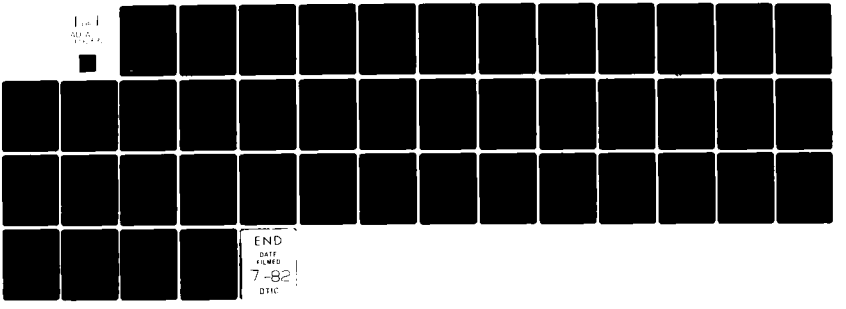
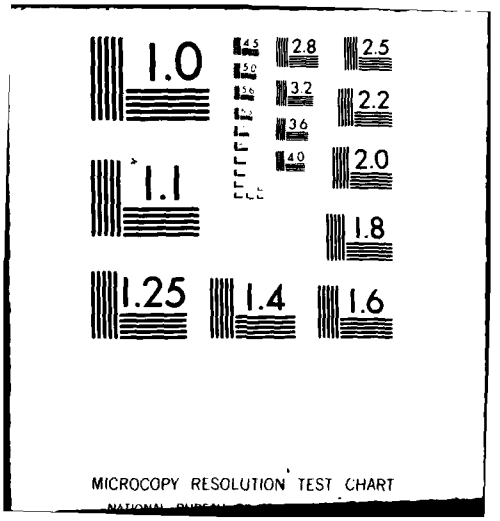
AD-A116 332 ARMY ELECTRONICS RESEARCH AND DEVELOPMENT COMMAND WS--ETC F/G 9/2
RESEARCH ON INTERACTIVE ACQUISITION AND USE OF KNOWLEDGE.(U)
MAY 82 M E STICKEL, B GROSZ, N HAAS N00039-80-C-0575
UNCLASSIFIED NL



END
DATE FILMED
7-82
DTIC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
MICROCOPY RESOLUTION TEST CHART

(12)

RESEARCH ON INTERACTIVE ACQUISITION AND USE OF KNOWLEDGE

Interim Report for the Period July 1981–January 1982

SRI Project 1894

May 1982

Edited by: Mark E. Stickel, Computer Scientist
Artificial Intelligence Center
Computer Science and Technology Division

Prepared for:

Defense Advanced Research Projects Agency
Information Processing Techniques Office
1400 Wilson Boulevard
Arlington, Virginia 22209

Attention: Cmdr. Ronald B. Ohlander

Preparation of this paper was supported by the Defense Advanced Research Projects Agency under Contract N00039-80-C-0575 with the Naval Electronic Systems Command.

SRI International®

333 Ravenswood Ave. • Menlo Park, CA 94025
(415) 859-6200 • TWX: 910-373-2046 • Telex: 334 486

Contents

A

1. Introduction

SRI International is engaged in a long-term research effort under DARPA sponsorship to develop techniques for facilitating the acquisition of knowledge by computers. The ultimate goal is to build KLAUS (Knowledge-Learning And -Using System), a computer program that could acquire a model of a domain of interest by being instructed in English. The model could then be used for domain-related tasks. Systems based on the KLAUS concept would be useful for a variety of applications, including sophisticated interfaces to computer software, advanced database systems, and intelligent computer assistants.

This report covers work done on the KLAUS project during the period July 1981 to January 1982. Section 2 describes DIALOGIC, the natural-language processing system used in MICROKLAUS, which is our current preliminary implementation of the KLAUS concept. The DIALOGIC system translates English sentences into representations of their literal meaning in the context of an utterance. These representations, or *logical forms*, are intended to be a purely formal language that is as close as possible to the structure of natural language, while providing the semantic compositionality necessary for meaning-dependent computational processing. The design of DIALOGIC (and of its constituent modules) was influenced by the goal of using it as a core language-processing component in a variety of systems, some of which are transportable to new domains of application.

Section 3 describes a new scheme for syntax-directed translation of natural language into logical form. That scheme, comprising the basis of an English translation system called PATR, was used to specify a semantically interesting fragment of English that included 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.

Section 4 describes a new theorem-proving program developed for KLAUS. It combines the use of nonclausal resolution and connection graphs. The use of nonclausal resolution as the inference system eliminates some of the redundancy and unreadability of clause-based systems. The use of a connection graph restricts the search space and facilitates graph searching for efficient deduction. This theorem-

proving program is used as the deduction component of both MICROKLAUS (of which DIALOGIC, described in Section 2, is the natural-language-processing component) and the PATR-based question-answering system described in Section 3.

2. DIALOGIC: A Core Natural-Language-Processing System

This section was written by Barbara Grosz, Norman Haas, Gary Hendrix, Jerry Hobbs, Paul Martin, Robert Moore, Jane Robinson, and Stanley Rosenschein. Norman Haas and Gary Hendrix are now with Symantec in Sunnyvale, California.

2.1. Overview

The DIALOGIC system translates English sentences into representations of their literal meaning in the context of an utterance. These representations, or *logical forms*, are intended to be a purely formal language that is as close as possible to the structure of natural language, while providing the semantic compositionality necessary for meaning-dependent computational processing. The design of DIALOGIC (and of its constituent modules) was influenced by the goal of using it as the core language-processing component in a variety of systems, some of which are transportable to new domains of application.

Currently DIALOGIC is a core component of KLAUS and three other systems being developed in several different research projects at SRI. One is the TEAM project,[1] whose goal is to provide natural-language access to large databases through systems that are easily adaptable to a wide range of new applications. A second project is investigating the problem of providing natural-language access to text.[2] A third, in which DIALOGIC also plays an important role, is examining the development of formal grammars.[3]

DIALOGIC is divided into five modules coordinated by the DIAMOND executive system. DIAMOND is a modification of the executive system used in the SRI speech-understanding project [Wa78] as well as in a task-dialogue interpretation system [Ro80]. It provides the formal language for defining the grammar and the control for parsing English sentences and translating them into logical-form expressions.

