JOYCE FRIEDMAN AND DAVID S. WARREN

A PARSING METHOD FOR MONTAGUE GRAMMARS

ABSTRACT. The main result in this paper is a method for obtaining derivation trees from sentences of certain formal grammars. No parsing algorithm was previously known to exist for these grammars.

Applied to Montague's PTQ the method produces all parses that could correspond to different meanings. The technique directly addresses scope and reference and provides a framework for examining these phenomena. The solution for PTQ is implemented in an efficient and useful computer program.

This research was supported in part by National Science Foundation Grants BNS 76-23840 and MCS 76-04297.

INTRODUCTION

This paper describes an original parsing algorithm that handles quantifier scope and pronoun reference. The parsing method is applicable to formal grammars that use substitution rules to bind variables. The parsing problem for these grammars has not previously been investigated. In developing the algorithm we had in mind the grammar presented by Richard Montague in *The proper treatment of quantification in ordinary English* (PTQ). Our solution provides a general framework in which to consider problems of scope and reference. It is implemented in a practical parsing system that can be used in linguistic investigations.

The parsing method obtains derivation trees from sentences of a grammar. A major difficulty arises because each sentence in such a grammar has infinitely many derivations. Our method obtains a finite set of representative parses that is adequate to characterize the full infinite set. The finite set contains all parses that could reasonably correspond to different meanings.

The interesting and difficult part of the problem is how to incorporate the rules of quantification. Our algorithm solves this in a natural way that provides a more general framework in which to address problems related to scope and reference.

THE PARSING PROBLEM

The formal grammars investigated here are exemplified by the grammar of Montague's PTQ. PTQ gives both the syntax and semantics of a fragment of English. The syntax is presented by an inductive definition of the sets of phrases, beginning with sets of basic lexical phrases. Compound phrases combine subphrases of specified syntactic categories to create new phrases of a particular category. The semantics is given by translation to an intensional logic. Each derivation tree for a phrase translates to a distinct logical formula. Parsing a phrase to find the possible derivation trees is thus a necessary first step in finding its meanings.

A sentence may have many derivation trees. This does not necessarily make it ambiguous. The logical formulas obtained may be equivalent, so that two derivation trees correspond to the same meaning. To find all meanings, we must find at least one derivation tree for each possible meaning.

The grammar of PTQ actually gives infinitely many derivation trees for each sentence, as we show below. Since there are infinitely many derivations, no parser can produce them all. However, each phrase has only finitely many meanings. We would like to produce only a finite set of parse trees, and at the same time obtain all meanings.

One might attempt to use the inductive definition directly and simply add a criterion for stopping after all the meanings have been obtained. No straightforward way of doing this is apparent. Instead of attempting to provide a stopping criterion, we chose a more efficient alternative approach that yields only the derivation trees of interest. We define a restricted grammar, which yields only finitely many derivation trees for any input and is sufficient to produce all meanings of any sentence.

Uses of a Parser

For complex grammars a parsing method can be a valuable tool in theoretical investigations. As examples, we cite two discussions from the literature:

(1) In a discussion of an extension to PTQ, Thomason (1974) writes:

It is possible to construct rigorous proofs that certain expressions ... are not generable as sentences in this fragment. Such results require a large number of lemmas, whose proofs, though not difficult, involve a detailed examination of the syntactic rules of the fragment....

Though it's clear that the mere fact that *one* way of trying to generate an expression is blocked will not suffice to demonstrate that *all* ways will be blocked, transformationalists have tended to accept such 'demonstrations' and have neglected to develop adequate mechanisms for establishing ungrammaticality. I believe that this has been unfortunate; it eliminates a powerful incentive to make rules simple and their formulations precise.

Parsing can show that an expression is not generable. Indeed, this requires only a recognizer, a simpler algorithm that does not yield the derivation trees but merely decides whether or not one exists. Once it is shown that the parsing method is correct, proofs for an individual expression are no longer needed. The parsing method will determine its derivations if it is generable. Notice also that the need to prepare the rules of the grammar for formal use can provide the 'powerful incentive' suggested by Thomason.

(2) Since our parsing method produces all semantically distinct derivation trees, it can be used to demonstrate that a certain meaning can be obtained only by using a particular rule. Partee (1975) writes:

The rule of CN-scope quantification plays a crucial role in certain cases, although the crucial examples are much less obvious in PTQ than they would be if Montague had included CN-phrases like *friend of him*...

(60) Every man who has lost a pen who does not find it will walk slowly.

The reader can verify that the CN-scope quantification rule must be applied to the CN-phrase *man such that he has lost him*$_1$ *such that he does not find him*$_1$ in order to derive (60) on the reading where *a pen* has narrower scope than *every* but binds the pronoun *it*, and that there is no way to derive the sentence on that reading without the rule of CN-scope quantification.

How could the reader verify this? One way would be to construct each derivation of the sentence not using the rule of CN-scope quantification and to translate it into intensional logic. This could establish that the reading discussed is not obtained. This process is not one to which the term 'verification' is usually applied. It is in the spirit of a creative proof, unless a parsing algorithm is available to produce the derivations. Using our algorithm we find that there are ten parses of the closest sentence of PTQ, namely,

Every man such-that he has lost a pen such-that he doesn't find it will walk slowly.

By examining the translations, we see that the only parse to produce the desired meaning does indeed use the CN-scope quantification rule.

Our algorithm is implemented in a computer program, which applies it to any input and gives the parse trees automatically. The program frees the linguist from the time and tedium of accurately following the steps of the algorithm. Clearly, using the parsing program does not take away the need for creativity in constructing examples. Their verification can appropriately be done by a computer.

PARSING PTQ

Each of our derivation trees is exactly the same as a derivation tree for PTQ. The language of the modified grammar is essentially the same as the language of PTQ; we chose not to include sentences containing free variables because they do not occur in English. They could easily be included. With this exception the language is that of PTQ. Henceforth, when we refer to PTQ, we refer to the modified grammar and language.

The reader is assumed to be familiar with the syntactic rules of PTQ. We include them in an Appendix for easy reference.*

Infinitely Many Derivations

Every sentence in the language of PTQ has an infinite number of different derivations in the grammar of PTQ. There are at least three ways that this can happen. First, vacuous applications of rules S14–S16 can be nested arbitrarily deeply. For example, by rule S14, we may generate a sentence by substituting ***a unicorn***, a term phrase, for he_0 in the sentence *John loves Mary*. This yields the structure of Figure 1 for the sentence *John loves Mary*. Since any term could be substituted in place of ***a unicorn***, or ***a unicorn*** could be substituted any number of times, we clearly have an infinite number of derivations. Second, from any derivation tree that contains some variable he_i, another can be obtained by choosing a new variable he_j that does not occur in the tree, and uniformly replacing all occurrences of he_i with he_j. This gives a new derivation. For instance, by replacing he_0 by he_{437} in the tree of Figure 1, we get another derivation for *John loves Mary*. Finally, any one of the rules S14–S16 can be used repeatedly to substitute one free variable for another. In this way also, infinitely many derivations for a single sentence may be obtained.

A parser for a grammar that has infinitely many derivations for a single sentence cannot be useful in the ways discussed above: if there are

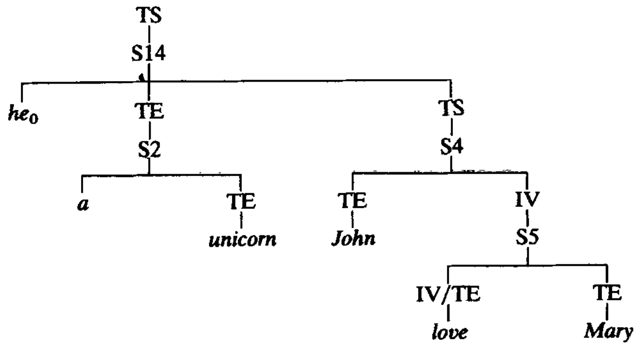


Fig. 1 Vacuous Application of S14.

