[Mechanical Translation and Computational Linguistics, vol.8, nos.3 and 4, June and October 1965]

Machine Methods for Proving Logical Arguments Expressed in English*

by Jared L. Darlington, Research Laboratory of Electronics, Massachusetts Institute of Technology

This paper describes a COMIT *program that proves the validity of logical arguments expressed in a restricted form of ordinary English. Some special features include its ability to translate an input argument into logical notation in four progressively refined ways, of which the first pertains to propositional logic and the last three to first-order functional logic; and its ability in many cases to select the "correct" logical translation of an argument, i.e., the translation that yields the simplest proof. The logical evaluation part of the program uses a proof procedure algorithm that is an amalgam of the "one-literal clause rule" of Davis-Putnam and the "matching algorithm" of Guard. It is particularly efficient in proving theorems whose matrices in conjunctive normal form contain one or more one-literal clauses (atomic wffs), but it will also prove theorems whose matrices contain only polyliteral clauses. The program has been run on the I.B.M. 7094 computers at M.I.T. and utilizes the time-sharing facilities provided by Project* MAC *and the Computation Center.*

Introduction

A considerable amount of work has recently been done in the general area of automatic translation of ordinary language into the terminology of symbolic logic. We shall not attempt here to give a general description of this work, since it has already been summarized and discussed in some detail by R. F. Simmons in section 7 of his excellent report, “Answering English Questions by Computer: a Survey”[1]. Suffice it to say that no one has essayed the construction of a *general* logic translation program that would, taking account of all the amphibolies and polysemies of natural language, unambiguously parse any English sentence and translate it into the notation of symbolic logic. The syntactic and semantic problems involved are just as difficult, if not more so, than those of translating between natural languages. The existing logic translation schemes are based, therefore, on systems of restricted English, with limited grammars and vocabularies. They are, for all that, at least potentially quite useful for posing questions and submitting problems to computers in ordinary language, so long as the restrictions of the input language are simple and clear enough to be easily grasped by the user, and so long as provision is made for the user to correct his mistakes and rephrase his problem if he doesn't get it right the first time. In this connection, the time-sharing systems that are being installed in several computation centers are particularly useful, in that they permit the programming of error-detection devices that immediately reject ungrammatical sequences, misspelled words, etc., and allow the user sitting at a console to retype the problem in whole or in part.

The logic translation program developed by the present author differs from some of the others in placing primary emphasis on the evaluation of arguments, a traditional concern of the logician since the advent of the Aristotelian theory of the syllogism. An argument may be defined semantically as a group of propositions organized into premisses and conclusion, where the propositions that constitute the premisses provide evidence for the truth of the conclusion. Or an argument may be defined syntactically as a string of permissible sentences that are divided into premisses and conclusion by a syntactic marker, such as a word like 'therefore' or 'since'. Our program, for example, requires one of the sentences of the string to begin with 'therefore', and takes the sentence or sentences to the left of 'therefore' to be the premisses and those to the right to be the conclusion. This syntactic definition of 'argument' itself constitutes one of the restrictions of our input language, since there are many arguments that occur in ordinary language in which the order of premisses and conclusion is inverted, as in arguments of the form

p because q

or in which the relation between premisses and conclusion is not explicitly denoted by any connective words but is simply understood, as in

X is not expected to accompany the team on the next road trip. His ankle injury will probably keep him out of action for several more weeks.

* This work was supported in part by the Joint Services Electronics Program under contract DA36-039-AMC-03200(E); and in part by the National Science Foundation (Grant GN-244). An abbreviated version of the paper was read at the IFIP Congress 65 in New York City in May, 1965.

in which the second sentence states the evidence for the expectation expressed by the first sentence. This argument lies outside the scope of our program for another reason: its evaluation requires the techniques of inductive rather than deductive logic. Our program will prove arguments only if they are deductively valid, in the sense that to assume the premisses true and the conclusion false would be self-contradictory. A deductively invalid argument may of course be inductively valid, if the premisses provide good evidence for the conclusion, but we have not attempted to include a set of rules for testing the inductive validity of arguments, though the program could be adapted for this purpose.

Directly related to this emphasis on the evaluation of arguments is another difference between our program and the others, namely, the fact that our program must distinguish several "levels of analysis" or ways of translating the sentences of an input argument. A propositional logic analysis is entirely adequate to prove an argument like

> If Henry is a member of the Socialist Party (SP), then Henry is not a member of the Progressive Party (PP). Henry is a member of the PP. Therefore Henry is not a member of the SP.

which may be symbolized in propositional logic as

> p implies not-q, q, therefore not-p

but it will not suffice for an argument like

> All circles are figures. Therefore all who draw circles draw figures.2

which may be symbolized in first-order functional logic as

> (Ax) (Cx implies Fx). Therefore (Ay) ((Ez) (Cz & Dyz) implies (Ew) (Fw & Dyw)).

To symbolize this argument in terms of propositional logic would yield

> p, therefore q

which is clearly invalid. Our program, in fact, is capable of providing up to four progressively refined logical translations for an input argument. The first of these translations, "Analysis I," pertains to propositional logic, and the last three, "Analyses II, III and IV," to first-order functional logic. In Analysis I, each sentence or sentential clause is replaced by a single propositional letter, while in Analyses II, III, and IV, the sentences and sentential clauses are symbolized in terms of quantifiers, variables, individual constants, and unary, binary, and ternary predicates. In Analysis II, all nouns, adjectives, relative clauses, and prepositional phrases are symbolized as unary predicates and are replaced by terms of the form "P/.n," where 'n' denotes a numerical subscript of less than 500. Analysis III differs from II in employing binary and ternary predicates, i.e., two- and three-term relations, in addition to unary predicates. Transitive verbs, prepositions, and phrases like 'is greater than' and 'is a member of are treated as binary relations and are replaced by terms of the form "P/.n," where 'n' denotes a numerical subscript equal to or greater than 500, and verbs like 'gives' are treated as ternary relations if they are accompanied by an indirect object, while nouns and adjectives continue to be symbolized as unary predicates as in II. Analysis IV differs from II and III solely in its treatment of phrases like 'the king of France', i.e., definite descriptions. Analyses II and III regard such phrases as proper names and replace them by individual constants, i.e., terms of the form "IND/.n," while IV analyses them as asserting the unique existence of the subject referred to. Each of these four translations thus embodies more of the meaning of the input sentences than its predecessors, but in logical analysis the aim is not to express as much of the meaning as possible, as in translation between natural languages, but rather to discover how much of the meaning it is necessary to consider in order to prove the argument valid.

The fact that an argument may be logically symbolized in several different ways raises the question of which analysis should be selected to provide the input for the logical evaluation part of the program. Rather than starting the logical computation with the simplest analysis or the most detailed analysis, the program employs a criterion, based on the amount of repetition between the premises and conclusion, to decide which of the four analyses is likeliest to yield the simplest proof. This decision, however, is not final: if it appears that the argument as symbolized cannot be proven, the operator may interrupt the logical computation and direct the program to try proving a formula resulting from another analysis of the argument. This type of operator intervention is easily accomplished in the M.I.T. time-sharing system, into which the program has been incorporated.

In addition to permitting a considerable amount of operator control over the course of a running program, the use of time-sharing has, as we have discovered, several further advantages over batch processing. For example, it is quicker and easier using time-sharing to check out and debug new routines, take dumps, etc., and it is simpler to save and resume compiled programs. Time-sharing has one minor disadvantage insofar as our own program is concerned, which is that our program has grown too large for the COMIT time-sharing system to compile. We have therefore split up the program into three convenient sections, called "DA COMIT," "DB COMIT," and "DC COMIT," and designed to run consecutively. The three sections of the program have all been compiled and saved (and named "DA SAVED," "DB SAVED," and "DC SAVED," respectively), so one section may be resumed as soon as the previous section is finished, and the effect is that of running a single program; we shall therefore continue to speak of DA, DB, and DC as constituting one program. The three sections do correspond quite closely to natural divisions of the program, since DA does the look-up and parsing of the input sentences, DB does the logical translation of the parsed sentences, and DC does the logical evaluation of the resulting formulae. The division between DA, DB, and DC corresponds, up to a point, to Yngve's[3] conception of mechanical translation as requiring three principal stages, i.e., analysis of the input sentences, conversion of the structures of the input sentences into corresponding structures of the output language, and synthesis of the output sentences. Roughly speaking, DA and DB correspond to the first two of Yngve's three stages, but DC does not correspond to his third stage. Our program does not have to synthesize the output sentences, since validity is a matter of logical form or structure rather than content, and the evaluation routine DC operates solely on the logical forms of the sentences. We shall be discussing these three sections of the program in greater detail in the remainder of the paper.

Please note our use of quotation marks: throughout the paper we follow the convention for the use of single quotes (inverted commas) that is explained in W. V. Quine's *Mathematical Logic*[4], according to which a word, phrase, or sentence that is "mentioned" (as opposed to "used") is enclosed within single quotes, and the quotation is regarded as naming the entity within the quotes. For this reason, it is necessary to place any punctuation marks that are not actually part of the sequence named outside the single quotes, lest the punctuation marks be construed as part of the name of an entity. This convention accords with the current usage of many logicians, though it conflicts with the more journalistic policy of placing quotation marks outside commas and periods regardless of logic. We do, however, follow current journalistic procedure in placing double quotes, and single quotes that delimit quotations within quotations, outside commas and periods; and we occasionally omit quotes altogether where no ambiguity is likely to result.

Initial Stages of the Program—Lookup and Parsing

The operator at the time-sharing console starts the program by typing 'RESUME DA', or simply 'R DA'. He then proceeds to type in an argument. After the last sentence, he types 'DONE', which signals to the program that the input is finished. The program then proceeds to look up each word and punctuation mark of the argument in a dictionary, or "list rule," whose function is to supply subscripts specifying the syntactic class or classes to which a word may belong. There are nine principal syntactic classes, denoted by the literal subscripts

ADJN, CONJ, DET, NOT, P, PREP, PRNAME, RELPR, *and* VPOS.

The category ADJN comprises both adjectives and nouns, which may be lumped together since the logic translation routine regards both adjectives and nouns as unary predicates. An incidental advantage of this procedure is that it avoids parsing problems stemming from the fact that nouns frequently occur in adjectival positions, as in 'birthday present' (though it does not avoid the problem that many such expressions are idiomatic), or from the fact that adjectives frequently occur in nominal positions, as in 'none but the brave deserve the fair'. The category CONJ comprises the conjunctive words

and, iff (if and only if), implies, nor, or, *and* then.

('But' is regarded as a variant of 'and', and is changed to 'and' during the lookup.) The category DET comprises the five determiners

all, some, no, only, *and* the.

('Each' and 'every' are changed to 'all', and 'a' and 'an' are changed to 'some'.) The category NOT includes negative particles, of which 'not' is the only one employed at present. The category P comprises punctuative words, whose primary function is to separate sentences or sentential clauses. In addition to the conjunctive words, and the period and comma, the category P includes

both, either, if, neither, that (in the context 'implies that'), and therefore.

The remaining categories are as follows: PREP includes the prepositions, PRNAME includes the proper names, RELPR includes the relative pronouns, and VPOS includes both transitive and intransitive verbs. In addition to the nine primary syntactic categories, there are three secondary categories, so called because they figure only in a routine, directly following the dictionary lookup, that performs some verbal rearrangements and simplifications, and they are eliminated before the program enters the parsing routine. Of these three secondary categories, COMP denotes comparative particles like 'as', 'than', 'more', and 'less'; COMPADJ includes comparative forms of adjectives; and VAUX includes auxiliary verbs, like 'will', 'have', and 'do'. The vocabulary that the program employs is chosen mainly from the examples that are submitted to the program. It is, however, unnecessary to recompile the program every time it is desired to submit an argument with new vocabulary, since words that are not found in the program's dictionary may be typed directly into the workspace from the console, along with their appropriate subscripts. A word thus typed in goes onto a supplementary shelf, where it may be found if it recurs in the argument. This supplementary dictionary does not become a permanent addition to the dictionary of the compiled program, so if it is planned to use the new vocabulary at all frequently, it is better to recompile the program with the new words added to the list rule. The dictionary has been simplified by listing only the singular forms of regular nouns and the infinitives of regular verbs, so if a word is not found in the dictionary the program (employing a variant of the method of "longest match") reduces it to a singular noun or a verbal infinitive, if possible, and looks it up again. Nouns remain in the singular, since the determiner of a noun provides the translation routine with enough information about number (logically speaking, 'all man' is just as good as 'all men'), and verbs remain in the present infinitive, thereby facilitating the reduction of certain verbal forms to others, as will be explained later on, when we discuss propositional logic translation. The dictionary lookup and syntactic subscripting procedures are summarized in the following outline.

OUTLINE OF THE DICTIONARY LOOKUP AND SYNTACTIC SUBSCRIPTING ROUTINE

Input shelf is Shelf 9, output shelf is Shelf 2, supplementary dictionary is Shelf 100.

```
1. Start. Read in next word, W, from input shelf.
 1.1. Succeed: go to 2.
 1.2. Fail: DONE.

2. Look up W in list.
2.1. Succeed: put appropriate subscripts (/ADJN, /DET, /CONJ, etc.) on W; queue W onto output shelf; go to 1.
2.2. Fail: look up W in supplementary dictionary.
     Succeed: go to 2.1.
     Fail: does W end in 'ies' or 'ied'?
       Yes: change 'ies' ('ied') to 'y'; go to 2.
       No: does W end in 's'?
         Yes: go to 3.
         No: does W end in 'd'?
           Yes: go to 3.
           No: does W end in 'e'?
           Yes: if W results from deletion of final 'd' or 's', go to 3. If not, go to 4.
           No: does W end in a double consonant?
             Yes: if W results from deletion of final 'ed', go to 3. If not, go to 4.
             No: go to 4.

3. Delete final letter of W; go to 2.

4. Ask operator, "What part of speech is W?" Operator responds by typing in an item of the form
   —/SUB +
```

where 'SUB' denotes one of the nine principal syntactic categories ADJN, DET, etc. (The plus sign has no significance other than the fact that the COMIT "format s input," which allows input items to be subscripted, requires that each input item be followed by the punctuation mark '+'.) The program then creates the item

W/SUB

and adds it to the supplementary dictionary. In some cases the operator must retype W; e.g., if W is 'sold', an irregular past tense verbal form, the operator types

SELL/VPOS+

in order to reduce it to the present infinitive. The program does this automatically for past tenses of regular verbs.

When 4 is finished, go to 1.

After all the words and punctuation marks of the input sentences have been subscripted, the program performs a series of verbal rearrangements and simplifications which, for want of a better word, we may call "transformations." These transformations are essentially of six types, and are performed in the following order.

(1) Structures of the form

$1/COMP + $1/ADJN + $1/COMP

and

$1/COMPADJ + $1/COMP

e.g.,

AS/COMP + GREAT/ADJN + AS/COMP, MORE/COMP + TALL/ADJN + THAN/COMP, GREATER/COMPADJ + THAN/COMP,

are compressed into one word and are given the subscript /COMPADJ, thereby becoming

ASGREATAS/COMPADJ, MORETALLTHAN/COMPADJ, GREATERTHAN/COMPADJ,

etc. (The '$1' symbol in COMIT denotes any single constituent.)

(2) The verbal auxiliaries WILL/VAUX, HAVE/VAUX, DO/VAUX, etc., are eliminated, and any negative particles are placed after their verbs. For example,

WILL/VAUX + COME/VPOS, HAVE/VAUX + COME/VPOS, DO/VAUX + COME/VPOS,

etc., are reduced to COME/VPOS, and

WILL/VAUX + NOT/NOT + COME/VPOS, HAVE/VAUX + NOT/NOT + COME/VPOS, DO/VAUX + NOT/NOT + COME/VPOS,

etc.. are reduced to COME/VPOS + NOT/NOT. Any verbal auxiliary that is not accompanied by a main verb is itself taken as a main verb, and has its subscript /VAUX replaced by /VPOS.

(3) Structures of the form

IS/VPOS + $1/COMPADJ

and

IS/VPOS + NOT/NOT + $1/COMPADJ

delete the IS/VPOS and change the subscript/COMPADJ to /VPOS. For example, IS/VPOS + GREATERTHAN/COMPADJ, AND IS/VPOS + NOT/NOT + ASGREATAS/COMPADJ

are converted into

GREATERTHAN/VPOS, and ASGREATAS/VPOS + NOT/NOT.

