Translating English into logical form

20th Meeting of the ACL, 1983.

TRANSLATING ENGLISH INTO LOGICAL FORM*

Stanley J. Rosenschein
Stuart M. Shieber

Artificial Intelligence Center
SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025

ABSTRACT

A scheme for syntax-directed translation that mirrors compositional model-theoretic semantics is discussed. The scheme is the basis for an English translation system called PATR and was used to specify a semantically interesting fragment of English, including such constructs as tense, aspect, modals, and various lexically controlled verb complement structures. PATR was embedded in a question-answering system that replied appropriately to questions requiring the computation of logical entailments.

I INTRODUCTION

When contemporary linguists and philosophers speak of "semantics," they usually mean model-theoretic semantics—mathematical devices for associating truth conditions with sentences. Computational linguists, on the other hand, often use the term "semantics" to denote a phase of processing in which a data structure (e.g., a formula or network) is constructed to represent the meaning of a sentence and serve as input to later phases of processing. (A better name for this process might be "translation" or "transduction.") Whether one takes "semantics" to be about model theory or translation, the fact remains that natural languages are marked by a wealth of complex constructions—such as tense, aspect, moods, plurals, modality, adverbials, degree terms, and sentential complements—that make semantic specification a complex and challenging endeavor.

Computer scientists faced with the problem of managing software complexity have developed strict design disciplines in their programming methodologies. One might speculate that a similar requirement for manageability has led linguists (since Montague, at least) to follow a discipline of strict compositionality in semantic specification, even though model-theoretic semantics *per se* does not demand it. Compositionality requires that the meaning of a phrase be a function of the meanings of its immediate constituents, a property that allows the grammar writer to correlate syntax and semantics on a rule-by-rule basis and keep the specification modular. Clearly, the natural analogue to compositionality in the case of translation is *syntax-directed translation;* it is this analogy that we seek to exploit.

We describe a syntax-directed translation scheme that bears a close resemblance to model-theoretic approaches and achieves a level of perspicuity suitable for the development of large and complex grammars by using a declarative format for specifying grammar rules. In our formalism, translation types are associated with the phrasal categories of English in much the way that logical-denotation types are associated with phrasal categories in model-theoretic semantics. The translation types are classes of *data objects* rather than abstract *denotations,* yet they play much the same role in the translation process that denotation types play in formal semantics.

In addition to this parallel between logical types and translation types, we have intentionally designed the language in which translation rules are stated to emphasize parallels between the syntax-directed translation and corresponding model-theoretic interpretation rules found in, say, the GPSG literature [Gazdar, forthcoming]. In the GPSG approach, each syntax rule has an associated semantic rule (typically involving functional application) that specifies how to compose the meaning of a phrase from the meanings of its constituents. In an analogous fashion, we provide for the translation of a phrase to be synthesized from the translations of its immediate constituents according to a local rule, typically involving *symbolic application* and λ-conversion.

It should be noted in passing that doing translation rather than model theoretic interpretation offers the temptation to abuse the formalism by having the "meaning" (translation) of a phrase depend on *syntactic* properties of the translations of its constituents—for instance, on the order of conjuncts in a logical expression. There are several points to be made in this regard. First, without severe *a priori* restrictions on what kinds of objects can *be* translations (coupled with the associated strong theoretical claims that such restrictions would embody) it seems impossible to prevent such abuses. Second, as in the case of programming languages, it is reasonable to assume that there would emerge a set of stylistic practices that would govern the actual form of grammars for reasons of manageability and esthetics. Third, it is still an open question whether the model-theoretic program of strong compositionality will actually succeed. Indeed, whether it succeeds or not is of little concern to the computational linguist, whose systems, in any event, have no direct way of using the sort of abstract model being proposed and whose systems must, in general, be based on deduction (and hence translation).

The rest of the paper discusses our work in more detail. Section II presents the grammar formalism and describes PATR, an implemented parsing and translation system that can accept a grammar in our formalism and uses it to process sentences. Examples of the system's operation, including its application in a simple deductive question-answering system, are found in Section III. Finally, Section IV describes further extensions of the formalism and the parsing system. Three appendices are included: the first contains sample grammar rules; the second contains meaning postulates (axioms) used by the question-answering system; the third presents a sample dialogue session.

