# Using Semantics in Non-Context-Free Parsing of Montague Grammar[1]

**David Scott Warren**

**Department of Computer Science**
**SUNY at Stony Brook**
**Long Island, NY 11794**

**Joyce Friedman**

**University of Michigan**
**Ann Arbor, MI**

**In natural language processing, the question of the appropriate interaction of syntax and semantics during sentence analysis has long been of interest. Montague grammar with its fully formalized syntax and semantics provides a complete, well-defined context in which these questions can be considered. This paper describes how semantics can be used during parsing to reduce the combinatorial explosion of syntactic ambiguity in Montague grammar. A parsing algorithm, called *semantic equivalence parsing,* is presented and examples of its operation are given. The algorithm is applicable to general non-context-free grammars that include a formal semantic component. The second portion of the paper places semantic equivalence parsing in the context of the very general definition of an interpreted language as a homomorphism between syntactic and semantic algebras (Montague 1970).**

## Introduction

The close interrelation between syntax and semantics in Montague grammar provides a good framework in which to consider the interaction of syntax and semantics in sentence analysis. Several different approaches are possible in this framework and they can be developed rigorously for comparison. In this paper we develop an approach called *semantic equivalence parsing* that introduces logical translation into the ongoing parsing process. We compare this with our earlier *directed process* implementation in which syntactic parsing is completed prior to translation to logical form.

Part I of the paper gives an algorithm that parses a class of grammars that contains both essentially context-free rules and non-context-free rules as in Montague's 1973 PTQ. Underlying this algorithm is a nondeterministic syntactic program expressed as an ATN. The algorithm introduces *equivalence parsing,* which is a general execution method for nondeterministic programs that is based on a recall table, a generalization of the well-formed substring table. Semantic equivalence, based on logical equivalence of formulas obtained as translations, is used. We discuss the consequences of incorporating semantic processing into the parser and give examples of both syntactic and semantic parsing. In Part II the semantic parsing algorithm is related to earlier tabular context-free recognition methods. Relating our algorithm to its predecessors gives a new way of viewing the technique. The algorithmic description is then replaced by a description in terms of refined grammars. Finally we suggest how this notion might be generalized to the full class of Montague grammars.

The particular version of Montague grammar used here is that of PTQ, with which the reader is assumed to be conversant. The syntactic component of PTQ is an essentially context-free grammar, augmented by some additional rules of a different form. The non-

[1] A preliminary version of this paper was presented at the symposium on Modelling Human Parsing Strategies at the University of Texas at Austin, March 24-26, 1981. The work of the first author was supported in part by NSF grant IST 80-10834.



0362-613X/82/030123-16$03.00

context-free aspects arise in the treatment of quantifier scope and pronouns and their antecedents. Syntactically each antecedent is regarded as substituted into a place marked by a variable. This is not unlike the way fillers are inserted into gaps in Gazdar's 1979 treatment. However, Montague's use of variables allows complicated interactions between different variable-antecedent pairs. Each substitution rule substitutes a term phrase (NP) for one or more occurrences of a free variable in a phrase (which may be a sentence, common noun phrase, or intransitive verb phrase). The first occurrence of the variable is replaced by the phrase; later occurrences are replaced by appropriate pronouns. The translation of the resulting phrase expresses the coreferentiality of the noun phrase and the pronouns. With substitution, but without pronouns, the only function of substitution is to determine quantifier scope.

### Directed Process Approach

One computational approach to processing a sentence is the directed process approach, which is a sequential analysis that follows the three-part presentation in PTQ. The three steps are as follows. A purely syntactic analysis of a sentence yields a set of parse trees, each an expression in the disambiguated language. Each parse tree is then translated by Montague's rules into a formula of intentional logic to which logical reductions are immediately applied. The reduced formulas can then be interpreted in a model. The directed process approach is the one taken in the system described by Friedman, Moran, and Warren 1978a,b.

Semantic equivalence parsing is motivated by the observation that the directed process approach, in which all of the syntactic processing is completed before any semantic processing begins, does not take maximal advantage of the coupling of syntax and semantics in Montague grammars. Compositionality and the fact that for each syntactic rule there is a translation rule suggest that it would be possible to do a combined syntactic-semantic parse. In this approach, as soon as a subphrase is parsed, its logical formula is obtained and reduced to an extensionalized normal form. Two parses for the same phrase can then be regarded equivalent if they have the same formula.

The approach to parsing suggested by Cooper's 1975 treatment of quantified noun phrases is like our semantic equivalence parsing in storing translations as one element of the tuple corresponding to a noun phrase. Cooper's approach differs from the approach followed here because he has an intermediate stage that might be called an "autonomous syntax tree". The frontier of the tree is the sentence; the scope of the quantifier of a noun phrase is not yet indicated. Cooper's approach has been followed by the GPSG system (Gawron et al. 1982) and by Rosenschein and Shieber 1982. Neither of those systems treats pronouns. In Montague's approach, which we follow here, the trees produced by the parser are expressions in the disambiguated language, so scope is determined, pronoun antecedents are indicated, and each tree has a unique (unreduced) translation. The descriptions of the systems that use Cooper's approach seem to imply that they use a second pass over the syntax tree to determine the actual quantifier scopes in the final logical forms. Were these systems to use a single pass to produce the final logical forms, the results described in this paper would be directly applicable.

## 1. Equivalence Parsing

### Ambiguity

Ambiguity in Montague grammar is measured by the number of different meanings. In this view syntactic structure is of no interest in its own right, but only as a vehicle for mapping semantics. Syntactic ambiguity does not directly correspond to semantic ambiguity, and there may be many parses with the same semantic interpretation. Further, sentences with scope ambiguity, such as *A man loves every woman,* require more than one parse, because the syntactic derivation determines quantifier scope.

In PTQ there is infinite syntactic ambiguity arising from three sources: alphabetic variants of variables, variable for variable substitutions, and vacuous variable substitution. However, these semantically unnecessary constructs can be eliminated, so that the set of syntactic sources for any sentence is finite, and a parser that finds the full set is possible. (This corresponds to the "variable principle" enunciated by Janssen 1980 and used by Landsbergen 1980.) This approach was the basis of our earlier PTQ parser (Friedman and Warren 1978).

However, even with these reductions the number of remaining parses for a sentence of reasonable complexity is still large compared to the number of non-equivalent translations. In the directed process approach this is treated by first finding all the parses, next finding for each parse a reduced translation, and then finally obtaining the set of reduced translations. Each reduced translation may, but does not necessarily, represent a different sentence meaning. No meanings are lost. Further reductions of the set of translations would be possible, but the undecidability of logical equivalence precludes algorithmic reduction to a minimal set.

### The ATN Program

In the underlying parser the grammar is expressed as an augmented transition network (ATN) (Woods 1973). Both the syntactic and the semantic parsers use this same ATN. The main difficulty in construct-

ing the ATN was, as usual, the non-context-free aspects of the grammar, in particular the incorporation of a treatment of substitution rules and variables. The grammar given in PTQ generates infinitely many derivations for each sentence. All but finitely many of these are unnecessary variations on variables and were eliminated in the construction of the ATN. The ATN represents only the reduced set of structures, and must therefore be more complex.

### Equivalence Testing

In order to say what we mean by semantic equivalence parsing, we use Harel's 1979 notion of *execution method* for nondeterministic programs. An execution method is a deterministic procedure for finding the possible execution paths through a nondeterministic program given an input. For an ATN, these execution paths correspond to different parses. Viewing parsing in this way, the only difference between the usual syntactic parsing and semantic equivalence parsing is a difference in the execution method. As will be seen, semantic equivalence parsing uses semantic tests as part of the execution method.

We call the execution method we use to process a general ATN *equivalence parsing* (Warren 1979). Equivalence parsing is based on a recall table. The recall table is a set of buckets used to organize and hold partial syntactic structures while larger ones are constructed. Equivalence parsing can be viewed as processing an input sentence and the ATN to define and fill in the buckets of the recall table. The use of the recall table reduces the amount of redundant processing in parsing a sentence. Syntactic structures found along one execution path through the ATN need not be reconstructed but can be directly retrieved from the recall table and used on other paths. The recall table is a generalization of the familiar well-formed substring table (WFST) to arbitrary programs that contain procedure calls. Use of the WFST in ATN parsing is noted in Woods 1973 and Bates 1978. Bates observes that the WFST is complicated by the HOLDs and SENDRs in the ATN. These are the ATN actions that correspond to parameter passing in procedures and are required in the ATN for PTQ to correctly treat the substitution rules.

In the Woods system the WFST is viewed as a possible optimization, to be turned on when it improves parsing efficiency. In our system the recall table is an intrinsic part of the parsing algorithm. Because any ATN that naturally represents PTQ must contain left recursion, the usual depth-first (or breadth-first or best-first) ATN parsing algorithm would go into an infinite loop when trying to find all the parses of any sentence. The use of the recall table in equivalence parsing handles left-recursive ATNs without special consideration (Warren 1981). As a result there is no need to rewrite the grammar to eliminate left-recursive rules as is usually necessary.

