edinburgh, June, 1972.

Fillmore, C.J., "Verbs of Judging: an Exercise in Semantic Description". In Fillmore and Langendoen, Studies in Linguistic Semantics. New York: Holt, Rinehart, and Winston, 1971.

Fitzpatrick, Eileen and Sager, Naomi, "The Lexical Subclasses of the Linguistic String Parser". American Journal of Computational Linguistics 1(1974).

Givon, Talmy, "The Time-Axis Phenomenon". Language 49,4 (1973): 890-925.

Grishman, Ralph, "Implementation of the String Parser of English". In Rustin, Natural Language Processing. New York: Algorithmics Press, 1973.

Harris, Z.S., "Two Systems of Grammar: Report and Paraphrase". In Harris, Structural and Transformational Linguistics. Dordrecht, Holland: D. Reidel Publishing Co., 1970.

Joshi, A.K., "A Class of Transformational Grammars". In M. Gross, M. Halle, and M. Schutzenberger, Formal Analysis of Natural Language. Amsterdam: North-Holland, 1973.

Joshi, Aravind K., and Ralph M. Weischedel, "Computation of a Subclass of Inferences: Presupposition and Entailment". Technical Report #74, Information and Computer Science Department, University of California, Irvine, Irvine CA, 1976.

Joshi, A.K., S. Kosaraju, and H.M. Yamada, "String Adjunct Grammars I and II, Information and Control, September, 1972 and October, 1972.

Karttunen, Lauri, "On the Semantics of Complement Sentences". In Papers from the Sixth Regional Meeting of the Chicago Linguistic Society. Chicago: University of Chicago, 1970.

Karttunen, Lauri and Peters, Stanley, "Conventional Implicature in Montague Grammar". Presented at the First Annual Meeting of Berkeley Linguistic Society, February 15, 1975. Berkeley, CA.

Keenan, Edward L., "Two Kinds of Presupposition in Natural Language". In Fillmore and Langendoen, Studies in Linguistic Semantics. New York: Holt, Rinehart, and Winston, 1971.

Keenan, Edward L. "On Semantically Based Grammar". Linguistic Inquiry III (1972): 413-461.

Kiparsky, Paul and Kiparsky, Carol, "Fact". In Steinberg and Jakobovits, Semantics. New York: Cambridge University Press, 1971.

Lakoff, George, "Presupposition and Relative Well-formedness". In Steinberg and Jakobovits, Semantics. New York: Cambridge University Press, 1971.

Sager, Naomi, "Syntactic Analysis of Natural Language". In Alt. F., and Rubinoff, M., Advances in Computers 8. New York: Academic Press, 1967.

Sager, Naomi, "The String Parser for Scientific Literature". In Rustin, Natural Language Processing. New York: Algorithmics Press, 1973.

Schank, Roger C., et. al. "Inference and Paraphrase by Computer". Journal ACM 22,3 (July 1975): 309-328.

Weischedel, Ralph M., Computation of a Unique Class of Inferences: Presupposition and Entailment. Unpublished Doctoral dissertation, University of Pennsylvania, 1975.

Wilks, Yorick, "A Preferential, Pattern-seeking, Semantics for Natural Language Inference". Artificial Intelligence 6 (1975): 53-74.

Winograd, Understanding Natural Language, New York: Academic Press, 1972.

Woods, W.A., "Transition Network Grammars for Natural Language Analysis". Comm.ACM 13,10 (October 1970): 591-606.

Woods, W.A., "An Experimental Parsing System for Transition Network Grammars". In Rustin, Natural Language Processing. New York: Algorithmics Press, 1973.

Appendix A

Since one way to evalute a natural language system is by the extent and complexity of syntactic constructions in the system, this appendix specifies a context-free language which includes all sentences the system is prepared to handle. The context-free language is specified by a recursive transition network (Woods (1970)).

A recursive transition network (RTN) is a generalization of a finite-state machine. An RTN is a graph with a finite number of labelled states; labelled, directed arcs; a distinguished start state; and a subset of final states. Arc labels may be either terminal symbols or state names. An input string of terminal symbols is accepted by an RTN if there is a path from the start state to the final state which consumes the string. A transition from one state to another in such a path may be made in one of two ways:

1) if the arc is labelled by a terminal symbol, the transition indicated by the arc is made iff that terminal is currently pointed to in the input string; 2) if the arc is labelled by a state name, the transition is made iff there is a substring beginning at the current symbol in the input string such that the substring is accepted by a path from the state named on the arc to a final state.

We have used the following nonterminals to represent syntactic categories (which would appear in the lexicon for a word). This is a shorthand for a subgraph of two states; the first state being labelled by that syntactic category, and the second state being a final state. Arcs between the two states would each be labelled by a word in the lexicon having that syntactic subcategory.