* The rules given here first appeared in Hintikka *et al.* (Eds.), *Approaches to Natural Language*, D. Reidel, Dordrecht, 1973, pp. 224–225.

infinitely many derivations, we can never obtain them all. Furthermore, at no point can we be sure that we have found all the interesting ones. No parser that represents PTQ directly can avoid these problems.

Which Derivations?

The question now arises as to which of the infinitely many derivations of a given sentence to take as interesting. The principle used to make these decisions is based on the semantics associated with the structures. Variations of a structure which in all circumstances have the same semantics are not generated.

There is a technical problem with the semantics of sentences containing vacuous applications of rules S14–S16. The parse tree of Figure 1 contains a vacuous substitution of *a unicorn*. For this tree the sentence *John loves Mary* has the semantics 'there is an X such that X is a unicorn and John loves Mary'. Since *John loves Mary* does not necessarily entail the existence of unicorns, this parse is clearly not desired. One way to eliminate it is to reformulate the substitution rules S14–S16 so that vacuous applications are not allowed. Vacuous applications of S3, while perhaps not being acceptable syntactically, have no similar problem with their semantics.

This parsing system produces no derivations that contain a vacuous application of any rules S14–S16. Only one variant of each derivation is produced, that is, no two trees which differ only in variable numbers are produced. Also no tree which contains a use of a substitution rule in which one variable is substituted for another is produced. However, vacuous applications of S3 are produced.

It is important to notice that (1) no meanings are lost, and (2) for a given sentence and meaning, any lost tree is like one kept except in having variable for variable substitutions or in having different variables. Thus in the Partee example cited above, where the only trees produced for a particular meaning used the CN-scope rule S15, the lost trees for this meaning must also have used S15.

TRANSITION NETWORK GRAMMARS

Montague gave his inductive definition of the language of PTQ using English as the metalanguage. We give our inductive definition of the language as a nondeterministic algorithm expressed in a formalism appropriate to a computer, an augmented transition network (ATN). The ATN was first developed as a representation of natural language grammars by

Thorne, Bratley, and Dewar (1968). Woods (1970) further developed the concept into an extended form that is readily understandable and usable. The ATN used here is based on that form.

An augmented transition network represents a grammar as an underlying context-free component with augmentations of various sorts. The grammar of PTQ has an underlying categorial, hence context-free, component that is augmented with substitution rules. Thus the ATN seems a natural representation.

An ATN represents a grammar as a network. It may be used for either parsing or generation. A parser takes as inputs an ATN and a sentence. Its output is the set of parses of that sentence with respect to the ATN. A parse is a derivation tree or analysis tree for the sentence. It corresponds to a path through the ATN.

We now describe the ATN formalism. As our example we use the context-free part of the grammar of PTQ. We develop this grammar in steps, going from a simple finite-state network through a series of increments until a final context-free net is reached. The remainder of this section is thus essentially a tutorial on ATN's. Readers familiar with the ATN formalism can move rapidly to the discussion, in the next section, of the more interesting aspects of the solution to the parsing problem for PTQ.

Finite-state Networks

A finite-state grammar can be represented graphically as a network of nodes and directed arcs. The nodes correspond to nonterminal symbols and the arc labels to terminal or category symbols. The sequence of terminals that make up a sentence can be read by concatenating the arc labels on a path from the initial node to a final node.

For example, the finite-state grammar of Figure 2 can be represented as Net-1, the network of Figure 3.

The nonterminal category TS, sentence, labels the initial node. We use TS for the type of sentences (Montague's t), and TE for the type of terms (Montague's T). The lexical or terminal categories are bte, basic terms; biv/te, basic transitive verbs; and biv, basic intransitive verbs. A lexicon

TS → bte IV
IV → biv/te TE
IV → biv
TE → bte

Fig. 2 A Finite-State Grammar.

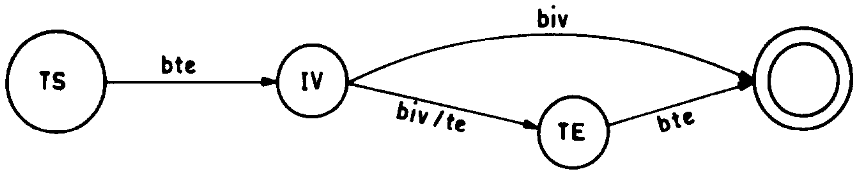


Fig. 3 Net-1, a Finite-State Network.

gives the English phrases of each terminal category. Suppose bte contains *John* and *Mary*, biv/te contains *loves*, and biv contains *runs*. Then *John loves Mary* and *John runs* are among the sentences represented.

Net-1 is simply an alternative way of presenting the grammar of Figure 2. The grammar can be used in generating sentences or in parsing them, and the same is true of the network. In describing a network we sometimes refer to an arc as 'accepting' a phrase, but we may equally well think of it as 'generating' the phrase. The net itself is independent of its use.

The set of sentences of this net is finite, but that is not intrinsic to finite-state networks. If we add an arc from node TS to itself and label it with the category bts/ts containing *necessarily*, then for each sentence *S* of the language there is also the sentence *necessarily S*.

Recursive Transition Networks

A first step in increasing the power of the representation is to allow recursion by the use of nonterminal symbols as arc labels, as in Net-2 of Figure 4.

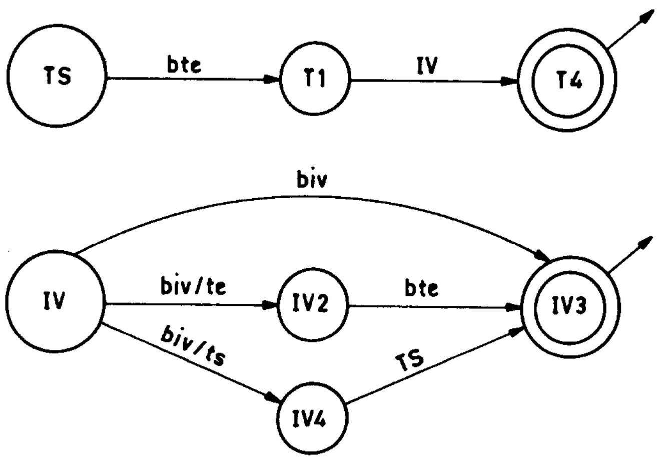


Fig. 4 Net-2, a Recursive Transition Network.

These nonterminal symbols also name nodes. To traverse an arc labeled with a nonterminal symbol, one traverses the subnet whose initial node has that name. The nodes with double circles are final states; the outgoing arrow indicates a return to the position in the higher net from which this net was entered.