In a general nondeterministic program, a bucket in the recall table corresponds to a particular subroutine and a set of values for the calling parameters and return parameters. For an ATN a bucket is indexed by a triple: (1) a grammatical category, that is, a subnet to which a PUSH is made, (2) the contents of the SENDR registers at the PUSH and the current string, and (3) the contents of the LIFTR registers at the POP and the then-current string. A bucket contains the members of an equivalence class of syntactic structures; precisely what they are depends on what type of equivalence is being used.

What makes equivalence parsing applicable to non-context-free grammars is that its buckets are more general than the cells in the standard tabular context-free algorithms. In the C-K-Y algorithm (Kasami 1965), for example, a cell is indexed only by the starting position and the length of the parsed segment, i.e., the current string at PUSH and POP. The cell contents are nonterminals. In our case all three are part of the bucket index, which also includes SENDR and LIFTR register values. The bucket contents are equivalence classes of structures.

### Sentence Recognition

For sentence recognition all parses are equivalent. So it is enough to determine, for each bucket of the recall table, whether or not it is empty. A sentence is in the language if the bucket corresponding to the sentence category (with empty SENDR registers and full string, and empty LIFTR registers and null string) is nonempty. The particular forms of the syntactic structures in the bucket are irrelevant; the contents of the buckets are only a superfluous record of the specific syntactic structures. The syntactic structure is never tested and so does not affect the flow of control. Thus which buckets are nonempty depends only on what other buckets are nonempty and not on what those other buckets contain. For sentence recognition, when the execution method constructs a new member of a bucket that is already nonempty, it may or may not add the new substructure, but it does *not* need to use it to construct any larger syntactic structures. This is because the earlier member has already verified this bucket as nonempty. Therefore this fact is already known and is already being used to determine the nonemptiness of other buckets. To find all parses, however, equivalence parsing *does* use all members of each bucket to construct larger structures.

It would be possible first to do recognition and determine all the nonempty buckets in the recall table, and then to go back and take all variants of one single parse that can be obtained by replacing any substructure by another substructure from the same bucket.

This is essentially how the context-free parsing algorithms constructed from the tabular recognition methods work. This is not how the equivalence parsing algorithm works. When it obtains a substructure, it immediately tries to use it to construct larger structures.

The difference described above between sentence recognition and sentence parsing is a difference only in the execution methods used to execute the ATN and not in the ATN itself. This difference is in the test for equivalence of bucket contents. In sentence recognition any two syntactic structures in a bucket are equivalent since we only care whether or not the substring can be parsed to the given category. At the other extreme, in finding all parses, two entries are equivalent only if they are the identical structure. For most reasonable ATNs, including our ATN for PTQ, this would not happen; distinct paths lead to distinct structures.

Semantic parsing is obtained by again modifying only the equivalence test used in the execution method to test bucket contents. For semantic parsing two entries are equivalent if their logical translations, after logical reduction and extensionalization, are identical to within change of bound variable.

### Small Grammar

For our examples, we introduce in Figure 1 a small subnet of the ATN for PTQ. Arcs with fully capitalized labels are PUSH arcs; those with lower case labels are CAT arcs. Structure-building operations are indicated in parentheses. This net implements just three rules of PTQ. Rule S4 forms a sentence by concatenating a term phrase and an intransitive verb phrase; S11 conjoins two sentences, and S14,i substitutes a term phrase for the syntactic variable $he_i$ in a sentence. S4 and S11 are context-free rules; S14,i is one of the substitution rules that make the grammar non-context-free and is basic to the handling of quantifiers, pronouns, and antecedents. The ATN handles the substitution by using a LIFTR to carry the variable-binding information. The LIFTR is not used for the context-free rules.

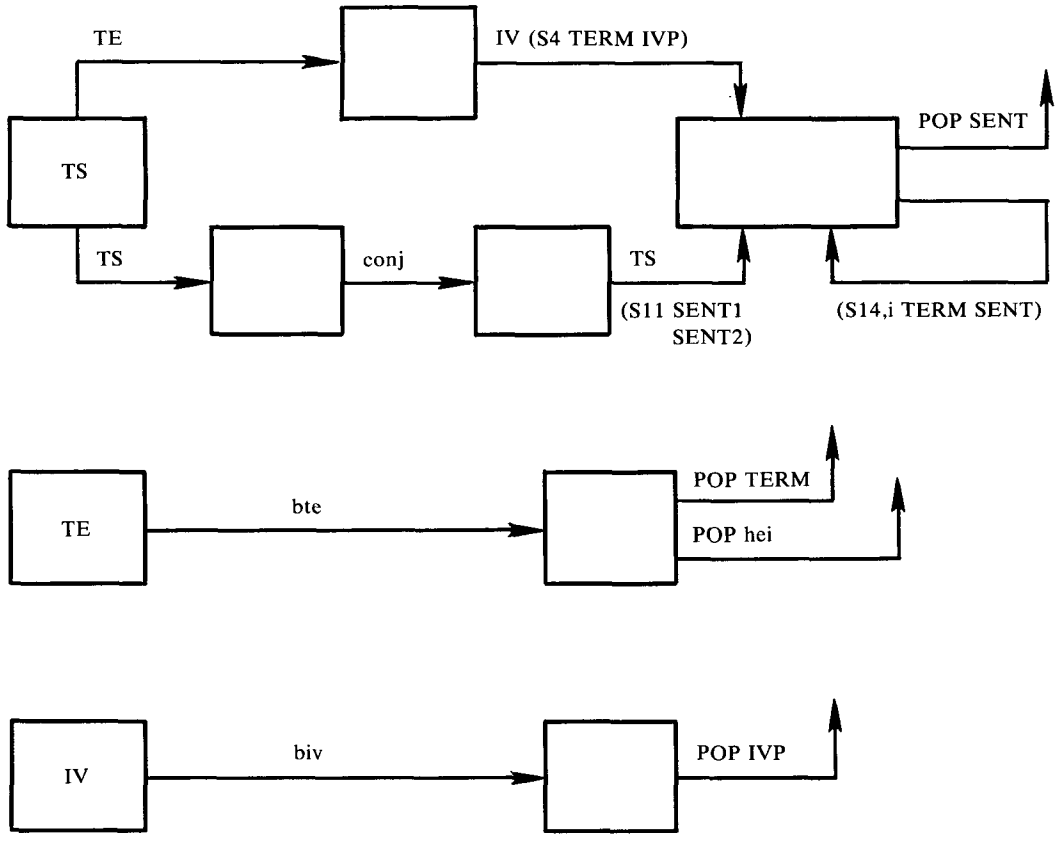


Figure 1. Subnet of the ATN for PTQ.

**Example 1:** *Bill walks*

The first example is the sentence *Bill walks.* This sentence has the obvious parse using only the context-free rule S4. It also has the parse using the substitution rule. We will carry through the details of its parse to show how this substitution rule is treated in the parsing process.

In the trace PUSHes and POPs in the syntactic analysis of this sentence are shown. The entries are in chronological order. The PUSHes are numbered sequentially for identification. The PUSH number uniquely determines a) the category to which the PUSH is made, b) the remainder of the sentence being parsed at the time of the PUSH, and c) the contents of the SENDR registers at the time of the PUSH, called the PUSH environment. At each POP a bucket and an element in that bucket are returned. The bucket name at a POP is made up of the corresponding PUSH number, the remaining input string, and the contents of the LIFTR registers, which are called the POP environment. The element in the bucket is the tree that is returned. For brevity we use in the trace only the first letters of the words in the sentence; for example, *Bill walks* becomes Bw.

**Trace of** *Bill walks*

| PUSH: | | | | Bucket: | | | Contents: |
|---|---|---|---|---|---|---|---|
| # | CAT | Str | Env | from | Str | Env | Tree |
| 1 | TS | Bw | null | | | | |

[Parsing begins with a PUSH to the sentence category passing the entire string and an empty or null environment.]

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 2 | TE | Bw | null | | | | |

[In the sentence subnet we first PUSH to find a TE.]

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | 2 | w | null | Bill |

[The TE subnet finds and POPs the term Bill to return from PUSH 2.]

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 3 | IV | w | null | | | | |
| | | | | 3 | $\epsilon$ | null | walk |
| | | | | 1 | $\epsilon$ | null | (S4 Bill walk) |

[Now since a tree is returned to the top level, covers the whole string, and the returned environment is null, the tree is printed. The parses are always the trees in bucket 1-$\epsilon$-null. The execution method now backs up; there are no more POPs from PUSH 3; there is another from PUSH 2.]

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | 2 | w | (he0 B) | he0 |

[Continuing forward with the new environment...]