V	verb
VN	nominalized verb
VEN	past participle of verb
VING	present participle of verb
PRO	pronoun

DET	determiner
TITLE	title
A	adjective
D	adverb
N	noun
P	preposition

The six graphs are DS for declarative sentences, NP for noun phrases, LS for subject shapes, OBJ for object shapes, POBJ for passive forms of object shapes, and T(x) for tensed verbs x.

All items in lower case are constants. All upper case items are state names, corresponding to nonterminal symbols. The special symbols are as follows:

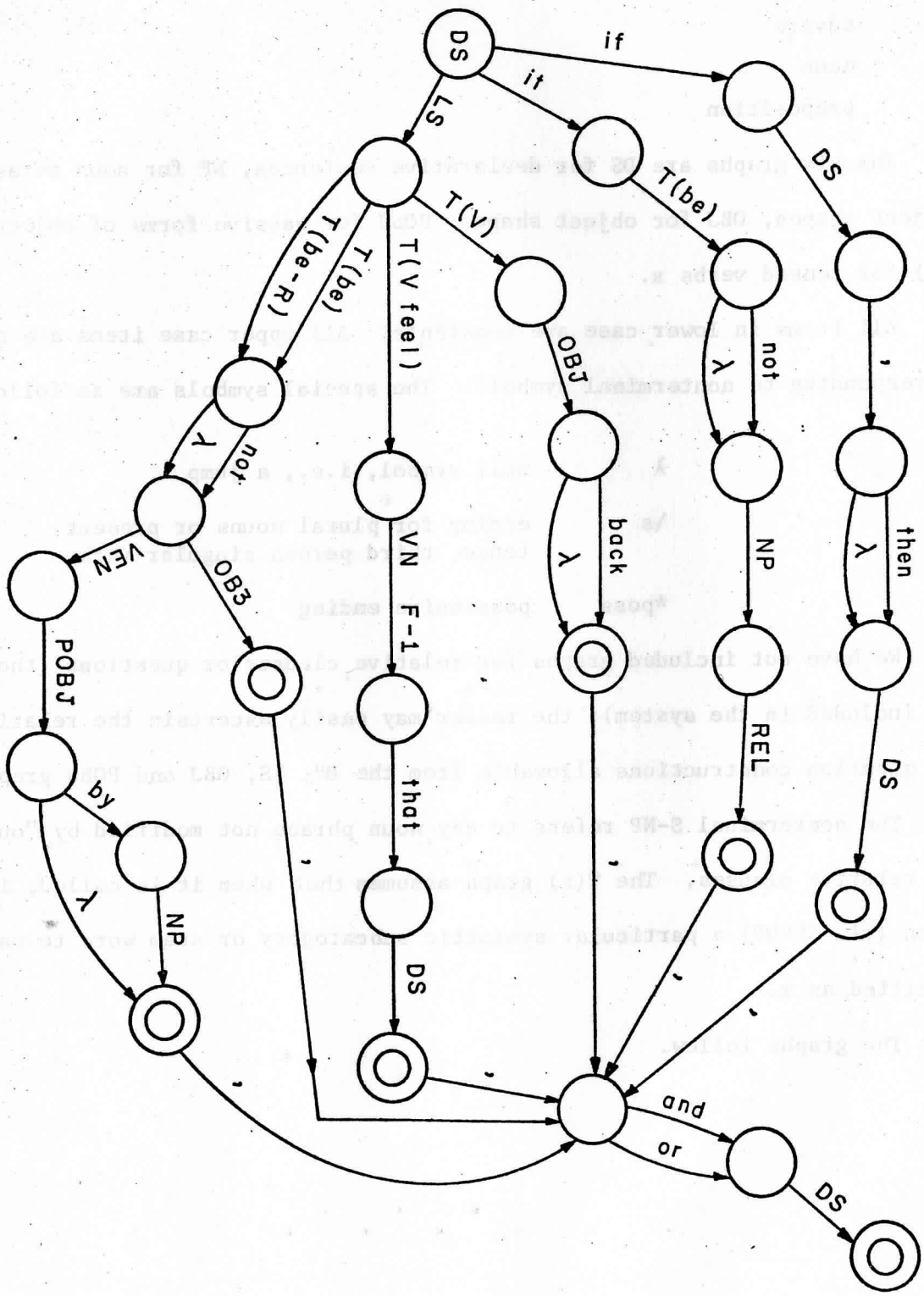
λ	null symbol, i.e., a jump
\s	ending for plural nouns or present tense, third person singular verbs
*poss	possessive ending

We have not included graphs for relative clauses or questions (though these are included in the system); the reader may easily ascertain the relative clause and question constructions allowable from the DS, LS, OBJ and POBJ graphs.