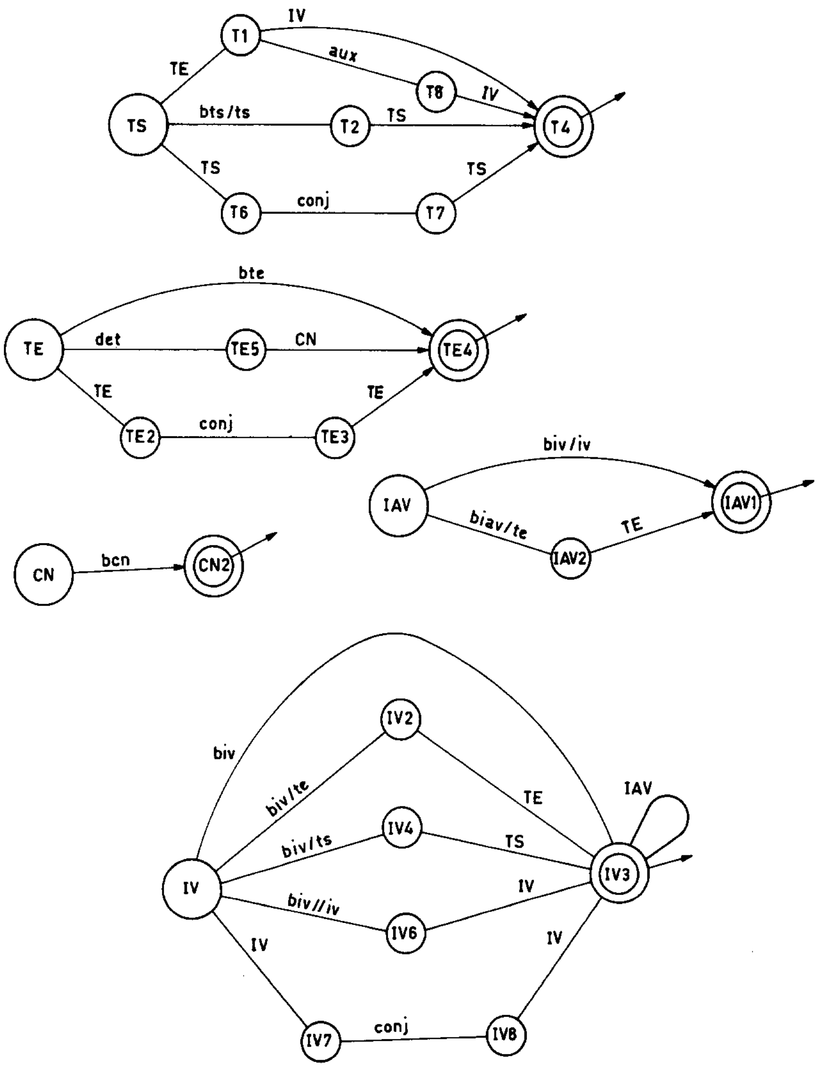


Fig. 5 Net-3, a Context-Free Subnet of the PTQ Net.

Suppose biv/ts contains *believes-that* and *asserts-that*. Then Net-2 represents all the sentences previously obtained, and also all sentences in which the verb phrase is *believes-that S* or *asserts-that S*, where *S* is any sentence.

Recursive transition networks correspond exactly to context-free grammars. Figure 5 shows Net-3, the part of the PTQ net that includes the syntactic rules S2, S4 through S13, and S17. All of these rules are essentially context-free: they concatenate their subparts.

Consider the simple sentence *John loves Mary*. *John* and *Mary* are words in the basic category bte, basic term phrases; *loves* is a basic transitive verb, category biv/te. *John loves Mary* is obtained on exactly one path through Net-3. The net is entered at TS, the sentence node. A sentence can begin with a term TE, a sentence-modifier bts/ts (of which there is only one, *necessarily*), or a sentence TS. The TE arc leading from the initial TS node represents entry into the TE subnet. *John* is a bte, so the TE subnet returns from TE4 to the TS net at node T1. The possible continuations are the IV and aux arcs; here the IV arc is used. IV is a nonterminal symbol and as such it names a subnet. Examination of the IV subnet shows that an IV can begin with a word of any of four basic categories or can be a conjunction of IV's. Here the second arc, biv/te, is the only successful category arc. The TE subnet is then entered again. Since *Mary* is a bte, the TE net is traversed on its first arc, returning to the final state IV3 of the IV net, and from here to T4, the final state of the TS net. Return from the TS net signals that a sentence has been recognized. Figure 6 shows schematically this path through the net.

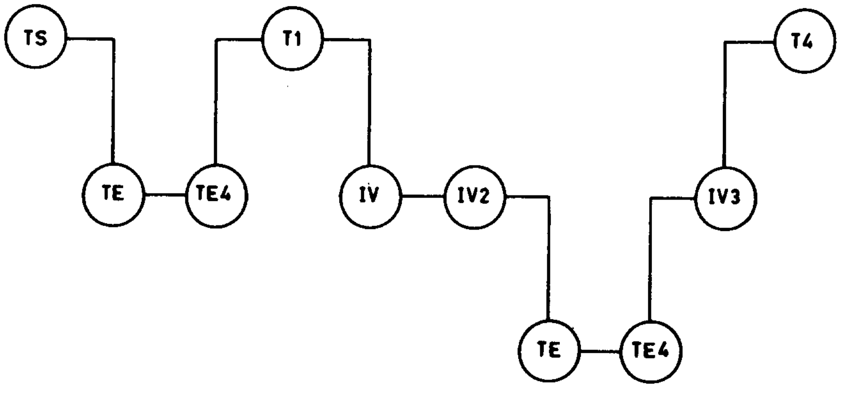


Fig. 6 Path Through Net-3 for *John loves Mary*.

In the discussion so far we have shown how the net recognizes a sentence. For this it suffices to establish that there is some path for the sentence through the net. Multiple paths correspond to syntactic ambiguity, because each path through the net yields a different derivation of the sentence. Since the meaning of a sentence is a function of the derivation tree, semantic ambiguity may also result. Since we are interested in obtaining all meanings of a sentence, the net is used as a parser, that is to produce sentence structure. This is done by building a tree in each subnet to represent the structure of the string recognized by the subnet. This tree is then returned to a higher net where it is used as a subtree in a larger construction.

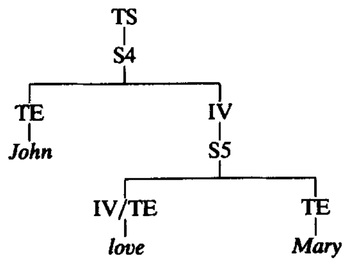


Fig. 7 Parse Tree for *John loves Mary.*

Figure 7 shows the parse tree for *John loves Mary.* Category names are included in the figures to aid the discussion. They are not produced by the parser, since the rule names uniquely determine them.

Augmented Transition Networks

An augmented transition network is a recursive transition network in which computations are performed while traversing the path through the net. These computations are specified by writing instructions on the arcs. They alter the conditions under which the arc can be traversed and also allow information to be saved and used later. Since these instructions can be arbitrary, the language that is represented by the network no longer need be context-free. These augmentations do not alter the structure of the net.

The first augmented network has the structure of Net-3 (Figure 5), but with additional computations on certain arcs. These particular additions implement language features that could alternatively be treated in a larger context-free net. Verb type is handled by transmitting information from each arc labeled IV to the IV subnet to indicate whether the verb form should be finite or infinite. The subnet uses this and additional information from the lexicon to decide on the correct form. Problems peculiar to conjunction are handled similarly.

PARSING FOR SCOPE AND REFERENCE

At this point our introductory description of the ATN formalism is complete. Net-3 gave all the essentially context-free rules. We can now proceed to describe our particular solution to the problems in parsing Montague grammars.

The most interesting contribution of our algorithm is in the treatment of pronouns and substitution rules. Montague gives four rules for quantification and pronoun reference. These are the relative clause rule S3 and the three quantification rules S14 through S16, which are not context-free.

We describe our solution as though we were first finding an underlying context-free structure and then traversing it to resolve questions of reference and quantification. In actuality the algorithm makes only one pass. It handles pronouns and quantifiers as it builds the tree.

Context-free Treatment of Pronouns

To help describe how the substitution rules are parsed, we first give an incomplete treatment of pronouns which is context-free. Net-4 (Figure 8) shows a grammar that has pronouns and has a simpler version of S3. Net-4 differs structurally from Net-3 only in its CN and TE subnets.

Net-4 accepts sentences with pronouns and with relative clauses, but does not associate the pronouns with antecedents. Thus, for example, it accepts *She runs* and *Mary loves a man such-that it runs.* The treatment of pronouns is complicated by the need to check their case. We assume first that *he*, *she*, *it*, *him*, and *her* are added as basic pronouns and are appropriately marked for case as NOM or ACC or both. To extend the net for pronouns, one new arc is added at TE to accept pronouns as term phrases.