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 4 | IV | w | (he0 B) | | | | |
| | | | | 4 | $\epsilon$ | null | walk |

[Note that this is not the same bucket as on the previous PUSH 3 because the PUSH environments differ.]

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | 1 | $\epsilon$ | (he0 B) | (S4 he0 walk) |

[The tree has been returned and covers the whole string. However, the returned environment is not null so the parse fails, and the execution method backs up to seek another return from PUSH 1.]

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | 1 | $\epsilon$ | null | (S14,0 Bill (S4 he0 walk)) |

[This is another element in bucket 1-$\epsilon$-null; it is a successful parse so it is printed out. Execution continues but there are no more parses of the sentence.]

**Discussion**

In this trace bucket 1-$\epsilon$-null is the only bucket with more than one entry. The execution method was *syntactic parsing,* so each of the two entries was returned and printed out. For *recognition,* these two entries in the bucket would be considered the same and the second would not have been POPped. Instead of continuing the computation up in the subnet from which the PUSH was made, this path would be made to fail and the execution method would back up. For *semantic equivalence parsing,* the bucket contents throughout would not be the syntax trees, but would instead be their reduced extensionalized logical formulas. (Each such logical formula represents the equivalence class of the syntactic structures that correspond to the formula.) For example, bucket 2-w-null would contain $\lambda PP\{\hat{}b\}$ and bucket 3-c-null would contain walk'. The first entry to bucket 1-$\epsilon$-null would be the formula for (S4 Bill walk), that is, $walk_*'(b)$. The entry to bucket 1-$\epsilon$-null on the last line of the trace

would be the formula for (S14,0 Bill (S4 he0 walk)), which is also $walk_*'(b)$. Therefore, this second entry would not be POPped.

Buckets also serve to reduce the amount of repeated computation. Suppose we have a second PUSH to the same category with the same string and environment as an earlier PUSH. The buckets resulting from this new PUSH would come out to be the same as the buckets from the earlier PUSH. Therefore the buckets need not be recomputed; the results of the earlier buckets can be used directly. This is called a "FAKEPUSH" because we don't actually do the PUSH to continue through the invoked subnet but simply do a "FAKEPOP" using the contents of the previously computed buckets.

Consider, as an example of FAKEPOP, the partial trace of the syntactic parse of the sentence *Bill walks and Mary runs* (or Bw&Mr for short). The initial part of this trace, through step 4, is essentially the same as the trace above for the shorter sentence *Bill walks.*

**Trace of** *Bill walks and Mary runs*

| PUSH: | | | | Bucket: | | | Contents: |
|---|---|---|---|---|---|---|---|
| # | CAT | Str | Env | from | Str | Env | Tree |
| 1 | TS | Bw&Mr | null | | | | |
| 2 | TE | Bw | null | | | | |
| | | | | 2 | w&Mr | null | Bill |
| 3 | IV | w | null | | | | |
| | | | | 3 | &Mr | null | walk |
| | | | | 1 | &Mr | null | (S4 Bill walk) |

[A tree has been returned to the top level, but it does not cover the whole sentence, so the path fails and the execution method backs up.]

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | 2 | w&Mr | (he0 B) | he0 |
| 4 | IV | w&Mr | (he0 B) | | | | |
| | | | | 4 | &Mr | null | walk |
| | | | | 1 | &Mr | (he0 B | (S4 he0 walk) |

[Again a tree has been returned to the top level, but it does not span the whole string, nor is the returned environment null, so we fail.]

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | 1 | &Mr | null | (S14,0 Bill (S4 he0 walk)) |

[Again we are the top level; again we do not span the whole string, so again we fail.]

| | | | |
|---|---|---|---|
| 5 | TS | Bw&Mr | null |

[This is the second arc from the TS node of Figure 1. The PUSH to TE (2 above) has completely failed. However, this PUSH, TS-Bw&Mr-null has been done before; it is PUSH 1. We already have two buckets from that PUSH: 1-&Mr-null containing two trees, and 1-&Mr-(he0 B) with one tree. There is no need to re-enter this subnet; the buckets and their contents tell us what would happen. Therefore we FAKEPOP one subtree and its bucket and follow that computation to the end; later we will return to FAKEPOP the next one.]

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | 1(5) | &Mr | null | (S4 Bill walk) |
| 6 | TS | Mr | null | | | | |
| 7 | TE | Mr | null | | | | |
| | | | | 7 | r | null | Mary |

[This computation continues and parses the second half of this sentence. Two parses are produced:
(S11 (S4 Bill walk) (S4 Mary run)) and
(S11 (S4 Bill walk) (S14,0 Mary (S4 he0 run)))
After this, the execution method fails back to the FAKEPOP at PUSH 5, and another subtree from a bucket from PUSH 2 is FAKEPOPped.]

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | 1(5) | &Mr | null | (S14,0 Bill (he0 walk)) |

[And the computation continues, eventually producing a total of ten parses for this sentence.]

(In the earlier example of *Bill walks,* these FAKEPOPs are done, but their computations immediately fail, because they are looking for a conjunction but are at the end of the sentence.)

### Results of Parsing

The sentence *Bill walks and Mary runs* has ten syntactic structures with respect to the PTQ grammar. The rules S4, S11, and S14,i can be used in various orders. Figure 2 shows the ten different structures in the order they are produced by the syntactic parser.

The nodes in the trees of Figure 2 that are in italics are the syntactic structures used for the first time. The nodes in standard type are structures used previously, and thus either are part of an execution path in common with an earlier parse, or are retrieved from a bucket in the recall table to be used again. Thus the number of italicized nodes measures in a crude way the amount of work required to find all the parses for this sentence.