(4) Structures of the form

$1/VPOS + $1/COMPADJ, AND $1/VPOS + NOT/NOT + $1/COMPADJ

have the $1/VPOS and the $1/COMPADJ compressed into one word, which is subscripted with /VPOS. For example,

RUN/VPOS + ASFASTAS/COMPADJ, AND SEE/VPOS + NOT/NOT + FARTHERTHAN/COMPADJ,

are converted into

RUNASFASTAS/VPOS, AND SEEFARTHERTHAN/VPOS + NOT/NOT.

(5) Structures of the form

$1/VPOS + $1/PREP,

and

$1/VPOS + NOT/NOT + $1/PREP

have the $l/VPOS and the $1/PREP temporarily compressed and looked up in a special dictionary to see whether they can form a single relation. If so, they remain compressed, and are subscripted with /VPOS. For example,

STOP/VPOS + IN/PREP

and

GET/VPOS + NOT/NOT + TO/PREP

become

STOPIN/VPOS

and

GETTO/VPOS + NOT/NOT,

while

OWN/VPOS + TO/PREP

remains uncompressed.

(6) Finally, the dummy word

ONE/ADJN

is inserted in a couple of special cases, in order to facilitate the subsequent parsing. For example,

THERE + IS/VPOS

becomes

SOME/DET + ONE/ADJN + IS/VPOS,

and any determiner not directly followed by a $1/ADJN is provided with ONE/ADJN. For example,

ALL/DET + WHO/RELPR + DRAW/ADJN, VPOS + CIRCLE/ADJN, VPOS

becomes

ALL/DET + ONE/ADJN + WHO/RELPR + DRAW/ADJN, VPOS + CIRCLE/ADJN, VPOS.

As a result of the dictionary lookup and preliminary transformations, each item of the input text should be subscripted with one or more of the subscripts denoting the nine principal syntactic categories. Any secondary subscripts should have disappeared by this time, but if any remain, they will cause the program to stop with an appropriate error comment. The next step is to parse the input sentences according to the following grammar, which is presented in the exact form in which it appears in the program, i.e., as a list rule, or dictionary of symbols. The COMIT notation, which the program employs, is explained in greater detail in *An Introduction to* COMIT *Programming*[5] and COMIT *Programmers' Reference Manual*[6]. A good informal presentation is "A Programming Language for Mechanical Translation"[7], by V. H. Yngve.

GRAMMAR OF THE PROGRAM, IN THE FORM OF A COMIT LIST RULE

```
–P05 S = NP +V + OR + NP + VP*0 + OR + NP + VP*1+ *(+ –/DET–
        +–/ADJN+–/PRNAME    *
     SNOVP = NP + *( + –/DET+ –/ADJN+ –/PRNAME    *
     SNONP = V + OR + VP*0 + OR + VP*L + *(+ –/VPOS    *
     NP= – /PRNAME + OR + NP*0 + OR + NP*1 + *( + –/DET–
         +–/ADJN+–/PRNAME    *
     NP*0=ADJNCL + OR + NP*2 + *(+–/ADJN    *
     NP*L=–/DET + NP*0+*(+–/DET    *
     NP*2 = ADJNCL + RELCL + OR + ADJNCL + PPCL–
            + *(+– /ADJN    *
     ADJNCL= –/ADJN + OR + ACL*0 + OR + ACL*L–
              + *(+–/ADJN    *
     ACL*0 = – /ADJN + ADJNCL + *(+–/ADJN    *
     ACL*L=–/ADJN + ACL*2 + *(+–/ADJN    *
     ACL*2 = – /CONJ + ADJNCL + *( + – /CONJ    *
     VP*0 = V + NP + *( + –/VPOS    *
     VP*L=VP*0 + PPCL+*(+–/VPOS    *
     V = – /VPOS + OR + VNEG + *(+– /VPOS    *
     VNEG = – /VPOS + – /NOT + *( + – /VPOS    *
     IVP=NP + V+*(+ –/DET + –/ADJN+ –/PRNAME    *
     RELCL = RCL*1 + OR + RCL*2+*(+–/RELPR    *
     PPCL=PPCL*L + OR + PPCL*2 + *( + –/PREP    *
     RCL*L = RCL*2 + RCL*3 + *(+–/RELPR    *
     PPCL*L =PPCL*2 + PPCL*3 + *(+ –/PREP    *
     RCL*2 = –/RELPR + V + OR + – /RELPR + VP*0 + OR–
              + –/RELPR + VP*1 + OR + – /RELPR + IVP–
              + *(+– /RELPR    *
     PPCL*2 =– /PREP + NP+*(+ –/PREP    *
     RCL*3 =– /CONJ + RELCL + *( + –/CONJ    *
     PPCL*3 = – /CONJ + RELCL + *( + – /CONJ    *
```

The left half of each list subrule of P05 is a symbol of the grammar, and the right half of each rule gives all the ways of rewriting the symbol in the left half. If there are more than one expansion for a symbol, they are separated by OR. At the end of each rule is a * (followed by one or more terms of the form —/SUB. These items denote all the possible initial words of the possible expansions. Thus, the symbol SNONP may be rewritten as V or VP*0 or VP*l, but any clause of these three types must begin with a lexical item of the form $1/VPOS. This information is included in the right half of each rule because it enables the parsing routine to be written more efficiently than otherwise—if a sentence is being parsed and the next lexical item to be accounted for is an ADJN, then the next structure could not possibly be a V, VP*0, or VP*l, or, for all that, an SNONP. The asterisk at the far right of each list subrule is the go-to; in COMIT, if a rule or subrule bearing the asterisk go-to is successfully executed, then control passes to the next rule (not subrule) in sequence.

The parsing program will parse complete sentences (denoted by S), "sentences" lacking a main verb phrase (denoted by SNOVP), and "sentences" lacking a main noun phrase (denoted by SNONP). All three types are illustrated by the compound sentence

Jack and Jill goup the hill and godown the hill.
(Jack and Jill go up the hill and go down the hill.)

whose parsing will treat 'Jack' as an SNOVP, 'Jill goup the hill' as an S, and 'godown the hill' as an SNONP. A routine directly following the parsing expands SNOVP's into S's, by borrowing the main verb phrases from the immediately following S's and SNONP's, and expands SNONP's into S's, by borrowing the main noun phrases from the immediately preceding S's and SNOVP's. The sample sentence will then be expanded into

Jack goup the hill and Jack godown the hill and
Jill goup the hill and Jill godown the hill.

In addition to parsing S's, SNOVP's and SNONP's, the parsing routine has the task of determining the beginnings and ends of these structures. It assumes that a sentence or sentential clause begins with the first non-P word (i.e., the first word not bearing the subscript /P) that it encounters, and it stops with the longest sentence or sentential clause directly followed by a P-word that it can find.

The parsing routine is a straightforward program that attempts to generate all the sentences of the grammar from left to right by successively applying the phrase structure rules to the expansion of symbols, thereby generating successive word-class symbols that are matched against the words of the input sentence. If a word-class symbol matches the corresponding word in the input sentence, the sentence is provisionally accepted, but if they do not match, the analysis is rejected. The proposed parsings, or partial analyses, of the input sentence are stored in pushdown form on Shelf 1. Each analysis is of the form

.........+ *Q/.n + X +.................. + **

in which the part of the formula to the left of the marker *Q has already been found to be compatible with the sentence being parsed, the numerical subscript /.n on *Q is the number of words taken account of so far increased by 1, X is the next symbol to be tested, the part of the formula between X and ** is the proposed parsing for the rest of the sentence, and the marker ** denotes the end of the analysis and separates it from the other analyses on the same shelf. An analysis is read in from Shelf 1, and the symbol x directly to the right of *Q is tested. If X is a word-class symbol, it will be of the form —/SUB, where SUB may be an ADJN, DET, etc., and the next word (nth word) of the sentence is looked at to see whether it has the subscript /SUB. If it does, then the analysis is confirmed, any subscripts other than SUB on the word are deleted, the marker *Q is moved to the right of the next symbol, the numerical subscript /.n on *Q is increased by 1, and the analysis is stored at the front of Shelf 1. If, however, the word does not have the subscript —/SUB, then the analysis is invalidated. If the symbol X directly to the right of *Q is not of the form —/SUB, then it is looked up in the list P05 to determine its possible expansions, a new analysis is created for each expansion, the marker *Q is moved to the right of the symbol expanded, and the new analyses are stored at the front of Shelf 1. This procedure is described in greater detail in the following outline.

OUTLINE OF THE PARSING ROUTINE

Shelf 9 is input shelf, Shelf 6 is output shelf, Shelf 1 is for the partial parsings, Shelf 8 is for the complete parsings, Shelf 4 is for all the expansions of a given symbol X under analysis, and Shelves 2, 3, and 5 are for temporary storage of parts of the formula under analysis.

1. *Start.* Has first item of Shelf 9 a /P subscript?
1.1. *Yes:* delete any numerical subscript; queue item onto Shelf 6; go to 1.
1.2. *No*: is Shelf 9 empty?

1.21. *Yes:* DONE.
1.22. *No*: subscript first item of Shelf 9 with /.1, second item with /.2, etc.; initialize Shelf 1 With *Q/.1 + SNONP + ** + *Q/.1 + SNOVP + ** + *Q/.1 + S + **; go to 2.

2. Read in from Shelf 1 up to and including first **.
2.1. Succeed: locate item of Shelf 9 with same numerical subscript as *Q in workspace; make a copy of this item, and place it at front of Shelf 9; queue everything up to but not including *Q onto Shelf 3; go to 3.
2.2. Fail: go to 8.

3. Is *Q directly followed by an item of the form —/SUB?

3.1. *Yes:* move *Q to right of —/SUB; insert first item on Shelf 9 between them. This results in a sequence of the form

—/SUB + W/SUB2 + *Q/.n

Go to 4.

3.2. *No*: *Q is directly followed by a symbol, say X. Move *Q to right of X; queue X + *Q onto Shelf 3, leaving copy of X in workspace; store remainder of formula temporarily on Shelf 2; go to 6.

4. Is — /SUB1 equal to, or a part of, SUB2?

4.1. *Yes:* formula is a possible parsing; go to 5.

4.2. *No*: delete workspace and Shelf 3; go to 2.

•5. Is *Q directly followed by **?

•5.1. *Yes:* formula is a complete parsing. Delete *Q; queue formula in workspace onto Shelf 3; transfer parsed sentence from Shelf 3 to Shelf 8; go to 2.

5.2. *No*: formula is a partial parsing. Queue workspace onto Shelf 3; transfer formula from Shelf 3 to front of Shelf 1; go to 2.

6. Look up X in list P05; store part of formula up to but not including * ((i.e., the possible expansions of X) on Shelf 4; delete *(. The items —/SUB remaining in the workspace denote possible initial words of structures on Shelf 4. Read in next item, W, from Shelf 9. Do any of the items —/SUB in the workspace have the same literal subscript as W?

6.1. *Yes:* parsing is legitimate so far; go to 7.

6.2. *No*: parsing is illegitimate; clear workspace, and Shelves 2, 3, and 4; go to 2.

7. Read in next expansion of X from Shelf 4.

7.1. *Succeed:* store expansion on Shelf 5; assemble partial parsing as follows: copy of Shelf 3 + Shelf 5 + copy of Shelf 2; shelve resulting formula onto front of Shelf 1; go to 7.

7.2 *Fail:* clear Shelves 2 and 3; go to 2.

8. Find last word, w, in workspace that occurs before a \$1/P; record the numerical subscript /.n of W; erase formula in workspace up to and including w; shelve everything after w onto front of Shelf 9; determine which parsing(s) on Shelf 8 take account of exactly n words, and discard the others. Are there any parsings left?

8.1. *Yes:* go to 9.

8.2. *No*: stop with error comment.

9. Is there exactly one parsing?

9.1. *Yes:* go to 10.

9.2. *No*: give each parsing a number, and ask operator which one he wants. Operator responds by typing

–/.n+

where n is the number of the desired parsing. Go to 9.1.

10. Check formula for wellformedness, using SCOPE routine (described below). Is formula wellformed?

10.1. *Yes:* queue formula, followed by *), onto Shelf 6; go to 2.

10.2. *No*: stop with error comment.

A typical sentence that the program has parsed is

All who support Ickes will vote for Jones.

which is a paraphrase of 'Whoever supports Ickes will vote for Jones', the first sentence of an example from I.M. Copi's *Symbolic Logic*[8]. The parsing is given below.

S + NP + NP*1 + ALL/DET + NP*0 + NP*2 + ADJNCL + ONE/ADJN + RELCL + RCL*2 + WHO/RELPR + VP*0 + V + SUPPORT/VPOS + NP + ICKES/PRNAME + VP*0 + V + VOTEFOR/VPOS + NP + JONES/PRNAME + *) + ./P

The SCOPE routine that the program employs serves the primary purpose of determining the extent of a formula or section of a formula, and the secondary purpose of testing the wellformedness of a formula. Directly following the parsing routine, each symbol of the parsed formula is given a numerical subscript through a list lookup (any old numerical subscripts are automatically deleted), as follows: each symbol that is expanded into two symbols is given the numerical subscript /.1 (these include S, NP*1, NP*2, ACL*0, ACL*1, ACL*2, VP*0, VP*1, VNEG, IVP, RCL*1, PPCL*1, RCL*2, PPCL*2, RCL*3, PPCL*3); and each symbol that is rewritten as one symbol is given the subscript /.0 (these include SNOVP, SNONP, NP, NP*0, ADJNCL, V, RELCL, PPCL). The remaining symbols are all lexical items, and are given the subscript /.32767 (equal to minus one, mod 2^{15}). The SCOPE routine determines the scope of a symbol X by putting the marker —/.1 immediately to the left of X, and then reading from left to right. Each item W encountered in the left-to-right search raises the subscript on the marker by the numerical subscript on W. The search ends when the count goes to zero. The essence of the SCOPE routine is the one-rule loop

```
SCOPE $0 + $l/.G0 + $1 = 2/.1.*3 + 3 //*Q7 2 SCOPE
```

The \$0 finds the left end of the workspace; the \$l/.G0 finds the marker, so long as its subscript is greater than zero; and the \$1 finds the item directly to the right of the marker. The loop can terminate in either of two ways, namely, if the count on the marker goes to zero, or if the workspace becomes empty except for the marker. The second contingency constitutes an error condition, indicating that the formula does not contain enough lexical items, so it is necessary to check the workspace after the failure of the loop to see whether the count actually has gone to zero. The SCOPE routine may thus be used to test the wellformedness of a parsed sentence, as follows: after the loop terminates, test whether count has gone to zero. If not, formula contains too few words, and is illformed. If so, check whether any words remain in workspace. If so, formula contains too many words, and is illformed. If not, formula is wellformed.

Propositional Logic Translation

Once the input argument is parsed, and all the SNOVP's and SNONP's have been expanded into complete s's, the program attempts a propositional logic analysis of the argument. This involves replacing each s and its corresponding sentence by a different propositional symbol, A/V, B/V, C/V, etc. Identical sentences are replaced by the same propositional symbol, and contradictory sentences, i.e., sentences that differ only in that the main verb of one is followed by a NOT are replaced by contradictory symbols, e.g., A/V and A/V, NOT. (The SCOPE routine can be used to find the main verb of any sentence, by first finding the main verb phrase, whether it be V, VP*0, or VP*l, and then finding the first verb of the main verb phrase. The main verb thus located is subscripted with /MAIN.) The criterion of synonymy that the program employs, i.e., that of complete identity in wording and word-order, is on the face of it extremely strict, but its effects are somewhat mitigated by the initial dictionary lookup and its ensuing "tranformations," which frequently reduce two apparently different sentences to the same wording and word-order. All verbal forms, as previously noted, are reduced to the present infinitive. This may be justified by the consideration that verbal tenses are largely irrelevant to the statement of logical implications. For example, the idea (or proposition) that the butler's presence implies his being seen may be expressed in a wide variety of ways, some of which are obtainable by substituting different forms of the verb 'to be' in the sentential pattern

If the butler ——present then he ——— be seen.

Some of the possible substitutions are the pairs 'were', 'would be'; 'had been', 'would have been'; and 'be', 'will be'. They may all be regarded as variants of the basic implication

If the butler be present then he (the butler) be seen.