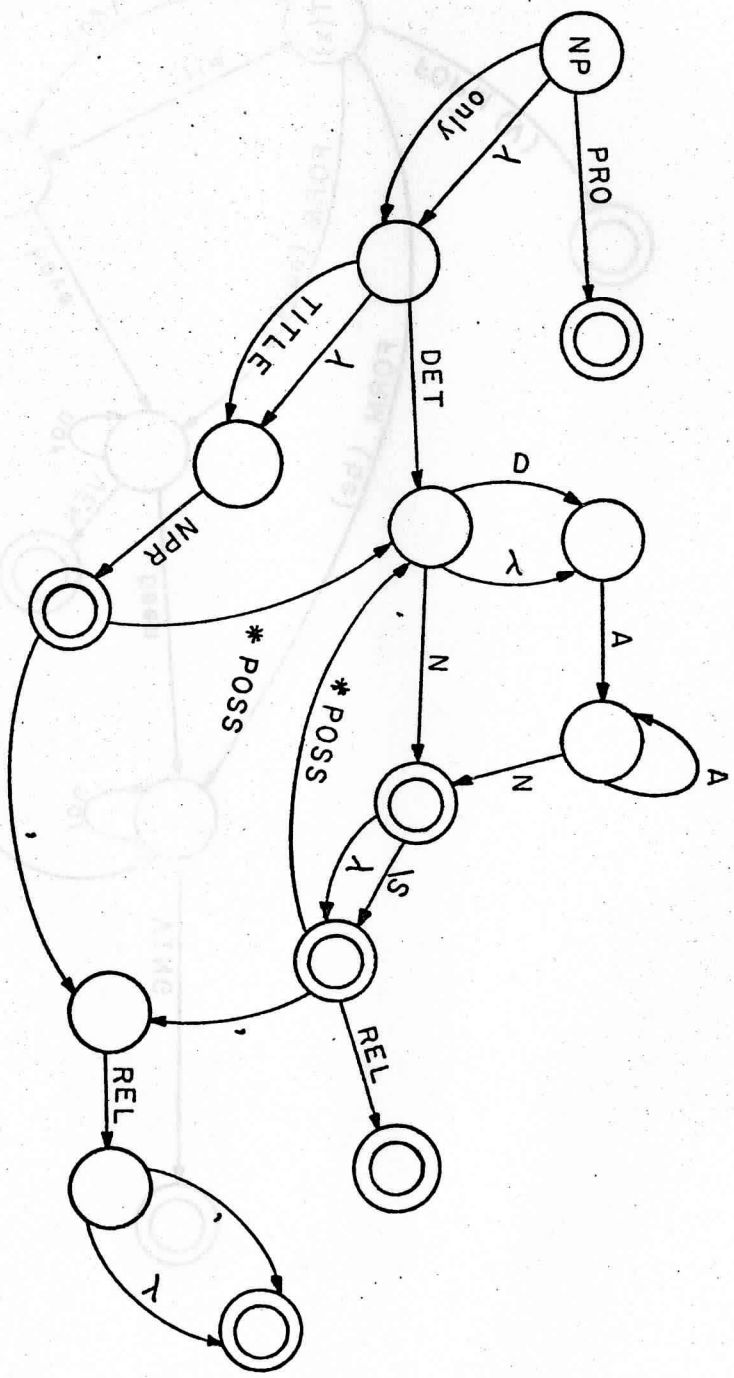
The nonterminal S-NP refers to any noun phrase not modified by "only" or by any relative clauses. The T(x) graph assumes that when it is called, it has been given (via SENDR) a particular syntactic subcategory or stem word to match, specified as x.

The graphs follow.