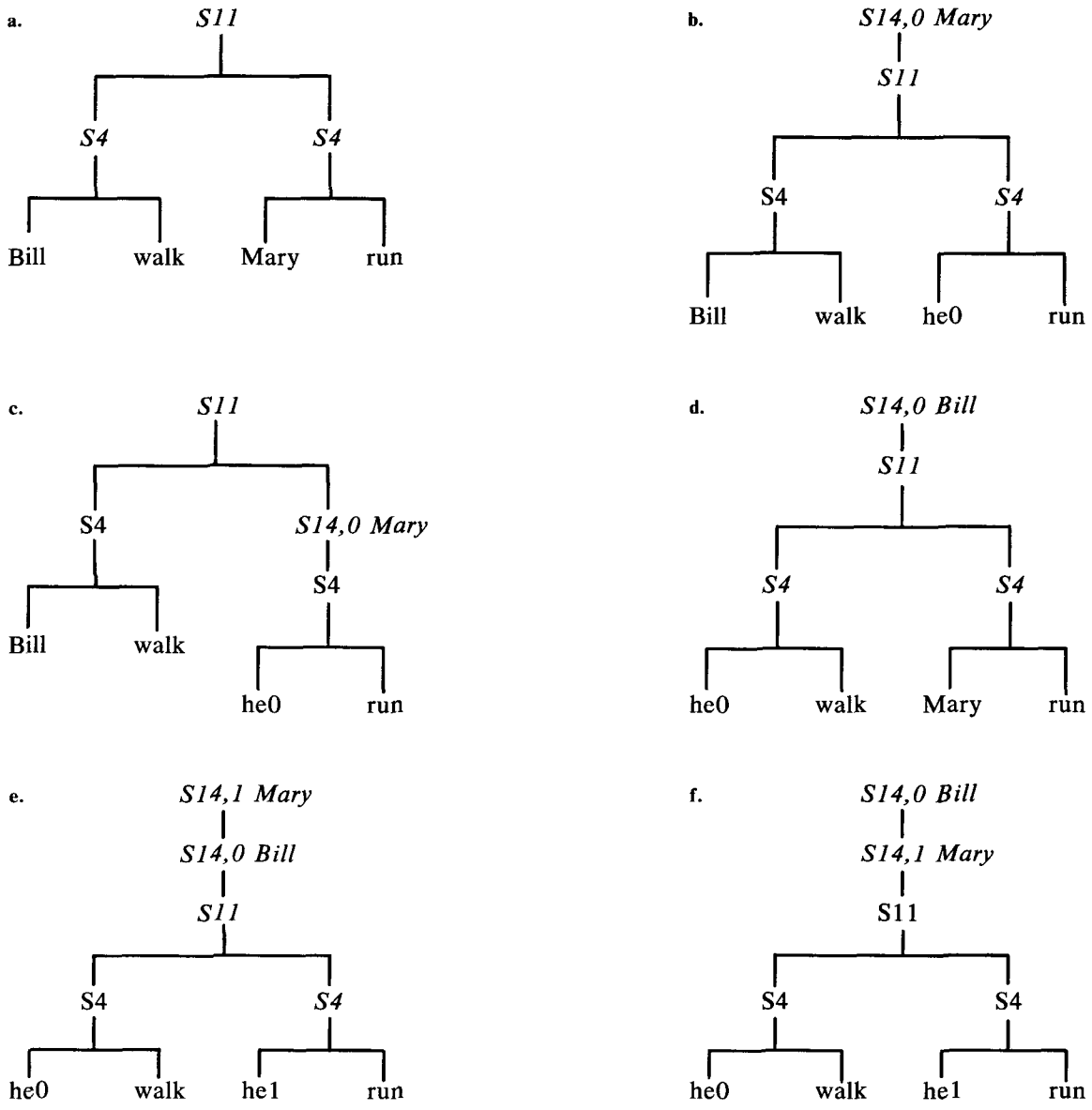


**Figure 2.** Ten parses for *Bill walks and Mary runs.*

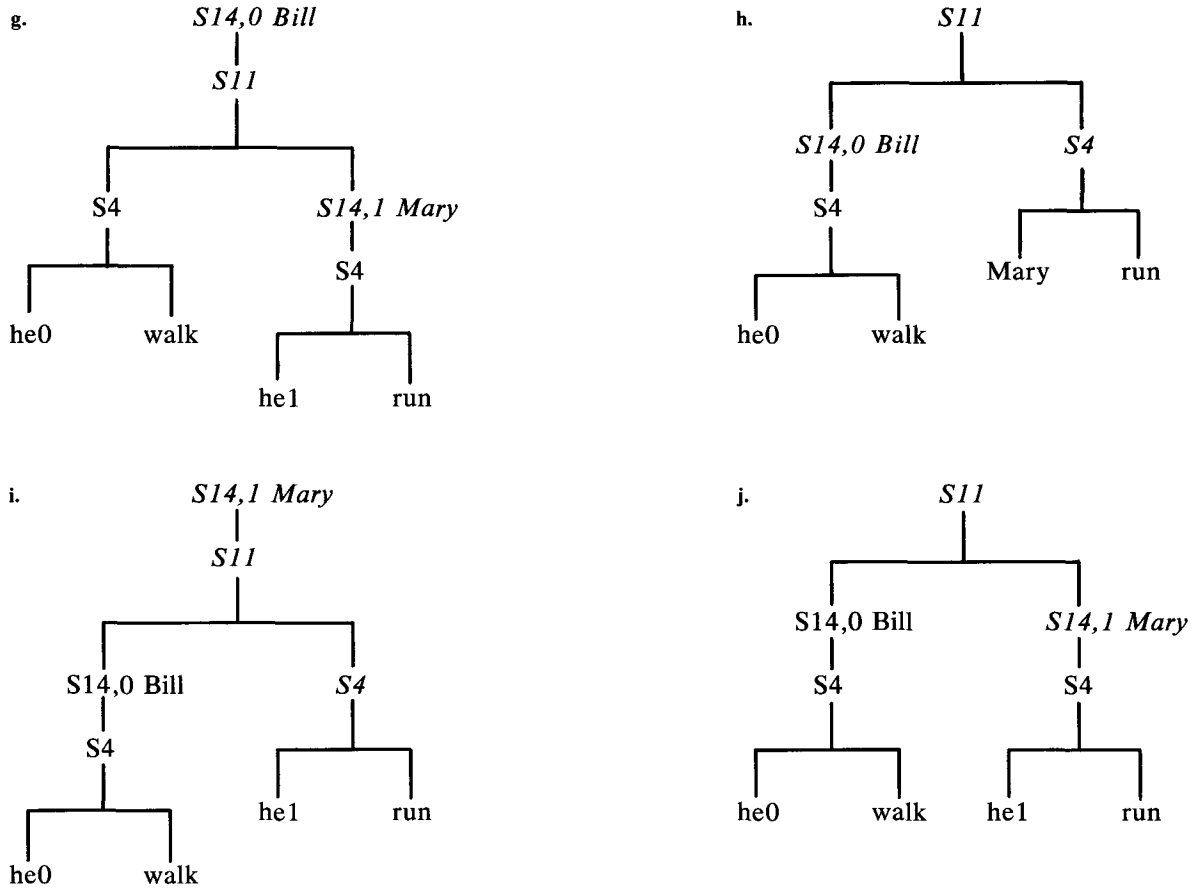


**Figure 2.** *continued*

### Example of Semantic Equivalence Parsing

This sentence, *Bill walks and Mary runs,* is one for which semantic parsing is substantially faster. It is unambiguous; its only reduced extensionalized logical translation is "walk$_*$'(b)&run$_*$'(m)". In the directed process parser, all ten trees of Figure 2 are found. They will all have the same translation. In semantic parsing only one is found. Here the method works to advantage because both parses of the initial string *Bill walks* result in the same environment for parsing *Mary runs.* These two parses go into the same bucket so only one needs to be used to construct larger structures. We trace the example.

| PUSH: | | | | Bucket: | | | Contents: | |
|---|---|---|---|---|---|---|---|---|
| # | CAT | Str | Env | from | Str | Env | Formula | ? |
| 1 | TS | Bw&Mr | null | | | | | |
| 2 | TE | Bw | null | | | | | |
| | | | | 2 | w&Mr | null | λPP{$^\wedge$b} | y |
| 3 | IV | w | null | | | | | |
| | | | | 3 | &Mr | null | walk' | y |
| | | | | 1 | &Mr | null | walk$_*$'(b) | y |
| [Fail] | | | | | | | | |
| | | | | 2 | w&Mr | (he0 B) | λPP{x0} | y |
| 4 | IV | w&Mr | (he0 B) | | | | | |
| | | | | 4 | &Mr | null | walk' | y |
| | | | | 1 | &Mr | (he0 B) | walk'($^\vee$x0) | y |
| [Fail] | | | | | | | | |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | 1 | &Mr | null | walk$_*$'(b) | n |

[This formula is the translation of the syntactic structure using S14,0 to substitute Bill into "he0 walks". This is the same bucket and the same translation as obtained at the return from 1 after PUSH 3 above, so we do not POP (indicated by the 'n' in the final column), but instead fail back.]

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 5 | TS | Bw&Mr | null | | | | | |

[FAKEPOP, since this is a repeat PUSH to this category with these parameters. There are two buckets: 1-&Mr-null, which in syntactic parsing had two trees but now has only one translation, and bucket 1-&Mr-(he0 B) with one translation. So we FAKEPOP 1-&Mr-null.]

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | (FAKEPOP) | 1(5) | &Mr | null | walk$_*$'(b) | y |
| 6 | TS | Mr | null | | | | | |
| 7 | TE | Mr | null | | | | | |
| | | | | 7 | r | null | Mary | y |
| 8 | IV | r | null | | | | | |
| | | | | 8 | $\epsilon$ | null | run' | y |
| | | | | 6 | $\epsilon$ | null | run$_*$'(m) | y |
| | | | | 1 | $\epsilon$ | null | walk$_*$'(b)&run$_*$'(m) | y |

[This is a successful parse. The top level prints out the translation and then the execution method fails back.]

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | 7 | (he0 M) | null | $\lambda$PP{x0} | y |
| 9 | IV | r | (he0 M) | | | | | |
| | | | | 9 | $\epsilon$ | null | run' | y |
| | | | | 6 | $\epsilon$ | null | run$_*$'($^\vee$x0) | y |
| | | | | 1 | $\epsilon$ | (he0 M) | walk$_*$'(b)&run$_*$'($^\vee$x0) | y |

[Fail because we are at the top level and the environment is not null.]

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | 1 | $\epsilon$ | null | walk$_*$'(b)&run$_*$'(m) | n |

[Again we want to enter a translation into bucket 1-$\epsilon$-null. This translation duplicates the one already there. So it is not returned and we fail back.]

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | 6 | $\epsilon$ | null | run$_*$'(m) | n |

[This again duplicates a bucket and its contents, so we fail back to the second FAKEPOP from PUSH 5. Now we use the other bucket: 1-&Mr-(he0 B).]

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | (FAKEPOP) | 1(5) | &Mr | (he0 B) | walk$_*$'($^\vee$x0) | y |
| 10 | TS | Mr | (he0 B) | | | | | |
| 11 | TE | Mr | (he0 B) | | | | | |
| | | | | 11 | r | null | $\lambda$P{x0} | y |
| 12 | IV | r | (he0 B) | | | | | |
| | | | | 12 | $\epsilon$ | null | run' | y |
| | | | | 10 | $\epsilon$ | null | run$_*$'(m) | y |
| | | | | 1 | $\epsilon$ | (he0 B) | walk$_*$'($^\vee$x0)&run$_*$'(m) | y |

[Fail at the top level since the environment is not null.]

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | 1 | $\epsilon$ | (he0 B) | walk$_*$'(b)&run$_*$'(m) | n |

[This duplicates a bucket and its contents, so we do not POP it but fail back.]