A test in this new pronoun arc checks that the pronoun has the required case. The required case is determined before entering the TE subnet, either in the subject arc at TS or in the object arc at IV2.

Without relative clauses the CN subnet had only to look for a basic common noun and return it. To allow relative clauses the CN net is

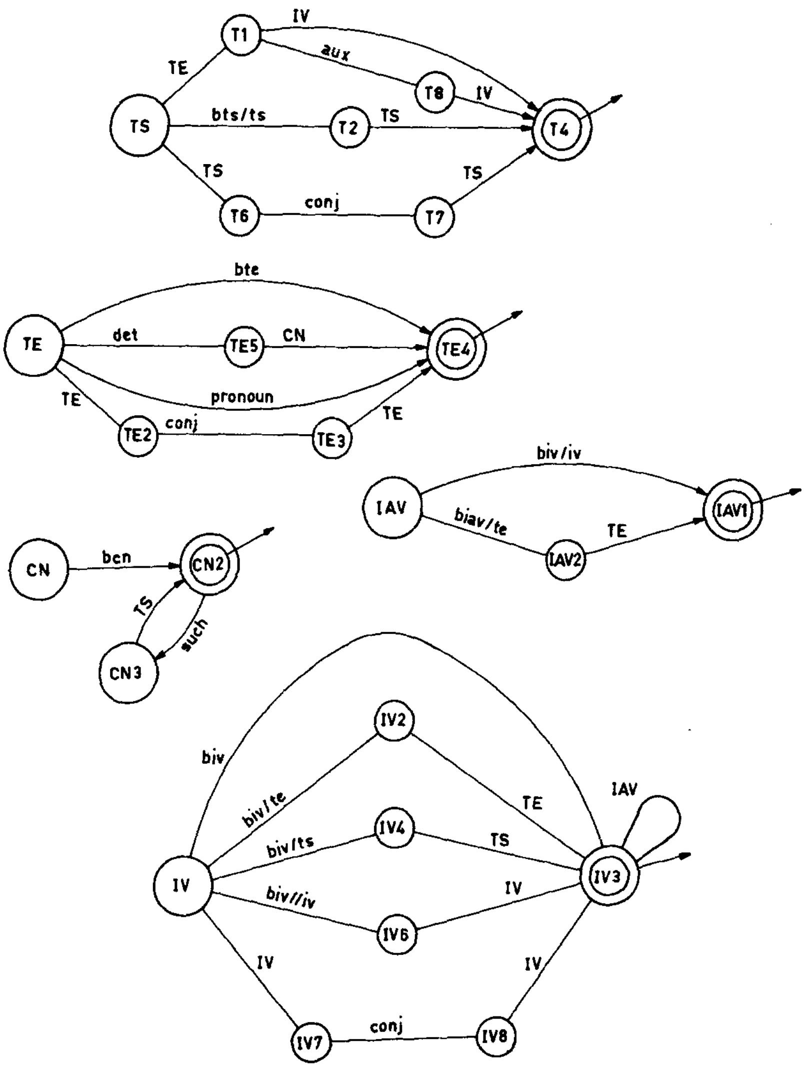


Fig. 8 Net-4, the Context-free Net with Pronouns and Relative Clauses.

modified. Node CN still has only its one arc. Node CN2 gets an additional arc which recognizes *such-that*. At CN3 the sentence of the relative clause is found and the whole S3 construction stored. This yields, for example, the tree of Figure 9.

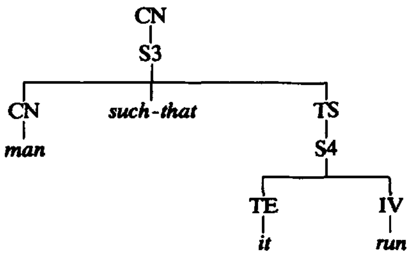


Fig. 9 Common Noun Tree.

Rules of Quantification

The crucial final step in constructing a parser for PTQ is to add to the net the full version of S3 and the substitution rules S14 through S16. These rules present difficulties in the design of the algorithm; it needs to be more complex than might at first appear. There are constraints on the ways that the substitution rules can be applied. Substitution rules combine a term and a subtree containing free variables in a way similar to the way in which a quantification rule in predicate logic combines a variable and a subformula. An application of a substitution rule consists in substituting a term for all occurrences of a variable in a subphrase. In translation the term corresponds to a quantifier, the variable to its bound variable, and the subphrase to its scope. Much of the ambiguity in the formal English of PTQ arises from differences in the scope and order of applications of the substitution rules.

Net-4 contains all of the structure of PTQ, but is not yet augmented for the strictly non-context-free parts of S3 and the substitution rules S14–S16. This is done in PTQNET, which is structurally the same as Net-4 (Figure 8).

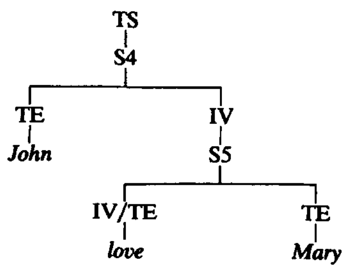
In order to find a derivation for a sentence that includes an application of a substitution rule, the fact of the use of the rule, the scope of the rule, and the variables bound by this use must all be determined. To help understand how a parse tree containing an occurrence of a substitution rule is constructed, we may think of PTQNET as traversing the Net-4 context-free parse tree and performing operations on that tree to obtain the full parse. The order of traversal of the parse tree is left-to-right and depth-first. For example, in Figure 7 above, the nodes are traversed in the following order: TS, first TE, TS, IV, IV/TE, IV, second TE, IV, TS. Note that each nonterminal node is traversed more than once.

In general, any term phrase encountered can either be entered immediately into the tree, or can be substituted in later by an application of a substitution rule at some higher node. Therefore, a choice is made whenever a term phrase is encountered. One alternative is to insert the term phrase directly. The other is to enter a variable in place of the phrase and later bind this variable through the application of a substitution rule. Different parses result from these alternatives.

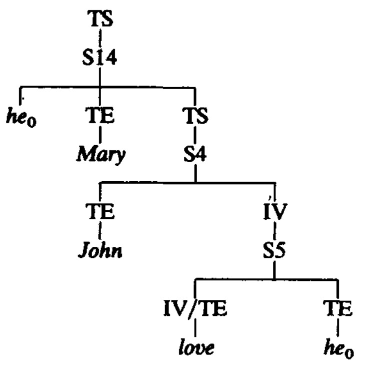
To handle the substitution rules, two sets of bindings are maintained at each node in the tree. A binding is a paired variable and term phrase. Each binding will correspond to an instance of a substitution rule in which a phrase is substituted for its variable. The FVB-set (Free Variables Below) contains bindings created at this node or passed up from FVB-sets below. After a binding is used in a substitution rule, its variable is no longer free so the binding does not get passed up to the FVB-set above. Consequently, the completed FVB-set is the set of bindings for all free variables dominated by the current node. The second set is the SA-set (Substitutions Above). The SA-set for a node consists of the bindings passed down from the SA-set and FVB-set of its dominating node. The SA-set and the FVB-set of a node together contain each binding that corresponds to a substitution that will include the node in its scope.

We now outline the general method for parsing substitution rules. It is illustrated by examples below. The FVB-sets and SA-sets for each node are determined dynamically during a traversal of the context-free parse tree. When a term that is not a pronoun is encountered, it may be replaced by a new free variable and the corresponding binding placed in the FVB-set. The FVB-set for each node is also extended each time the node is reached from below by including the bindings in the FVB-set of the dominated node. On the final traversal through an IV, TS, or CN node, a binding in its FVB-set indicates that a substitution rule may be applied there. If it is applied, that binding is not passed up to the FVB-set of the dominating node. Pronouns are handled through the use of the SA-sets, which contain bindings for all substitution application above the node. When a node is reached from above, its SA-set is created as the union of both the FVB-set and the SA-set of the immediately dominating node. When a pronoun is encountered, its antecedent is determined by searching the SA-set for a binding that contains a term phrase of the correct gender.