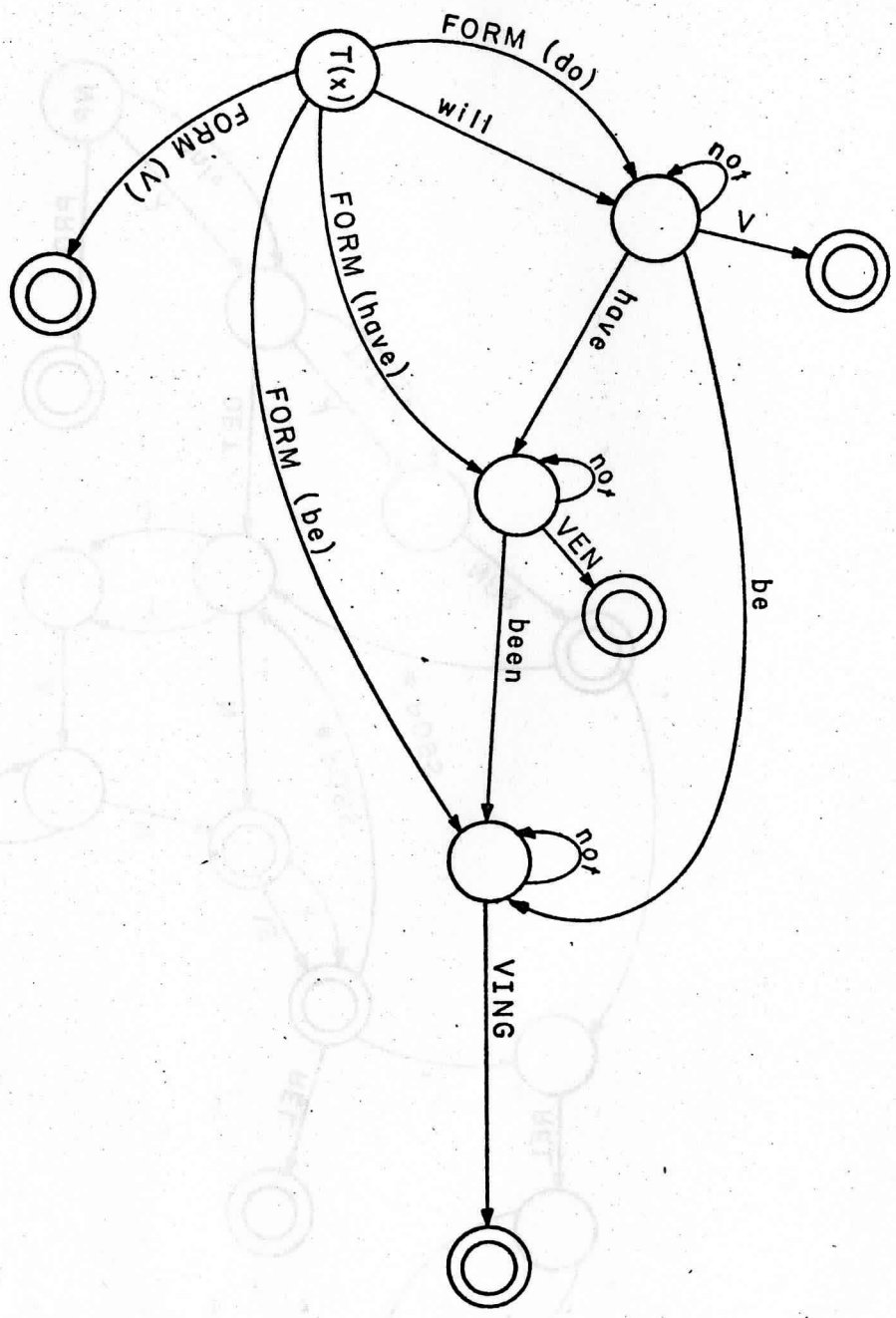
DS (Declarative Sentences)



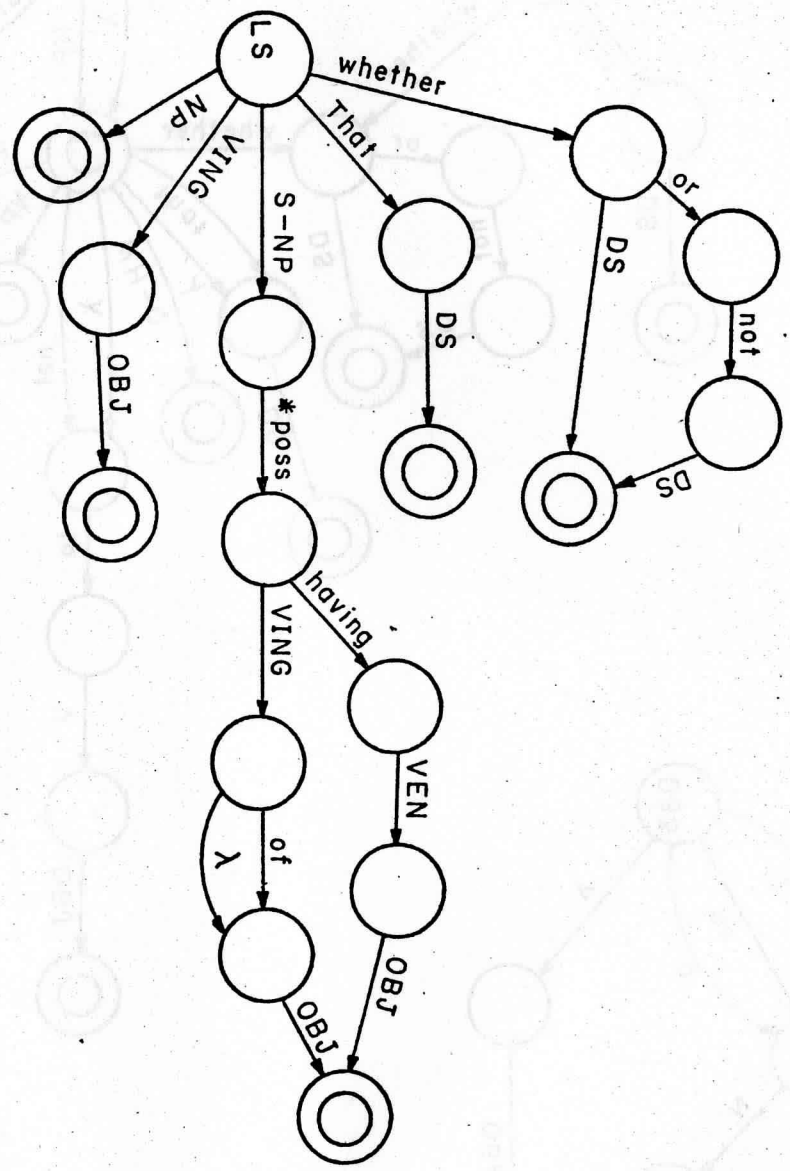
NP (Noun phrases)