---

[1]Sponsored by the Defense Advanced Research Projects Agency under Contract N00039-80-C-0645.
[2]Funded by the National Library of Medicine under Grant 1RO1-LM03611.
[3]Sponsored by the National Science Foundation under Grant IST-8103550.

The five modules are (1) the DIAGRAM grammar; (2) a set of semantic translators; (3) a set of basic semantic functions; (4) a scoping algorithm (for quantifiers and sentence operators); (5) a set of basic pragmatic functions. The remainder of this section describes these components of DIALOGIC and presents an example illustrating how they coordinate with one another in the interpretation of an utterance. A description of the logical form that is the target of DIALOGIC's interpretation processes may be found in [Mo81].

2.2. DIAGRAM

DIAGRAM is a general grammar of English. It now contains about 125 rule schemata, equivalent to about 800 individual rules. These define all the common sentence types, complex auxiliaries and modals, complex noun phrases, nominalized sentences, all the common quantifiers, relative clauses, verbs with sentential complements, comparative and measure expressions, subordinate clauses and other adverbial modifiers. Conjunction, however, is limited to a few placeholders, pending further study of the problems it poses for constraining the number of syntactic analyses. A detailed description of DIAGRAM is contained in [Ro82]. In a formal sense, DIAGRAM is an augmented phrase-structure grammar. The lexicon categorizes words and associates attributes with them that are used in the rules. Each rule has associated with it a *constructor* that expresses the constraints on its application and also a *translator* (described in the next subsection) that produces the corresponding logical form.

Phrases inherit attributes from their constituents and acquire attributes from the larger phrases that contain them. These attributes are used to impose context-sensitive constraints upon the acceptance of an analysis. Before constructing a node in the parse tree corresponding to the application of a rule, the executive invokes the rule's constructor to test for admissibility. In addition to accepting or rejecting a rule application, the constructors can assign scores that allow the listing of alternative analyses in a preferred order. The result of applying the grammar to analysis of an input is one or more *annotated parse trees.*

Attributes and annotations are not limited to syntactic information. The translators, described next, specify how the translation of a phrase into logical form is to be defined in terms of the attributes of the words and phrases that compose it. This coupling of syntax and semantics (for which *attribute*

grammars [Ti80] were originally designed) is convergent with current formal theories of natural language that advocate constructing a syntax and semantics that "work in tandem" [DW81, KB-, Ga80, La76.]

Future work on DIAGRAM will include efforts to extend both its coverage and its formalism. In extending the formalism, our dual objective is to capture certain linguistic generalizations (e.g., dative movement) and to make the task of developing a large grammar more manageable. To accomplish this, we are exploring the use of metarules [Ga80].

2.3. Translators

Following the syntactic analysis of an utterance, a sequence of semantic translators is invoked to build the logical form that corresponds to a literal interpretation of the utterance in context. The translator for each phrase-structure rule specifies how the various constituents of the phrase are to be combined to form an interpretation of the whole phrase. It prescribes the predicate-argument structures that correspond to the grammatical construction or, more generally, the operator-operand structures.

Although the translators operate top-down (the translator for each node invokes the translators for its children), the translation is in effect built bottom-up—since, typically, the first thing a translator for a nonterminal node does is to invoke the translators for each of its constituents, usually left to right. However, the top-down nature of the translation process is significant, because it means that information located above a node and to its left is available when the node is translated. In addition to producing the logical form, the translators determine the syntactic constraints upon and preferences for either coreference or noncoreference of noun phrases, especially pronouns, in accordance with an algorithm described in [Ho76].

2.4. Basic Semantic Functions

To insulate changes in the grammar from those that occur in logical form, the construction of the latter is isolated from the translator procedures by calls on basic semantic functions [Ko79]). The actual construction of a logical form is done in two phases: (1) logical-form fragments (lffs) are attached to the parse tree by the basic semantic functions; (2) the final logical form is assembled from these by the scoping algorithm.

Lffs are assigned only to certain nodes in the parse tree. Usually the lff at an NP node will encode the properties held by the entity the NP describes [e.g., "X such that EMPLOYEE(X) & OLD(X)" for "old employee"] and the fragment for a clause-level construction (e.g., a VP) will encode the predicate-argument structure of the clause.

The basic semantic functions also leave markers on the parse tree to indicate such things as the type of quantifier or determiner associated with a noun phrase. These markers are used by the scoping algorithm to determine the final logical form for the utterance. (Note that the lffs and markers left by the basic semantic functions may be viewed as further annotations to the parse tree.)

DIALOGIC currently includes eleven basic semantic functions. Six of these do most of the work of building lffs for standard noun phrases and clauses. The others are concerned with adding such things as mode, degree, and adverbial modification to clauses. As more precise specifications are defined for encoding those phenomena in logical form, we expect to eliminate some of these basic semantic functions.

2.5. Scoping of Quantifiers and Other Sentential Operators

The scoping algorithm is designed to collect the logical-form fragments from the parse tree and produce the possible scopings of quantifiers and other scoped operators. The scoping algorithm used in DIALOGIC (adapted from that in [He78]) produces all the scopings that do not violate the hard rules of English scoping, and then ranks them according to a score computed by a set of specialist critics. Each critic is a function that returns a score for some aspect of the conflicting rules of quantification in English; e.g., the left-right scope critic lowers the score of scopings that involve permuting the left-outermost default ordering of quantifiers. All critics receive equal weight in the present implementation, but the design of the system does allow for differential weighting.

The current set of critics is concerned with such things as changes in sentence order and the relative scoping of quantifiers of different strengths. The scoping of nonstandard quantifiers and of the generalized negative ("not," "no one," "nothing," "none") remains to be done.

2.6. Basic Pragmatic Functions

Basic pragmatic functions are intended to fulfill several roles in DIALOGIC, all concerned with

certain kinds of indeterminacies in logical form whose resolution requires pragmatic information. The four primary uses of basic pragmatic functions in the current system are (1) to provide a context-specific interpretation of certain terms that have only vague meanings in themselves (e.g., prepositions like "of" and "in," or inherently vague verbs like "have"); (2) to establish the specific relationship underlying any given noun-noun combination; (3) to identify the referents of pronouns; and (4) to interpret a limited range of metonymy (e.g., the use of "blonds" to mean "people with blond hair"). At present, only a small core of pragmatic functions is implemented, each of which handles but a subset of the cases it is intended to cover.

2.7. Example

To illustrate how the different modules of DIALOGIC contribute to the interpretation of an utterance, we shall consider the example

What SRI employees have children older than 15 years?