The algorithm is illustrated by the example of Figure 10. It gives two parses for the simple sentence *John loves Mary.* The parse in Figure 10a is



10a Using NET-5.



10b Using PTQNET.

Fig. 10 Example for S14.

obtained using Net-4. The parse in Figure 10b uses the substitution rule S14 and thus requires PTQNET.

Consider how to construct the tree 10b during a traversal of tree 10a. This example shows how the FVB-sets are maintained and used to determine applications of a substitution rule. The first entry into an FVB-set occurs when *Mary* is encountered and we are about to return to the immediately dominating IV node. At this point it can be decided that this term, *Mary*, is the term substituted over some dominating node by a substitution rule. In this case the term is not to be entered into the tree at this point. Instead a new variable is taken and paired with *Mary* to generate the binding (*he*$_0$ *Mary*). The variable, *he*$_0$, is entered in the tree instead of *Mary*. The binding is entered into the FVB-set for the TE node since this node dominates the free variable *he*$_0$. When return is made to the immediately dominating IV node, this binding is added to its FVB-set, since it also spans this free variable. Similarly, the binding is added to the FVB-set of node TS. Note that 10b contains a copy of 10a with *Mary* replaced by *he*$_0$. Now since we are about to leave a TS node and its FVB-set contains a binding to which S14 can apply, we choose to apply it here. The binding indicates the term to be substituted for the free variable, so we construct the tree of 10b. Note that the binding (*he*$_0$ *Mary*) does not then go up to the FVB-set of the new top TS node, since *he*$_0$ is not free below it.

This example shows how the FVB-sets are used to determine the scope of substitution rule instances. There is another similar parse of this sentence, *John loves Mary*, in which the substitution rule has a different scope. This parse, shown in Figure 11, uses the substitution rule S16, which substitutes over intransitive verb phrases. Traversal continues as before

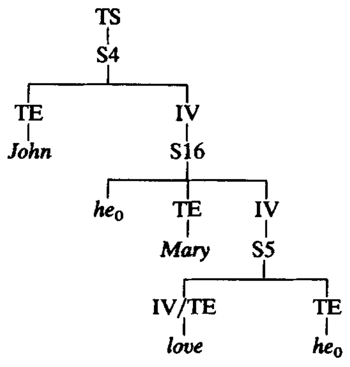


Fig. 11 Example for S16.

until we are about to return from the IV node back to the TS node. Remember that the FVB-set for this IV node contains the binding (*he*$_0$ *Mary*). At this point we decide to insert an application of rule S16 since there is a free variable below waiting to be bound. The instance of rule S16 is inserted in the tree here; the binding does not go up to the FVB-set of the dominating TS node as it did in Figure 10. The tree of Figure 11 results.

Note that new bindings are added to the FVB-sets only when a term is encountered. Since substitution rules are only entered in a tree as the result of a binding, no vacuous applications are ever generated. Also note that for the tree to be acceptable as a parse, the FVB-set of the top TS node must be empty, indicating that all the free variables in the tree have been substituted for.

Parsing Pronouns

The other major problem with respect to the substitution rules is the correct parsing of pronouns. Recall that, using Net-4, antecedents of pronouns were not found. Sentences that contained a pronoun with no antecedent or with an apparent antecedent of different gender were accepted. In PTQNET, the SA-sets are used to solve this problem. Consider the parses of Figure 12 for the sentence, *John loves Mary and she loves him.* Tree 12a is a Net-4 parse and tree 12b is a PTQNET parse. Tree 12b can be constructed during a traversal of tree 12a in the left-to-right depth-first order (that is, TS-0, TS-1, TE-1, TS-1, IV-1, IV/TE-1, IV-1, TE-2, IV-1, TS-1, TS-0, TS-2, TE-3, TS-2, IV-2, IV/TE-2, IV-2, TE-4, IV-2, TS-2, TS-0.) Suppose that in the traversal of the first half of this tree *John* and *Mary* are both replaced by variables and entered in the FVB-sets. After returning to TS-0 and before moving on to TS-2, (*he*$_0$ *John*) and (*he*$_1$ *Mary*)

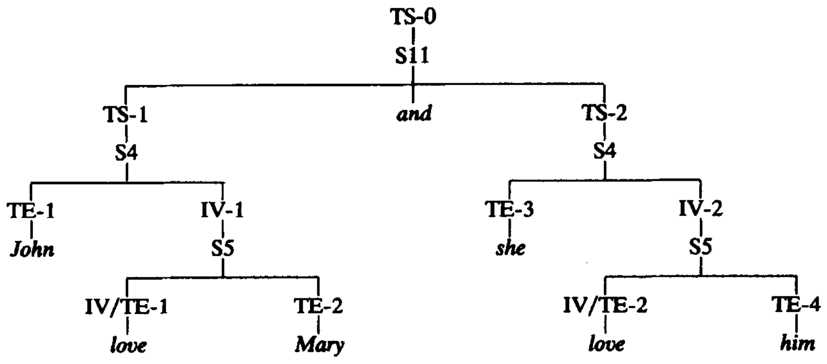


Fig. 12a Net-4 Parse.

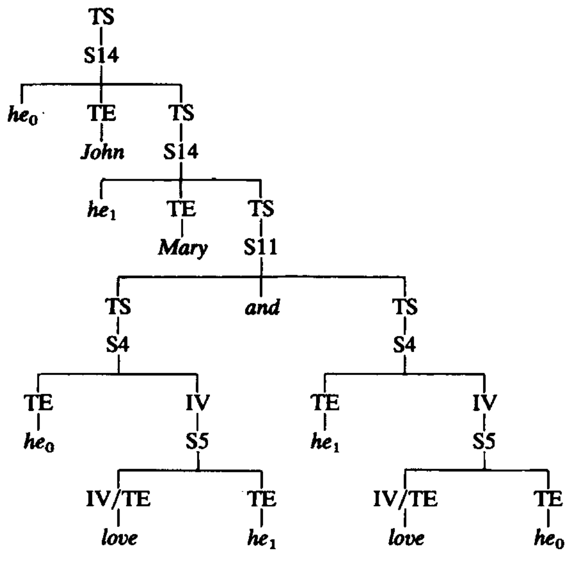


Fig. 12b PTQNET Parse.

Pronoun Antecedents
John loves Mary and she loves him

are in the FVB-set of TS-0. When we encounter the pronouns in the second half of the sentence, we will need to know what substitutions will have them within their scope. The SA-set for each node will contain these bindings. Since (*he*$_0$ *John*) and (*he*$_1$ *Mary*) are free below TS-0, they must be substituted in over TS-0 and so also over TS-2. Therefore, we must add these bindings to the SA-set of TS-2. And since the SA-sets are passed down the tree, we also add the bindings to the SA-set of TE-3. Now when we encounter *she*, we scan the SA-set and find the binding (*he*$_1$ *Mary*). Since *Mary* and *she* have the same gender, *Mary* may be the antecedent of *she*. So the variable of that binding, *he*$_1$, replaces *she* in the tree. Similarly, when *him* is encountered, (*he*$_0$ *John*) is in the SA-set of TE-4. So *he*$_0$ replaces *him* indicating that *John* is the antecedent of *him*.