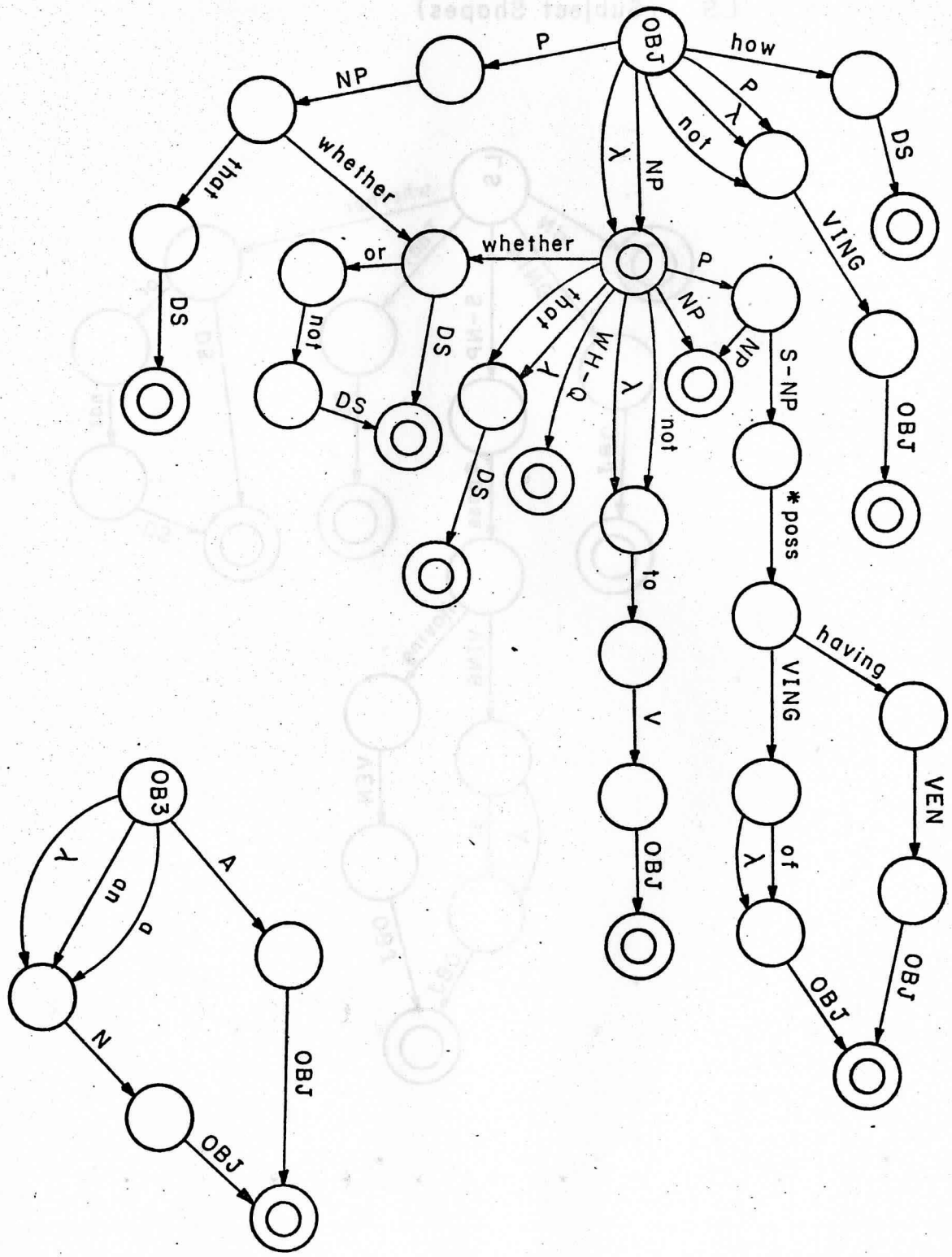
T(x) (Tensed elements)