The propositional logic translation routine may be illustrated by the following example, which is a paraphrase of an example from I. M. Copi's *Introduction to Logic* [9], and has been successfully processed by our program.

If I buy a new car or fix my old car then I'll get to Canada and stop in Duluth. If I stop in Duluth then I'll visit my parents. If I visit my parents then I'll stay in Duluth but if I stay in Duluth then I'll not get to Canada. Therefore I'll not fix my old car.

The lookup and parsing transform this argument into the following:

If I buy some new car or I fix my old car then I getto Canada and I stopin Duluth. If I stopin Duluth then I visit my parents. If I visit my parents then I stayin Duluth and if I stayin Duluth then I getto not Canada. Therefore I fix not my old car.

Replacement of sentences by variables yields:

If A/V or B/V then C/V and D/V. If D/V then F/V. If F/V then H/V and if H/V then C/V,NOT. Therefore B/V,NOT.

in which

A/V	= I buy some new car
B/V	= I fix my old car
C/V	= I getto Canada
D/V	= I stopin Duluth
F/V	= I visit my parents
H/V	= I stayin Duluth

At this stage, the decision whether to go further with the propositional logic analysis is made, the criterion being that, if one or more propositional letters occur both in the premisses and in the conclusion, then the propositional logic routine is carried out to its conclusion, but if there is no such repetition of terms, then the assumption is made that the propositional logic analysis could not possibly be successful, and the program proceeds with the functional logic analyses, i.e., Analyses II, III, and IV. The particular example under consideration does, however, pass the test, since the term B occurs both in the premisses and in the conclusion, so the partially translated argument is converted into a fully parenthesized formula of propositional logic, i.e.

((((((A)OR(B))IMPLIES((C)AND(D)))AND((D)IMPLIES (F)))AND(((F)IMPLIES(H))AND((H)IMPLIES (NOT(C)))))IMPLIES(NOT(B)))

This involves the application of a set of rules for the insertion of parentheses in such a way that the scope of every C-word (i.e., word corresponding to a logical connective) is made perfectly precise. For sentences containing fewer than two binary connectives, this problem is trivial: P becomes (P), and P AND Q becomes ((P) AND (Q)). A great many sentences containing two or more binary connectives likewise involve no difficulty; e.g., IF P, THEN Q OR R becomes ((P) IMPLIES ((Q) OR (R))), and P AND EITHER Q OR R becomes ((P) AND ((Q) OR (R))). There do, nonetheless, exist ambiguous or borderline cases, such as P AND Q OR R, concerning which it is useless to lay down general rules, except perhaps the rule that the input language should be restricted so as to exclude them. Ambiguous sentences or clauses are characterized by the fact that they do not contain sufficient clues or indications as to where to place the parentheses. These clues (of which the unambiguous clauses contain a sufficiency) are of several types. They include:

(i) relative strength of connectives
(ii) placement of "groupers," i.e., IF, BOTH, EITHER, and NEITHER.
(iii) placement of punctuation marks, such as commas and periods; and
(iv) "symmetry" of connectives.

As for (i), in a sentence like P IMPLIES Q AND R, the AND may be said to be "stronger" than the IMPLIES, in that the Q and R are bound together more strongly by the AND than are the P and the Q by the IMPLIES, resulting in ((P) IMPLIES ((Q) AND (R))) as the natural grouping. As for (ii) and (iii), the amphiboly of P AND Q OR R may be resolved either by employing a grouper, as in P AND EITHER Q OR R, or by inserting a comma, as in P, AND Q OR R, and in P AND Q, OR R. Or a combination of groupers and commas may be used. (Apropos, employing the grouper BOTH would not materially affect this example, as BOTH P AND Q OR R is still ambiguous.) Point (iv) is perhaps the hardest to formalize, but it is exhibited in clauses like P IMPLIES Q OR R IMPLIES S, and P OR Q AND R OR S, in which the middle connective seems to be the fundamental one regardless of the intrinsic "strength" of the connectives. This factor of symmetry apparently operates most strongly in clauses containing three connectives in which the two "outer" connectives are the same, but may differ from the "inner" one. It is debatable, though, whether the notion of symmetry of connectives can be extended beyond, or even as far as, clauses containing five connectives.

Our program exploits all four types of clues, and incorporates them into a set of rules for the placement of parentheses (see below). These rules are applied in sequence to a sentence or clause until the main connective is located. Two more clauses are then marked off, i.e., that to the left of the main connective and that to the right of it. The leftmost clause is then subdivided in the same way into two new clauses. This procedure is repeatedly applied until all the clauses are fully parenthesized, where the criterion of full parenthesization is that every connective occur in the context '). . .('. If the program fails to find the main connective of a given clause, it concludes that the clause is ambiguous, prints it out with a comment to that effect, and proceeds to parenthesize the rest of the sentence.

The rules for parenthesizing and grouping are stated in the following outline.

OUTLINE OF THE PARENTHESIZING AND GROUPING ROUTINE

The rules listed below are applied in sequence to an initially parenthesized clause "C," until the basic connective of c has been found.

1. If C contains no C-words, C is assumed to be fully parenthesized.
2. If C contains exactly one C-word, the one C-word is basic. Furthermore, if the one C-word is NOR, i.e., if C is of the form NEITHER+P+NOR+Q, then C is replaced by a clause of the form ((P) AND (Q)).
3. If C contains exactly one C-word directly preceded by a comma, that C-word is basic, unless it occurs between IF and THEN.
4. If C contains exactly three C-words, and if C is "symmetrical," then the middle C-word is basic. Furthermore, if C is of the form NEITHER P * Q NOR R * S, where * may be AND, OR, IMPLIES, or IFF, then C is replaced by a clause of the form ((NOT(P * Q)) AND (NOT(R * S))).
5. If all the C-words in C are AND, or if all the C-words in C are OR, then the first C-word is basic.
6. If C contains an AND+IF, not occurring between IF and THEN, then the AND is basic, unless C also contains an OR+IF not occurring between IF and THEN.
7. If C contains an AND+EITHER or an AND+NEITHER, then the AND is basic, unless it is preceded by an IF.
8. If C contains an OR+IF, not occurring between IF and THEN, then the OR is basic, unless C also contains an AND+IF not occurring between IF and THEN.
9. If C contains an OR+EITHER or an OR+NEITHER, then the OR is basic, unless it is preceded by an IF.
10. If C is of the form EITHER...............OR Q, then the last OR is basic.
11. If all the C-words in C are NOR, C is converted into an equivalent formulation employing NOT and AND, and the first AND is basic.
12. If C is of the form NEITHER .... NOR Q, then C is replaced by a clause of the form ((NOT ( )) AND (NOT(Q))).
13. If C contains exactly one IMPLIES+THAT, the IMPLIES is basic, unless it is preceded by an IF.
14. If C contains exactly one IMPLIES, the IMPLIES is basic, unless it is preceded by an IF.
15. If C contains exactly one IFF, the IFF is basic, unless it is preceded by an IF.
16. If C contains a THEN, the THEN is basic. The IF . . . THEN is replaced by IMPLIES.

At the conclusion of the parenthesization, the formula is "tidied up" by erasing all superfluous groupers, i.e., all P-words that are not C-words.

In the argument used to illustrate propositional logic translation, the partially translated formula is converted into a fully parenthesized formula of propositional logic, through application of the above set of rules, as follows.

*****

(If A/V OR B/V THEN C/V AND D/V) (Input)
((A/V OR B/V) IMPLIES (C/V AND D/V)) (Rule 16)
(((A/V) OR (B/V)) IMPLIES (C/V AND D/V)) (Rule 2)
(((A/V) OR (B/V)) IMPLIES ((C/V) AND (D/V))) (Rule 2)

*****

(IF D/V THEN F/V) (Input)
((D/V) IMPLIES (F/V)) (Rule 2)

*****

(IF F/V THEN H/V AND IF H/V THEN C/V,NOT) (Input)
((IF F/V THEN H/V) AND (IF H/V THEN C/V,NOT)) (Rule 4)
(((F/V) IMPLIES (H/V)) AND (IF H/V THEN C/V,NOT))
(Rule 2)
(((F/V) IMPLIES (H/V)) AND ((H/V) IMPLIES (C/V,NOT)))
(Rule 2)

* * * * *

(B/V,NOT) (Input)
(B/V,NOT) (Rule 1)
* * * * *

The fully parenthesized formulae corresponding to the sentences of the argument are combined into a single formula of implicational form, according to the following procedure. The sentences left of THEREFORE are taken to be the premisses, and are separated from those to the right of THEREFORE, which are taken to be the conclusion. If there are more than one premiss, e.g.,

(P1). (P2). (P3)........

they are combined into the formula

(((P1) AND (P2)) AND (P3)) ..

The sentences of the conclusion are combined in the same way. Finally, the premisses are combined with the conclusion, by changing THEREFORE to IMPLIES, and putting a set of parentheses around the entire formula, i.e.,

(Premisses) THEREFORE (Conclusion)

become

((Premisses) IMPLIES (Conclusion))

The fully parenthesized formula is next tested for validity, using the Wang propositional calculus algorithm[10]. The principal proof procedure that the program employs is a combination of the "one-literal clause rule" of Davis-Putnam[11] and the "matching algorithm" of Guard[12], and it forms the body of the DC section of the program. As it is desired to obtain an immediate verdict as to the validity of the propositional logic formulation, and as it is inconvenient to switch over to DC and back to DA again, since they are compiled separately, the Wang algorithm is employed to test the propositional logic formulae for validity. It provides a short and neat test of validity, and it is easy to stick onto the end of the propositional logic translation routine. It requires that the formula to be tested be in Polish prefix notation, and our program accomplishes this conversion by means of a short routine that is a modification of a method devised by Yngve. This routine is described below.

OUTLINE OF ROUTINE FOR TRANSLATING A
FULLY PARENTHESIZED FORMULA INTO
POLISH PREFIX NOTATION

Shelf 1 is output shelf; Shelf 2 is input shelf; input formula is stored in expanded form on Shelf 2.

```
1. Read in next item from Shelf 2.
   Succeed: go to 2.
   Fail: DONE.

2. Is item a *) ?
   Yes: erase it; erase first *( on Shelf 1; go to 1.
   No: is it a binary connective?
      Yes: place it directly left of first *( on Shelf 1; go
      to 1.
      No: store it at front of Shelf 1; go to 1.
```

This routine leaves the formula in reverse Polish notation. It is, however, a simple matter to reverse it back again. The formula of our example then becomes

IMPLIES + AND + AND + AND + IMPLIES + A/V + B/V + IMPLIES + C/V + D/V + IMPLIES + D/V + F/V + AND + IMPLIES + F/V + H/V + IMPLIES + H/V + NOT + C/V + NOT + B/V

The formula is now ready to be tested by the Wang algorithm, and the answer 'valid' is readily obtained.

The programming of the Wang algorithm and the more extensive proof procedure algorithm employed in section DC of the program illustrate the wide applicability of COMIT. Originally designed as a programming language for mechanical translation[7], it has also proved useful for nonlinguistic types of problems, and is no less efficient in this area than many other list-processing languages. Our program for the Wang algorithm runs quite rapidly, and proves reasonably long formulae in one or two seconds or less. Our proof procedure program for functional logic runs less rapidly, but this is attributable to the greater difficulty of proving theorems in functional logic rather than to any deficiency in COMIT. These proof procedure programs are described in greater detail in the section entitled "Methods of Logical Evaluation."

If the propositional logic routine gives the answer 'valid' for a formula, then the program stops. If, however, the answer 'invalid' is given, or if the earlier test for the feasibility of a propositional logic analysis was negative, then the parsed argument is written out into "Channel A" (actually called "A CHANEL"), from where it is read in at the start of the next section of the program, i.e., DB.

Functional Logic Translation

Section DB of the program, which translates the parsed arguments provided by DA into functional logic notation, is based on the interaction of three principal routines, i.e., "PHI," "SFORM," and "LF." The routine PHI determines the sentence or part of a sentence that should be analysed next, SFORM converts this string into a quasi-logical formula, and LF translates the quasi-logical formula into a complete formula of functional logic. We shall first give an example of the procedure, and then discuss it in detail.

> All who support Ickes will vote for Jones. Everyone whom Anderson will vote for is a friend of Harris. Jones is a friend of no one who is a friend of Kelly. Harris is a friend of Kelly. Therefore Anderson will not support Ickes.

The first sentence of this argument, which is a paraphrase of an example from I. M. Copi's *Symbolic Logic*[8] was used in a previous example. As pointed out earlier, a ONE/ADJN was inserted between 'All' and 'who', the 'will' was deleted, and the 'vote for' was compressed to form a new verb, 'votefor'. At the start of Analysis II, an additional change is made, i.e., all the words of the predicate are compressed into a single symbol, which is regarded as an intransitive verb. The parsed sentence is thereby changed into the form given below, complete with subscripts.

S/.1 + NP/.0 + NP*1/.1 + ALL/.32767,DET + NP*0/.0 + NP*2/.1 + ADJNCL/.0 + ONE/.32767,ADJN + RELCL/.0 + RCL*2/.1 + WHO/.32767,RELPR + VP*0/.1 + V/.0 + SUPPORT/.32767,VPOS + NP/.0 + ICKES/.32767,PRNAME + V/.0 + VOTEFORJONES/.32767,VPOS,MAIN

The routine SFORM then determines the quasi-logical form of the parsed sentence, i.e.,

All + X/A + PHI/.1,A + P/.2,A

("All A such that PHI/.1,A is P/.2,A.")

in which

PHI/.1 = NP*0/.0 + NP*2/.1 + ADJNCL/.0 + ONE/.32767,ADJN + RELCL/.0 + RCL*2/.1 + WHO/.32767,RELPR + VP*0/.1 + V/.0 + SUPPORT/.32767,VPOS + NP*0/.0 + ICKES/.32767,PRNAME

and

P/.2 = VOTEFORJONES/.32767,VPOS,MAIN

Each PHI, followed by the string that it denotes, is stored on Shelf 9. Also, each IND/.n, followed by the proper name that it denotes, is stored on Shelf 16; each P/.n (n less than 500), followed by the unary predicate that it denotes, is stored on Shelf 17; and each P/.n (n equal to or greater than 500), followed by the binary or ternary predicate that it denotes, is stored on Shelf 18. Shelf 17 is initialized with

P/.1 + ONE + P/.0 + IS

so the unary predicate VOTEFORJONES is denoted by P/.2, whose numerical subscript is greater by one than that of the largest P already on Shelf 17.

The routine LF converts the quasi-logical formula into a complete formula of functional logic, i.e.,

(A/Q X/A)((PHI/.1,A) IMPLIES/OP (P/.2,A))

The translation, however, is not finished until all the PHI's have been replaced by complete logical formulae. The PHI routine reads in PHI/.1,A + (etc.) from Shelf 9, and replaces it by

((P/.1.A) AND/OP (PHI/.1,A))

in which

P/.1 = ONE

and

PHI/.1=RELCL/.0 + RCL*2/.1 + WHO/.32767,RELPR + VP*0/.1 + V/.0 + SUPPORT/.32767,VPOS + NP*0/.0 + ICKES/.32767,PRNAME

This substitution is made in the partially translated formula, which then becomes

(A/Q X/A)(((P/.1,A) AND/OP (PHI/.1.A)) IMPLIES/OP (P/.2.A))

The routines SFORM and LF next convert PHI/.1,A into P/.3,A, in which

P/.3 = SUPPORTICKES

so the complete translation of the first premiss, resulting from Analysis II, is

(A/Q X/A)(((P/.1,A) AND/OP (P/.3,A)) IMPLIES/OP (P/.2,A))

Finally, the formula is simplified by eliminating the dummy term P/.1,A, yielding

(A/Q X/A)((P/.3,A) IMPLIES/OP (P/.2,A))

as the final version.