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | 11 | r | (he1 M) | $\lambda$PP{x1} | y |
| 13 | IV | r | (he0 B)<br>(he1 M) | | | | | |
| | | | | 13 | $\epsilon$ | null | run' | y |
| | | | | 10 | $\epsilon$ | (he1 M) | run$_*$'($^\vee$x1) | y |
| | | | | 1 | $\epsilon$ | (he0 B)<br>(he1 M) | walk$_*$'($^\vee$x0)&run$_*$'($^\vee$x1) | y |

[Fail at top level because environment is not null.]

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | 1 | $\epsilon$ | (he1 M) | walk$_*$'(b)&run$_*$'($^\vee$x1) | y |

[Fail at top level because environment is not null.]

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | 1 | $\epsilon$ | null | walk$_*$'(b)&run$_*$'(m) | n |

[Duplicate bucket and translation, so fail.]

| | | | | |
|---|---|---|---|---|
| 1 | ∈ | (he0 B) | $walk_*'(^vx0)\&run_*'(m)$ | n |

[Duplicate bucket and translation, so fail.]

| | | | | |
|---|---|---|---|---|
| 1 | ∈ | null | $walk_*'(b)\&run_*'(m)$ | n |

[Duplicate, so fail.]

| | | | | |
|---|---|---|---|---|
| 10 | ∈ | null | $run_*'(m)$ | n |

[Duplicate, so fail.]

This completes the trace of the semantic parse of the sentence.

### Results of Parsing

Figure 3 displays in graphical form the syntactic structures built during the semantic parsing of *Bill walks and Mary runs* traced above. A horizontal line over a particular node in a tree indicates that the translation of the structure duplicated a translation already in its bucket, so no larger structures were built using it. Only parse a) is a full parse of the sentence and thus it is the only parse returned. All the others are aborted when they are found equivalent to earlier partial results. These points of abortion in the computation are the points in the trace above at which a POP fails due to the duplication of a bucket and its contents.

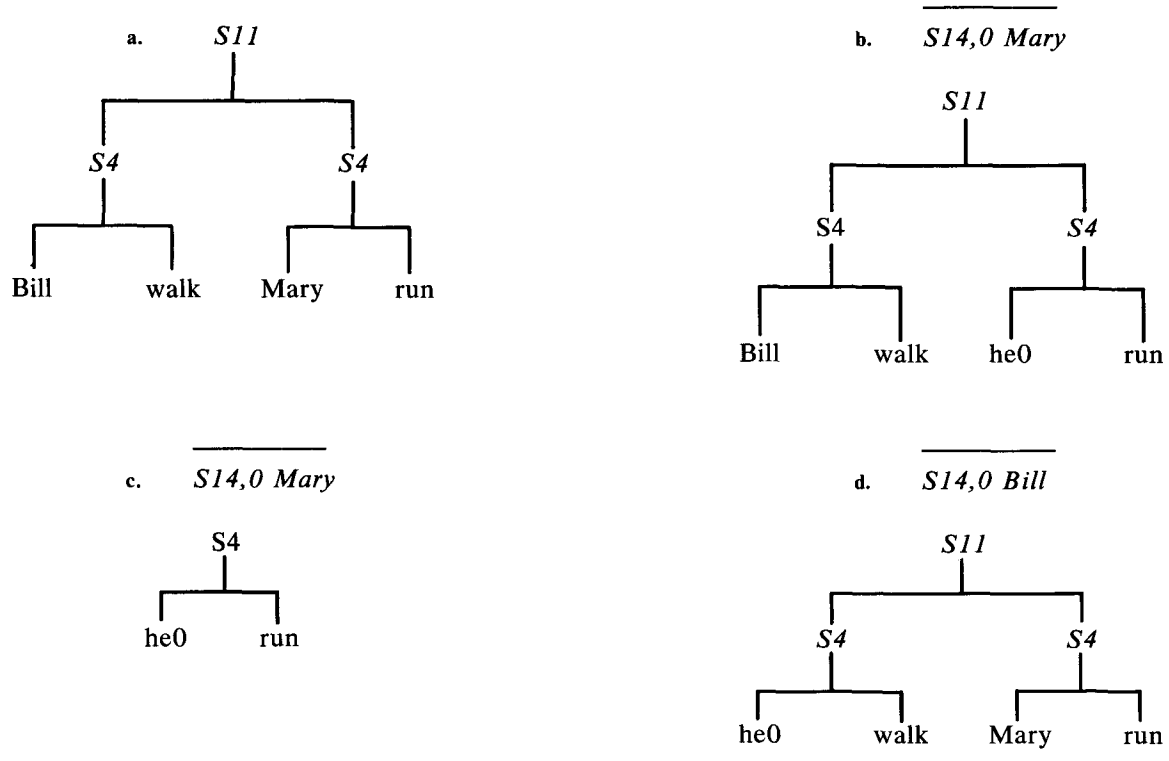


**Figure 3.** Semantic parses of *Bill walks and Mary runs.*

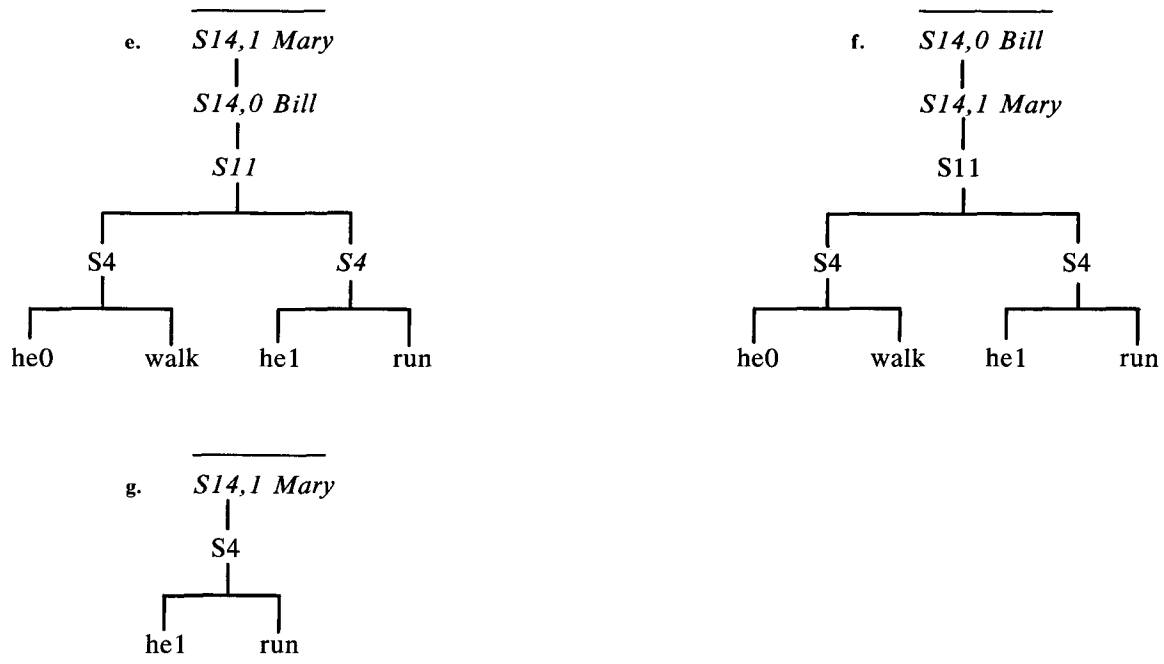


**Figure 3.** *continued*

Note that construction of parse c) is halted when a translation is built that duplicates the translation of the right S4 subtree of parse a). This corresponds to the failure due to duplicate bucket contents in bucket 6-$\epsilon$-null following PUSH 9 in the trace above. Similarly parse g) is aborted before the entire tree is built. This corresponds to the failure in the final line of the trace due to a duplicate translation in bucket 10-$\epsilon$-null. Semantic parses that would correspond to syntactic parses h), i), and j) of Figure 2 are not considered at all. This is because bucket 1-&Mr-null contains two syntactic structures, but only one translation. Thus in semantic equivalence parsing we only do one FAKEPOP for this bucket for PUSH 5. In syntactic parsing the other parses are generated by the FAKEPOP of the other structure in this bucket.

### Reducing the Environment

The potential advantage of semantic equivalence parsing derives from treating partial results as an equivalence class in proceeding. A partial result consists of a structure, its extensionalized reduced translation, and a set of parameters of the parse to that point. These parameters are the environment for parsing the phrase. Consider the sentence *John loves Mary* and its parses:

(1) (S4 John (S5 love Mary))
(2) (S4 John (S16,0 Mary (S5 love he0)))
(3) (S14,0 John (S4 (he0 (S5 love Mary)))
(4) (S14,0 John (S4 he0 (S16,1 Mary
(S5 love he1))))
(plus 3 more)

On reaching the phrase *love Mary* in parse (3) the parameters are *not* the same as they were at that point in parse (1), because the pair (he0 John) is in the environment. Thus the parser is not able to consult the recall table and immediately return the already parsed substructure. Instead it must reparse *love Mary* in the new context.