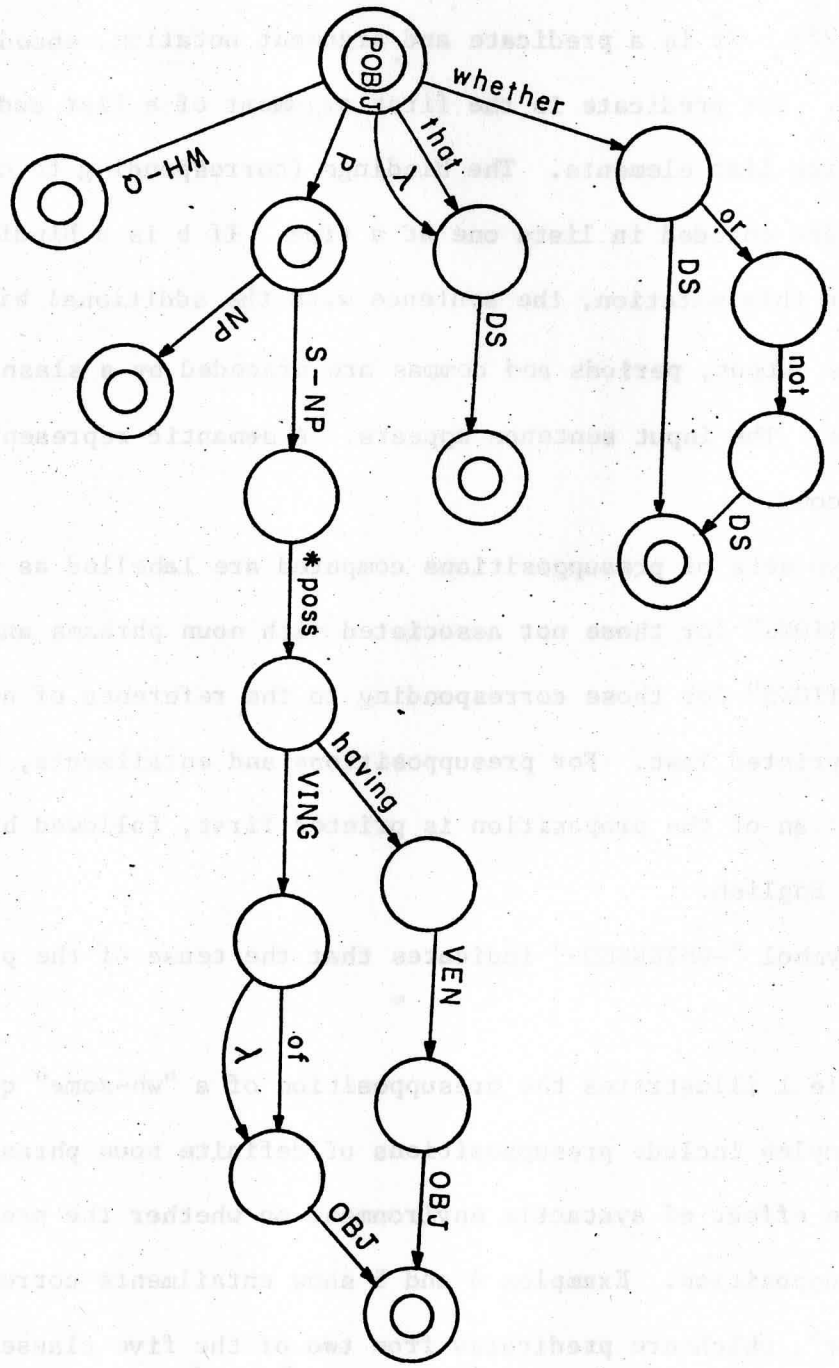
LS (Subject Shapes)



OBJ and OB3 (Object Shapes)



POBJ (Object Shapes for Passive Sentences)



Appendix B: Sample Terminal Sessions

Several example sentences are presented here with their computed presuppositions and entailments. The semantic representations used follow those suggested by Keenan (1972). It is a predicate and argument notation, encoded as list structures. The predicate is the first argument of a list and its arguments are the remaining list elements. The bindings (corresponding to noun phrases) for variables are encoded in lists one at a time. If b is a binding and r is a sentence in this notation, the sentence with the additional binding is (b r).

In the output, periods and commas are preceded by a slash because of LISP conventions. The input sentence appears. A semantic representation of the sentence appears second.

The two sets of presuppositions computed are labelled as follows: "NON-NP PRESUPPOSITIONS" for those not associated with noun phrases and "NP-RELATED PRESUPPOSITIONS" for those corresponding to the referents of noun phrases. Entailments are printed last. For presuppositions and entailments, the semantic representation of the proposition is printed first, followed by a simple translation to surface English.

The symbol "-UNTENSED-" indicates that the tense of the proposition is not known.

Example 1 illustrates the presupposition of a "wh-some" question. Almost all of the examples include presuppositions of definite noun phrases. Examples 2 and 3 reflect the effect of syntactic environment on whether the predicate "be a problem" has a presupposition. Examples 4 and 5 show entailments corresponding to "force" and prevent", which are predicates from two of the five classes of predicates identified by Karttunen (1970). Example 6 is a yes-no question; the entailment of "continue" is therefore blocked, but the presupposition of "continue" remains. Examples 7 and 8 illustrate the projection problem.

WHO TRANSLATED THE ASSIGNMENT ?