The logical form for this query—the target of the interpretation processes—is (lowercase is used to indicate variables, uppercase to indicate constants and predicates):

```
[QUERY (WH employee1 (AND (EMPLOYEE employee1)
                          (EMPLOYEES-COMPANY-OF employee1 SRI))
                    (SOME child2 (CHILD child2) (AND (CHILD-OF employee1 child2)
                                                     ((*MORE* OLD) child2 (YEAR 15]
```

This corresponds roughly to a formal representation for "Who is each employee such that the company of the employee is SRI and some child of the employee is older than fifteen years?"

During DIAMOND's parsing phase, the parse tree in Figure 1 is constructed. At this point, the attributes annotating the tree encode such properties as the type of noun (count, mass, unit) and the syntactic number. These attributes have been used during the parsing phase to rule out certain alternative structures.

Once this structure is built, the translators are invoked. In combination with the basic semantic functions, the translators assign additional attributes to nodes in the tree, encoding such information as the quantifiers (type, strength, and the variables they bind) and the heads of noun phrases. For example, the head of the WHNP, "what SRI employees", is a variable of type EMPLOYEE that is

bound by a wh-type quantifier. Attributes also encode the underlying predicate-argument structures for verb phrases and adjectives as well as the lffs to be used in constructing the final logical form for the utterance.

In the sentence of Figure 1, the nodes WHNP and S are annotated as being quantified, WHNP with a wh-type quantifier and S as a "query." Although every rule has an associated translator, only some of these result in lffs being attached to nodes. For this example, the nodes marked with ** in the original parse tree are the only ones for which lffs are produced.

The fragment attached to each of these nodes is as follows:

NOUN1	(*NN* employee1 SRI)
NOUN2	(EMPLOYEE employee1)
PREDICATE	(*HAVE employee1 child2)
NOUN3	(CHILD child2)
NCOMP	((*MORE* OLD) child2 (YEAR 15))

EMPLOYEE, CHILD, and OLD are unary predicates that are part of the conceptual model of the domain. *MORE* maps a predicate into a comparative along the scale corresponding to the predicate. *NN* and *HAVE are dummy predicates that indicate the need to invoke the basic pragmatic functions.

After the translation process is complete, the final logical form is assembled by a procedure that considers alternative quantifier scopings (using the quantifier-related annotations left on the parse tree) and invokes the basic pragmatic functions as needed. The basic pragmatic functions use information in the conceptual model of the domain to transform (*NN* employee1 SRI)—corresponding to the noun-noun compound "SRI employee"—into (EMPLOYEE-OF employee1 SRI) and (*HAVE employee1 child2) into (CHILD-OF employee1 child2).

The nodes with either quantifier or logical-form markings are the only ones considered by the TEAM scoping algorithm. Besides the WH quantifying employee1, TEAM recognizes that a default existential quantifier must be created for child2, so SOME is added. The scope rules force QUERY to have the widest scope; this position is contested only if there are multiple sentential markers. Both orderings of the WH and SOME quantifiers are generated. The two resulting quantified statements correspond to (WH employee1 (SOME child ...) ...) and (SOME child (WH employee ...) ...)

Next the scope critic functions evaluate the different scopings; only three of the critics are

relevant. One critic considers the left-right node ordering and prefers the first scoping because it comes closer to the surface form. One critic prefers scopings in which WH outscopes an adjacent existential; it too upgrades the score of the first and downgrades the score of the second. The other critic knows that default existential quantifiers need the narrowest possible scope; it too selects the first scoping.

2.8. Summary

Because of the modularization in DIALOGIC, changes in one part of the system reverberate very little in other components. Changes in the constraints imposed on the phrase-structure rules in the grammar have no effect on any other part of the system. A change in a rule itself necessitates a change in the corresponding translator, but the basic semantic functions do not need to be revised. Similarly, a change in the logical form or in the data structures within which it is implemented requires a corresponding change in the basic semantic functions, but not in the grammar or translators.

In addition to extending DIALOGIC as mentioned in the foregoing sections, we are also investigating possible revisions of the translation phase (as currently performed by the translators and basic semantic functions) to allow translation into logical form to be specified declaratively. In this new approach (described in the following section), logical types are associated with phrasal categories, and the translation of a phrase is synthesized from the translations of its immediate constituents according to a local rule, which typically involves functional application.