Analysis III produces a more refined logical translation of the first premiss. The words of the predicate, i.e., VOTE + FOR + JONES, are not compressed as they are in Analysis II, so the quasi-logical form of the parsed sentence is

ALL + X/B + PHI/.1,B + P/.500 + IND/.0

in which

PHI/.1 = NP*0/.0 + NP*2/.1 + ADJNCL/.0 + ONE/.32767,ADJN + RELCL/.0 + RCL*2/.1 + WHO/.32767,RELPR + VP*0/.1 + V/.0 + SUPPORT/.32767,VPOS + NP/.0 + ICKES/.32767,PRNAME

and

P/.500 = SUPPORT

and

IND/.0 = JONES

whose logical translation is
(A/Q X/B)((PHI/.1,B) IMPLIES/OP (P/.500 X/B IND/.0))

PHI/.1,B is next replaced by

((P/.1,B) AND/OP (PHI/.1,B))

in which

P/.1 = ONE

and

PHI/.1 = RELCL/.0 + RCL*2/.1 + WHO/.32767,RELPR + VP*0/.1 + V/.0 + SUPPORT/.32767,VPOS + NP/.0 + ICKES/.32767,PRNAME

yielding the formula

(A/Q X/B)(((P/.1,B) AND/OP (PHI/.L,B)) IMPLIES/OP (P/.500 X/B IND/.0))

PHI/.1,B is next converted into P/.501 + X/B + IND/.1, and P/.1,B is eliminated, yielding the formula

(A/Q X/B)((P/.501 X/B IND/.1) IMPLIES/OP (P/.500 X/B IND/.0))

in which

P/.501 = SUPPORT

and

IND/.1 = ICKES

Since the first premiss contains no NP'S beginning with THE, Analysis IV gives the same result as Analysis III. This is also true of the remaining sentences of the argument. The translations of the premises and conclusion, resulting from Analyses II and III, are given below.

* * * * *

First premiss
II (A/Q X/A) ((P/.3,A/ IMPLIES/OP (P/.2,A))
III (A/Q X/B)((P/.501 X/B IND/.1) IMPLIES/OP (P/.500 X/B IND/.0))

*****

Second premiss
II (A/Q X/D)((P/.5,D) IMPLIES/OP (P/.4,D))
III (A/Q X/E)((P/.500 IND/.3 X/E) IMPLIES/OP (P/.502 X/E IND/.2))

*****

Third premiss
II (P/.6 IND/.0)
III (A/Q X/G)((P/.502 X/G IND/.4) IMPLIES/OP (NOT(P/.502 IND/.0 X/G)))

*****

Fourth premiss
II (P/.7 IND/.2)
III (P/.502 IND/.2 IND/.4)

*****

Conclusion
II (NOT(P/.3) IND/.3)
III (NOT(P/.501 IND/.3 IND/.1))

*****

The complete lexicon for the above argument is as follows.

IND/.4 + KELLY + IND/.3 + ANDERSON + IND/.2 + HARRIS + IND/.1 + ICKES + IND/.0 + JONES

P/.7 + FRIENDOFKELLY + P/.6 + FRIENDOFNOONEWHOFRIENDOFKELLY + P/.5 + ANDERSONVOTEFOR + P/.4 + FRIENDOFHARRIS + P/.3 + SUPPORTICKES + P/.2 + VOTEFORJONES + P/.L + ONE + P/.0 + IS

P/.502 + FRIENDOF + P/.501 + SUPPORT + P/.500 + VOTEFOR

After the program has completed all the functional logic analyses (i.e., II, III, and IV) for an input argument, it selects one of them as the basis of the proof that will be attempted in Section DC of the program. In making this choice, the program makes a list of the terms in the premisses and conclusion, where a "term" may be a propositional letter (e.g., A/V, B/V, etc.), an individual name (e.g., IND/.0, IND/.1, etc.), or a unary, binary, or ternary predicate (e.g., P/0, P/.l, ... P/.500, P/.501, etc.). It then searches for repetition of terms between premisses and conclusion. The repetition of at least one term between the premisses and conclusion may be stated as a necessary condition of validity of a nontrivial argument, i.e., an argument with nonselfcontradictory premisses and nontautological conclusion. If an analysis of an argument contains no repetition, then it is ruled out, but if it contains some repetition, then it is regarded as providing the basis of a possible proof. In Analysis I, the repetition of just one term is sufficient to justify having a go at a proof in propositional logic; if the argument cannot be proven in propositional logic, the Wang algorithm will quickly determine this, and send the program on into Analyses II, III, and IV. For these last three analyses, something a little stronger is required than repetition of just one term. In fact, the program looks for the simplest analysis in which all the terms of the conclusion are repeated in the premisses. This criterion is still not strong enough, mainly because there are some arguments with short conclusions containing just a few terms, all of which are repeated in the premisses under Analysis II, but the arguments nevertheless require more refined analyses for the premisses. The program, therefore, looks for internal repetition within the premisses. The analysis that is finally selected as the basis for the attempted proof is the simplest analysis according to which all the terms of the conclusion are repeated in the premisses and according to which at least one term of the premisses is repeated in the premisses. If such an analysis cannot be found, then the program settles for Analysis IV. The criterion as thus defined is adequate for all the examples that have been submitted to the program thus far. It seems neither too weak nor too strong, in that it takes account of the fact that some repetition of terms is a necessary condition of validity of a non-trivial argument, but it does not require 100 per cent repetition. It is, however, a purely pragmatic criterion, and there is no guarantee that it will always work, so we have designed the program in such a way that, in ease of failure of the criterion, the operator may specify an alternative analysis. In order to facilitate selection by the operator, should it be necessary, the formulae resulting from Analyses II, III, and IV, i.e., the output of Section DB, are written out into Channel B, whence they are read in at the start of DC. The formula selected by the program is stored first, and it is the one that will be tested in the absence of any contrary instructions by the operator. If the operator decides that the formula selected cannot be proven (the logical evaluation part of the program is a proof procedure rather than a decision procedure, and is therefore incapable of rejecting invalid formulae, except in Analysis I), he may interrupt the evaluation, restart DC, and type in — .2+, —/.3+, or —/.4+ at the start, depending on which analysis he wishes the program to try.

For the example that we have been considering, the propositional logic analysis, i.e.,

A. B. C. D. THEREFORE E.

is rejected by the criterion, since there is no repetition at all between premisses and conclusion. In Analysis II, there are two terms in the conclusion, i.e., P/.3 and IND/.3, of which only the first recurs in the premisses, so Analysis II is also rejected by the criterion. Analysis III, however, is accepted by the criterion, since all three terms of the conclusion, i.e., IND/.1, IND/.3, and P .501, recur in the premisses, and several terms occur more than once in the premisses; the formula resulting from Analysis III is in fact a theorem and is subsequently proven in Section DC.

Once an analysis is selected, by the program or by the operator, the premisses and conclusion are combined into a single formula of conditional form, in which the conjunction of the premisses is taken to imply the conclusion. The method by which this is accomplished was described earlier in the section on propositional logic translation. If the formula pertains to functional logic, the additional step is performed of putting it into prenex normal form, in which all the quantifiers are on the left, and the scope of each quantifier is the entire formula to the right of it. The prenex normal form of a formula is required by the functional logic evaluation program. It is arrived at through the application of the PRNX routine, which is based upon the repeated application of the following standard set of logical equivalences, until all the quantifiers are on the left. ('P' is any formula that contains no free occurrence of V; 'OP' may be 'AND', 'OR', or 'IMPLIES'; and 'Q' may be 'Q/ALL' or 'Q/SOME'.)

P OP (QX)(FX) = (QX)(P OP FX)
(AX) (FX) IMPLIES P = (EX) (FX IMPLIES P)
(EX) (FX) IMPLIES P = (AX) (FX IMPLIES P)
(QX)(FX) AND/OR P = (QX)(FX AND/OR P)

Negated quantifiers are eliminated by the application of the pair of equivalences

NOT(AX)(FX) = (EX)(NOT FX)
NOT(EX)(FX) = (AX)(NOT FX)

The PRNX routine operates as follows.

OUTLINE OF THE PRNX ROUTINE

Universal quantifiers, i.e., (A/Q X/A), (A/Q X/B), etc., are changed to Q/ALL,A, Q/ALL,B, etc. Existential quantifiers, i.e., (E/Q X/A), (E/Q X/B), etc., are changed to Q/SOME,A, Q/SOME,B, etc. Shelf 1 is for initial Q's.

```
1. Start. Is first item in workspace a Q?
1.1. Yes: Queue item onto Shelf 1; go to 1.
1.2. No: Read up to first Q.

1.21. Succeed: go to 2.
1.22. Fail: queue workspace onto Shelf 1; Shelf 1 contains prenex formula; Done.
2. (In the following, 'Q' refers to the first Q in the workspace.) Is Q preceded by *(+NOT?
2.1. Yes: change *(+NOT+Q/ALL to Q/SOME+*( + NOT; change *(+NOT+Q/SOME to Q/ALL+*( + NOT; go to 3.
2.2. No: go to 3.
3. Apply whichever one of the following rules is appropriate.
    ((P) OP Q(R)) = Q((P) OP (R));
    (Q/ALL(P) IMPLIES(R)) = Q/SOME((P) IMPLIES(R));
    (Q/SOME (P) IMPLIES (R)) = Q/ALL ((P) IMPLIES (R));
    (Q(P) AND/OR (R)) = Q((P) AND/OR (R));
go to 1.
```

The prenex normal form of the formula resulting from Analysis III of our example is

(E/Q X/B) (E/Q X/E)(E/Q X/G) ((((((P/.501 X/B IND/.1) IMPLIES (P/.500 X/B IND/.0)) AND ((P/.500 IND/.3 X/E) IMPLIES (P/.502 X/E IND/.2))) AND ((P/.502 X/G IND/.4) IMPLIES (NOT(P/.502 IND/.0 X/G)))) AND (P/.502 IND/.2 IND/.4)) IMPLIES (NOT(P/.501 IND/.3 IND/.1)))

The overall plan of Section DB, which translates the parsed sentences of the input arguments into logical notation according to Analyses II, III, and IV, is given below.

OUTLINE OF SECTION DB

Shelf 22 is input shelf for parsed sentences; Shelves 19, 20, and 21 are output shelves for storing translations resulting from Analyses II, III, and IV, respectively; Shelf 25 is for recording which analysis the program is in at a given time, and is initialized with 'II+III+IV'; Shelf 10 is copy of input Shelf 22; Shelf 9 records the PHI's and the material that they abbreviate; Shelves 1, 12, and 24 are for storing translated or partially translated formulae.

1. *Start.* Is anything on Shelf 25?
1.1. *Yes:* put copy of Shelf 22 on Shelf 10; go to 2.
1.2. *No*: Analyses II, III, and IV are finished; go to 8.

2. Read in parsed sentence from Shelf 10.
2.1. *Succeed:* go to 3.
2.2. *Fail:* go to 7.

3. Is program in Analysis II?
3.1. *Yes:* compress all the words in the predicate (except NOT) into one word, and subscript it with /VPOS. If predicate is positive, it will be of the form V + $1/VPOS; if negative, it will be of the form V+VNEG+$1/VPOS+NOT. Go to 4.
3.2. *No*: go to 4.

4. Is program in Analysis III?
4.1. *Yes:* if there is any noun phrase whose first word is THE, compress all the words in the noun phrase into one word, and subscript it with /PRNAME; noun phrase will then be of the form NP + $1/PRNAME; go to 5.
4.2. *No*: go to 5.

5. Enter SFORM, and determine quasi-logical form of parsed formula; enter LF, and determine logical translation of quasi-logical formula; if any PHI's are created in SFORM, store them on Shelf 9, followed by the material that they abbreviate. Is Shelf 12 empty?
5.1. *Yes:* store formula on Shelf 12; go to 6.
5.2. *No*: formula is the logical translation of a certain PHI/.n; replace all occurrences of PHI/.n in the formula on Shelf 12 with copies of the formula in the workspace; go to 6.

6. Read in next PHI from Shelf 9.
6.1. *Succeed:* go to 5.
6.2. *Fail:* transfer formula from Shelf 12 to Shelf 24; use ** to mark end of formula; go to 2.

7. Combine formulae on Shelf 24 into a single formula of conditional form, in which the conjunction of the premisses implies the conclusion; store formula on Shelf 19, 20, or 21, depending on whether program is in Analysis II, III, or IV; delete first item on Shelf 25; go to 1.

8. Apply selection criterion to formulae on Shelves 19, 20, and 21 to decide which one is likeliest to yield the simplest proof; write out formulae into Channel B, with the selected one first; each formula is followed by —/.n, where n is the number of the analysis that produced the formula; *done.*

The routines SFORM, LF, and PHI are the principal subroutines of DB. Instead of attempting to describe them verbally in detail, we shall reproduce the actual COMIT rules that embody these routines, accompanied by a paragraph or so of explanation in each case. The expression '$0', which occurs frequently in these three routines, is a feature of the time-sharing version of COMIT but is not explained in the COMIT manuals. It denotes the beginning or end of the workspace.

SFORM ROUTINE

(For translating parsed sentences into quasi-logical formulae)

Shelf 1 is input shelf for sentence or part of sentence whose quasi-logical form is to be determined; Shelf 9 is for PHI's; Shelf 11 is for variables X/A, X/B, etc.; Shelf 14 is output shelf; Shelf 15 records largest PHI currently on Shelf 9 (Shelf 15 is initialized with PHI/.0); Shelf 16 is for terms IND/.n; Shelf 17 is for terms P/.n (n less than 500) denoting unary predicates (Shelf 17 is initialized with P/.1+ONE+P/.0 + IS); Shelf 18 is for terms P/.n (n equal to or greater than 500) denoting binary and ternary predicates.

```
SFORM $//*A1 1    *
* $0 + VP*1=–/.1 + –/SVP1//*S10 2    SCOPE
* $0 + NP + $1/PRNAME + $ = 3 + 4 + –/QSH14 //*SI 2,*A16 2,–
*S10 3    INDCHECK
* $0 + NP + NP*1 + $L/DET + $=4 + 5 //*Q14 1,*Q8 2,*N15 1   SF5
* $0 + V + $1/VPOS = 3+ –/QSH14//*S10 2,*A17 2    P1CHECK
* $0 + V+VNEG + $1/VPOS + NOT = 4 + 5+ –/QSH14 //*S4 2,–
*S10 3,*A17 2    P1CHECK
* $0 + VP*0 + V + IS + $ = P/.0 + 5 //*Q14 1,*Q1 2    SFORM
* $0 + VP*0 + V + VNEG + IS + NOT + $ = P/.0,NOT + 7//*Q14 1,–
*Q12    SFORM
* $0 + VP*0 + V+$L/VPOS + $=4 + 5+ –/QSH14 //*Q1 2,$S10 3,–
*A18 2    P2CHECK
* $0 + VP*0 + V + VNEG + $1 + $1 + $ = 5 + 7+–/QSH14 + NOT //–
*Q1 2,*S10 3,*A18 2,*S4 4    P2CHECK
* $0 + IVP+$ +V+$1 + $0 = 3+VP*0 +4 +5 //*Q1 1 2 3 4,*N14 1,–
*Q1 1    SFORM
* $0 + IVP+$ + V+ $3+ $0= 3+VP*0 + 4 +5//*Q1 1 2 3 4,*N14 1,–
*Q11SFORM
* *X //*Q14 1    *
* $1    Z
SFO $//*A14 1    *
* Y=    NV
*    LF
NV $ = X+1 //*N11 1    $
NV1 $1 + $ + Y + PHI+$ = 1+4 + 2 + L+4/$*1 + 5 //*Q14 3 4 5 6,–
*A9 3    *
* $0 + $1 + $1 + $ + 3=4 + 5/$*2//*X9    SF1
OV$ + X = 2 + L + 2    NVI
SF1 $//*A14 1    *
* Y=    SF2
* LF
SF2 $0 + THE + X + PHI + P/.0,NOT + SOME + Y + PHI    NV
```

```
* $0 + THE + X + PHI + P/.0 + NO + Y + PHI    SF3
*   SF4
SF3 P/.0,NOT    OV
*   NV
SF4 $0+ $1/DET +PHI+ P/.0 +SOME+ Y + PHI    OV
* $0 + $1/DET + X + PHI + P/.0 + NO + Y + PHI    OV
*   NV
SF5 PHI= 1/.I1 + 1/.I1 //*S15 2    SF6
* $=PHI/.0    SF5
SF6 $=Y+1 + 1+ –/.1+ –/SF7//*Q14 1 2,*Q9 3,*S10 5,–
•A8 5    SCOPE
SF7 $//*A8 1,*Q1 1,*A7 1,*Q9 1    SFORM
SVP1 $//*A8 1    *
* $0 + PPCL + PPCL*2 + $L/PREP + $ = 4 + 5 //*Q2 1 2,*A7 1    *
* $ + $1/VPOS = 1 + 2 + X//*Q1 1,*N2 3,*K2 3    *
*  $1 + $ = 1/.32767,VPOS,MAIN + 2 //*Q1 1 2,*A2 1,–
* Q1 1    SFORM
INDCHECK $1 = 1/–$    *
* $1 + $ + $1 +1 + $ = 3+ 2 + 3 + 4 + 5 //*S14 1,*Q16 2 3 4 5,–
* N10 1    $
* $1 + $1 + $ = 2/.I1 + 2/.I1 + L + 2 + 3//*S14 1,*Q16 2 3 4 5,–
* N10 1    $
* $1=IND/.0 + IND/.0+1 //*S14 1,*Q16 2 3,*N10 1    $
P1CHECK $1 = 1/–$    *
* $1 + $ + $1 + 1 + $ = 3 + 2 + 3 + 4 + 5 //*Q17 2 3 4 5,–
* N4  2    NCHECK
* $1 + $1 + $ = 2/.I1 + 2/.I1 + L + 2 + 3 //*Q17 2 3 4 5,–
* N4 2    NCHECK
P2CHECK $1 = 1/–$    *
* $1 + $ + $1 + 1 + $ = 3 + 2 + 3 + 4 + 5 //*Q18 2 3 4 5,–
* N4 2    NCHECK
*  $1  +$1+$  =2/.I1+2/.I1  +  L  +  2  +  3//*Q18  2  3  4  5,–
* N4 2    NCHECK
* $1=P/.500 + P/.500 + 1 //*Q18 2 3,*N4 2    NCHECK
QSH14  $//*N14  1,*Q14  1    SFORM
NCHECK $1 + NOT=1/NOT //*S14  1,*N10 1    $
* $1 + $//*S14 1,*S4 2,*N10 1    $
SCOPE $0 + $1/.G0 + $1 = 2/.I.*3 + 3 //*Q7 2    SCOPE
* $0 + $1/.0 + $ = 3//*Q8 1,*N10 1    $
*   Z
```