*This research was supported by the Defense Advanced Research Projects Agency under Contract N00039-80-C-0575 with the Naval Electronic Systems Command. The views and conclusions contained in this document are those of the authors and should not be interpreted as representative of the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the United States Government.

II A GRAMMAR FORMALISM

A General Characterization

Our grammar formalism is best characterized as a specialized type of augmented context-free grammar. That is, we take a grammar to be a set of context-free rules that define a language and associate structural descriptions (parse trees) for each sentence in that language in the usual way. Nodes in the parse tree are assumed to have a set of features which may assume binary values (*True* or *False*), and there is a distinguished attribute—the "translation"—whose values range over a potentially infinite set of objects, i.e., the translations of English phrases.

Viewed more abstractly, we regard translation as a binary relation between word sequences and logical formulas. The use of a relation is intended to incorporate the fact that many word sequences have several logical forms, while some have none at all. Furthermore, we view this relation as being composed (in the mathematical sense) of four simpler relations corresponding to the conceptual phases of analysis: (1) LEX (lexical analysis), (2) PARSE (parsing), (3) ANNOTATE (assignment of attribute values, syntactic filtering), and (4) TRANSLATE (translation proper, i.e., synthesis of logical form).

The domains and ranges of these relations are as follows:

Word Sequences $-LEX\rightarrow$
Morpheme Sequences $-PARSE\rightarrow$
Phrase Structure Trees $-ANNOTATE\rightarrow$
Annotated Trees $-TRANSLATE\rightarrow$
Logical Form

The relational composition of these four relations is the full translation relation associating word sequences with logical forms. The subphases too are viewed as *relations* to reflect the inherent nondeterminism of each stage of the process. For example, the sentence "a hat by every designer sent from Paris was felt" is easily seen to be nondeterministic in LEX ("felt"), PARSE (postnominal modifier attachment), and TRANSLATE (quantifier scoping).

It should be emphasized that the correspondence between *processing* phases and these conceptual phases is loose. The goal of the separation is to make specification of the process perspicous and to allow simple, clean implementations. An actual system could achieve the net effect of the various stages in many ways, and numerous optimizations could be envisioned that would have the effect of folding back later phases to increase efficiency.

B The Relations LEX, PARSE, and ANNOTATE

We now describe a characteristic form of specification appropriate to each phase and illustrate how the word sequence "John went" is analyzed by stages as standing in the translation relation to "(past (go john))" according to the (trivial) grammar presented in Figure 1.

```
RULES:
  Constant COMP' = (λ P (λ Q (λ X (P (Q X)))))
  S → NP VP
    Trans: VP'[NP']
  VP → TENSE V
    Anno: [ ¬Transitive(V) ]
    Trans: { COMP'[TENSE'][V'] }

LEXICON:
  NP → John
    Anno: [ Proper(NP) ]
    Trans: { John }
  TENSE → &past
    Trans: { (λ X (past X)) }
  V → go
    Anno: [ ¬Transitive(V) ]
    Trans: { (λ X (go X)) }
```

Figure 1: Sample specification of augmented phrase structure grammar

Lexical analysis is specified by giving a kernel relation between individual words and morpheme sequences[1] (or equivalently, a mapping from words to sets of morpheme sequences), for example:

```
John → (john);
went → (&past go);
persuaded → (&past persuade),
            (&ppl persuade);
```

The kernel relation is extended in a standard fashion to the full LEX relation. For example, "went" is mapped onto the single morpheme sequence (&past go), and "John" is mapped to (john). Thus, by extension, "John went" is transformed to (John &past go) by the lexical analysis phase.

Parsing is specified in the usual manner by a context-free grammar. Utilizing the context-free rules presented in the sample system specification shown in Figure 1, (John &past go) is transformed into the parse tree

```
(S (NP john)
  (VP (TENSE &past)
    (V go)))
```

Every node in the parse tree has a set of associated features. The purpose of ANNOTATE is to relate the bare parse tree to one that has been enhanced with attribute values, filtering out those that do not satisfy stated syntactic restrictions. These restrictions are given as Boolean expressions associated with the context-free rules; a tree is properly annotated only if all the Boolean expressions corresponding to the rules used in the analysis are simultaneously true. Again, using the rules of Figure 1,