Relative Clauses

Applications of S3, the rule which introduces *such-that* relative clauses, are handled in this same framework. The only difference is that there is no possible scope ambiguity, given the Net-4 context-free parse. When *such-that* is encountered, the common noun is paired with a new variable and this binding is placed directly into the SA-set. It could also be placed in the FVB-set and then immediately removed after the *such-that* phrase is traversed. The construction of the SA-sets for lower levels from the SA-sets and FVB-sets of the current level places this pair in the SA-set of every subordinate node, which allows pronouns encountered during the parse of the sentence subtree to be coreferential with the comnon noun heading the *such-that* phrase.

The sentence *John loves a woman such-that she loves him* illustrates the processing of a sentence with a *such-that* phrase. Tree 13a is its Net-4 parse and tree 13b is the PTQNET parse. When *such-that* is encountered while traversing tree 13a, (*he*$_1$ *woman*) is added to the SA-set of node TS

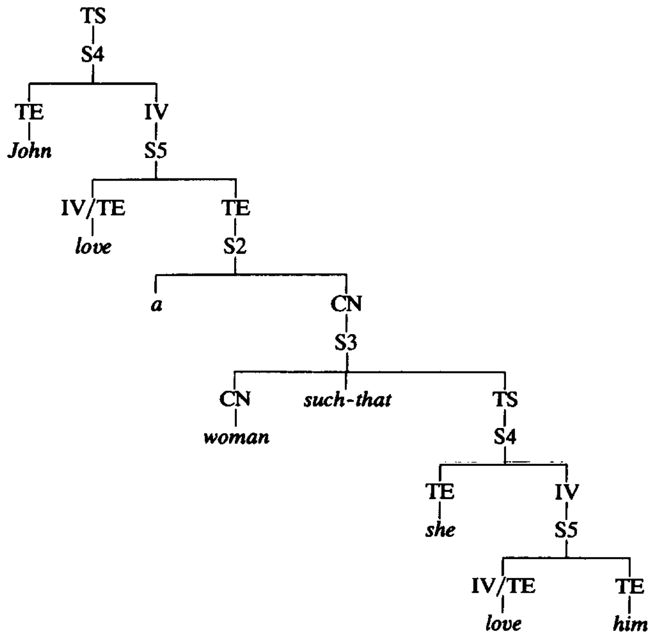


Fig. 13a Net-4 Parse.

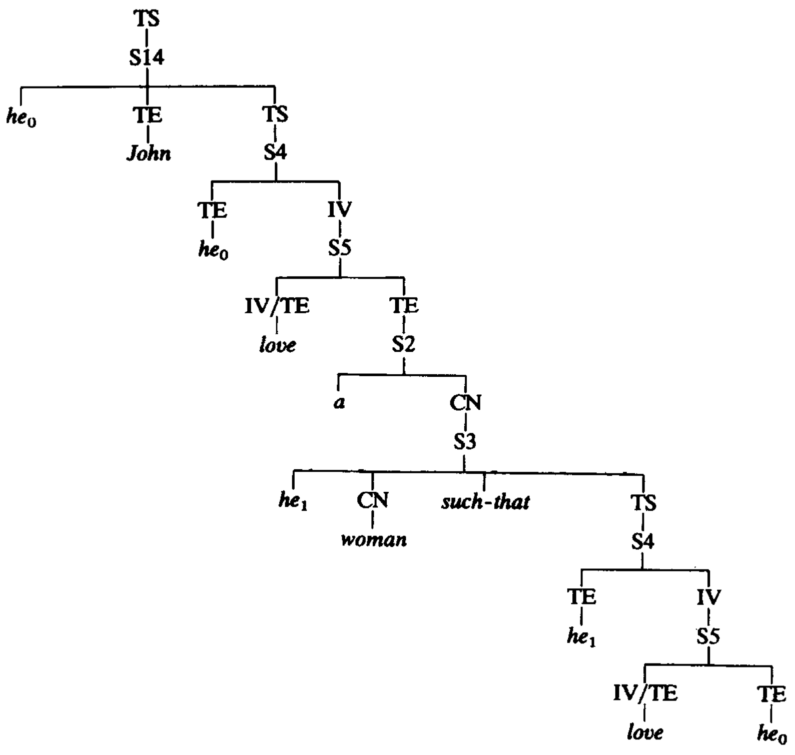


Fig. 13b PTQNET Parse.

Parsing Relative Clauses
John loves a woman such-that she loves him

so that during the traversal of this sentence subtree, this binding will be in the SA-sets. Then when *she* is encountered, the SA-set is scanned and it is found that *she* can be coreferential with *woman.*

Ambiguities of Scope and Reference

At each point in the traversal that the FVB-set or the SA-set is scanned to determine a possible next action, the possibility of ambiguity arises. For example there is another parse of the sentence of Figure 12. The two occurrences of rule S14 could appear in the opposite order, with *John* being substituted into the subsentence first and *Mary* later. This parse is found because any pair can be chosen from the FVB-set when inserting a substitution rule in a parse. Similarly parses which differ only in the referents of pronouns are found because any pair with the correct gender can be chosen from the SA-set.

Trees 14a and 14b show two parses of the sentence ***John or Bill loves Mary and she loves him.*** These parses differ only in that when ***him*** was encountered, in the first case the binding (***he***$_0$ ***John***) was chosen from the SA-set whereas in the second tree at that same point (***he***$_1$ ***Bill***) was chosen. These trees clearly differ semantically. In the first Mary loves John and in the second she loves Bill. In still another parse, not shown, Mary loves John or Bill.

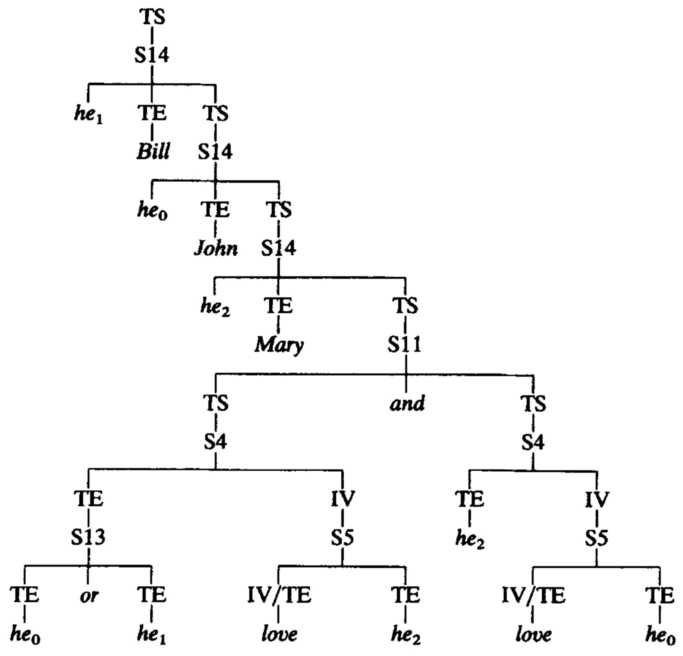


Fig. 14a Mary loves John.