SEMANTIC REPRESENTATION

((THE ASSIGNMENT /, X0077) ((E ONE /, X0076) (ASK I YOU (WH-SOME X0076 (IN-THE-PAST (TRANSLATE X0076 X0077))))))

NON-NP PRESUPPOSITIONS

NP-RELATED PRESUPPOSITIONS

((E ASSIGNMENT /, X0077) (*UNTENSED (IN-THE-SHARED-INFO X0077)))

SOME ASSIGNMENT EXIST -UNTENSED- IN THE SHARED INFORMATION .

((THE ASSIGNMENT /, X0077) ((E ONE /, X0076) (IN-THE-PAST (TRANSLATE X0076 X0077))))

SOME ONE TRANSLATED THE ASSIGNMENT .

ENTAILMENTS

THAT THE DISCUSSIONS ARE IRRELEVANT IS A PROBLEM /.

SEMANTIC REPRESENTATION

```
((((COLLECTIVE DISCUSSION /, X0013) (NUMBER X0013 TWO-OR-MORE)) /, X0
014) (ASSERT I (IN-THE-PRESENT (PROBLEM (FACT (IN-THE-PRESENT (NOT (R
ELEVANT X0014)))))))
```

NON-NP PRESUPPOSITIONS

```
((((COLLECTIVE DISCUSSION /, X0013) (NUMBER X0013 TWO-OR-MORE)) /, X0
014) (IN-THE-PRESENT (NOT (RELEVANT X0014))))
```

THE DISCUSSIONS ARE IRRELEVANT .

NP-RELATED PRESUPPOSITIONS

```
((((E DISCUSSION /, X0013) (NUMBER X0013 TWO-OR-MORE)) /, X0014) (*UN
TENSED (IN-THE-SHARED-INFO X0014)))
```

SOME DISCUSSIONS EXIST -UNTENSED- IN THE SHARED INFORMATION

.

ENTAILMENTS

WHETHER THE DISCUSSIONS ARE IRRELEVANT IS A PROBLEM /.

SEMANTIC REPRESENTATION

((((COLLECTIVE DISCUSSION /, X0017) (NUMBER X0017 TWO-OR-MORE)) /, X0
018) (ASSERT I (IN-THE-PRESENT (PROBLEM (WHE (IN-THE-PRESENT (NOT (RE
LEVANT X0018)))))))

NON-NP PRESUPPOSITIONS

NP-RELATED PRESUPPOSITIONS

((((E DISCUSSION /, X0017) (NUMBER X0017 TWO-OR-MORE)) /, X0018) (*UN
TENSED (IN-THE-SHARED-INFO X0018)))

SOME DISCUSSIONS EXIST -UNTENSED- IN THE SHARED INFORMATION
.

ENTAILMENTS

MARY WAS FORCED TO LEAVE /.

SEMANTIC REPRESENTATION

((MARY /, X0052) (ASSERT I (IN-THE-PAST (CAUSE NIL (COME-ABOUT (EVENT (LEAVE X0052)))))))

NON-NP PRESUPPOSITIONS

NP-RELATED PRESUPPOSITIONS

((MARY /, X0052) (*UNTENSED (IN-THE-SHARED-INFO X0052)))

MARY EXIST -UNTENSED- IN THE SHARED INFORMATION .

ENTAILMENTS

((MARY /, X0052) (IN-THE-PAST (LEAVE X0052)))

MARY LEFT .

THE CROWDED LECTURES PREVENTED THE STUDENTS FROM LEARNING /.

SEMANTIC REPRESENTATION

```
((((COLLECTIVE STUDENT /, X0059) (NUMBER X0059 TWO-OR-MORE)) /, X0060
) ((((((COLLECTIVE LECTURE /, X0056) (CROWDED X0056)) /, X0057) (NUMB
ER X0057 TWO-OR-MORE)) /, X0058) (ASSERT I (IN-THE-PAST (CAUSE X0058
(NOT (COME-ABOUT (EVENT (LEARN X0060 NIL)))))))))
```

NON-NP PRESUPPOSITIONS

NP-RELATED PRESUPPOSITIONS

```
((((((E LECTURE /, X0056) (CROWDED X0056)) /, X0057) (NUMBER X0057 TW
O-OR-MORE)) /, X0058) (*UNTENSED (IN-THE-SHARED-INFO X0058)))
```

SOME CROWDED LECTURES EXIST -UNTENSED- IN THE SHARED INFORMATION .

```
((((E STUDENT /, X0059) (NUMBER X0059 TWO-OR-MORE)) /, X0060) (*UNTEN
SED (IN-THE-SHARED-INFO X0060)))
```

SOME STUDENTS EXIST -UNTENSED- IN THE SHARED INFORMATION .

ENTAILMENTS

```
((((COLLECTIVE STUDENT /, X0059) (NUMBER X0059 TWO-OR-MORE)) /, X0060
) (NOT (IN-THE-PAST (LEARN X0060 NIL))))
```

IT IS NOT THE CASE THAT THE STUDENTS LEARNED .