[1] Of course, more sophisticated approaches to morphological analysis would seek to analyze the LEX relation more fully. See, for example, [Kartunnen, 1982] and [Kaplan, 1981].

```
(S (NP john)
  (VP (TENSE &past)
    (V go)))
```

is transformed into

```
(S (NP: Proper
     john)
  (VP: ¬Transitive
    (TENSE &past)
    (V: ¬Transitive
      go)))
```

C The Relation TRANSLATE

Logical-form synthesis rules are specified as augments to the context-free grammar. There is a language whose expressions denote translations (syntactic formulas); an expression from this language is attached to each context-free rule and serves to define the composite translation at a node in terms of the translations of its immediate constituents. In the sample sentence, TENSE' and V' (the translations of TENSE and V respectively) would denote the λ-expressions specified in their respective translation rules. VP' (the translation of the VP) is defined to be the value of (SAP (SAP COMP' TENSE') V'), where COMP' is a constant λ-expression and SAP is the *symbolic-application operator*. This works out to be (λ X (past (go X))). Finally, the symbolic application of VP' to NP' yields (past (go John)). (For convenience we shall henceforth use square brackets for SAP and designate (SAP α β) by $\alpha[\beta]$.)

Before describing the symbolic-application operator in more detail, it is necessary to explain the exact nature of the data objects serving as translations. At one level, it is convenient to think of the translations as λ-expressions, since λ-expressions are a convenient notation for specifying how fragments of a translation are substituted into their appropriate operator-operand positions in the formula being assembled–especially when the composition rules follow the syntactic structure as encoded in the parse tree. There are several phenomena, however, that require the storage of more information at a node than can be represented in a bare λ-expression. Two of the most conspicuous phenonema of this type are quantifier scoping and unbounded dependencies ("gaps").

Our approach to quantifier scoping has been to take a version of Cooper's storage technique, originally proposed in the context of model-theoretic semantics, [Cooper, forthcoming] and adapt it to the needs of translation. For the time being, let us take translations to be ordered pairs whose first component (the *head*) is an expression in the target language, characteristically a λ-expression. The second component of the pair is an object called *storage*, a structured collection of sentential operators that can be applied to a sentence matrix in such a way as to introduce a quantifier and "capture" a free variable occurring in that sentence matrix.[2]

For example, the translation of "a happy man" might be < m , (λ S (some m (and (man m)(happy m)) S)) >.[3] Here the head is *m* (simply a free variable), and storage consists of the λ-expression (λ S ...). If the verb phrase "sleeps" were to receive the translation < (λ X (sleep X)), ϕ > (i.e., a unary predicate as head and no storage), then the symbolic application of the verb phrase translation to the noun phrase translation would compose the heads in the usual way and take the "union" of the storage yielding < (sleep m), (λ S (some m (and (man m)(happy m)) S)) >.

We define an operation called "pull.s," which has the effect of "pulling" the sentence operator out of storage and applying it to the head. There is another pull operation, pull.v, which operates on heads representing unary predicates rather than sentence matrices. When pull.s is applied in our example, it yields < (some m (and (man m)(happy m)) (sleep m)), ϕ >, corresponding to the translation of the clause "a happy man sleeps." Note that in the process the free variable m has been "captured." In model-theoretic semantics this capture would ordinarily be meaningless, although one can complicate the mathematical machinery to achieve the same effect. Since *translation* is fundamentally a *syntactic* process, however, this operation is well-defined and quite natural.

To handle gaps, we enriched the translations with a third component: a variable corresponding to the gapped position. For example, the translation of the relative clause "...[that] the man saw" would be a triple: < (past (see X Y)), Y, (λ S (the X (man X) S))>, where the second component, Y, tracks the free variable corresponding to the gap. At the node at which the gap was to be discharged, λ-abstraction would occur (as specified in the grammar by the operation "ungap") producing the unary predicate (λ Y (past (see X Y))), which would ultimately be applied to the variable corresponding to the head of the noun phrase.