In the first and main section of the SFORM routine, the principal noun phrases and verb phrases of the parsed sentence are looked up, and replaced by abbreviations. Proper names are replaced by terms IND/.n, other noun phrases are replaced by expressions of the form $l/DET+Y+PHI/.n, and verbs are replaced by terms P/.n (or P/.n/NOT) denoting unary, binary, or ternary predicates. The terms IND/.n and P/.n are determined from Shelves 16, 17, and 18. If a term is not found on its appropriate shelf, a copy of it is put there, and its numerical subscript /.n is increased by 1. The terms PHI/.n abbreviate noun phrases minus their determiners; each PHI/.n, followed by the material that it abbreviates, is stored on Shelf 9, and its numerical subscript /.n is greater by 1 than that of the largest PHI already on Shelf 9. The rest of SFORM replaces the terms Y with the variables X/A, X/B, etc., from Shelf 11. In some cases, mainly those in which the main verb is P/.0 (IS) and is directly followed by SOME or NO, both PHI's in the formula are replaced by the same X; otherwise, the Y's are replaced by different X's. In either case, each PHI is given the literal subscript of its immediately preceding X.

The program next enters the LF routine, which translates the quasi-logical formulae into fully parenthesized formulae of functional logic.

LF ROUTINE

(For translating quasi-logical formulae into fully parenthesized formulae of functional logic)
Program uses Shelves 1 and 14 for storage of formulae or parts of formulae.

```
LF  ALL+$  +  $1/NOT  =  SOME  +  2  +  3        *
* NO + $1 + NOT = ALL + 2 + 3/–NOT    *
* $ I/NOT + ALL = 1 + SOME    *
* $1/NOT + NO=1/–$,.*1 + SOME    *
* P/.0 + NO = 1/NOT + SOME    *
LF1 $0 + $1 + $1 + $0 = *( + 3 + 2 + *) LF16
* $0 + $1 + $1 + $1 + $0=*( + 3 + 2 + 4 + *) LF16
* $0+$1+$1+$1+P+ $0=*(+2+3+*) +*(+ *( +4 + *) + IA + *( + –
5/$*3+ *) + *) LF2
* $0 + $1 + $1 + $1 + $1 + $0=*( + 3 + 2 + 4 + 5+*)    LF16
* $0+$1+P+$1+ $1+$1+$0=*(+4 +5+*)+ *( + *( + 6+*) + IA + –
*( + 3 + 2 + 5+ *) + *)    LF2
* $0+$1+$1+$1+P+$1+$0=*(+2+3+*) + *( + *( +4+ *) + IA + –
*( + 5 + 3 + 6 + °) + *)    LF2
* $0 + $1 + $1 + $1 + $1+$3+ $0+*(+2+3+*)+*( + *(+4 + *) + –
IA + 3 + 5 + 6 //*Q14 1 2 3 4 5 6 7 8 9    LF1
* $0+ $1+P + X +$1+$1+$1+$0=*(+5+6+*) + *( + *( + 7+*) + –
IA+*( + 3 + 2 + 4 + 6 + *) + *)    LF2
* $0+ $1+P + $1 + X + $1 + $1+$0=*(+4 +5+*)+*(+*(+6+*)+ –
IA+*( + 3 + 2 + 5 + 7+*) + *)    LF2
* $0 + $1+ $1+ $1+P+$1+ $1+$0=*(+2 + 3 + *) +*( + *( + 4 + –
*) + IA+*( + 3 + 5 + 6 + 7+*) + *)    LF2
* $0 + $1+P+ $1+ $1+ $1+ $3+$0=*(+4 + 5 + *) + *(+*( + 6 + –
*) + IA + 2 + 3 + 5 + 7//*Q14 1 2 34 5 6 7 8 9    LF1
* $0+ $1+ $1+ $1+P+$=*(+2+3+*)+ *( + *( +4 + *) + IA + 3 + –
5 + 6 //*Q14 123456789    LF1
LF2 $ = X+1 //*A14 1    *
* $=–/.0 + 1 + 1 //*S14 3    *
LF3 $1 + $ + IA=1/.I1    LF3
* $1 + $ = 1    *
LF4 $1/.G1 = 1/.D1 + *) //*Q14 2    LF4
* $//*A14 1    *
LF5 $ + ALL + $ + IA = 1 + A/Q + 3 + IMPLIES/OP //–
*Q14 1 2 3 4    LF5
* $ = X+1 //*A14 1    *
LF6 $ + SOME + $ + IA = 1 + E/Q + 3 + AND/OP //–
*Q14 1 2 3 4    LF6
* $ = X+1 //*A14 1    *
LF7 NO + $ + IA + $ = A/Q + 2 + IMPLIES/OP + *( + NOT + –
4 + *)    LF7
LF8 $ + ONLY+$ + IA = 1+2 + 3 + 4–/.1 //*Q14 1 2 3 4    LF9
*   LF11
```

```
LF9 $0 + $1 + *( = 2/.I1 + 3//*Q14 1 2    LF9
* $0 + $1 + *) = 2/.DL + 3//*Q14 1 2    LF10
* $1 + $1 //*Q14 1 2    LF9
LF10 $0 + I1/.GO    LF9
* $1 = 1 + *Q//*A14 1     *
* *) + *Q = 2+L     *
* ONLY + $ + $3 + IA + *(+ P/NOT + $ + *) + *Q = E/Q + 2 + –
*( + NOT + 3 + *) + AND/OP + 5 + 6/ – NOT + 7 + 8    LF8
* ONLY + $ + $3 + IA + *( + $ + *) + *Q = A/Q +2+5+6+7+–
IMPLIES/OP + 3    LF8
LF11 *( + THE + $1 + $3 + $1 + $1 + IA = 2 + 3 + 5 + 7 //*N11 4,–
*Q14 1 2 3 4    LF11
* //*X14     *
LF12 THE + $1+$1+$1+$=*(+ E/Q + 2 + *) + *( + *( + 3 + *) + –
AND/OP+ 5+*(+ A/Q + 4 + *) + *( + *( + *( + PHI/.*3,$°4 + *) + –
IMPLIES/OP + *(+*=+2 + 4 +*) + *) + AND/OP + *) //–
*Q14  28    LF12
* $ = 1 + X //*A14 2     *
LF13 *(+$1/Q+X+*)+ *( + *( + PHI + *) + $1/OP + *( + P/.0,NOT–
 + 3 + 3 + *) + *) = 6 + NOT + 6 + 7 + 8 + 8    LF13
LF14 *( + $1/Q + X + *)+*( + *( + PHI + *) + $1/OP + *( + P/.0–
 + 3 + 3 + *)+*) = 6 + 7 + 8    LF14
LF15 NOT+*( + $1/NOT+$ + *) = 3/–NOT + 4    LF15
* NOT+*( + P + $+*) = 3/NOT + 4    LF15
LF16 P/.0 = * = /$*1,–.    LF16
Q $//*Q14 1    SH24CHECK
```

The first five rules of the program perform a few simple verbal changes, such as elimination of double negatives, and conversion of a sentential form like 'ALL X/A IS/NOT X/B' into the logically more accurate form 'SOME X/A IS/NOT X/B'. The second main section of the program, headed by the rule LF1, searches for a rule that applies to the sentential form of the sentence, and translates or partially translates it into logical notation, queuing the translated part onto Shelf 14, and in some cases leaving part of the formula behind in the work-space for further translation. The term 'IA' is used in this section to stand for 'IMPLIES/AND'. The rest of the program adjusts the parenthesization, decides whether IA is IMPLIES or AND, inserts negatives in sentences that contain NO, rearranges sentences contain ONLY into equivalent forms containing ALL, and performs a special set of operations on sentences containing THE so as to make explicit the fact that such sentences express the unique existence of objects possessing certain properties.

PHI ROUTINE

(For selecting the input phrases for the SFORM and LF routines)

Shelf 13 is input shelf, and is initialized with first PHI+.........on Shelf 9; Shelves 7, 8, and 26 are for temporary storage.

```
PHI $//*A13 1     *
* $1/ADJN + $ //*S26 2    PHI01
*    PHI02
PHI01 RELCL +$ = 1 + 2 + X //*A26 3    PHI02
* PPCL+$ = 1 + 2 + X //°A26 3    PHI02
* $ = 1 + X//*A26 2     *
* PHI+$ + $L/ADJN + $ = 1 + 3 + 4+ –/14PAR //*S13 3 1,–
*S10 4,*A17 3    P1CHECK
PHI02 $0 + PHI + RELCL + RCL*2 + $ = 2 + 4 + 5//*S13 1,–
*Q7 2 3    PHI03
* $0 + PHI + RELCL + RCL*1 + RCL*2 + $ = 2+ –/.1 + 5 + 6 + –
–/PHI04 //*S13 1,*S10 5    SCOPE
* $0 + PHI + PPCL + PPCL*2 + $ = 2 + 4 + 5 //*S13 1,–
* Q7 2 3    PHI05
* $0 + PHI + PPCL + PPCL*1 + PPCL*2 + $ = 2+ –/.1 + 5 + 6 + –
–/PHI06 //*S13 1,*S10 5    SCOPE
* PHI+$1     Z
* PHI= //*S13 1     RPL01
PHI03 $ = X + X //*N13 1,*A8 2,*S13 2 1,*A7 1     *
* $2 + $ = 2 //*S7 1,*N25 1     *
* $ = 1 + RC+1 //*S25 1,*K2 3    DAN
PHI04 $ = X + X //*N8 1,*N8 2     *
* RCL*3 + $1/CONJ    PHI03
*    Z
PHI05 $ = X + X //*N13 1,*A8 2,*S13 2 1,*N25 1     *
* $ = 1 + PP+1//*S25 1,*K2 3    DAN
PHI06 $ = X + X //*N8 1,*N8 2     *
* PPCL*3 + $1/CONJ    PHI05
*    Z
DAN $1 //*L1    DAN1
*    Z
* DAN1 PPII=    PPII
          PPIII=   PPIV
          PPIV=    PPIV
          RCII=    RCII
          RCIII=   RCIV
          RCIV=    RCIV
PPII  $//*A7  1  RC201
PPIV $//*A7 1 *
* $1 + $1 + $ = *X + VP*0 + V + 2/–$,VPOS + 3//*Q14 1,–
*Q1 2 3 4 5    SFORM
RCII $//*N7 1     *
* $1/VP    RCII
* $ = 1 + X//*A7 2     *
* $0 + $1 + $1/VPOS + NOT //*S4 4    RC201
* VNEG + $1/VPOS + NOT + $0 //*S4 3     *
RC201 $ + $L/.G32766 = 2 //*Q2 1    RC201
* $ = X+ –/14PAR //*A2 1,*K1,*S10 2,*A17 2    P1CHECK
RCIV $ = *X //*S14 1,*A7 1,*S1 1    SFORM
NEWPHI $//*A9 1     *
* PHI + $ + PHI + $ //*Q13 1 2,*Q9 3 4    PHI
* PHI+$//*Q13 1 2     PHI
* $//*A12  1     *
* $ =*–*.THE – LOGICAL – TRANSLATION – IS*. – *. + 1 + 1 + –
** //*WAL1,*WSL2,*Q1 3 4     *
* $//*A2 1,*A3 1,*A4 1,*A5 1,*A6 1,*A7 1,*A8 1,*A9 1,*A12 1,–
*A13 1,*A14 1,*A15 1,*A23 1,*A24 1,*N22 1     *
* THEREFORE = 1 + 1 //*S1 1,*S22 2     *
* $//*S22 1    H
RPL01 $//*A24 1     *
```

```
* $ = 1 + 1//*S14 1,*S24 2,*N13 1    *
8 $1 = 1/–. + 1 //*S13 2,*A14 2    *
RPL02 $L + $ + P/.L500=L + 2 + 3/$*1 //*Q7 2 3    RPL02
* $1 + $ = 1+X + 2//*A7 2    *
RPL03 $1 + $ + *X = 1 + 2 + X/$*1 //*Q7 2 3    RPL03
* $1 + $ = 2 //*Q7 1    *
RPL04 $ = X + X+** + X //*N11 1,*A7 2,*A12 4    *
RPL05 $1 + $ + ** + $ + *( + 1 + *) = 1 + 2 + 3 + 4 + 2 //–
*Q12 4 5    RPL05
* $1 + $ + ** + $ = 1 + 3 + 4 //*A12 2    *
* $1=PHI/.*1    *
* $1+$+1+$=1/$*3+2+3+ 4 //*S13 1,*Q12 2 3 4    RPL01
* $1 + $ = 2 //*Q12 1,*A24 1    NEWPHI
*   Z
SH24CHECK $//*A24 1    *
* $1 + $ = *(+1 + 2 + AND/P.CONJ + X + *) //*A14 5,–
*Q24 1 2 3 4 5 6    SH12CHECK
* $//*A14 1,*Q24 1    *
SH12CHECK $//*N12 1    *
*  $1 //*S12 1    PHI
* $ //*A24 1,*S12 1    NEWPHI
```

The first seven rules of the PHI routine deal with terms $1/ADJN that do not occur within a relative clause or prepositional phrase. Such terms are replaced by terms p/.n (n less than 500) regardless of which analysis the program is in. The rules "PHI02" through "RCIV" deal with any relative clauses and prepositional phrases that the formula may contain. The treatment of such sequences does depend on the particular analysis that the program is in. In Analysis II, all the words in a relative clause (minus the relative pronoun) or prepositional phrase are compressed to form a single term, which is subscripted with /VPOS. If the relative clause is negative, the $1/VPOS is followed by NOT.) The new term thus formed is replaced by a term p/.n (or P/.n,NOT), denoting a unary predicate. Thus, the relative clause 'who climb the hill' becomes CLIMBTHEHILL/VPOS, and the prepositional phrase 'in the house' becomes INTHEHOUSE/VPOS, and the resulting terms are looked up on Shelf 17 in order to determine the P's that should replace them. Analyses III and IV may be treated together at this point in the program, since the analysis of definite descriptions, which is the only respect in which they differ, was performed at an earlier stage. Analyses III and IV treat relative clauses and prepositional phrases essentially as propositional functions; that is, a relative clause like 'who climb the hill' (whose parsed form is RCL*2 + WHO/RELPR + VP*0 + V + CLIMB/VPOS + NP + NP*1 + THE/DET + NP*0 + ADJNCL + HILL/ADJN) is converted into '*X climb the hill' (i.e., *X + VP*0 + V + CLIMB/VPOS + NP + NP*1 + THE/DET + NP*0 + ADJNCL + HILL/ADJN), and its quasi-logical form is determined from SFORM. Likewise, a prepositional phrase like 'in the house' (whose parsed form is PPCL*2 + IN/PREP + NP + NP*1 + THE/DET + NP*0 + ADJNCL — HOUSE/ADJN) is converted into '*X in the house' (i.e., *X + VP*0 + V + IN/VPOS + NP + NP*1 + THE/DET + NP*0 + ADJNCL + HOUSE/ADJN), and its quasi-logical form is also determined from SFORM. The initial preposition of a prepositional phrase is subscripted with /VPOS so that the SFORM routine will interpret it as a binary relation. This device avoids the necessity of adding to SFORM, a special set of rules dealing with prepositions and is purely a matter of programming convenience. The term '*X', which is used in the analysis of relative clauses and prepositional phrases, is replaced, as soon as the PHI under analysis has been completely translated, by a term 'X' bearing the same literal subscript as the PHI. The part of the routine following the rule "RCIV" is not strictly speaking part of the PHI routine, but is concerned with setting up Shelf 13, and with substituting the part of the formula that has just been translated in the appropriate places in the main formula.