This environment problem arises because the ATN is designed to follow PTQ in treating pronouns by the non-context-free substitution rules. We have also considered, but have not to this point implemented, alternative ways of treating variables to make partial results equal. One way would be not to pass variable bindings down into lower nets at all. Thus the PUSH environment would always be null. Since these bindings are used to find the antecedent for a pronoun, the way antecedents are determined would have to be changed. An implementation might be as follows: On encountering a pronoun during parsing, replace it by a new he-variable. Then pass back *up* the tree information concerning both the variable number used and the pronoun's gender. At a higher point in the tree, where the substitution rule is to be applied, a determination can be made as to which of the substituted terms could be the antecedent for the pronoun. The variable number of the pronoun can then be changed to agree with the variable number of its antecedent term by a variable-for-variable substitution. Finally the substitution rule can be used to substitute the term into the phrase for all occurrences of the variable. Note that this alternative process would construct trees that do have substitution rules to substitute variables for varia-

bles, contrary to the variable principle mentioned above. We also note that with this modification a pronoun is *not* associated with its antecedent when it is first encountered. Instead the pronoun is saved and at some later point in the parse the association is made. This revised treatment is related computationally to that proposed in Cooper 1975.

### Evaluation of Semantic Equivalence Parsing

The question of the interaction of syntax and semantics in parsing was introduced early in computational linguistics. Winograd 1971 argued for the incorporation of semantics as early as possible in the recognition process, in order to reduce the amount of syntactic processing that would be needed. Partial parses that had no interpretation did not need to be continued. The alternative position represented by Woods's early work (Woods and Kaplan 1971) was basically the inverse: less semantic processing would be needed if only completed parses were interpreted. This argument is based on the idea of eliminating uninterpretable parses as soon as possible.

This advantage, if it is one, of integrated syntactic and semantic procedures does not occur here because the semantic aspect does not eliminate any logical analyses. The translation of a structure to a formula is always successful, so no partial parse is ever eliminated for lack of a translation. What happens instead is that several partial parses are found to be equivalent because they have the same translation. In this case only a representative of the set of partial parses needs to be carried forward.

A further expansion of equivalence parsing would be *interpretation equivalence parsing*. Sentence processing would take place in the context of a specified model. Two structures would be regarded as equivalent if they had the same denotation in the model. More partial structures would be found equivalent under the equivalence relation than under the reduce-extensionalize relation, and fewer structures would need to be constructed. Further, with the interpretation equivalence relation, we might be able to use an inconsistent denotation to eliminate an incorrect partial parse. For example, consider a sentence such as *Sandy and Pat are running and she is talking to him.* In this case, since the gender of Sandy and Pat cannot be determined syntactically, these words would have to be marked in the lexicon with both genders. This would result in multiple logical formulas for this sentence, one for each gender assumption. However, during interpretation equivalence parsing, the referents for *Sandy* and *Pat* would be found in the model and the meaning with the incorrect coreference could be rejected.

Logical normal forms other than the reduced, extensionalized form used above lead to other reasonable versions of equivalence parsing. For example, we could further process the reduced, extensionalized form to obtain a prenex normal form with the matrix in clausal form. We would use some standard conventions for naming variables, ordering sequences of the same quantifier in the prefix, and ordering the literals in the clauses of the matrix. This would allow the algorithm to eliminate, for example, multiple parses arising from various equivalent scopes and orderings of existential quantifiers.

The semantic equivalence processor has been implemented in Franz Lisp. We have applied it to the PTQ grammar and tested it on various examples. For purposes of comparison the directed process version includes syntactic parse, translation to logical formula and reduction, and finally the reduction of the list of formulas to a set of formulas. The mixed strategy yields exactly this set of formulas, with one parse tree for each. Experiments with the combined parser and the directed parser show that they take approximately the same time for reasonably simple sentences. For more complicated sentences the mixed strategy usually results in less processing time and, in the best cases, results in about a 40 percent speed-up. The distinguishing characteristic of a string for which the method yields the greatest speed-up is that the environment resulting from parsing an initial segment is the same for several distinct parses.

The two parsing method we have described, the sequential process and the mixed process, were obviously not developed with psychological modeling in mind. The directed process version of the system can be immediately rejected as a possible psychological model, since it involves obtaining and storing all the structures for a sentence before beginning to interpret any one of them. However, a reorganization of the program would make it possible to interpret each structure immediately after it is obtained. This would have the same cost in time as the first version, but would not require storing all the parses.

Although semantic equivalence parsing was developed in the specific context of the grammar of PTQ, it is more general in its applicability. The strict compositionality of syntax and semantics in PTQ is the main feature on which it depends. The general idea of equivalence parsing can be applied whenever syntactic structure is used as an intermediate form and there is a syntax-directed translation to an output form on which an equivalence relation is defined.

## 2. Input-Refined Grammars

We now switch our point of view and examine equivalence parsing not in algorithmic terms but in formal grammatical terms. This will then lead into showing how equivalence parsing relates to *Universal*

*Grammar* (UG) (Montague 1970). The basic concept to be used is an *input-refined grammar*. We begin by defining this concept for context-free grammars and using it to relate the tabular context-free recognition algorithms of Earley 1970, Cocke-Kasami-Younger (Kasami 1965), and Sheil 1976 to each other and eventually to our algorithm.

Given a context-free grammar G and a string s over the terminal symbols of G, we define from G and s a new grammar $G_s$, called an *input-refinement of G*. This new grammar $G_s$ will bear a particular relationship to G: $L(G_s) = \{s\} \cap L(G)$, i.e., $L(G_s)$ is the singleton set {s} if s is in L(G), and empty otherwise. Furthermore, there is a direct one-to-one relationship between the derivations of s in G and the derivations of s in $G_s$. Thus the problem of recognizing s in G is reduced to the problem of determining emptiness for the grammar $G_s$. Also, the problem of parsing s with respect to the grammar G reduces to the problem of exhaustive generation of the derivations of $G_s$ (there is at most one string). Each of the tabular context-free recognition algorithms can be viewed as implicitly defining this grammar $G_s$ and testing it for emptiness. Emptiness testing is essentially done by reducing the grammar, that is by eliminating useless symbols and productions. The table-constructing portion of a tabular recognition algorithm, in effect, constructs and reduces the grammar $G_s$, thus determining whether or not it is empty. The tabular methods differ in the construction and reduction algorithm used.

In each case, to turn a tabular recognition method into a parsing algorithm, the table must first be constructed and then reprocessed to generate all the parses. This corresponds to reprocessing the grammar $G_s'$, the result of reducing the grammar $G_s$, and using it to exhaustively generate all derivations in $G_s$.

Rather than formally defining $G_s$ from a context-free grammar G and a string s in the general case, we illustrate the definition by example. The general definition should be clear.

Let G be the following context-free grammar:

Terminals: {a,b}
Nonterminals: {S}
Start Symbol: S
Productions: $S \rightarrow S\ S\ a$
$S \rightarrow b$
$S \rightarrow \epsilon$
(S produces the empty string)

Let s be the string bba. $G_{bba}$ is defined from G and bba:

Terminals: {a,b}
Nonterminals: $\{a_1,a_2,a_3,b_1,b_2,b_3,$
($t_i$ for t a terminal of G and $1 \le i \le$ length(s))
$S_{123},S_{12},S_1,S_{23},S_2,S_3,$
($A_x$ for each nonterminal A of G and each x a nonempty subsequence of <1,2,3,...,length(s)>)
$S^0,S^1,S^2,S^3\}$
($A^i$ for each nonterminal A of G and i, $0 \le i \le$ length(s))
Start Symbol: $S_{123}$
Productions: [from G production: $S \rightarrow S\ S\ a$]
$S_{123} \rightarrow S_{12}S^2a_3$
$S_{123} \rightarrow S_1S_2a_3$
$S_{123} \rightarrow S^0S_{12}a_3$
$S_{12} \rightarrow S_1S^1a_2$
$S_{12} \rightarrow S^0S_1a_2$
$S_1 \rightarrow S^0S^0a_1$
$S_{23} \rightarrow S_2S^2a_3$
$S_{23} \rightarrow S^1S_2a_3$
$S_2 \rightarrow S^1S^1a_2$
$S_3 \rightarrow S^2S^2a_3$
[from G production: $S \rightarrow b$]
$S_1 \rightarrow b_1$
$S_2 \rightarrow b_2$
$S_3 \rightarrow 3$
[from G production: $S \rightarrow \epsilon$]
$S^0 \rightarrow \epsilon$
$S^1 \rightarrow \epsilon$
$S^2 \rightarrow \epsilon$
$S^3 \rightarrow \epsilon$
[for the terminals]
$b_1 \rightarrow b$
$b_2 \rightarrow b$
$a_3 \rightarrow a$

These productions for $G_s$ were constructed by beginning with a production of G, adding a subscript or a superscript to the nonterminal on the LHS to obtain a nonterminal of $G_s$, adding single subscripts to all terminals and sequence subscripts to some nonterminals on the RHS so that the concatenation of all subscripts on the RHS equals the subscript on the LHS. For the RHS nonterminals without subscripts, add the appropriate subscript. Also, to handle the terminals, for each $t_i$ add the production $T_i \rightarrow t$ where t is the $i^{th}$ symbol in s.