It turns out that triples consisting of <head, var, storage> are adequate to serve as translations of a large class of phrases, but that the application operator needs to distinguish two subcases (which we call type A and type B objects). Until now we have been discussing type A objects, whose application rule is given (roughly) as

$$\langle hd,var,sto\rangle[\langle hd',var',sto'\rangle] = \langle (hd\ hd'),var \cup var',\ sto \cup sto'\rangle \quad ,$$

where one of var or var' must be null. In the case of type B objects, which are assigned primarily as translations of determiners, the rule is

$$\langle hd,var,sto\rangle[\langle hd',var',sto'\rangle] = \langle var,\ var',\ hd(hd') \cup sto \cup sto'\rangle \quad .$$

For example, if the meaning of "every" is

$$every' = \langle(\lambda\ P\ (\lambda\ S\ (every\ X\ (P\ X)\ S))),\ X,\ \phi\rangle$$

and the meaning of "man" is

$$man' = \langle\ man,\ \phi,\ \phi\ \rangle \quad ,$$

then the meaning of "every man" is

$$every'[man'] = \langle\ X\ ,\ \phi,\ (\lambda\ S\ (man\ X)\ S)\rangle \quad ,$$

as expected.

Nondeterminism enters in two ways. First, since pull operations can be invoked nondeterministically at various nodes in the parse tree (as specified by the grammar), there exists the possibility of computing multiple scopings for a single context-free parse tree. (See Section III.B for an example of this phenomenon.) In addition, the grammar writer can specify explicit nondeterminism by associating several distinct translation rules with a single context-free production. In this case, he can control the application of a translation schema by

[2] In the sample grammar presented in Appendix A, the storage-forming operation is notated *mk.mbd*.

[3] Following [Moore, 1980], a quantified expression is of the form (*quantifier, variable, restriction, body*)

specifying for each schema a *guard*, a Boolean combination of features that the nodes analyzed by the production must satisfy in order for the translation schema to be applicable.

D Implementation of a Translation System

The techniques presented in Sections II.B and II.C were implemented in a parsing and translation system called PATR which was used as a component in a dialogue system discussed in Section III.B. The input to the system is a sentence, which is preprocessed by a lexical analyzer. Parsing is performed by a simple recursive descent parser, augmented to add annotations to the nodes of the parse tree. Translation is then done in a separate pass over the annotated parse tree. Thus the four conceptual phases are implemented as three actual processing phases. This folding of two phases into one was done purely for reasons of efficiency and has no effect on the actual results obtained by the system. Functions to perform the storage manipulation, gap handling, and the other features of translation presented earlier have all been realized in the translation component of the running system. The next section describes an actual grammar that has been used in conjunction with this translation system.

III EXPERIMENTS IN PRODUCING AND USING LOGICAL FORM

A A Working Grammar

To illustrate the ease with which diverse semantic features could be handled, a grammar was written that defines a semantically interesting fragment of English along with its translation into logical form [Moore, 1981]. The grammar for the fragment illustrated in this dialogue is compact occupying only a few pages, yet it gives both syntax and semantics for modals, tense, aspect, passives, and lexically controlled infinitival complements. (A portion of the grammar is included as Appendix A.)[4] The full test grammar, loosely based on DIAGRAM [Robinson, 1982] but restricted and modified to reflect changes in approach, was the grammar used to specify the translations of the sentences in the sample dialogue of Appendix C.

B An Example of the System's Operation

The grammar presented in Appendix A encodes a relation between sentences and expressions in logical form. We now present a sample of this relation, as well as its derivation, with a sample sentence: "Every man persuaded a woman to go."

Lexical analysis relates the sample sentence to two morpheme streams:

```
▸ every man &ppl persuade a woman to go
▸ every man &past persuade a woman to go.
```

The first is immediately eliminated because there is no context-free parse for it in the grammar. The second, however, is parsed as

```
[S (SDEC (NP (DETP (DDET (DET every)))
             (NOM (NOMHD (NOUN (N man)))))
         (PREDICATE (AUXP (TENSE &past))
                    (VPP (VP (VPT (V persuade)))
                             (NP (DETP (A a))
                                 (NOM (NOMHD (NOUN (N woman)))))
                             (INFINITIVE (TO to)
                                         (VPP (VP (VPT (V go]    .
```