The program has successfully translated and proven a number of examples from I. M. Copi's *Introduction to Logic* and *Symbolic Logic,* and we present them below in summary form. Most of the examples (with the exception of "CIRCLE") required a certain amount of pre-editing in order to make their sentences conform to the restrictions imposed by the program's grammar; for these examples we present the original version along with the pre-edited one, so that the reader may have an idea of the sort of rewording and paraphrasing that is necessary. For each example, the analysis that was chosen by the selection criterion as the basis of the proof is denoted by an asterisk.

DULUTH

If I buy a new car this spring or have my old car fixed, then I'll get up to Canada this summer and stop off in Duluth. I'll visit my parents if I stop off in Duluth. If I visit my parents they'll insist upon my spending the summer with them. If they insist upon my spending the summer with them I'll be there till autumn. But if I stay there till autumn then I won't get to Canada after all! So I won't have my old car fixed.9

Pre-edited version:

If I buy a new car or fix my old car then I'll get to Canada and stop in Duluth. If I stop in Duluth then I'll visit my parents. If I visit my parents then I'll stay in Duluth but if I stay in Duluth then I'll not get to Canada. Therefore I'll not fix my old car.

* *Analysis I*:

(((A/V) OR (B/V)) IMPLIES ((C/V) AND (D/V))) . ((D/V) IMPLIES (F/V)) . (((F/V) IMPLIES (H/V)) AND ((H/V) IMPLIES (C/V.NOT))) . THEREFORE (B/V,NOT) .

Prenex version of selected formula:

((((((A)OR(B))IMPLIES((C)AND(D)))AND((D)IMPLIES

(F)))AND(((F)IMPLIES(H))AND((H)IMPLIES(NOT(C)))))
IMPLIES(NOT(B)))

Lexicon:

A/V =	I buy some new car.
B/V =	I fix my old car.
C/V =	I getto Canada.
D/V =	I stopin Duluth.
F/V =	I visit my parents.
H/v =	I stayin Duluth.

KELLY

Whoever supports Ickes will vote for Jones. Anderson will vote for no one but a friend of Harris. No friend of Kelly has Jones for a friend. Therefore, if Harris is a friend of Kelly, Anderson will not support Ickes.8

Pre-edited version:

All who support Ickes will vote for Jones. Everyone whom Anderson will vote for is a friend of Harris. Jones is a friend of no one who is a friend of Kelly. Harris is a friend of Kelly. Therefore Anderson will not support Ickes.

Analysis I:

A/V. B/V. C/V. D/V. THEREFORE E/V.

Analysis II:

(A/Q X/A)((P/.3,A) IMPLIES (P/.2,A)) . (A/Q X/D) ((P/.5,D) IMPLIES (P/.4,D)) . (P/.6 IND/.0) . (P/.7 IND/.2) . THERE–FORE (NOT(P/.3) IND/.3) .

* *Analysis* III:

(A/Q X/B)((P/.501 X/B IND/.L) IMPLIES (P/.500 X/B IND/.0)) . (A/Q X/E)((P/.500 IND/.3 X/E) IMPLIES (P/.502 X/E IND/.2)) . (A/Q X/G)((P/.502 X/G IND/.4) IMPLIES (NOT(P/.502 IND/.0 X/G))) . (P/.502 IND/.2 IND/.4) .THERE–FORE (NOT(P/.501 IND/.3 IND/.1)) .

Analysis IV:

(A/Q X/C)((P/.501 X/C IND/.1) IMPLIES (P/.500 X/C IND/.0)) . (A/Q X/F)((P/.500 IND/.3 X/F) IMPLIES (P/.502 X/F IND/.2)) . (A/Q X/H)((P/.502 X/H IND/.4) IMPLIES (NOT(P/.502 IND/.0 X/H))) . (P/.502 IND/.2 IND/.4) . THEREFORE (NOT(P/.501 IND/.3 IND/.1)).

Prenex form of selected formula:

(E/Q X/B)(E/Q X/E)(E/Q X/G) ((((((P/.501 X/B IND/.1) IMPLIES (P/.500 X/B IND/.0)) AND ((P/.500 IND/.3 X/E) IMPLIES (P/.502 X/E IND/.2))) AND ((P/.502 X/G IND/.4) IMPLIES (NOT(P/.502 IND/.0 X/G)))) AND (P/.502 IND/.2 IND/.4)) IMPLIES (NOT(P/.501 IND/.3 IND/.1)))

Lexicon:

A/V =	All one who support Ickes votefor Jones.
B/V =	All one whom Anderson votefor friendof Harris
C/V =	Jones friendof no one who friendof Kelly.
D/V =	Harris friendof Kelly.
E/V =	Anderson support not Ickes.

IND/.4 + KELLY + IND/.3 + ANDERSON + IND/.2 + HARRIS + IND/.L + ICKES + IND/.0 JONES

P/.7 + FRIENDOFKELLY + P/.6 + FRIENDOFNOONEWHO FRIENDOFKELLY + P/.5 + ANDERSONVOTEFOR + P/.4 + FRIENDOFHARRIS + P/.3 + SUPPORTICKES + P/.2 + VOTE FORJONES

P/.502 + FRIENDOF + P/.501 + SUPPORT + P/.500 + VOTEFOR

CIRCLE

All circles are figures. Therefore all who draw circle draw figures.2

Analysis I:

A/V. THEREFORE B/V.

Analysis II:

(A/Q X/A)((P/.2,A) IMPLIES (P/.3,A)) . THEREFORE (A/Q X/D)((P/.5,D) IMPLIES (P/.4.D)) .

Analysis III:

(A/Q X/B)((P/.2,B) IMPLIES (P/.3,B)) . THEREFORE (A/Q X/E)((E/Q X/G)((P/.2,G) AND (P/.500 X/E X/G)) IMPLIES (E/Q X/F)((P/.3,F) AND (P/.500 X/E X/F)))

**Analysis* IV:

(A/Q X/C)((P/.2,C) IMPLIES (P/.3,C)) . THEREFORE (A/Q X/H)((E/Q X/J)((P/.2,J) AND (P/.500 X/H X/J)) IMPLIES (E/Q X/I)((P/.3,I) AND (P/.500 X/H X/I)))

Prenex form of selected formula:

(E/Q X/C) (A/Q X/H) (A/Q X/J) (E/Q X/I) (((P/.2,C) IMPLIES (P/.3,C)) IMPLIES (((P/.2,J) AND (P/.500 X/H X/J)) IMPLIES ((P/.3,I) AND (P/.500 X/H X/I))))

Lexicon:

A/V = All circle is some figure.
B/V = All who draw some circle draw some figure.

P/.5 + DRAWSOMECIRCLE + P/.4 + DRAWSOMEFIGURE + P/.3 + FIGURE + P/.2 + CIRCLE

P/.500 + DRAW

PROFESSOR

There is a professor who is liked by every student who likes any professor at all. Every student likes some professor or other. Therefore there is a professor who is liked by all students.[13]

Pre-edited version:

There is a professor whom every student who likes some professor likes. Every student likes some professor. Therefore there is a professor that all students like.

Analysis I:

A/V. B/V. THEREFORE C/V.

Analysis II:

(E/Q X/A)((P/.2,A) AND (P/.3,A)) . (A/Q X/H) ((P/.4,H) IMPLIES (P/.5,H)) . THEREFORE (E/Q X/M) ((P/.2,M) AND P/.6,M)) .

**Analysis* III:

(E/Q X/B)((P/.2,B) AND (A/Q X/C) (((P/.4,C) AND E/Q X/D)((P/.2,D) AND (P/.500 X/C X/D))) IMPLIES (P/.500 X/C X/B))) . (A/Q X/I)((P/.4,I) IMPLIES (E/Q X/J) ((P/.2,J) AND (P/.500 X/I X/J))) . THEREFORE (E/Q X/N) ((P/.2,N) AND (A/Q X/O)((P/.4,O) IMPLIES (P/.500 X/O X/N))) .

Analysis IV*:*

(E/Q X/E)((P/.2,E) AND (A/Q X/F) (((P/.4,F) AND (E/Q X/G)((P/.2,G) AND (P/.500 X/F X/G))) IMPLIES (P/.500 X/F X/E))) . (A/Q X/K)((P/.4,K) IMPLIES (E/Q X/L) ((P/.2,L) AND (P/.500 X/K X/L))) . THEREFORE (E/Q X/P) ((P/.2,P) AND (A/Q X/Q)((P/.4,Q) IMPLIES (P/.500 X/Q X/P))) .

Prenex form of selected formula:

(A/Q X/B)(E/Q X/C) (E/Q X/D) (E/Q X/I) (A/Q X/J) (E/Q X/N) (A/Q X/O)((((P/.2,B) AND (((P/.4,C) AND ((P/.2,D) AND (P/.500 X/C X/D))) IMPLIES (P/.500 X/C X/B))) AND ((P. .4,I) IMPLIES ((P/.2,J) AND (P/.500 X/I X/J)))) IMPLIES ((P .2,N) AND ((P/.4,O) IMPLIES (P/.500 X/O X/N))))

Lexicon:

A/V = Some one is some professor whom all student who like some professor like.
B/V = All student like some professor.
C/V = Some one is some professor that all student like.

P/.6 – ALLSTUDENTLIKE + P/.5 + LIKESOMEPROFESSOR + P/.4 + STUDENT + P/.3 + ALLSTUDENTWHOLIKESOMEPROFESSORLIKE + P/.2 + PROFESSOR

P/.500 + LIKE

RED

It is a crime to sell an unregistered gun to anyone. All the weapons that Red owns were purchased by him from either Lefty or Moe. So if one of Red's weapons is an unregistered gun, then if Red never bought anything from Moe, Lefty is a criminal.[14]

Pre-edited version:

Everyone who sells an unregistered gun to someone is a criminal. Lefty sold all the weapons that Red owns to Red. Red owns a weapon that is an unregistered gun. Therefore Lefty is a criminal.

Analysis I:

A/V. B/V. C/V. THEREFORE D/V.

Analysis II:

(A/Q X/A)((P/.2,A) IMPLIES (P/.3,A)) . (P/.6 IND/.0) . (P/.8 IND/.1) . THEREFORE (P/.3 IND/.0) .

* *Analysis* III:

(A/Q X/B) ((E/Q X/C)(((P/.4,C) AND (P/.5,C)) AND (E/Q X/D)(P/.500 X/B X/C X/D)) IMPLIES (P/.3,B)) . (A/Q X/H) (((P/.7,H) AND (P/.501 IND/.L X/H)) IMPLIES (P/.500 IND/.0 X/H IND/.1)) . (E/Q X/J) (((P/.7, J) AND ((P/.4,J) AND (P/.5,J))) AND (P/.501 IND/.1 X/J)) . THEREFORE (P/.3 IND/.0) .

Analysis IV*:*

(A/Q X/E)((E/Q X/F)(((P/.4,F) AND (P/.5,F)) AND (E/Q X/G)(P/.500 X/E X/F X/G)) IMPLIES (P/.3,E)) . (A/Q X/I)(((P/.7,I) AND (P/.501 IND/.1 X/I)) IMPLIES (P/.500 IND/.0 X/I IND/.1)) . (E/Q X/K) (((P/.7,K) AND ((P/.4,K) AND (P/.5,K))) AND (P/.501 IND/.1 X/K)) . THEREFORE (P/.3 IND/.0) .

Prenex form of selected formula:

(E/Q X/B) (E/Q X/C) (E/Q X/D) (E/Q X/H) (A/Q X/J) (((((((P/.4,C) AND (P/.5,C)) AND (P/.500 X/B X/C X/D)) IMPLIES (P/.3,B)) AND (((P/.7,H) AND (P/.501 IND/.1 X/H)) IMPLIES (P/.500 IND/.0 X/H IND/.1))) AND (((P/.7,J) AND ((P/.4,J) AND (P/.5,J))) AND (P/.501 IND/.L X/J))) IMPLIES (P/.3 IND/.0))

Lexicon:

A/V = All one who sell some unregistered gun to some one is some criminal.
B/V = Lefty sell all weapon that Red own to Red.
C/V = Red own some weapon that is some unregistered gun.
D/V = Lefty is some criminal.

IND/.1 + RED + IND/.0 + LEFTY
P/.8+ OWNSOMEWEAPONTHATISSOMEUNREGISTEREDGUN + P/.7 + WEAPON + P/.6 + SELLALLWEAPONTHATREDOWN–TORED + P/.5 + GUN + P/.4 + UNREGISTERED + P/.3 + CRIMINAL + P/.2 + SELLSOMEUNREGISTEREDGUNTOSOMEONE

P/.501 + OWN + P/.500 + SELLTO

DESK

Everything on my desk is a masterpiece. Anyone who writes a masterpiece is a genius. Someone very obscure wrote some of the novels on my desk. Therefore someone very obscure is a genius.[15]

Pre-edited version:

Everything on my desk is a masterpiece. Everyone who writes a masterpiece is a genius. Some obscure one wrote some of the novels on my desk. Therefore some obscure one is a genius.

Analysis I:

A/V. B/V. C/V. THEREFORE D/V.

Analysis II:

(A/Q X/A)((P/.2,A) IMPLIES (P/.3,A)) . (A/Q X/F) ((P/.6,F) IMPLIES (P/.7,F)) . (E/Q X/K) ((P/.9,K) AND (P/.8,K)) . THEREFORE (E/Q X/R) ((P/.9,R) AND (P/.7,R)) .

**Analysis* III:

(A/Q X/B)((E/Q X/C)(((P/.4,C) AND (P/.5,C)) AND (P/.500 X/B X/C)) IMPLIES (P/.3,B)) . (A/Q X/G)((E/Q X/H) ((P/.3,H) AND (P/.501 X/G X/H)) IMPLIES (P/.7,G)) . (E/Q X/L) ((P/.9,L) AND (E/Q X/M) (((P/.10,M) AND (E/Q X/N) (((P/.4,N) AND (P/.5,N)) AND (P/.500 X/M X/N))) AND (P/.501 X/L X/M))) . THEREFORE (E/Q X/S) ((P/.9,S) AND (P/.7,S)) .

Analysis IV:

(A/Q X/D)((E/Q X/E)(((P/.4,E) AND (P/.5,E)) AND (P/.500 X/D X/E)) IMPLIES (P/.3,D)) . (A/Q X/I)((E/Q X/J) ((P/.3,J) AND (P/.501 X/I X/J)) IMPLIES (P/.7,I)) . (E/Q X/O) ((P/.9,O) AND (E/Q X/P)(((P/.10,P) AND (E/Q X/Q) (((P/.4,Q) AND (P/.5,Q)) AND (P/.500 X/P X/Q))) AND (P/.501 X/O X/P))) . THEREFORE (E/Q X/T) ((P/.9,T) AND (P/.7,T)) .