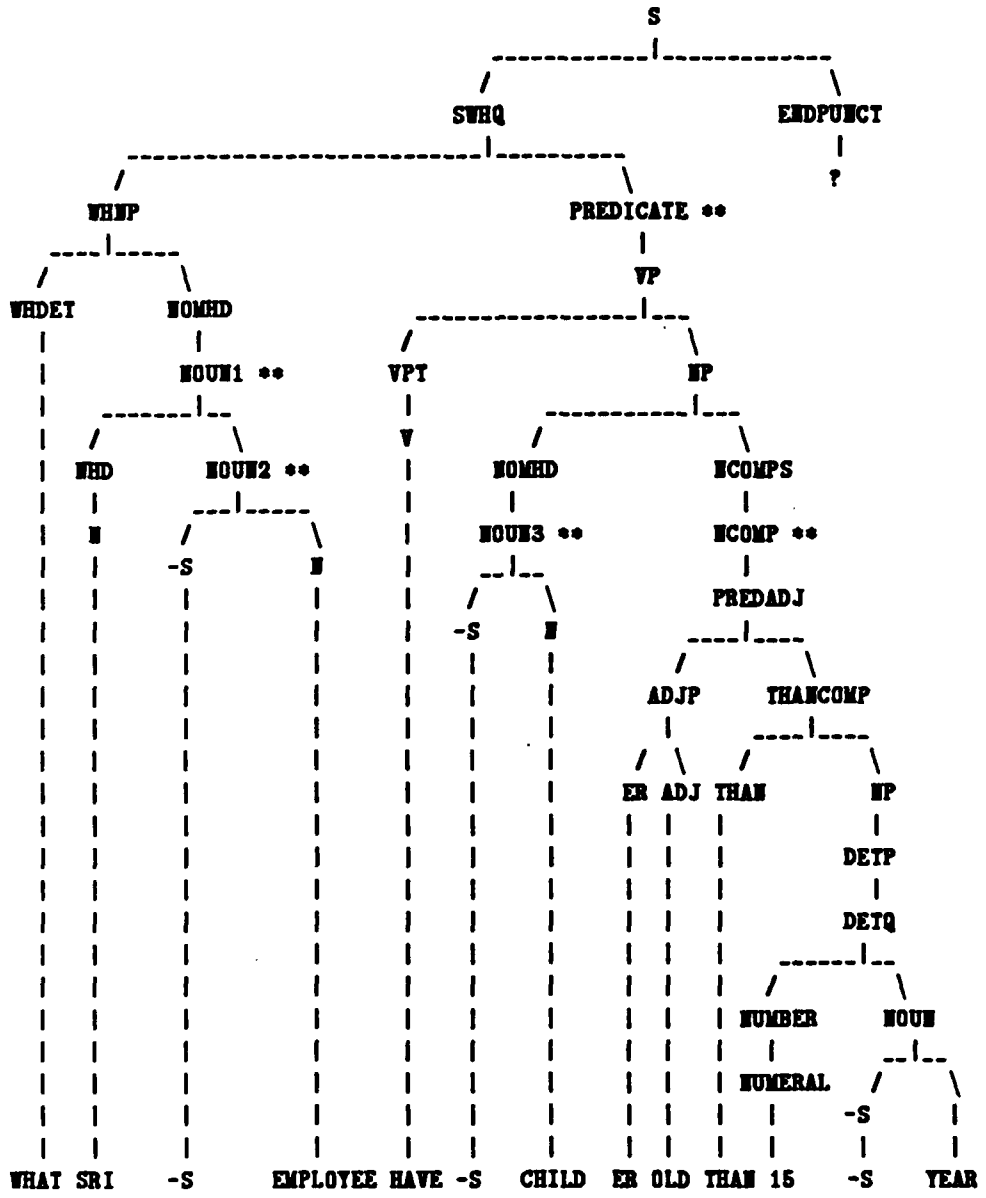


Figure 1. Parse Tree for "What SRI employees have children older than 15 years?"

3. Translating English into Logical Form

This section was written by Stanley J. Rosenschein and Stuart M. Shieber.

3.1. 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 [Ga80]. 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, for reasons of manageability and esthetics, would govern the actual form of grammars. 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 must generally be based on deduction (and hence translation).

The rest of this section discusses our work in more detail. Section 3.2 presents the grammar formalism and describes PATR, an implemented parsing and translation system that can accept a grammar in our formalism and use it to process sentences. Examples of the system's operation, including its application in a simple deductive question-answering system, are found in Section 3.3. Finally, Section 3.4 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.

3.2. A Grammar Formalism

3.2.1 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, in the usual way, associate structural descriptions (parse trees) for each sentence in that language. Nodes in the parse tree are assumed to have a set of features that 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

```
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:
   Mr → 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

in many ways, and numerous optimizations could be envisioned that would have the effect of folding back later phases to increase efficiency.

3.2.2 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.

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,

[1] Of course, more sophisticated approaches to morphological analysis would seek to analyze the LEX relation more fully. See, for example, [KR81] and [KK81].

"went" is mapped to 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. Through utilization of 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,

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

is transformed into

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

3.2.3 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 single 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, [Co-] 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

---

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

[3] Following [Mo81], a quantified expression is of the form (*quantifier, variable, restriction, body*)

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"), thereby 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

$$<hd,var,sto>[<hd',var',sto'>] = <(hd\ hd'),var \cup var',\ sto \cup sto'> \quad ,$$

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

$$<hd,var,sto>[<hd',var',sto'>] = <var,\ var',\ (hd\ hd') \cup sto \cup sto'> \quad .$$

For example, if the meaning of "every" is

$$every' = <(\lambda\ P\ (\lambda\ S\ (every\ X\ (P\ X)\ S))),\ X,\ \phi>$$

and the meaning of "man" is

$$man' = <man,\ \phi,\ \phi> \quad ,$$

then the meaning of "every man" is

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

as expected.

Nondeterminism arises in two ways. First, since pull operations can be invoked nondeterminis-

tically 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 3.3.2 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 specifying for each schema a *guard*, i.e., a Boolean combination of features that the nodes analyzed by the production must satisfy for the translation schema to be applicable.

3.2.4 Implementation of a Translation System

The techniques presented in Sections 3.2.2 and 3.2.3 were implemented in a parsing and translation system called PATR, which was used as a component in a dialogue system discussed in Section 3.3.3. 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.

3.3. Experiments in Producing and Using Logical Form

3.3.1 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 [Mo81]. 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 [Ro82] 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.

3.3.2 An Example of the System's Operation

The grammar presented in Appendix A encodes a relation between sentences and logical-form expressions. 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 acquires the

[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.

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 3.2.3. (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))>
       [<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))>
```

[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.

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, thus 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. After 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)))))
```

3.3.3 A Simple Question-Answering System

As mentioned in Section 3.1, 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 [Mo81], which were then postprocessed into a form suitable for a first-order deduction system. We used a connection graph theorem prover, described in Section 4 of this report. (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, etc.—and that, moreover, it was not restricted to

[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.

```
(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

extensional evaluation in a database, 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.)

```
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 8: Translation to LF and Reduction to FOL

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. [Mo80] 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 Appendix B).

3.4. 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, along with 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 [PR81], and methods for generating the augmented phrase-structure grammar through a metarule formalism similar to that of [Ko80]. We have also experimented with alternative parsing algorithms, including a chart parser [BK79] adapted to carry out annotation and translation in the manner described in this paper.

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.
```

4. A Nonclausal Connection-Graph Resolution Theorem-Proving Program

This section was written by Mark E. Stickel.

4.1. Introduction

This section describes some of the theory and features of a nonclausal connection-graph resolution theorem-proving program being developed as a reasoning component for KLAUS.

The most important characteristics of the program are

- Nonclausal resolution is used as the inference system, eliminating some of the redundancy and unreadability of clause-based systems.
- A connection graph is used to represent permitted resolution operations, restricting the search space and facilitating the use of graph searching for efficient deduction.
- Heuristic search and special logical connectives are used for program control.

The following sections will describe these aspects of the program, citing disadvantages and difficulties as well as advantages, and will be followed by a description of the implementation status of the program and future plans for it.

4.2. Nonclausal Resolution

One of the most widely criticized aspects of resolution theorem proving is its use of clause form for wffs. The principal criticisms are

- Conversion of a wff to clause form may eliminate pragmatically useful information encoded in the choice of logical connectives (e.g., $\neg P \vee Q$ may suggest case analysis while the logically equivalent $P \supset Q$ may suggest chaining).
- Use of clause form may result in a large number of clauses being needed to represent a wff, as well as in substantial redundancy in the resolution search space.