DID MARY CONTINUE STRIKING JOHN ?

SEMANTIC REPRESENTATION

((JOHN /, X0109) ((MARY /, X0108) (ASK I YOU (WHE (IN-THE-PAST (CONTI
NUE (EVENT (STRIKE X0108 X0109)) NIL))))))

NON-NP PRESUPPOSITIONS

((JOHN /, X0109) ((MARY /, X0108) ((((E TIME /, X0110) (IMMEDIATELY-B
EFORE X0110 NIL)) /, X0111) (AT-TIME (IN-THE-PAST (HAVE-EN (BE-ING (S
TRIKE X0108 X0109)))) X0111))))

MARY HAD BEEN STRIKING JOHN .

NP-RELATED PRESUPPOSITIONS

((MARY /, X0108) (*UNTENSED (IN-THE-SHARED-INFO X0108)))

MARY EXIST -UNTENSED- IN THE SHARED INFORMATION .

((JOHN /, X0109) (*UNTENSED (IN-THE-SHARED-INFO X0109)))

JOHN EXIST -UNTENSED- IN THE SHARED INFORMATION .

ENTAILMENTS

THE PROFESSOR DOUBTED THAT JOHN MANAGED TO TRANSLATE AN ASSIGNMENT /.

SEMANTIC REPRESENTATION

((E ASSIGNMENT /, X0042) ((JOHN /, X0041) ((THE PROFESSOR /, X0039) (ASSERT I (IN-THE-PAST (BELIEVE X0039 (NOT (IN-THE-PAST (COME-ABOUT (E VENT (TRANSLATE X0041 X0042)))))))))))

NON-NP PRESUPPOSITIONS

((THE PROFESSOR /, X0039) (*UNTENSED (HUMAN X0039)))

THE PROFESSOR BE -UNTENSED- HUMAN .

((E ASSIGNMENT /, X0042) ((JOHN /, X0041) ((THE PROFESSOR /, X0039) (IN-THE-PAST (BELIEVE X0039 (IN-THE-PAST (ATTEMPT (EVENT (TRANSLATE X0 041 X0042)))))))))

THE PROFESSOR BELIEVED THAT JOHN ATTEMPTED TO TRANSLATE SOME ASSIGNMENT .

NP-RELATED PRESUPPOSITIONS

((E PROFESSOR /, X0039) (*UNTENSED (IN-THE-SHARED-INFO X0039)))

SOME PROFESSOR EXIST -UNTENSED- IN THE SHARED INFORMATION .

((JOHN /, X0041) (*UNTENSED (IN-THE-SHARED-INFO X0041)))

JOHN EXIST -UNTENSED- IN THE SHARED INFORMATION .

ENTAILMENTS

((E ASSIGNMENT /, X0042) ((JOHN /, X0041) ((THE PROFESSOR /, X0039) (IN-THE-PAST (BELIEVE X0039 (NOT (IN-THE-PAST (TRANSLATE X0041 X0042))))))))

THE PROFESSOR BELIEVED THAT IT IS NOT THE CASE THAT JOHN TRANSLATED SOME ASSIGNMENT .

JOHN WON /, AND MARY FELT DISAPPOINTMENT THAT HE WON /.

SEMANTIC REPRESENTATION

((MARY /, X0050) ((JOHN /, X0048) (ASSERT I (AND (IN-THE-PAST (WIN X0048)) (IN-THE-PAST (DISAPPOINT (FACT (IN-THE-PAST (WIN X0048))) X0050))))))

NON-NP PRESUPPOSITIONS

((JOHN /, X0048) (IF-THEN (IN-THE-PAST (WIN X0048)) (IN-THE-PAST (WIN X0048))))

IF JOHN WON THEN JOHN WON .

((MARY /, X0050) ((JOHN /, X0048) (IF-THEN (IN-THE-PAST (WIN X0048)) (*UNTENSED (HUMAN X0050)))))

IF JOHN WON THEN MARY BE -UNTENSED- HUMAN .

NP-RELATED PRESUPPOSITIONS

((JOHN /, X0048) (*UNTENSED (IN-THE-SHARED-INFO X0048)))

JOHN EXIST -UNTENSED- IN THE SHARED INFORMATION .

((JOHN /, X0048) (IF-THEN (IN-THE-PAST (WIN X0048)) ((MARY /, X0050) (*UNTENSED (IN-THE-SHARED-INFO X0050)))))

IF JOHN WON THEN MARY EXIST -UNTENSED- IN THE SHARED INFORMATION .

ENTAILMENTS

((JOHN /, X0048) (IN-THE-PAST (WIN X0048)))

JOHN WON .

((MARY /, X0050) ((JOHN /, X0048) (IN-THE-PAST (DISAPPOINT (FACT (IN-THE-PAST (WIN X0048))) X0050))))

THE FACT THAT JOHN WON DISAPPOINTED MARY .