It is straightforward to show inductively that if a nonterminal symbol generates any string at all it generates exactly the substring of s that its subscript determines. Symbols with superscripts generate the empty string. Also a parse tree of $G_s$ can be converted to a parse tree of G by first deleting all terminals (each is dominated by the same symbol with a subscript) and then erasing all superscripts and subscripts on all symbols in the tree. Conversely, any parse tree for s in G can be converted to a parse tree of s in $G_s$ by adding appropriate subscripts and superscripts to all the symbols of the tree and then adding the terminal symbols at the leaves.

It is clear that $G_s$ is not in general a reduced grammar. $G_s$ can be reduced to $G_s'$ by eliminating unproductive and unreachable symbols and the rules involving them. Reducing the grammar will determine whether or not $L(G_s)$ is empty. By the above discussion, this will determine whether s is in L(G), and thus an algorithm for constructing and reducing the refined grammar $G_s$ from G and s yields a recognition algorithm. Also, given the reduced grammar $G_s'$, it is straightforward, in light of the above discussion, to generate all parses of s in G: simply exhaustively generate the parse trees of $G_s'$ and delete subscripts and superscripts.

The tabular context-free recognition methods of Cocke-Kasami-Younger, Earley, and Sheil can all be understood as variations of this general approach. The C-K-Y recognition algorithm uses the standard bottom-up method to determine emptiness of $G_s$. It starts with the terminals and determines which $G_s$ nonterminals are productive, eventually finding whether or not the start symbol is productive. The matrix it constructs is essentially the set of productive nonterminals of $G_s$.

Sheil's well-formed substring table algorithm is the most obviously and directly related. His simplest algorithm constructs the refined grammar and reduces it top-down. It uses a top-down control mechanism to determine the productivity only of nonterminals that are reachable from the start symbol. The well-formed substring table again consists essentially of the reachable, productive nonterminals of $G_s$.

Earley's recognition algorithm is more complicated because it simultaneously constructs and reduces the refined grammar. It can be viewed as manipulating sets of subscripted nonterminals and sets of productions of $G_s$. The items on the item lists, however, correspond quite directly to reachable, productive nonterminals of $G_s$.

The concept of input-refined grammar provides a unified view of the tabular context-free recognition methods. Equivalence parsing as described in Part I above is also a tabular method, although it is not context-free. It applies to context-free grammars and also to some grammars such as PTQ that are not context-free. We next relate it to the very general class of grammars defined by Montague in UG.

### Universal Grammar and Equivalence Parsing

In the following discussion of the problem of parsing in the general context of Montague's definitions of a language (which might more naturally be called a grammar) and an interpretation, we assume the reader is familiar with the definitions in UG (Montague 1970). We begin with a formal definition of a refinement of a general disambiguated language. A particular type of refinement, input-refinement, leads to an equivalence parsing algorithm. This generalizes the procedure for input-refining a grammar shown above for the special case of a context-free grammar. We then discuss the implications for equivalence parsing of using the formal interpretation of the language. Finally we show how the ATN for PTQ and semantic equivalence parsing fit into this general framework.

Recall that a *disambiguated language* $\Omega = \langle A, F_\gamma, X_\delta, S, \delta_0\rangle_{\gamma\in\Gamma,\delta\in\Delta}$ can be regarded as consisting of an algebra $\langle A,F_\gamma\rangle_{\gamma\in\Gamma}$, with proper expressions A and operations $F_\gamma$, basic expressions $X_\delta$ for each category index $\delta\in\Delta$, a set of syntactic rules S, and a sentence category index $\delta_0\in\Delta$. A *language* is a pair $\langle\Omega,R\rangle$ where $\Omega$ is a disambiguated language and R is a binary relation with domain included in A. Given a disambiguated language

$$\Omega = \langle A, F_\gamma, X_\delta, S, \delta_0\rangle_{\delta\in\Gamma,\ \delta\in\Delta},$$

a disambiguated language

$$\Omega' = \langle A, F_\gamma, X'_{\delta'}, S', \delta_0'\rangle_{\gamma\in\Gamma,\ \delta'\in\Delta'}$$

is a *refinement of* $\Omega$ if there is a refinement function $d:\Delta'\to\Delta$ from the category indices of $\Omega'$ to those of $\Omega$ such that

1) $X'_\delta \subseteq X_{d(\delta')}$,
2) If $\langle F_\gamma,\langle\delta_1',\delta_2',\ldots,\delta_n'\rangle,\delta'\rangle \in S'$, then $\langle F_\gamma,\langle d(\delta_1'),d(\delta_2'),\ldots,d(\delta_n')\rangle, d(\delta')\rangle \in S'$, and
3) $d(\delta_0') = \delta_0$.

(Note that the proper expressions A, the operation indexing set $\Gamma$, and the operations $F_\gamma$ of $\Omega$ and $\Omega'$ are the same.)

The word *refinement* refers to the fact that the catgories of $\Omega$ are split into finer categories. Condition 1 requires that the basic expressions of a refined category come from the basic expressions of the category it refines. Condition 2 requires that the new syntactic rules be consistent with the old ones. Note that Condition 2 is *not* a biconditional.

If $\Omega'$ is a refinement of $\Omega$ with refinement function d, $\langle C'_{\delta'}\rangle_{\delta'\in\Delta'}$ is the family of syntactic categories of $\Omega'$ and $\langle C_\delta\rangle_{\delta\in\Delta}$ is the family of syntactic categories of $\Omega$, then $C'_{\delta'}\subseteq C_{d(\delta')}$.

As a simple example of a refinement, consider an arbitrary disambiguated language $\Omega' = \langle A, F_\gamma, X'_{\delta'}, \delta_0'\rangle_{\gamma\in\Gamma,\delta'\in\Delta'}$. Now let $\Omega$ be the disambiguated language $\langle A, F_\gamma, X_a, S, a\rangle_{\gamma\in\Gamma}$, in which the set of category names is the singleton set $\{a\}$. $X_a = \cup_{\delta'\in\Delta'} X_{\delta'}$. Let S be $\{\langle F_\gamma, \langle a,a,\ldots,a\rangle, a\rangle : \gamma\in\Gamma$ and the number of a's agrees with the arity of F$\}$. Then $\Omega'$ is a refinement of $\Omega$, with refinement function $d:\Delta'\to\{a\}$, $d(\delta') = a$ for all $\delta'\in\Delta'$. Note that the disambiguated language $\Omega$ is completely determined by the algebra $\langle A,F_\gamma\rangle_{\gamma\in\Gamma}$, and is the natural disambiguated language to associate with it. Thus in a formal sense, we can view a disambiguated language as a refinement of its algebra.

As a more intuitive example of refinement, consider an English-like language with categories term (TE) and intransitive verb phrase (IV) that both include singular and plural forms. The language generated would then allow subject-verb disagreement (assuming the ambiguating relation R does not filter them out). By refining category TE to $TE_{sing}$ and $TE_{pl}$ and category IV to $IV_{sing}$ and $IV_{pl}$, and having syntactic rules that combine category $TE_{sing}$ with $IV_{sing}$ and $TE_{pl}$ with $IV_{pl}$ only, we obtain a refined language that has subject-verb agreement. A similar kind of refinement could eliminate such combinations as "colorless green ideas", if so desired.

With this definition of refinement, we return to the problem of parsing a language $L = <\Omega, R>$. The problem can now be restated: find an algorithm that, given a string $\xi$, constructs a disambiguated language $\Omega_\xi$ that is an input-refinement of $\Omega$. That is, $\Omega_\xi$ is a refinement in which the sentence category $C'_{\delta}$ is exactly the set of parses of $\xi$ in L. Finding this algorithm is equivalent to solving the parsing problem. For given such an algorithm, the parsing problem reduces to the problem of generating all members of $C'_{\delta_0}$.

In the case of a general language $<\Omega, R>$, it may be the case that for $\xi$ a string, the input-refined language $\Omega_\xi$ has finitely many categories. In this case the reduced grammar can be computed and a recursive parsing algorithm exists. If the reduced grammar has infinitely many categories, then the string has infinitely many parses and we are not, in general, interested in trying to parse such languages. It may happen, however, that $\Omega_\xi$ has infinitely many categories, even though its reduction has only finitely many. In this case, we are not guaranteed a recursive parsing algorithm. However, if this reduced language can be effectively constructed, a recursive parsing algorithm still exists.