While parsing is being done, annotations are added to each node of the parse tree. For instance, the NP → DETP NOM rule includes the annotation rule AGREE(NP, DETP, Definite). AGREE is one of a set of macros defined for the convenience of the grammar writer. This particular macro invocation is equivalent to the Boolean expression Definite(NP) ⇔ Definite(DETP). Since the DETP node itself has the annotation Definite as a result of the preceding annotation process, the NP node now gets the annotation Definite as well. At the bottom level, the Definite annotation was derived from the lexical entry for the word "every".[5] The whole parse tree receives the following annotation:

```
[S (SDEC (NP: Definite
           (DETP: Definite
             (DDET: Definite
               (DET: Definite
                 every)))
           (NOM (NOMHD (NOUN (N man)))))
         (PREDICATE (AUXP (TENSE &past))
                    (VPP (VP: Active
                           (VPT: Active,Transitive,TakesInf
                             (V: Active,Transitive,TakesInf
                               persuade)))
                         (NP (DETP (A a))
                           (NOM (NOMHD (NOUN (N woman)))))
                         (INFINITIVE (TO to)
                           (VPP (VP: Active
                                  (VPT: Active
                                    (V: Active
                                      go]    .
```

Finally, the entire annotated parse tree is traversed to assign translations to the nodes through a direct implementation of the process described in Section II.C. (Type A and B objects in the following examples are marked with a prefix 'A:' or 'B:'.) For instance, the VP node covering (persuade a woman to go), has the translation rule VPT'[NP'][INFINITIVE']. When this is applied to the translations of the node's constituents, we have

```
<A: (λ X (λ P (λ Y (persuade Y X (P X)))>
  [<A: X2, φ, (λ S (some X2 (woman X2) S))>]
    [<A: (λ X (go X))>]
```

which, after the appropriate applications are performed, yields

```
<A: (λ P (λ Y (persuade Y X2 (P X2)))), φ,
  (λ S (some X2 (woman X2) S))>
```

[4] Since this is just a small portion of the actual grammar selected for expository purposes, many of the phrasal categories and annotations will seem unmotivated and needlessly complex. These categories and annotations are utilized elsewhere in the test grammar.

[5] Note that, although the annotation phase was described and is implemented procedurally, the process actually used guarantees that the resulting annotation is exactly the one specified declaratively by the annotation rules.

```
[<A: (λ X (go X))>]

== <A: (λ Y (persuade Y X2 (go X2))), φ,
        (λ S (some X2 (woman X2) S))>    .
```

After the past operator has been applied, we have

```
<A: (λ Y (past (persuade Y X2 (go X2)))), φ,
  (λ S (some X2 (woman X2) S))>    .
```

At this point, the pull operator (pull.v) can be used to bring the quantifier out of storage, yielding[6]

```
<A: (λ Y (some X2 (woman X2) (past (persuade Y X2 (go X2))))),
  φ, φ>    .
```

This will ultimately result in "a woman" getting narrow scope. The other alternative is for the quantifier to remain in storage, to be pulled only at the full sentence level, resulting in the other scoping. In Figure 2, we have added the translations to all the nodes of the parse tree. Nodes with the same translations as their parents were left unmarked. From examination of the S node translations, the original sentence is given the fully-scoped translations

```
(every X2 (man X2)
    (some X1 (woman X1) (past (persuade X2 X1 (go X1)))))
```

and

```
(some X1 (woman X1)
    (every X2 (man X2) (past (persuade X2 X1 (go X1)))))    .
```

C A Simple Question-Answering System

As mentioned in Section I, we were able to demonstrate the semantic capabilities of our language system by assembling a small question-answering system. Our strategy was to first translate English into logical formulas of the type discussed in [Moore, 1981], which were then postprocessed into a form suitable for a first-order deduction system.[7] (Another possible approach would have been to translate directly into first-order logic, or to develop direct proof procedures for the non-first-order language.) Thus, we were able to integrate all the components into a question-answering system by providing a simple control structure that accepted an input, translated it into logical form, reduced the translation to first-order logic, and then either asserted the translation in the case of declarative sentences or attempted to prove it in the case of interrogatives. (Only yes/no questions have been implemented.)

The main point of interest is that our question-answering system was able to handle complex semantic entailments involving tense, modality, and so on—that, moreover, it was not restricted to extensional evaluation in a data base, as with conventional question-answering systems. For example, our system was able to handle the entailments of sentences like

▸ John could not have been persuaded to go.

(The transcript of a sample dialogue is included as Appendix C.)

[6] For convenience, when a final constituent of a translation is ϕ it is often not written. Thus we could have written <A: (λ Y (some ...) ...)> in this case.