- Clause form is difficult to read and not human-oriented.

The clausal resolution rule can be easily extended to general quantifier-free wffs [Mu82,MW80]. Proofs of soundness and completeness are in [Mu82]. Where clausal resolution resolves on clauses containing complementary literals, nonclausal resolution resolves on general quantifier-free wffs containing atomic wffs (atoms) occurring with opposite *polarity*, which is determined by the parity of the number of explicit or implicit negations in whose scope the atom appears (positive polarity if even, negative polarity if odd). In clausal resolution, resolved-on literals are deleted and remaining literals disjoined to form the resolvent. In nonclausal resolution, all occurrences of the resolved-on atom are replaced by **F**=false (**T**=true) in the wff in which it occurs positively (negatively). The resulting wffs are disjoined and simplified by truth-functional reductions that eliminate embedded occurrences of **T** and **F** and optionally perform simplifications such as $A \wedge \neg A \rightarrow \mathbf{F}$.

Definition 4.1. If A and B are ground wffs and C is an atom occurring positively in A and negatively in B, then the result of simplifying $A(C \leftarrow \mathbf{F}) \vee B(C \leftarrow \mathbf{T})$, where $X(Y \leftarrow Z)$ is the result of replacing every occurrence of Y in X by Z, is a *ground nonclausal resolvent* of A and B.

It is clear that nonclausal resolution reduces to clausal resolution when the wffs are restricted to be clauses. In the general case, however, nonclausal resolution has some novel characteristics as compared with clausal resolution. It is possible to derive more than one resolvent from the same pair of wffs, even resolving on the same atom, if the atom occurs both positively and negatively in both wffs (e.g., atoms within the scope of an equivalence occur both positively and negatively). Likewise, it is possible to resolve a wff against itself.

The ground nonclausal resolution rule can be lifted to nonground wffs by renaming parent wffs apart and unifying sets of atoms from each parent, one atom of each set occurring positively in the first wff and negatively in the second. As with clausal resolution, only single atoms need be resolved upon if the resolution operation is augmented by a factorization operation that derives a new wff by instantiating a wff by a most general unifier of two or more distinct atoms occurring in the wff (regardless of polarity).

A nonclausal resolution derivation of **F** from a set of wffs demonstrates the unsatisfiability of the set of wffs. Nonclausal resolution is thus, like clausal resolution, a refutation procedure. Variants

of the procedure that attempt to affirm rather than refute a wff are possible (e.g., see the variety of resolution rules in [MW80]), but are isomorphic to this procedure.

Although clause form is often criticized, use of nonclausal form has the disadvantage that most operations on nonclausal form are more complex than the same operations on clause form. The result of a nonclausal resolution operation is less predictable than the result of a clausal resolution operation. Clauses can be represented as lists of literals; sublists are appended to form the resolvent. Pointers can be used to share lists of literals between parent and resolvent [BM72]. With many simplifications such as $A \wedge \mathbf{T} \rightarrow A$ and $A \wedge \neg A \rightarrow \mathbf{F}$ being applied during the formation of a nonclausal resolvent, the appearance of a resolvent may differ substantially from its parents, making structure sharing more difficult.

For most forms of clausal resolution, an atom does not occur more than once in a clause. In nonclausal resolution, an atom may occur any number of times, with possibly differing polarity. In clausal resolution, every literal in the clause must be resolved upon for the clause to participate in a refutation. Thus if a clause contains a literal that is pure (cannot be resolved with a literal in any other clause), the clause can be deleted. This is not the case with nonclausal resolution; not all atom occurrences are essential in the sense that they must be resolved upon to participate in a refutation. For example, $\{P \wedge Q, \neg Q\}$ is a minimally inconsistent set of wffs, one of which contains the pure atom P. A more complicated definition of purity involving this notion of essential occurrences must be used. The subsumption operation must also be redefined for nonclausal resolution to take account of such facts as the subsumption of A by $A \wedge B$ as well as the clausal subsumption of $A \vee B$ by A.

[Mu82,MW80] suggest the extension of nonclausal resolution to resolving on nonatomic subwffs of pairs of wffs. For example, $P \vee Q$ and $(P \vee Q) \supset R$ could be resolved to obtain R. Resolving on nonatoms often permits significantly shorter and more readable refutations. However, there are several reasons for not doing this:

- It may be difficult to recognize complementary wffs. For example, $P \vee Q$ occurs positively in $Q \vee R \vee P$ and $\neg P \supset Q$.
- The effect of resolving a pair of wffs on nonatomic subwffs can be achieved by multiple resolution operations on atoms. Resolution on both atomic and nonatomic subwffs could result in redundant derivations.

- A connection-graph procedure would be complicated by the need to attach links to logical subwffs (e.g., $P \vee Q$ in $Q \vee R \vee P$) and link inheritance would be further complicated since subwffs of a resolvent may have no parent subwffs (e.g., when $P \vee Q$ and $\neg P \vee R$ are resolved, the resolvent $Q \vee R$ is a subwff of neither parent). Similar complications arise if equality inferences are used that introduce new structure into the result.

Although the nonclausal resolution rule in general seems adequate as compared with the above proposed extension to matching on nonatomic subwffs, the handling of the equivalence relation in [Mu82] is inadequate. In resolving $P \equiv Q$ and $(P \wedge R) \vee (\neg P \wedge S)$, it is possible to derive $Q \vee S$ and $\neg Q \vee R$, but not the more natural result of simply replacing P by Q. It is questionable whether handling the equivalence relation in nonclausal resolution without further extension is worthwhile in comparison with the representational advantages of negation normal form used in [An81,Bi81]. Another difficulty with the equivalence relation is that it sometimes needs to be removed during Skolemization. It is somewhat ineffective to provide an inference system which handles equivalence, but may require its removal during Skolemization. [MW82] provides extensions to nonclausal resolution that defer Skolemization and permit equivalence relations to be retained longer.

4.3. Connection Graphs

Connection-graph resolution was introduced in [Ko75]. It has the following advantages:

- The connection-graph refinement is quite restrictive. Many resolution operations permitted by other resolution procedures are not permitted by connection-graph resolution.
- The links associated with each wff function partially as indexing of the wffs. Effort is not wasted in the theorem prover examining the entire set of wffs for wffs that can be resolved against newly derived wffs.
- Links can be traversed by a graph-searching algorithm whereby each link traversal denotes a resolution operation. This can be done to plan a deduction without actually constructing it. This graph searching may resemble the searching performed for deduction in knowledge representation languages.