The ATN for PTQ represents the disambiguated language for PTQ in the UG sense. The categories of this disambiguated language correspond to the set of possible triples: PTQ category name, contents of SENDR registers at a PUSH to that subnet, contents of the LIFTR registers at the corresponding POP. The input-refined categories include the remainder of the input string at the PUSH and POP. Thus the buckets in the recall table are exactly the input-refined categories. The syntactic execution method is thus an exhaustive generation of all expressions in the sentence category of the input-refined disambiguated language.

### Semantic Equivalence Parsing in UG

In UG, Montague inclues a theory of meaning by providing a definition of interpretation for a language. Let $L = <<A,F_\gamma,X_\delta,S,\delta_0>_{\gamma\in\Gamma,\delta\in\Delta},R>$ be a language. An *interpretation* $\Psi$ for L is a system $<B,G_\gamma,f>_{\gamma\in\Gamma}$ such that $<B,G_\gamma>_{\gamma\in\Gamma}$ is an algebra similar to $<A,F_\gamma>_{\gamma\in\Gamma}$; i.e., for each $\gamma \in \Gamma$, $F_\gamma$ and $G_\gamma$ have the same number of arguments, and f is a function from $\cup_{\delta\in\Delta}X_\delta$ into B. Note that the algebra $<B,G_\gamma>_{\gamma\in\Gamma}$ need *not* be a free algebra (even though $<A,F_\gamma>_{\gamma\in\Gamma}$ must be). B is the set of meanings of the interpretation $\Psi$; $G_\gamma$ is the semantic rule corresponding to syntactic rule $F_\gamma$; f assigns meanings to the basic expressions $X_\gamma$. The meaning assignment for L determined by $\Psi$ is the unique homomorphism g from $<A,F_\gamma>_{\gamma\in\Gamma}$ into $<B,G_\gamma>_{\gamma\in\Gamma}$ that is an extension of f.

There are two ways to proceed in order to find all the meanings of a sentence $\xi$ in a language $L = <\Omega, R>$ with interpretation $\xi$. The first method is to generate all members of the sentence category $C'_{\delta_0}$ of the input-refined language $\Omega_\xi$. As discussed above, this is done in the algebra $<A,F_\gamma>_{\gamma\in\Gamma}$ of $\Omega_\xi$, using the syntactic functions $F_\gamma$ to inductively construct members of A from the basic categories of $\Omega_\xi$ and members of A constructed earlier and then applying g. The second method is to use the fact that g is a homomorphism from $<A,F_\gamma>_{\gamma\in\Gamma}$ into $<B,G_\gamma>_{\gamma\in\Gamma}$. Because g is a homomorphism, we can carry out the construction of the image of the sentence category entirely in the algebra $<B,G_\gamma>_{\gamma\in\Gamma}$ of the interpretation $\Psi$. We may use the G functions to construct inductively members of B from the basic semantic categories, that is, the images under g (and f) of the basic syntactic categories, and members of B already constructed. The advantage of carrying out the construction in the algebra of $\Psi$ is that this algebra may not be free, i.e., some element of B may have multiple construction sequences. By carrying out the construction there, such instances can be noticed and used to advantage, thus eliminating some redundant search. There are additional costs, however, associated with parsing in the interpretation algebra $\Psi$. Usually, the cost of evaluating a G function in the semantic algebra is greater than the cost of the corresponding F function in the syntactic algebra. Also in semantic parsing, each member of B as it is constructed is compared to the other members of the same refined category that were previously constructed.

In the PTQ parsing system discussed above, the interpretation algebra is the set of reduced translations. The semantic functions are those obtained from the functions given in the T-rules in PTQ, and reducing and extensionalizing their results. The directed process version of the parser finds the meanings in this algebra by the first method, generating all parses in the syntactic algebra and then taking their images under the interpretation homomorphism. Semantic equivalence parsing for PTQ uses the second method, carrying out the construction of the meaning entirely within the semantic algebra. The savings in the example sentence *Bill walks and Mary runs* comes about

because the algebra of reduced translations is not a free algebra, and the redundant search thus eliminated more than made up for the increase in the cost of translating and comparing formulas.

## Summary

We have described a parsing algorithm for the language of PTQ viewed as consisting of two parts, a nondeterministic program and an execution method. We showed how, with only a change to an equivalence relation used in the execution method, the parser becomes a recognizer. We then discussed the addition of the semantic component of PTQ to the parser. With again only a change to the equivalence relation of the execution method, the semantic parser is obtained. The semantic equivalence relation is equality (to within change of bound variable) of reduced extensionalized translations. Examples were given to compare the two parsing methods.

In the final portion of the paper we described how the parsing method initially presented in procedural terms can be viewed in formal grammatical terms. The notion of input-refinement for context-free grammars was introduced by example, and the tabular context-free recognition algorithms were described in these terms. We then indicated how this notion of refinement can be extended to the UG theory of language and suggested how our semantic parser is essentially parsing in the algebra of an interpretation for the PTQ language.

## References


Bates, Madeleine 1978 The theory and practise of augmented transition network grammars. In Bole, Ed., *Natural Lanauge Communication with Computers.* New York: 191-260.

Cooper, R. 1975 Montague's semantic theory and transformational syntax. Ph.D. thesis. Amherst, MA: University of Massachusetts.

Earley, Jay 1970 An efficient context-free parsing algorithm. *Comm. ACM* **13,** 94-102.

Friedman, J., Moran, D., and Warren, D.S. 1978a Evaluating English sentences in a logical model. Abstract 16, *Information Abstracts, 7th International Conference on Computational Linguistics.* Norway: University of Bergen (11 pp.).

Friedman, J., Moran, D., and Warren, D.S. 1978b Evaluating English sentences in a logical model, presented to the 7th International Conference on Computation Linguistics, University of Bergen, Norway (August 14-18). Report N-15. Ann Arbor, MI: University of Michigan, Computer and Communication Sciences Department (mimeographed).

Friedman, J. and Warren, D.S. 1978 A parsing method for Montague grammars. *Lingustics and Philosophy* **2,** 347-372.

Gawron, J.M., *et al.* 1982 The GPSG linguistic system. In *Proceedings 20th Annual Meeting of the Association for Computational Linguistics,*74-81.

Gazdar, G. 1979 English as a context-free language University of Sussex (mimeograph).

Harel, David 1979 On the total correctness of nondeterministic programs. IBM Research Report RC 7691.

Hintikka, J., Moravcsik, J., and Suppes, P., Eds. 1973 *Approaches to Natural Language.* Dordrecht: D. Reidel.

Janssen, T.W.V. 1978 Compositionality and the form of rules in Montague grammar. In Groenenijk, J. and Stokhof, M., Eds., *Proceedings of the Second Amsterdam Colloquium on Montague Grammar and Related Topics.* Amsterdam Papers in Formal Grammar, Volume II. University of Amsterdam, 211-234.

Janssen, T.W.V. 1980 On problems concerning the quantification rules in Montague grammar. In Roher, G., Ed., *Time, Tense, and Quantifiers.* Tuebingen, Max Niemeyer Verlag.

Kasami, T. 1965 An efficient recognition and syntax-analysis algorithm for context-free languages. Science Report AFCRL-65-758. Bedford, MA: Air Force Cambridge Research Laboratory.

Landsbergen, S.P.J. 1980 Adaptation of Montague grammar to the requirements of parsing. M.S. 11.646. Eindhoven, The Netherlands: Philips Research Laboratories.

Montague, Richard 1970 Universal grammar (UG). *Theoria* **36,** 373-398.

Montague, Richard 1973 The proper treatment of quantification in ordinary English. In Hintikka, Moravcsik, and Suppes 1973. Reprinted in Montague 1974, 247-270.

Montague, Richard 1974 *Formal Philosophy: Selected Papers of Richard Montague.* Edited and with an introduction by Richmond Thomason. New Haven, CT: Yale University Press.

Rosenschein, S.J. and Shieber, S.M. 1982 Translating English into logical form. In *Proceedings 20th Annual Meeting of the Association for Computational Linguistics,* 1-8.

Sheil, B.A. 1976 Observations on context-free parsing. *Statistical Methods in Linguistics* 71-109.

Warren, David S. 1979 Syntax and semantics in parsing: an application to Montague grammar. Ph.D. thesis. Ann Arbor, MI: University of Michigan.

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

Woods, W.A. and Kaplan, R.M. 1971 The Lunar Sciences Natural Language Information System. BBN Report No. 2265. Cambridge, MA Bolt Beranek and Newman.

Woods, W.A. 1973 An experimental parsing system for transition network grammars. In Rustin, R., Ed., *Natural Language Processing.* New York: Algorithmics Press, Inc., 111-154.