[7] We used a connection graph theorem prover written by Mark Stickel [Stickel, forthcoming].

```
(S: <A: (past (persuade X1 X2 (go X2))), φ,
    (λ S (every X1 (man X1) S))
    (λ S (some X2 (woman X2) S))>,
  <A: (some X2 (woman X2) (past (persuade X1 X2 (go X2)))), φ,
    (λ S (every X1 (man X1) S))>
  <A: (every X2 (man X2)
       (some X1 (woman X1) (past (persuade X2 X1 (go X2)))))>
  <A: (some X1 (woman X1)
       (every X2 (man X2) (past (persuade X2 X1 (go X2)))))>
 (SDEC
  (NP: <A: X1, φ, (λ S (every X1 (man X1) S))>
   (DETP: <B: (λ P (λ S (every X (P X) S))),
           X>
    (DDET (DET every)))
   (NOM: <A: (λ X (man X))>
    (NOMHD (NOUN (N man)))))
  (PREDICATE: <A: (λ X (past (persuade Y X2 (go X2)))), φ,
           (λ S (some X2 (woman X2) S))>,
         <A: (λ X (some X2 (woman X2)
                  (past (persuade Y X2 (go X2))))),
           φ, φ>
   (AUXP: <A: (λ P (λ X (past (P X))))>
    (TENSE &past))
   (VPP: <A: (λ Y (persuade Y X2 (go X2))), φ,
         (λ S (some X2 (woman X2) S))>
    (VP (VPT: <A: (λ X
                (λ P
                  (λ Y (persuade Y X (P Y))))>
          (V persuade)))
     (NP: <A: X2, φ, (λ S (some X2 (woman X2) S))>
      (DETP: <B: (λ P (λ S (some X (P X) S))),
              X>
       (A a))
      (NOM: <A: (λ X (woman X))>
       (NOMHD (NOUN (N woman)))))
     (INFINITIVE (TO: none
                  to)
      (VPP: <A: (λ X (go X))>
       (VP (VPT (V go]
```

Figure 2: Node-by-node translation of a sample sentence

The reduction of logical form to first-order logic (FOL) was parameterized by a set of recursive expansions for the syntactic elements of logical form in a manner similar to Moore's use of an axiomatization of a modal language of belief. [Moore, 1980[For example, (past P) is expanded, with respect to a possible world w, as

(some w2 (and (past w2 w) <P,w2>))

where "<P,w2>" denotes the recursive FOL reduction of P relative to the world w2. The logical form that was derived for the sample sentence "John went" therefore reduces to the first-order sentence

(some w (and (past w REALWORLD)(go w John))).

More complicated illustrations of the results of translation and reduction are shown in Figure 3. Note, for example, the use of restricted quantification in LF and ordinary quantification in FOL.

To compute the correct semantic entailments, the deduction system was preloaded with a set of meaning postulates (axioms) giving inferential substance to the predicates associated with lexical items (see

```
INPUT: every man must be happy
LF:  (every X (man X)
       (necessary (and (happy X)
                       (thing X))))
FOL: (every x0172
       (implies (man REALWORLD x0172)
         (every w0173
           (implies (poss REALWORLD w0173)
             (and (happy w0173 x0172)
               (thing w0173 x0172))))))

INPUT: bill persuaded john to go
LF:  (past (persuade bill john (go john)))
FOL: (some w0175
       (and (past w0175 REALWORLD)
         (some w0176
           (and (persuade w0175 bill john w0176)
             (go w0176 john)))))
```

Figure 3: Translation to LF and Reduction to FOL

Appendix B).

IV FURTHER EXTENSIONS

We are continuing to refine the grammar formalism and improve the implementation. Some of the refinements are intended to make the annotations and translations easier to write. Examples include:

- Allowing nonbinary features, including sets of values, in the annotations and guards (extending the language to include equality and set operations).
- Generalizing the language used to specify synthesis of logical forms and developing a more uniform treatment of translation types.
- Generalizing the "gap" variable feature to handle arbitrary collections of designated variables by using an "environment" mechanism. This is useful in achieving a uniform treatment of free word order in verb complements and modifiers.

In addition, we are working on extensions of the syntactic machinery, including phrase-linking grammars to handle displacement phenomena [Peters, 1981], and methods for generating the augmented phrase structure grammar through a metarule formalism similar to that of [Konolige, 1980]. We have also experimented with alternative parsing algorithms, including a chart parser [Bear, 1979] adapted to carry out annotation and translation in the manner described in this paper.