Connection-graph resolution is extended in a natural way to use the nonclausal resolution inference rule.

A connection graph is a set of wffs and a set of links that connect atoms occurring with positive polarity in one wff and negative polarity in the same or another wff. Performing the nonclausal resolution operation indicated by the link results in the production of a new connection graph with the resolved upon link eliminated and the nonclausal resolvent added. Roughly speaking, atoms of the nonclausal resolvent are linked only to to atoms to which atoms of the parent wffs were linked.

Definition 4.2. Let S be a set of ground wffs. Let L be

$$\{\langle C, A, B\rangle \mid A, B \in S, \textit{atom } C \textit{ occurs positively in } A \textit{ and negatively in } B\}$$

Then $\langle S, L\rangle$ is the *full connection graph* for S.

Definition 4.3. Let S be a set of ground wffs and L be its connection graph. Let $\ell = \langle C, A, B\rangle$ be an element of L and C be the nonclausal resolvent $A(C \leftarrow \mathbf{F}) \vee B(C \leftarrow \mathbf{T})$. Let S' be $S \cup \{C\}$. Let L' be

$$\begin{aligned}
&L - \{\ell\} \\
&\cup\{\langle E, C, D\rangle \mid \textit{atom } E \textit{ occurs positively in } C \textit{ and} \\
&\qquad \langle E, A, D\rangle \in L \textit{ or } \langle E, B, D\rangle \in L\} \\
&\cup\{\langle E, D, C\rangle \mid \textit{atom } E \textit{ occurs negatively in } C \textit{ and} \\
&\qquad \langle E, D, A\rangle \in L \textit{ or } \langle E, D, B\rangle \in L\} \\
&\cup\{\langle E, C, C\rangle \mid \textit{atom } E \textit{ occurs positively and negatively in } C \textit{ and} \\
&\qquad \langle E, A, A\rangle \in L, \langle E, B, B\rangle \in L, \langle E, A, B\rangle \in L, \textit{ or } \langle E, B, A\rangle \in L\}.
\end{aligned}$$

Then the connection graph $\langle S', L'\rangle$ is derived from $\langle S, L\rangle$ by *ground nonclausal connection-graph resolution.*

A nonclausal connection-graph resolution refutation of an input set of wffs is a derivation of a set of wffs including $\mathbf{F}$ by nonclausal connection-graph resolution from the full connection graph of the input set of wffs.

Ground nonclausal connection-graph resolution can be extended to the nonground case by including in the links the unifier of the atoms they connect, keeping wffs renamed apart, and by including links between variants of the same wff (to allow a wff to directly or indirectly resolve against a variant

of itself). Factorization must also be included. Either factors with appropriately inherited links must be added for each wff in the connection graph or special factor links can be used with link inheritance rules for both resolve and factor links after resolution and factorization operations.

The nonclausal connection-graph resolution procedure is sound and there is reason to believe it is complete. However, it has not yet been proved to be complete, and the history of proving completeness of connection-graph procedures for the simpler clausal case (see [Bi81]) suggests it may be difficult.

One reason it is difficult to prove the completeness of the connection-graph procedure is that the link inheritance rules exclude some links that would be present if the connection graph were merely an encoding of all permitted resolution operations for ordinary resolution. Exactly which links are excluded depends on the order in which resolution operations are performed. The effect of connection-graph resolution is to impose the following restriction: if a pair of atoms in a pair of wffs is resolved upon, atoms derived (in later resolution operations) from the resolved-on atoms cannot be resolved against each other. An example of this (and a reason for the power of this restriction) is that, if a set of wffs includes $P \vee Q$ and $\neg P \vee \neg Q$, these two wffs can be resolved upon P and Q—resulting in tautologies that are discarded; after that, neither wff can be resolved with an atom descended from the other, even though doing so would not result in a tautology.

Connection-graph resolution procedures can possibly be incomplete by succeeding in finding refutations when links are resolved upon in some orders, but not others. For example, consider the combination of linear resolution and connection-graph resolution for clauses. Each is complete, but the combination is not. If linear connection-graph resolution is applied to $\{ P \vee \neg Q, \neg P \vee \neg Q, Q \}$ with Q as top clause, depth-first search will find a refutation, but breadth-first search will not. This contrasts with the usual situation in which breadth-first search is "safe", always guaranteed to find a refutation if there is one. To see that it fails in this case, observe that after P and $\neg P$ are generated on the first level of breadth-first search, Q and $\neg Q$ have no links—and thus none of the three input clauses can be further resolved upon to lead to a refutation. P and $\neg P$ are linked, but cannot be resolved without violating the linear-resolution restriction.

A set of assertions in a connection graph can to some extent be regarded and treated as a semantic network—more so than the same set of assertions without the connection graph.

For example, the full connection graph for

$$elephant(Clyde)$$
$$elephant(x) \supset mammal(x)$$
$$elephant(y) \supset color(y, gray)$$
$$mammal(z) \supset animal(z)$$

would contain links between the following pairs of atoms

ℓ_1. $\langle elephant(Clyde), elephant(x) \rangle$
ℓ_2. $\langle elephant(Cldye), elephant(y) \rangle$
ℓ_3. $\langle mammal(x), mammal(z) \rangle$.

Answers to such queries as "What color is Clyde?" and "Is Clyde an animal?" can be found by graph searching with minimal analysis of the assertions, by traversing the links in the connection graph. Such searching can be made more efficient by labeling the links (e.g., *isa* for ℓ_1 and ℓ_3, *hascolor* for ℓ_2). The semantic content of the set of assertions is still conveyed by the assertions themselves, but control information is provided to a graph-searching procedure by the link labels.

Similar comments could be made regarding any logical representation. However, the use of a connection graph in which all permissible remaining resolution operations are encoded in explicit links can yield greater efficiency by eliminating traversal of multiple paths to the same goal. For example, suppose ℓ_3 is resolved upon, resulting in the added assertion

$$elephant(w) \supset animal(w)$$

and the added link

ℓ_4. $\langle elephant(Clyde), elephant(w) \rangle$.

The link ℓ_3 is deleted. There is still only one path or proof that Clyde is an animal, since the absence of ℓ_3 blocks the path or proof $elephant(Clyde) \rightarrow mammal(Clyde) \rightarrow animal(Clyde)$.