Prenex form of selected formula:

(E/Q X/B) (E/Q X/C) (E/Q X/G) (E/Q X/H) (A/Q X/L) (A/Q X/M) (A/Q X/N) (E/Q X/S) (((((((P/.4,C) AND (P/.5,C)) AND (P/.500 X/B X/C)) IMPLIES (P/.3,B)) AND (((P/.3,H) AND (P/.501 X/G X/H)) IMPLIES (P/.7,G))) AND ((P/.9,L) AND (((P/.10,M) AND (((P/.4,N) AND (P/.5,N)) AND (P/.500 X/M X/N))) AND (P/.501 X/L X/M)))) IMPLIES ((P/.9,S) AND (P/.7,S)))

Lexicon:

A/V = All one on my desk is some masterpiece.
B/V = All one who write some masterpiece is some genius.
C/V = Some obscure one write some novel on my desk.
D/V = Some obscure one is some genius.

P/.10 + NOVEL + P/.9 + OBSCURE + P/.8 + WRITESOME–NOVELONSOMEMYDESK + P/.7 + GENIUS + P/.6 + WRITE–SOMEMASTERPIECE + P/.5 + DESK + P/.4 + MY + P/.3 + MASTERPIECE + P/.2 + ONSOMEMYDESK

P/.501 + WRITE + P/.500 + ON

TAPPAN

The architect who designed Tappan Hall designs only office buildings. Therefore Tappan Hall is an office building.[16]

Pre-edited version:

The architect who designed Tappan-Hall designs only office-buildings. Therefore Tappan-Hall is an office-building.

Analysis I:

A/V. THEREFORE B/V.

Analysis II:

(P/.2 IND/.0) . THEREFORE (P/.3 IND/.1) .

Analysis III:

(A/Q X/A)((P/.500 IND/.0 X/A) IMPLIES (P/.3,A) . THERE–FORE (P/.3 IND/.1) .

**Analysis* IV:

(E/Q X/B) (((P/.4,B) AND (P/.500 X/B IND/.L)) AND (A/Q X/D)((((P/.4,D) AND (P/.500 X/D IND/.L)) IMPLIES (* = X/B X/D)) AND (A/Q X/C)((P/.500 X/B X/C) IMPLIES (P/.3,C)))) .THEREFORE (P/.3 IND/.1) .

Prenex form of selected formula:

(A/Q X/B) (E/Q X/D) (E/Q X/C) ((((P/.4,B) AND (P/.500 X/B IND/.1)) AND ((((P/.4,D) AND (P/.500 X/D IND/.1)) IMPLIES (*= X/B X/D)) AND ((P/.500 X/B X/C) IMPLIES (P/.3,C)))) IMPLIES (P/.3 IND/..1)) .

Lexicon:

A/V = The architect who design Tappan-Hall design only office-building.
B/V = Tappan-Hall is some office-building.

IND/.1 + TAPPAN-HALL + IND/.0 + THEARCHITECTWHODESIGNTAPPAN-HALL
P/.4 + ARCHITECT + P/.3 + OFFICE-BUILDING + P/.2 + DESIGNONLYOFFICE-BUILDING

P/.500 + DESIGN

The running times in seconds for the preceding examples are listed below.

Example	DA *(parsing)*	DB *(translation)*	DC *(proof)*	*Total time*
Duluth	63	—	—	63
Kelly	68	22	11	101
Circle	163	9	7	179
Professor	162	15	39	216
Red	244	16	30	290
Desk	133	32	39	204
Tappan	43	7	10	60

The reason for the blanks in Sections DB and DC of DULUTH is that the argument was parsed, translated, and proven in Section DA. The parsing and translation took 58 seconds, and the proof by the Wang algorithm took 5 seconds. For all the problems, it will be noted that the parsing consumed by far the greatest amount of time. A more efficient parsing system, therefore, is essential if the program is to become a really practical tool for logical analysis. A running time of five or six minutes is intolerable for two reasons: (1) at a peak hour of time-sharing usage, this amount of machine time is apt to consume twenty or thirty minutes of actual time at the console, and (2) the time allotments for most MAC users have recently been reduced to twenty or thirty minutes per shift per month (the above data, fortunately, were obtained before the reduced quotas were assigned). The large amount of time consumed by DA, however, is part of the price one pays for permitting syntactic homonymy and amphiboly. An earlier version of the program parsed the input sentences far more rapidly, but it required that each word be assigned a unique syntactic category in an *a priori* fashion, and its grammar was considerably more simplified in other respects than that employed by the present version of the program.

Three sentences of the input arguments were found to be amphibolous, and the choice of parsing was made by the operator at the console. In the CIRCLE example, the sentence 'All who draw circles draw figures' was transformed into 'All who draw circle draw figure', which admitted of two nonsentential and five sentential parsings, i.e.,

SNOVP + NP + NP*1 + ALL/DET + NP*0 + NP*2 + ADJNCL + ONE/ADJN + RELCL + RCL*2 + WHO/RELPR + IVP + NP + NP*0 + ADJNCL + ACL*0 + DRAW/ADJN + ADJNCL + ACL*0 + CIRCLE/ADJN + ADJNCL + DRAW/ADJN + V + FIGURE /VPOS + **/.1

SNOVP + NP + NP*1 + ALL/DET + NP*0 + NP*2 + ADJNCL + ONE/ADJN + RELCL + RCL*2 + WHO/RELPR + VP*0 + V + DRAW/VPOS + NP + NP*0 + ADJNCL + ACL*0 + CIRCLE/ADJN + ADJNCL + ACL*0 + DRAW/ ADJN + ADJNCL + FIGURE/ADJN + **/.2

S + NP + NP*1 + ALL/DET + NP*0 + NP*2 + ADJNCL + ONE/ADJN + RELCL + RCL*2 + WHO/RELPR + IVP + NP + NP*0 + ADJNCL + DRAW/ADJN + V + CIRCLE/ VPOS + VP*0 + V + DRAW/VPOS + NP + NP*0 + ADJNCL + FIGURE/ADJN + **/.3

S + NP + NP*1 + ALL/DET + NP*0 + NP*2 + ADJNCL + ONE/ADJN + RELCL + RCL*2 + WHO/RELPR + VP*0 + V + DRAW/VPOS + NP + NP*0 + ADJNCL + CIRCLE/ ADJN + VP*0 + V + DRAW/VPOS + NP + NP*0 + ADJNCL + FIGURE/ADJN + **/.4

S + NP + NP*1 + ALL/DET + NP*0 + NP*2 + ADJNCL + ONE/ADJN + RELCL + RCL*2 + WHO/RELPR + V + DRAW/VPOS + VP*0 + V + CIRCLE/VPOS + NP + NP*0 + ADJNCL + ACL*0 + DRAW/ADJN + ADJNCL + FIGURE/ADJN + **/.5

S + NP + NP*1 + ALL/DET + NP*0 + NP*2 + ADJNCL + ONE/ADJN + RELCL + RCL*2 + WHO/RELPR + IVP + NP + NP*0 + ADJNCL + ACL*0 + DRAW/ADJN + ADJNCL + CIRCLE/ADJN + V + DRAW/VPOS + V + FIGURE/VPOS + **/.6

S + NP + NP*1 + ALL/DET + NP*0 + NP*2 + ADJNCL + ONE/ADJN + RELCL + RCL*2 + WHO/RELPR + VP*0 + V + DRAW/VPOS + NP + NP*0 + ADJNCL + ACL*0 + CIRCLE/ADJN + ADJNCL + DRAW/ADJN + V + FIGURE/VPOS + **/.7

The fourth analysis was selected by the operator, since it is clearly necessary in this example to treat 'draw' as a verb and 'circle' and 'figure' as nouns.

In the RED example, the sentence 'Everyone who sells an unregistered gun to someone is a criminal' was transformed into 'All one who sell some unregistered gun to some one is some criminal', which admitted of two sentential parsings, i.e.,

S + NP + NP*L + ALL/DET + NP*0 + NP*2 + ADJNCL + ONE/ADJN + RELCL + RCL*2 + WHO/RELPR + VP*1 + VP*0 + V + SELL/VPOS + NP + NP*1 + SOME/DET + NP*0 + ADJNCL + ACL*0 + UNREGISTERED/ADJN + ADJNCL + GUN/ADJN + PPCL + PPCL*2 + TO/PREP + NP + NP*1 + SOME/DET + NP*0 + ADJNCL + ONE/ADJN + VP*0 + V + IS/VPOS + NP + NP*1 + SOME/DET + NP*0 + ADJNCL + CRIMINAL/ADJN – **/.1

S + NP + NP*1 + ALL/DET + NP*0 + NP*2 + ADJNCL + ONE/ADJN + RELCL + RCL*2 + WHO/RELPR + VP*0 + V + SELL/VPOS + NP + NP*1 + SOME/DET + NP*0 + NP*2 + ADJNCL + ACL*0 + UNREGISTERED/ADJN + ADJNCL + GUN/ADJN + PPCL + PPCL*2 + TO/PREP + NP + NP*1 + SOME/DET + NP*0 + ADJNCL + ONE/ADJN + VP*0 + V + IS/VPOS + NP + NP*1 + SOME/DET + NP*0 + ADJNCL + CRIMINAL/ADJN + **/.2

The first analysis was selected by the operator, since it links the prepositional phrase 'to someone' with 'sell' rather than with 'gun'.

In the DESK example, the sentence 'Some obscure one wrote some of the novels on my desk' was transformed into 'Some obscure one write some novel on my desk', which admitted of two sentential parsings, i.e.,

> S + NP + NP*1 + SOME/DET + NP*0 + ADJNCL + ACL*0 + OBSCURE/ADJN + ADJNCL + ONE/ADJN + VP*1 + VP*0 + V + WRITE/VPOS + NP + NP*1 + SOME/DET + NP*0 + ADJNCL + NOVEL/ADJN + PPCL + PPCL*2 + ON/PREP + NP + NP*0 + ADJNCL + ACL*0 + MY/ADJN + ADJNCL + DESK/ADJN + **/.1
>
> S + NP + NP*1 + SOME/DET + NP*0 + ADJNCL + ACL*0 + OBSCURE/ADJN + ADJNCL + ONE/ADJN + VP*0 + V + WRITE/VPOS + NP + NP*1 + SOME/DET + NP*0 + NP*2 + ADJNCL + NOVEL/ADJN + ACL*0 + MY/ADJN + ADJNCL + ACL*0 + MY/ADJN + ADJNCL + DESK/ADJN + **/.2

This example is analogous to the preceding one, in that the amphiboly results from the question of whether a prepositional phrase should be linked with a directly preceding noun or with a verb occurring earlier on in the sentence. In this case, however, the second analysis was selected, since the operator knew that the prepositional phrase 'on my desk should be linked with 'novel' rather than with 'write'. It could be argued that the amphibolies of the first two examples are somewhat artificial, in that they result merely from the limited nature of the grammar and do not really pertain to the original English sentences. In the third example, however, the amphiboly appears to be more genuine, since it is quite possible that the sentence could be referring to novels being written on my desk. In the next section, we shall discuss the logical evaluation part of the program.

Methods of Logical Evaluation

As pointed out earlier, the propositional logic formula produced by the program are tested by the Wang algorithm, which is included in Section DA, and the functional logic formulae are proven in Section DC by a method based on the one-literal clause rule of Davis-Putnam and the matching algorithm of Guard. Our COMIT version of the Wang algorithm is reproduced below.

THE WANG PROPOSITIONAL LOGIC ALGORITHM

```
            COM WANG
RCR $=–*.–TYPE–IN–FORMULA*. //*WAL1     *
* $//*RCR1     *
**.=0    *
* $ + *–+$ //*Q9 1 2 3    SUB
* $ = *–+1 //*Q9 1 2    *
SUB $//*N9 1     *
* $1 //*L1    SUB1
* $//*A2 1    COM
–SUB1 B = 1/ZB,OP//*Q2 1 SUB
      C = 1/ZC,OP//*Q2 1 SUB
      D=1/ZD,OP//*Q2 1 SUB
      F = 1/ZF,OP //*Q2 1    SUB
      I = 1/ZI,OP //*Q2 1    SUB
      * – = //*Q2 1     SUB
      , = 1/OP //*Q2 1     SUB
* $1 = 1/V //*Q2 1     SUB
WFF $1 + $ + *–+$ =1+2+3+4+–/WFF1+ –/S3 //*Q8 1 2,–
*Q6 3 4,*S10 5 6    SCOPE
*    WFF2
WFF1 $ = X + X //*A7 1,*A6 2     *
WFF2 $+*–+$1+$=1+2+3+4+ –/WFF3+ –/S3 //*Q6 1 2,–
*Q8 3 4,*S10 5 6    SCOPE
*    COM2
WFF3 $ = X + X //*A6 1,*A7 2    COM2
SCOPE $ = M/.L //*S7 1     *
* $//*N8 1     *
* F = //*Q7 1    S1
* $1/OP //*Q7 1,*N7 1     S4
* $1/V //*Q7 1,*N7 1     S2
* $=–*–THE–FORMULA –IS– DEFECTIVE – THERE – ARE –
* TOO–MANY–OPERATORS*.//*WAL1,*A7 1,*WSL1   END
S2 $1 = 1/.D1    *
* $1/.G0 //*S7 1    S1
* $//*N10 1     $
S3 $//*A8 1     *
* $1 + $=–*– THE –FORMULA– IS – DEFECTIVE – THERE –
– ARE – TOO – MANY – VARIABLES*. + 1 + 2 //*WAL1,–
*WSL2 3    END
* $//*N10 1     $
S4 $1 = 1/.I1 //*S7 1    S1
COM $ + , + $ + *– //*Q2 2     COM
* $ + *– //*Q2 1 2    *
COM1 $ + ,//*Q2 2    COM1
* $//*Q2 1,*A2 1    WFF
COM2 $ + ,= 1 //*Q2 1     COM2
* $//*Q2 1,*A2 1    TEST
TEST $ = *.–+1 + 1 //*WAL1 2     *
* $ + $1/OP+$=1+2+2+ 3+ –/PAREN //*Q7 1 2,*S10 3 5,–
*Q8 4     SCOPE
P1 $1+ $ + *–+$ + 1     VALID
* $1 + $ + *–//*Q7 1     P1
* $=X+1 //*A7 1     *
* $=–*. –FORMULA– IS –INVALID*. //*WAL1     END
VALID $ = X+1 //*A7 1     *
* $= – *. –VALID*. //*WAL1,*A9 1     *
* $1 + $ + ** + $ = L + 2 + 4//*S9 3    TEST
* $=–*.–FORMULA–IS–VALID*.//*WAL1     END
PAREN $ = *) //*Q7 1,*N10 1     *
* F=1 + X + X//*A7 2,*A8 3    $
* $=–/PAREN1 + L //*S10 2 1    SCOPE
PAREN1 $ = X + X+*) + X //*N10 1,*A7 2,*A8 4    $
ZF $1 = 0    *
P2A $ + * –+$ + F + *) = 5 +1 + 2 + 3    TEST
P2B F+$ + *) + $ + *–+$=4 + 5 + 6 + 2    TEST
ZC $1 = 0    *
P3A $ +*–+$+C+$+*)+$+*)+$=1+2+3+5+9+1+2+ 3 + –
7+ 9+**//*S9 11 10 9 8 7 6    TEST
P3BC + $ + *) + $ + *) = 2 + 4    TEST
```

```
ZD $1 = 0    *
P4A *–+$ + D + $ + *) + $+*) = 1 + 2 + 4 + 6    TEST
P4B $+D+$+*) + $ + *) +$+*–+$=L+3+7+8+9+L+5+7+–
8+9 + ** //*S9 11 10 9 8 7 6    TEST
ZI $1 = 0    *
P5A $ +*–+$+1+$+*)+$+*)+ $=1+5+ 2+3 + 7 + 9    TEST
P5B $ +1+$+*)+$+*)+$+*–+$ =1+5+7+8+9+1+7+ 8 + –
9+3 + **//*S9 11 10 9 8 7 6    TEST
ZB S1=0    *
P6A $+*–+$+B+$+*)+$+*)+$=5+1+2+3+7+ 9 + 7+1 + –
2+3+5+9+** //*S9 13 12 11 10 9 8 7    TEST
P6B $+ B+$+*)+$+*)+$+*–+$=3+5+1+7+8+9+1 + 7 + –
8+9 + 3 + 5 + ** //*S9 13 12 11 10 9 8 7    TEST
END          *
        END
```

At the start of the program, the operator types in a formula in the required Polish notation, e.g., 'ICTPFQBPQ'. The COMIT input mode "format c" that the program employs then expands the formula, adding a '*.' at the end; the sample formula thereby becomes 'I — C+F + P + F + Q + B + P + Q + *.'. The program then deletes the final '*.', adds a '*—' at the left of the workspace if the formula does not already contain this symbol, and provides the individual symbols with appropriate subscripts. The letters 'B', *'c'*, 'D', 'F', and 'I', which stand for 'iff', 'and', 'or', 'not', and 'implies', respectively, are regarded as operators and given the subscript '/OP', in addition to a distinctive subscript ('/ZB', '/ZC', etc.) whose use is explained below. The comma, which is also subscripted with ' OP', is used by Wang in the following way: if it appears on the left of the dash it signifies the conjunction of the two well-formed formulae (wffs) between which it occurs, and if it appears on the right of the dash it signifies the disjunction of the two wffs. The dash itself expresses implication or entailment; the Winer algorithm requires every formula to contain exactlv one dash, and thus treats every formula as an implication (a dash can always be placed at the far left of a formula that initially contains no dash without affecting validity). Any symbols in the input formula other than the dash and the operators are regarded as variables and subscripted with '/v' (the period or dot, which Wang uses to partially parenthesize the input formulae, is not required by our program and is not included). The formula is next tested for wellformedness by a series of rules that reject a formula if it contains too many or too few variables in relation to the number of operators, or if the variables and operators occur in an illegitimate order. Any commas are then deleted, and the program proceeds to the test of validity proper.