REFERENCES

Bear, John, and Lauri Karttunen. PSG: A Simple Phrase Structure Parser. *Texas Linguistic Forum*, vol. 14. 1979.

Cooper, Robin. *Quantification and Syntactic Theory.* Forthcoming. Reidel, Dordrecht.

Gazdar, Gerald. Phrase Structure Grammar. To appear in Jacobson, O. and G. K. Pullum (eds.) *On the Nature of Syntactic Representation.*

Kaplan, R. M., and Martin Kay. Personal communication. 1981.

Karttunen, Lauri, Rebecca Root, and Hans Uszkoreit. Morphological analysis of Finnish by computer. Paper presented at the ACL session of the 1981 LSA Annual Meeting, New York, December 1981.

Konolige, Kurt. Capturing linguistic generalizations with metarules in an annotated phrase-structure grammar. Proceedings of the 18th Annual Meeting of the Association for Computational Linguistics, University of Pennsylvania, Philadelphia, June 1980.

Moore, Robert C. Problems in Logical Form. Proceedings of the 19th Annual Meeting of the Association for Computational Linguistics, Stanford University, Palo Alto, June, 1981.

Moore, Robert C. Reasoning About Knowledge and Action. SRI International, Technical Note 191. October, 1980.

Peters, Stanley, and Robert W. Ritchie. Phrase Linking Grammars. December 1981. Unpublished manuscript.

Robinson, Jane. DIAGRAM: A Grammar for Dialogues. *Communications of the ACM, 25*:1 (January, 1982) 27–47.

Stickel, Mark. A Non-Clausal Connection Graph Resolution Theorem Proving Program. Forthcoming.