Graph searching in the connection graph to determine taxonomic relations quickly is a simple illustration of the more general notion, extensively explored in [An81,Si76], of using graph searching to determine the existence of refutations. The ideas and techniques developed there are applicable to nonclausal connection-graph resolution. Connection-graph resolution appears to offer the following advantages over these other schemes:

- Although graph searching can be done in the connection-graph resolution procedure, [An81,Si76] do not allow for the actual formation of resolvents. If their techniques for graph search were adopted as a device for planning or quick refutation, connection-graph resolution could be regarded as a superset of these other methods.
- The actual formation of resolvents and the resulting change in the connection graph are useful for retaining information during a refutation, as well as for conveying information (about usage of wffs, etc.) from one refutation or assertion to the next. (Here it is assumed that the theorem prover is being used with an assertional database to which queries are posed and assertions occasionally added and deleted, as opposed to the usual situation in theorem proving in which there is no persistent assertional database, all axioms being presented anew for each proof.)
- Connection-graph resolution provides a convenient, albeit unsophisticated, means of interleaving matching complementary literals and adding new instances of assertions (if more than one ground instance of a wff is required), as compared with the separate processes of searching for a mating, and quantifier duplication if the search fails [An81].

Of course, the argument in favor of performing only graph searching as in [An81, Si76] is that forming resolvents is expensive compared to traversing links, and the cost of creating and storing inherited links may be high.

A good system will probably have a mixture of resolution and graph searching, as in [BE81] for clausal connection-graph resolution. Graph searching is used in that system for look-ahead and to determine if a refutation exists within a certain number of steps. Simple graph searching is used (e.g., not looking for refutations in which wffs occur more than once), with the full complexity and completeness of connection-graph resolution in the background—ready to attempt a refutation.

One problem with graph searching to find refutations is in assessing the effectiveness of the procedure. In ordinary resolution theorem proving, effectiveness can be evaluated in part by examining the number of clauses generated, retained, used in the refutation, and so forth. [BE81] states "Within

this frame of reference it would be easy to design the 'perfect' proof procedure: the supervisor and the look-ahead heuristics would find the proof and then guide the system without any unnecessary steps through the search space." The amount of time used is a good measure for such a program, but should not be used to compare programs as there may be differences in the machines the programs run on and in the efficiency of the programs themselves (as opposed to the algorithms). In general, as [BE81] states, a measure incorporating both total time and space will be required, adding the further complication of evaluating time-space trade-offs.

4.4. Control

A link scheduler is used to specify a refutation search strategy. When an assertion is added by the user, it is linked in the connection graph to all previous assertions. When a resolvent is added, it is linked to the other assertions according to the link inheritance rules. All such added links are examined by the link scheduler. Three outcomes are possible:

- The link is deleted. For example, analysis may show that resolving on the link would create a tautology or pure wff that could not be used in a refutation, whereupon the link can be deleted.
- The link is retained, but not scheduled. Thus the link can be inherited, but cannot be resolved upon (though its descendants might be). This is done when combining connection-graph resolution with other refinements of resolution, such as ordering restrictions and the special logical-connective restrictions described below.
- The link is scheduled. It is given a numerical score and placed in the link schedule. The theorem prover operates by repeatedly resolving on the best scored link in the schedule, creating the resolvent, and scheduling the added links.

Scheduling of the links is done after all the new links have been added, so that the link scheduler can act on such important facts as the number of links attached to an atom.

Special logical connectives can be used to impose restrictions on the use of particular assertions. As in [Mo80], the following connectives denote the following procedural interpretations of $A \supset B$:

- $A \rightarrow B$. If literal A is ever asserted, assert B (forward chaining).

- $B \leftarrow A$. To prove literal B, try to prove A (backward chaining). Since a refutation procedure is being used, this is interpreted as "permit the resolution, on literal A, between $A \supset B$ and any wff having support and containing A".
- $A \Rightarrow B$. If literal A is ever asserted, also assert B and, to prove $\neg A$, try to prove $\neg B$.
- $B \Leftarrow A$. To prove literal B, try to prove A and, if $\neg B$ is ever asserted, also assert $\neg A$.
- $A \supset B$ and $\neg A \vee B$. Unrestricted and equivalent.

The use of both nonclausal resolution and these special logical connectives gives this program some resemblance to natural deduction [Bl77]. It represents an intermediate point between clausal resolution and natural deduction, with advantages of each. It differs from natural deduction, since, for example, a backward-chaining application of $A \supset B$ to C would result in $\neg A \vee C(B \leftarrow \mathbf{T})$ rather than $C(B \leftarrow A)$ (with perhaps only a single instance of B replaced, requiring additional operations to replace the other occurrences). The latter expression may be more natural, but the former is more concise because all occurrences of B are eliminated and only a single instance of A is added. Heuristic search is used in a manner similar to the way it is employed in a clausal system [St74] and in a natural-deduction system [Ty81].

4.5. Implementation Status and Future Plans

The theorem-proving program is implemented as a 4000+ line INTERLISP program and is presently being used as the deduction component of MICROKLAUS. Natural-language assertions and queries are translated by the DIALOGIC system (described in Section 2) into logical form. This logical form is further translated into predicate calculus for input to the theorem prover. The allowance for predicate variables extends the program slightly beyond ordinary first-order predicate calculus. Future work will expand the range of logical form handled, as not all logical forms that can be generated by DIALOGIC are presently being translated; the range of logical form generated by DIALOGIC is also being expanded). The program was also used as the deduction component of the simple question-answering system described in Section 3.3.3.

Besides the unification filtering provided by the connection graph, atoms in assertions are

indexed by predicate symbol so as to speed the addition to the connection graph of user input assertions when there is a large number of them. Wffs are also indexed by their propositional structure and predicate symbols to speed checking for alphabetic-commutative variants to be eliminated. More efficient indexing schemes will probably be tried and variant elimination replaced by subsumption.

Factorization has not yet been implemented in the program. When two wffs are resolved upon a pair of atoms, all atoms instantiated to be the same as the instantiated resolved-on atoms are replaced by **F** or **T**, but there is no effort to force additional atoms, by further instantiation, to be the same as the resolved-on atoms. Thus, only "obvious" factors are used. This is incomplete, but effective. Factor links will be added for completeness.