Constraints on Order of Substitution

This possibility of choosing elements from the FVB-set and SA-set allows the parser to find all parses which vary in the order of application of substitution rules and in the referents for pronouns. However, if all orders of substitution and all possible referents for pronouns are permitted, trees which are not parse trees for the input sentence are obtained in addition to all the correct parses. Consider the example, ***A woman such-that she loves a man such-that he loves her walks.*** Allowing all orders of application of substitution rules when parsing this sentence yields the tree of Figure 15. This is not a correct parse of the given sentence. When the syntactic rules are applied to generate the sentence, we get ***A woman such-that she loves a***

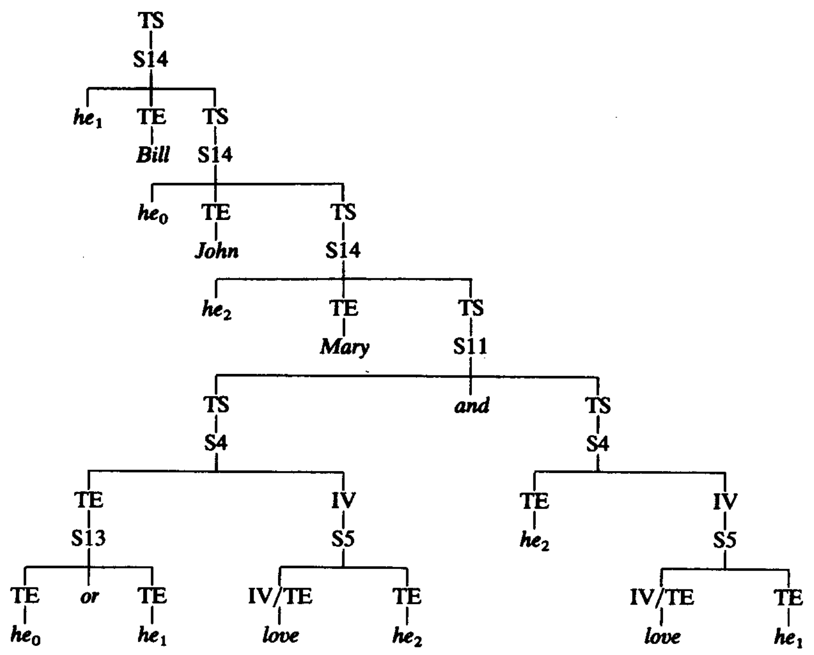


Fig. 14b Mary loves Bill.

Pronoun Ambiguity
John or Bill loves Mary and she loves him

man such-that he loves him$_0$ ***walks***. This still contains a free variable so the tree is not a parse for the original sentence.

The problem arises because terms can be nested in other terms. We eliminate the bad parses by keeping track of what terms are embedded in what other terms. The rule is that before any variable-term pair can be substituted over a phrase, that phrase must already contain all occurrences of the variable. That is, no variable-term pair can be substituted while occurrences of that variable remain in terms of other pairs in the FVB-set.

In the above false parse, the substitution of ***woman*** for ***he***$_0$ into ***he***$_0$ ***loves him***$_2$ violates this rule. It occurs at a CN-S3 node whose FVB-set also contains a variable-term pair with variable ***he***$_2$ and term ***a man such-that he loves him***$_0$. Thus, after the substitution a term still in the FVB-set contains an occurrence of ***he***$_0$. This pair is passed up the tree to nodes above the CN although ***he***$_0$ has been bound at the CN. When the ***he***$_2$-term pair is finally used, ***him***$_0$ will enter the tree above the point where ***he***$_0$ is bound. It will thus remain free.

To avoid this problem, when deciding to substitute a term into a subtree, it is checked that no term in the current FVB-set contains an occurrence of

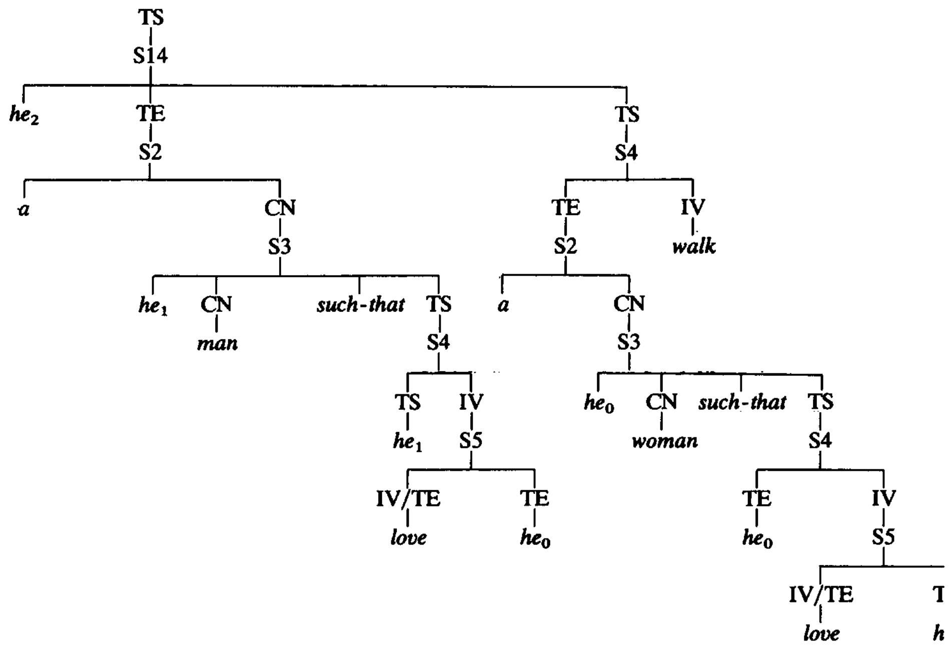


Fig. 15 Incorrect Parse Tree.
A woman such-that she loves a man such-that he loves her walks

the variable for which the chosen term is being substituted. If the rule is violated, the substitution is not made, since it could only lead to bad parses. There may of course be successful parses, but they will be arrived at in other ways. For example, the good parses of this sentence include some that substitute the he_2-term pair in over the subsentence he_0 *loves* him_2.

A MODIFICATION TO PTQ

Variants of PTQ can be investigated using this system. PTQ contains several errors in its treatment of conjoined phrases (see, for example, Bennett (1976) or Friedman (1977)). Our ATN contains an option that gives it a more sophisticated treatment of conjunction. On one choice the net preserves the errors in the grammar of PTQ; on the other, it represents a modified grammar that corrects them.

The choice of alternatives for conjoined verb phrases is implemented in passing information about verb types into the IV subnet. For example, if the errors are preserved, *John walks and talk* is accepted because the infinite verb *talk* is found by the IV subnet. Otherwise the net will look for a finite verb, so that *John walks and talks* will be accepted instead. The option also applies to pronouns, so that, for example, either *Bill loves Mary* *and Mary loves John or he* as in PTQ, or the corrected form *Bill loves Mary and Mary loves John or him* can be represented.

Computer implementation

The parsing algorithm has been implemented in the LISP programming language. It is one of a set of computer programs designed to aid in working with and testing PTQ and its extensions. The parsing system has a central role. It takes an English sentence and finds its derivation tree or trees. The parser produces all the correct derivation trees (to within well-specified equivalences) for any sentence of the PTQ fragment. The outputs of the parsing system are directly acceptable to a translation program that produces the meaning of the sentence in intensional logic, and applies various reductions to the formula. These formulas in turn can be interpreted in a possible worlds model by other programs. These programs will be described elsewhere.

Conclusions

Our parsing system for the English fragment of PTQ finds the derivation trees for any sentence. From the infinite set of trees defined by PTQ it selects a finite set that contains all trees that could have different semantics.

The grammar is represented as an ATN, which gives a natural treatment of the rules of PTQ. While most of the rules are essentially context-free, interesting problems are presented by the relative clause rule and the quantification rules. We have created a general variable-handling framework, which treats these in a natural way. The resulting system reflects the underlying structure of the phenomena. The framework for pronouns appears to have potential for the investigation of alternative approaches.

We are examining the possibilities for applying the parsing method to other Montague grammars. Our corrected treatment of conjunction is an example of a modification of PTQ. For extensions of PTQ that are *extensions by rule*, as, for example, Karttunen's (1977) treatment of questions, the extension will involve straightforward changes to the net. *Extensions by mechanism*, such as proposed by Bennett (1976) and Thomason (1976), may require more research, including perhaps some additional ways of augmenting the net. We are now exploring these questions.

University of Michigan
September 1977
Revised December 1977

JOYCE FRIEDMAN
DAVID S. WARREN

APPENDIX

SYNTACTIC RULES OF PTQ

Basic rules

S1. $B_A \subseteq P_A$ for every category A.