APPENDIX A. Sample Grammar Rules

The following is a portion of a test grammar for the PATR English translation system. Only those portions of the grammar utilized in analyzing the sample sentences in the text were included. The full grammar handles the following constructs: modals, adjectivals, tense, predicative and nonpredicative copulatives, adverbials, quantified noun phrases, aspect, NP, PP, and infinitival complements, relative clauses, yes/no questions, restricted wh-questions, noun-noun compounds, passives, and prepositional phrases as predicates and adjectivals.

```
========== Grammar Rules ==========

Constant EQ' = curry (LAMBDA (X Y) (equal X Y))

Constant PASS' =
<A: (LAMBDA P (LAMBDA X ((P X) Y))), NIL,
    (MK.MBD (QUOTE (LAMBDA S (some Y (thing Y) S)))) >

Constant PASSINF' =
<A: (LAMBDA P (LAMBDA I (LAMBDA X (((P X) I) Y)))), NIL,
    (MK.MBD (QUOTE (LAMBDA S (some Y (thing Y) S)))) >

AUXP -> TENSE;
  Translation:
    TENSE'

DDET -> DET;
  Annotation:
    [ Definite(DDET) ]
  Translation:
    DET'
```

```
DETP -> A;
  Annotation:
    [ ¬Definite(DETP) ]
  Translation:
    A'

DETP -> DDET;
  Annotation:
    [ AGREE(DETP, DDET, Definite) ]
  Translation:
    DDET'

INFINITIVE -> TO VPP;
  Annotation:
    [ AGREE(INFINITIVE, VPP, Gappy, Wh) ]
  Translation:
    pull.v(VPP')

NOM -> NOMHD;
  Annotation:
    [ AGREE(NOM, NOMHD, Gappy) ]
  Translation:
    NOMHD'

NOMHD -> NOUN;
  Translation:
    NOUN'

NOUN -> N;
  Translation:
    N'

NP -> DETP NOM;
  Annotation:
    [ AGREE(NP, NOM, Gappy) ]
    [ Predicative(NP) \/ ¬Predicative(NP) ]
    [ AGREE(NP, DETP, Definite) ]
  Translation:
    ¬Predicative(NP): DETP'[NOM']
    Definite(NP) & Predicative(NP): EQ'[DETP'[NOM']]
    ¬Definite(NP) & Predicative(NP): NOM'

PREDICATE -> AUXP VPP;
  Annotation:
    [ AGREE(PREDICATE, VPP, Active, Gappy, Wh) ]
  Translation:
    pull.v(AUXP'[VPP'])

S -> SDEC;
  Annotation:
    [ ¬Gappy(SDEC) ]
    [ ¬Wh(SDEC) ]
  Translation:
    SDEC'

SDEC -> NP PREDICATE,
  Annotation:
    [ Gappy(NP) \/ Gappy(PREDICATE) <=> Gappy(SDEC) ]
    [ ¬Predicative(NP) ]
    [ Wh(NP) \/ Wh(PREDICATE) <=> Wh(SDEC) ]
    [ ¬(Gappy(NP) & Gappy(PREDICATE)) ]
  Translation:
    pull.s(PREDICATE'[NP'])

VP -> VPT;
  Annotation:
    [ ¬Transitive(VPT) ]
    [ ¬TakesInf(VPT) ]
    [ Active(VPT) ]
    [ Active(VP) ]
  Translation:
    VPT'

VP -> VPT NP INFINITIVE;
  Annotation:
    [ TakesInf(VPT) ]
    [ Transitive(VPT) ]
    [ ¬Predicative(NP) ]
    [ AGREE(VP, VPT, Active) ]
    [ Wh(NP) \/ Wh(INFINITIVE) <=> Wh(VP) ]
    [ IF(Active(VPT),
         ((Gappy(NP) \/ Gappy(INFINITIVE)) <=> Gappy(VP)),
            & ¬(Gappy(NP) & Gappy(INFINITIVE)),
         (¬Gappy(VPT) & Gappy(NP))) ]
  Translation:
    Active(VP):  pull.v(VPT'[NP'][INFINITIVE'])
    ¬Active(VP):  pull.v(PASSINF'[VPT'][INFINITIVE'])

VPP -> VP;
  Annotation:
    [ AGREE(VPP, VP, Gappy, Wh) ]
    [ Active(VP) ]
  Translation:
    VP'

VPT -> V;
  Annotation:
    [ AGREE(VPT, V, Active, Transitive, TakesInf) ]
  Translation:
    V'


====================== Lexicon ========================

N -> man;
  Translation:
    <A: man, NIL, NIL >

N -> woman;
  Translation:
    <A: woman, NIL, NIL >

DET -> every;
  Annotation:
    [ Definite(DET) ]
  Translation:
    <B: (LAMBDA P (LAMBDA S (every X (P X) S))), X, NIL >

A -> a;
  Translation:
    <B: (LAMBDA P (LAMBDA S (some X (P X) S))), X, NIL >
```

```
V -> persuade;
  Annotation:
    [ Transitive(V) ]
    [ Active(V) \/ ¬Active(V) ]
    [ TakesInf(V) ]
  Translation:
    curry (LAMBDA (X P Y) (persuade Y X (P X)))

V -> go;
  Annotation:
    [ ¬Transitive(V) ]
    [ ¬TakesInf(V) ]
    [ Active(V) ]
  Translation:
    <A: go, NIL, NIL >

TENSE -> &past;
  Translation:
    curry (LAMBDA (P X) (past (P X)))
```

APPENDIX B. Meaning Postulates

```
[every w (every u (iff (past w u)
                       (not (past u w]
(every w (some u (past w u)))
[every w (every x (every y (every z (implies (promise w x y z)
                                             (past w z]
[every w (every x (every y (every z (implies (persuade w x y z)
                                             (past w z]
(every w (every x (thing w x)))
[every w (every x (every z (implies (want w x z)
                                    (past w z]
(every w (poss w w))
[every w (every u (implies (past w u)
                           (poss w u]
[every w (every u (every v (implies (and (past1 w u)
                                         (past1 u v))
                                    (past2 w v]
[every w (every z (implies (past2 w z)
                           (past w z]
[every w (every z (iff (past w z)
                       (past1 w z]
```

APPENDIX C. Transcript of Sample Dialogue

```
>> john is happy
OK.

>> is john happy
Yes.

>> is john a happy man
I don't know.

>> john is a man
OK.

>> is john a happy man
Yes.

>> no man could have hidden a book
OK.

>> did john hide a book
No.

>> bill hid a book
OK.

>> is bill a man
No.

>> was john a man
I don't know.

>> every man will be a man
OK.

>> will john be a man
Yes.

>> bill persuaded john to go
OK.

>> could john have been persuaded to go
Yes.

>> will john be persuaded to go
I don't know.
```