So far, only fairly simple evaluation functions have been used in the search control process. They are similar to those used in [St74], being weighted sums of the deduction depth of the wff (a measure of the effort required to derive the wff) and the number of atoms in the wff (a measure of the additional effort that will be needed to complete a refutation using the wff). Performance is generally superior to that in [St74]. In ordering restrictions, atoms are also evaluated according to how many links are connected to them, so that atoms with fewer links can be resolved upon preferentially. Not only is the immediate branching factor reduced, but there is also the prospect that the other atoms with more links will be instantiated and inherit fewer links when the resolution operation is performed. Interestingly, as was also noted in [St74], there can be negative interactions among individually good ideas on search control. For example, a strong length-preference strategy and the strategy of resolving on an atom with the fewest links are somewhat inconsistent. When there are many assertions about some predicate—some short and specific, others long and general—the atom with the fewest links is likely to be linked only to long and general assertions. Resolving on it thus may result in long resolvents that would be given low preference by a strong-length preference strategy. More work will be done on developing good evaluation functions. The most important extension of the program will be further development of connection-graph searching—both to provide input to evaluation functions and to perform entire deductions without creating any resolvents.

References

[An81] Andrews, P.B. Theorem proving via general matings. *J. ACM 28,* 2 (April 1981), 193–214.

[BK79] Bear, J. and Karttunen, L. PSG: a simple phrase structure parser. *Texas Linguistic Forum 14,* 1979.

[Bi81] Bibel, W. On matrices with connections. *J. ACM 28,* 4 (October 1981), 633–645.

[Bl77] Bledsoe, W.W. Non-resolution theorem proving. *Artificial Intelligence 9,* 1 (August 1977), 1–35.

[BE81] Bläsius, K., Eisinger, N., Siekmann, J., Smolka, G., Herold, A., and Walther, C. The Markgraf Karl refutation procedure (Fall 1981). *Proc. Seventh International Joint Conference on Artificial Intelligence,* Vancouver, British Columbia, Canada, August 1981, 511–518.

[BM72] Boyer, R.S. and Moore, J S. The sharing of structure in theorem-proving programs. In Meltzer, B. and Michie, D. (eds.). *Machine Intelligence 7.* Edinburgh University Press, Edinburgh, Scotland, 1972.

[Co–] Cooper, R. *Quantification and Syntactic Theory.* D.Reidel Publishing Co., Dordrecht, Holland. Forthcoming.

[DW81] Dowty, D.R., Wall, R.E., and Peters, S. *Introduction to Montague Semantics.* D.Reidel Publishing Co., Dordrecht, Holland, 1981.

[Ga80] Gazdar, G. Phrase structure grammar. July 1980. To appear in Jacobson, P. and Pullum, G.K. (eds.) *On the Nature of Syntactic Representation.*

[He78] Hendrix, G.G. Semantic aspects of translation. In [Wa78], pp. 193–226.

[Ho76] Hobbs, J. Pronoun resolution. Research Report 76-1, Department of Computer Sciences, City College, City University of New York, New York, New York, August 1976.

[KB–] Kaplan, R.M. and Bresnan, J.W. Lexical-functional grammar: a formal system for grammatical representation. Occasional Paper 13, Center for Cognitive Science, Massachusetts Institute of Technology, to appear.

[KK81] Kaplan, R.M. and Kay, M. Personal communication. 1981.

[KR81] Karttunen, L., Root, R., and Uszkoreit, H. Morphological analysis of Finnish by computer. Paper presented at the ACL session of the 1981 LSA Annual Meeting, New York, New York, December 1981.

[Ko79] Konolige, K.G. A framework for a portable natural language interface to large data bases. Technical Note 197, Artificial Intelligence Center, SRI International, Menlo Park, California, October 1979.

[Ko80] Konolige, K.G. 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, Pennsylvania, June 1980.

[Ko75] Kowalski, R. A proof procedure using connection graphs. *J. ACM 22,* 4 (October 1975), 572–595.

[La76] Landsbergen, S.P.J. Syntax and formal semantics of English in PHLIQA1. Preprints of the 6th International Conference on Computational Linguistics, Ottawa, Ontario, Canada, June 1976.

[MW80] Manna, Z. and Waldinger, R. A deductive approach to program synthesis. *ACM Transactions on Programming Languages and Systems 2,* 1 (January 1980), 90–121.

[MW82] Manna, Z. and Waldinger, R. Special relations in program-synthetic deduction. Technical Note 260, Artificial Intelligence Center, SRI International, Menlo Park, California, March 1982.

[Mo80] Moore, R.C. Reasoning about knowledge and action. Technical Note 191, Artificial Intelligence Center, SRI International, Menlo Park, California, October 1980.

[Mo81] Moore, R.C. Problems in logical form. Proceedings of the 19th Annual Meeting of the Association for Computational Linguistics, Stanford, California, June 1981.

[Mu82] Murray, N.V. Completely non-clausal theorem proving. *Artificial Intelligence 18,* 1 (January 1982), 67–85.

[PR81] Peters, S. and Ritchie, R.W. Phrase linking grammars. Unpublished manuscript, December 1981.

[Ro80] Robinson, A.E. Interpreting natural-language utterances in dialogs about tasks. Technical Note 210, Artificial Intelligence Center, SRI International, Menlo Park, California, March 1980.

[Ro82] Robinson, J.J. DIAGRAM: a grammar for dialogues. *Communications of the ACM 25,* 1 (January 1982), 27–47.

[Si76] Sickel, S. A search technique for clause interconnectivity graphs. *IEEE Transactions of Computers C-25,* 8 (August 1976), 823–835.

[St74] Stickel, M.E. The programmable strategy theorem prover: an implementation of the linear MESON procedure. Technical Report, Carnegie-Mellon University Computer Science Department, Pittsburgh, Pennsylvania, June 1974.

[Ti80] Tienari, M. On the definition of an attribute grammar. In Jones, N.D. (ed.) *Semantic-Directed Compiler Generation.* Proceedings of a Workshop, Aarhus, Denmark, January 1980. *Lecture Notes in Computer Science 94.* Springer-Verlag, Berlin, West Germany, 1980, pp. 408–414.

[Ty81] Tyson, W.M. *APRVR: A Priority-Ordered Agenda Theorem Prover.* Ph.D. Dissertation, University of Texas at Austin, Austin, Texas, 1981.

[Wa78] Walker, D.E. (ed.) *Understanding Spoken Language.* Elsevier, New York, New York, 1978.