S2. If $\zeta \in P_{CN}$, then $F_0(\zeta)$, $F_1(\zeta)$, $F_2(\zeta) \in P_T$,
where $F_0(\zeta) =$ **every** ζ,
$F_1(\zeta) =$ **the** ζ,
$F_2(\zeta)$ is **a** ζ or **an** ζ according as the first word in ζ takes **a** or **an**.

S3. If $\zeta \in P_{CN}$ and $\phi \in P_t$, then $F_{3,n}(\zeta, \phi) \in P_{CN}$, where $F_{3,n}(\zeta, \phi) = \zeta$ **such that** ϕ'; and ϕ' comes from ϕ by replacing each occurrence of **he**$_n$ or **him**$_n$ by $\left\{\begin{matrix}\textbf{he}\\ \textbf{she}\\ \textbf{it}\end{matrix}\right\}$ or $\left\{\begin{matrix}\textbf{him}\\ \textbf{her}\\ \textbf{it}\end{matrix}\right\}$ respectively, according as the first B_{CN} in ζ is of $\left\{\begin{matrix}\text{masc.}\\ \text{fem.}\\ \text{neuter}\end{matrix}\right\}$ gender.

Rules of functional application.

S4. If $\alpha \in P_{t/IV}$ and $\delta \in P_{IV}$, then $F_4(\alpha, \delta) \in P_t$, where $F_4(\alpha, \delta) = \alpha\delta'$ and δ' is the result of replacing the first *verb* (i.e., member of B_{IV}, B_{TV}, $B_{IV/t}$, or $B_{IV//IV}$) in δ by its third person singular present.

S5. If $\delta \in P_{IV/T}$ and $\beta \in P_T$, then $F_5(\delta, \beta) \in P_{IV}$, where $F_5(\delta, \beta) = \delta\beta$ if β does not have the form **he**$_n$ and $F_5(\delta,$ **he**$_n) = \delta$ **him**$_n$.

S6. If $\delta \in P_{IAV/T}$ and $\beta \in P_T$, then $F_5(\delta, \beta) \in P_{IAV}$.

S7. If $\delta \in P_{IV/t}$ and $\beta \in P_t$, then $F_6(\delta, \beta) \in P_{IV}$, where $F_6(\delta, \beta) = \delta\beta$.

S8. If $\delta \in P_{IV//IV}$ and $\beta \in P_{IV}$, then $F_6(\delta, \beta) \in P_{IV}$.

S9. If $\delta \in P_{t/t}$ and $\beta \in P_t$, then $F_6(\delta, \beta) \in P_t$.

S10. If $\delta \in P_{IV/IV}$ and $\beta \in P_{IV}$, then $F_7(\delta, \beta) \in P_{IV}$, where $F_7(\delta, \beta) = \beta\delta$.

Rules of conjunction and disjunction.

S11. If $\phi, \psi \in P_t$, then $F_8(\phi, \psi)$, $F_9(\phi, \psi) \in P_t$, where $F_8(\phi, \psi) = \phi$ **and** ψ, $F_9(\phi, \psi) = \phi$ **or** ψ.

S12. If $\gamma, \delta \in P_{IV}$, then $F_8(\gamma, \delta)$, $F_9(\gamma, \delta) \in P_{IV}$.

S13. If $\alpha, \beta \in P_T$, then $F_9(\alpha, \beta) \in P_T$.

Rules of quantification.

S14. If $\alpha \in P_T$ and $\phi \in P_t$, then $F_{10,n}(\alpha, \phi) \in P_t$, where either (i) α does not have the form **he**$_k$, and $F_{10,n}(\alpha, \phi)$ comes from ϕ by replacing the first occurrence of **he**$_n$ or **him**$_n$ by α and all other occurrences of ***he***$_n$ or **him**$_n$ by $\left\{\begin{matrix}\textbf{he}\\ \textbf{she}\\ \textbf{it}\end{matrix}\right\}$ or $\left\{\begin{matrix}\textbf{him}\\ \textbf{her}\\ \textbf{it}\end{matrix}\right\}$ respectively, according as the gender of the first B_{CN} or B_T in α is $\left\{\begin{matrix}\text{masc.}\\ \text{fem.}\\ \text{neuter}\end{matrix}\right\}$, or (ii) $\alpha =$ **he**$_k$, and $F_{10,n}(\alpha, \phi)$ comes from ϕ by replacing all occurrences of **he**$_n$ or **him**$_n$ by **he**$_k$ or **him**$_k$ respectively.

S15. If $\alpha \in P_T$ and $\zeta \in P_{CN}$, then $F_{10,n}(\alpha, \zeta) \in P_{CN}$.

S16. If $\alpha \in P_T$ and $\delta \in P_{IV}$, then $F_{10,n}(\alpha, \delta) \in P_{IV}$.

Rules of tense and sign.

S17. If $\alpha \in P_T$ and $\delta \in P_{IV}$, then $F_{11}(\alpha, \delta), F_{12}(\alpha, \delta), F_{13}(\alpha, \delta), F_{14}(\alpha, \delta), F_{15}(\alpha, \delta) \in P_t$, where:
$F_{11}(\alpha, \delta) = \alpha\delta'$ and δ' is the result of replacing the first verb in δ by its negative third person singular present;
$F_{12}(\alpha, \delta) = \alpha\delta''$ and δ'' is the result of replacing the first verb in δ by its third person singular future;
$F_{13}(\alpha, \delta) = \alpha\delta'''$ and δ''' is the result of replacing the first verb in δ by its negative third person singular future;
$F_{14}(\alpha, \delta) = \alpha\delta''''$ and δ'''' is the result of replacing the first verb in δ by its third person singular present perfect; and finally,
$F_{15}(\alpha, \delta) = \alpha\delta'''''$ and δ''''' is the result of replacing the first verb in δ by its negative third person singular present perfect.

REFERENCES

Bennett, Michael 'A variation and extension of a Montague fragment of English' in Partee, 1976, 119–163.

Friedman, Joyce 'An unlabeled bracketing solution to the problem of conjoined phrases in Montague's PTQ' Department of Computer and Communication Sciences, The University of Michigan, Ann Arbor, Mich., 1977; to appear in *Journal of Philosophic Logic.*

Hintikka, J., Moravcsik, J., and Suppes, P. (eds.), *Approaches to Natural Language,* Reidel, Dordrecht, 1973.

Karttunen, Lauri 'Syntax and semantics of questions' *Linguistics and Philosophy* 1 (1977), 3–44.

LaGaly, M., Fox, R., and Bruck, A. (eds.), *Papers from the Tenth Regional Meeting of the Chicago Linguistic Society*, University of Chicago, Chicago, 1974.
Montague, Richard 'The proper treatment of quantification in ordinary English' (PTQ), in Hintikka *et al.*, 1973, 221–242; reprinted in Montague, 1974, 247–270.
Montague, Richard *Formal Philosophy*: *Selected Papers of Richard Montague*, edited and with an introduction by Richmond Thomason, Yale University Press, New Haven, 1974.
Partee, Barbara 'Montague grammar and transformational grammar' *Linguistic Inquiry* VI (1975), 203–300.
Partee, Barbara (ed.), *Montague Grammar*, Academic Press, New York, 1976.
Thomason, Richmond H. 'Some complement constructions in Montague grammar' in LaGaly *et al.*, 1974, 712–722.
Thomason, Richmond H. 'Some extensions of Montague grammar' in Partee, 1976, 77–117.
Thorne, J. P., Bratley, P., and Dewar, H. 'The syntactic analysis of English by machine' in D. Michie (ed.), *Machine Intelligence 3*, American Elsevier, New York, 1968.
Woods, William A. 'Transition network grammars for natural language analysis' *Comm. ACM* 13 (1970), 591–606.