The validity test is based on a set of ten rules (i.e., P2A, P2B, P3A, P3B, P4A, P4S, P5A, P5B, P6A, and P6B) for the elimination of operators, and one rule (i.e., p1) for the testing of a formula all of whose operators have hem eliminated. These eleven rules are named in our program after the corresponding rules in Wang's statement of his algorithm. The program finds the leftmost operator in the formula, and eliminates it by whichever of the rules P2A-P6B is appropriate. At this point, the program exploits the "$ go-to" feature of COMIT in order to go directly to the section of the program that eliminates the leftmost operator. For example, if the operator is C/ZC,OP the program goes to rule zc, and thence directly to rule P3A (which eliminates C if it occurs to the right of the dash) or, if P3A is inapplicable, to rule P3B (which eliminates c if it occurs to the left of the dash). Certain of these elimination rules (e.g., P3A) create new branches of the formula, which are separated by double asterisks and stored on Shelf 9. After all the operators have been eliminated from the formula in the workspace, the test of validity embodied in rule P1 and its directly following rule is applied: if any propositional letter on the left of the dash is repeated on the right, the formula (specifically, the branch in the workspace) is valid; if not, the formula is invalid. This test is based on the fact that a statement of implicational form, whose antecedent is a conjunction of atomic wffs and whose consequent is a disjunction of atomic wffs, is tautologous if and only if at least one of the atomic wffs of the antecedent is repeated in the consequent; e.g., the formula

(P AND Q AND R) IMPLIES (P OR S OR T)

is a tautology, but

(P AND Q AND R) IMPLIES (U OR S OR T)

is not. (This test of validity bears a superficial resemblance to our selection criterion for deciding among Analyses I, II, III, and IV of an argument, which is based on repetition of terms between premisses and conclusion. We daren't press the analogy too far, however, since our criterion is not a conclusive test of validity but merely establishes *prima facie* evidence of validity.) The procedure of eliminating operators and testing for repetition continues until an invalid branch is found, in which case the formula is invalid, or until all the branches are proved valid, in which case the formula is valid.

The program described here exactly duplicates the print-out from Wang's program on page 18 of his article. Our running times range between 0.3 and 4.0 seconds (exclusive of compilation) per formula. The program was adapted for use in Section DA of our main program, where it provides a quick and easy test of validity for propositional logic formulae.

The functional logic evaluation program is an outgrowth of an earlier program embodying the Davis-Putnam proof procedure algorithm. The present program, like the Davis-Putnam algorithm, operates by *reductio ad absurdum:* accepting an input formula "F" in prenex form, it negates F, puts the matrix of not-F into conjunctive normal form, replaces each existentially qualified variable x in the matrix with a distinct function of the universally quantified variables that precede x in the string of quantifiers, and attempts to deduce a contradiction from the resulting formula not-F'. In attempting to deduce a contradiction from not-F', the Davis-Putnam algorithm exploits the Herbrand theorem, which provides an effective procedure for making substitutions for the variables thereby generating quantifier-free substitution instances (QFSI), and a corollary of which states that F is a theorem if and only if a finite conjunction of QFSI generated from not-F' is inconsistent. Our program, however, avoids actually generating the QFSI in most cases, by employing the matching algorithm of Guard to test whether two matrix clauses could possibly yield any contradictory QFSI. The use of matching cuts down on the amount of material necessary to produce a proof, and results in a more efficient proof procedure program. The revised program incorporating matching not only proves theorems that the earlier program was unable to prove because of limitations of time and storage, but also reduces the computation time (in some cases by a factor of 10) for many theorems that the earlier program was able to prove. The matching algorithm is formulated by its author as follows:

"*Definition:* The following algorithm which is to be applied to two atomic wffs B and C ... is called *matching.*

Step 1: Consider B and C as being stored at lines (1) and (2) respectively. Reletter the variables of line (2) so that it has no variables in common with line (1).

Step 2: Let us denote the n-th symbol—ignoring parentheses and commas—of line (1) by (l)n. Similarly we define (2)n.

Case a): If lines (1) and (2) are identical, the algorithm outputs (1) and stops.

Case b): Suppose n is the smallest integer such that (l)n is different from (2)n. Since wffs are involved and case a) does not hold, neither (l)n or (2)n can be vacuous. We consider four subcases:

i) Suppose (2)n is a variable, say X, while (l)n is a function or individual constant. Then call D the unique subformula of (1), starting at (l)n. If D contains X, output *does not match,* and stop. If D does contain X, substitute D for X everywhere in (1) and (2). Go back and repeat Step 2.
ii) Proceed as in i) if the roles of (1) and (2) are interchanged.
iii) If (l)n and (2)n are different variables, replace (2)n everywhere in (1) and (2) by (l)n.
iv) If (l)n and (2)n are different constants, output *does not match* and stop."[17]

An illustrative example that the author gives is the application of the matching algorithm to the following two clauses:

P(G(G(X,G(Y,X)),Z))
P(G(G(X,Y),G(X,X)))

These clauses turn out to have the formula

P(G(G(x,G(y,x)), G(x,x)))

as a "general matching formula," i.e., a QFSI common to both clauses and from which all other common QFSI can be generated (finding a general matching formula for two clauses is equivalent to proving that the two "match"). Two clauses that do not match are the following:

Q(X,X)
Q(Y,H(Y))

These violate Rule i) of the algorithm, since the subformula H(X), which the algorithm generates, contains X.

In our partial reconstruction of the Davis-Putnam algorithm, we have applied the matching algorithm in two ways:

(1) to test whether two one-literal clauses (atomic wffs) would generate contradictory QFSI, thereby proving the formula inconsistent; and

(2) to generate from a one-literal clause and a polyliteral clause of length n, one or more clauses of length n—1. I

Both (1) and (2) make use of what might be called "negative matching," where two formulae are said to match negatively, or to "N-match," if and only if one matches the negation of the other. An example of (1) would be the two one-literal clauses

F(X,Y) and not-F(P(X,Y),P(X,Y))

which generate an infinite number of contradictor] QFSI, e.g.,

F(P(a,a),p(a,a)) and not-F(P(a,a),P(a,a)).

An example of (2) would be the one-literal clause

F(y,P(x,y))

and the polyliteral clause

not-(F(x,y)) v G(y,x).

These two clauses together entail

G(P(x,y),y)

which is shorter by one literal than the original poly literal clause, and which, as it turns out, is a one-literal clause.

The latter example illustrates the following principle which we may call DS', since it is essentially an extension of the principle of Disjunctive Syllogism, i.e., "A and (not-A v B) entail B."

DS': Given a one-literal clause A and a polyliteral clause (B v R), where A and B N-match via the general matching formula, if B is positive and equals M, or if B is negative and equals M except for the negation sign, then A and (B v R) entail R. If, however, neither B nor its negation equal M, then A and (B v R) entail R', where R' is formed from R by making the same substitutions in R that would have to be made in B (or its negation) to make it equal M.

In the example used above to illustrate (2), the one-literal clause

F(y,P(x,y))

and the first term of the polyliteral clause

not-F(x,y) v G(y,x)

N-match via the general matching formula

F(y,P(x,y)).

In order to get F(x,y) to equal the general matching formula, it would be necessary to make the substitutions

x = *y* and y = P(x,y)

in F(x,y). These substitutions must also be made in the remainder of the polyliteral clause, i.e.,

G(y,x)

yielding the one-literal clause

G(P(x,y),y).

The general plan of our revised algorithm, then, is to search for an N-match among the one-literal clauses, thereby proving the formula inconsistent. The one-iteral clauses are separated from the polyliteral clauses, the former being stored on Shelf 6 and the latter on Shelf 9. If there is no N-match among the one-literal clauses on Shelf 6, then DS' is applied to the first polyliteral clause on Shelf 9. If DS' can be applied, then it produces one or more new clauses containing n-1 literals, where the original clause contained n literals. If n-1 = 1, i.e., if the new clauses generated are one-iteral clauses, then they are N-matched against the existing one-literal clauses on Shelf 6, where an N-Match proves the matrix inconsistent and the original formula valid. If a new one-literal clause does not N-Match the existing one-literal clauses, and if it is redundant, then it is deleted, but if it is not redundant, then it is stored at the front of Shelf 6, with the existing one-literal clauses. If n-1 is greater than 1, i.e., if the new clauses generated are polyliteral clauses, then they are stored at the front of Shelf 9 with the existing polyliteral clauses, and the program again attempts to apply DS' to the first polyliteral clause on Shelf 9. If, however, the first polyliteral clause contains no terms that N-match any of the one-literal clauses, then it is stored on another shelf, i.e., Shelf 13, on which the original polyliteral clauses as well as all the new polyliteral clauses are stored. If and when the original list of polyliteral clauses on Shelf 9 becomes exhausted without resulting in a proof, and if one or more new one-literal clauses have been generated in the course of running through Shelf 9, then the polyliteral clauses on Shelf 13 are transferred to Shelf 9, and the process begins anew. If, however, no more one-literal clauses were generated, or if there were no one-literal clauses to start with, then the algorithm reverts to the older method of generating QFSI and testing for consistency after each generation. The earlier version of our program used the Davis-Putnam algorithm for testing conjunctions of QFSI for consistency; the present version, however, uses only one of the three Davis-Putnam rules, the so-called "one-literal clause rule," which may be defined as follows:

One-literal clause rule: If P is a one-literal clause (i.e , a conjunct containing no disjunction operators, and which is therefore an atomic wff), then all conjuncts containing P and all single occurrences of not-P are deleted from C (it is assumed that all tautologous conjuncts have been previously deleted from c, so that no conjunct contains both P and not-P).

The one-literal clause rule is applied to a formula in conjunctive normal form until all the one-literal clauses (if any) are deleted or until two one-literal clauses are found to be mutually inconsistent. Since the application of the rule may produce new one-literal clauses, it is necessary to test the one-literal clauses for consistency after each application. The condition of inconsistency is the occurrence of two contradictory one-literal clauses, and the condition of consistency is the deletion of the entire formula. The Davis-Putnam algorithm contains two rules that are applied to the formula in the event that the one-literal clause rule fails to give a decision; our earlier program included these two rules, but they have been omitted from the present version of the program, partly because they use up a lot of machine time, but mainly because they are not guaranteed to return the program to the one-literal clause rule, which is the most efficient of the three rules. In place of the latter two Davis-Putnam rules, therefore, we have substituted a branching feature, based on the method described in Quine's *Methods of Logic*[18]. Whenever the one-literal clause rule cannot be applied, the formula is split into two branches, by finding the first term of the formula, assuming it first true and then false, and making appropriate cancellations. Letting P be the first term, the first branch is produced by deleting entire conjuncts containing P and individual occurrences of not-P; the second branch is produced by deleting entire conjuncts containing not-p and individual occurrences of P. The second branch is stored at the front of a shelf, and the first branch remains in the workspace where an attempt is made to apply the one-literal clause rule to it. If this attempt fails, the formula in the workspace is split again in the same way, the first branch remaining in the workspace and the second branch being stored at the front of the shelf. This procedure continues until a branch is obtained to which the one-literal clause rule can be applied. The consistency test terminates when one branch is proven consistent (i.e., is entirely deleted), in which case the entire formula is consistent, or when all branches are proven inconsistent (i.e., contain contradictory one-literal clauses), in which case the entire formula is inconsistent.

Most of the theorems that we have submitted to the proof procedure program have been proven solely by the use of the N-matching procedure, and do not require the generation and testing of QFSI. In fact, all the theorems that have resulted so far from the logic translation routine are of this sort; their negated matrices contain one or more one-literal clauses that are proven to be ultimately contradictory by the N-matching procedure. It may require as long as five or six minutes to prove theorems whose negated matrices contain no one-literal clauses, but we have not found this to be a serious difficulty since we have used the proof procedure program primarily in conjunction with the logic translation routine contained in Sections DA and DB of the program.

Conclusion

As we have seen, the program described in this paper is capable of proving a number of moderately complicated arguments in ordinary English that require propositional logic or first-order functional logic for their symbolization. We therefore regard the program as a significant contribution to our understanding of the logic of natural language sentences. This area of study is of interest not only for its own sake but also for its potential applications to question-answering and information-retrieval systems, where the answer to an inquiry posed in natural language is usually not explicitly stored but must be found by making deductions on the basis of the sentences or facts already stored. The most interesting part of the program, at least in our view, is not Section DA, since there are many dictionary lookup and parsing routines already in existence, nor is it Section DC, since logical proof-procedure programs are not exactly new either. Rather, it is the combination of linguistic and logical analysis that is found principally in Section DB, and the coming to grips with the special problems that arise from the union of the two, such as the matter of the correct level of analysis to employ for the logical translation of an argument. The selection criterion that we have devised for choosing Analysis I, II, III, or IV, though a bit crude, has nevertheless considerable practical value in that it enables the program to avoid a lot of unnecessary logical computation; it amounts, after all, to doing a certain amount of logical calculation in advance of entering the actual proof-procedure routine, and an important question, worthy of further investigation, is just how much of the logic can be done in this way. Perhaps in some cases it would be possible to prove some arguments valid or invalid without employing a proof-procedure algorithm at all, though this would require considerably more knowledge of the way in which people understand sentences and arguments in ordinary language and of their methods of reasoning. In particular a better understanding of the way in which a person rejects an invalid argument should be of considerable use, since a proof-procedure is designed to prove arguments valid rather than invalid, and in the areas that we view as providing the most important potential applications of our research, rejection is just as important as proof. In law, for example, it is just as important to know which actions are illegal as to know which are legal.

Apart from applications that require logical computation, a logic translation program such as ours may be of use in mechanical translation, or in abstracting and paraphrasing, where it would be desirable to store a concise formal representation of the input sentences. The logical language used would therefore be a kind of intermediate language, and would have to operate in conjunction with programs for translating the logical forms into sentences of the various output languages or for piecing them out with semantic content in various other ways. These purposes would of course require a logical language considerably more expressive than the first-order predicate calculus on which our program is based.

In view of the many actual and potential applications of our research into the logic of natural language sentences, therefore, it is clear that this subject is by way of becoming an important branch of computational linguistics, and that the further development of programs like the one that we have described will be of considerable practical as well as theoretical significance.

Received May 3, 1965

References

1. Simmons, R. F., "Answering English questions by computer: a survey," *Communications of the ACM,* Volume 8, No. 1, January 1965, pp. 53-70.
2. Copi, I. M., *Symbolic Logic,* Macmillan, New York, 1958, p. 140, example 3.
3. Yngve, V. H., "A framework for syntactic translation," *Mechanical Translation,* Vol. 4, No. 3, December 1957, pp. 59-65.
4. Quine, W. V., *Mathematical Logic,* Harvard University Press, Cambridge, Mass., 1958, pp. 23-26.
5. Yngve, V. H., *An Introduction to* COMIT *Programming,* M.I.T. Press Cambridge, Mass., 1962.
6. Yngve, V. H., COMIT *Program mers' Reference Manual,* M.I.T. Press, Cambridge, Mass., 1962.
7. Yngve, V. H., "A programming language for mechanical transla-

tion," *Mechanical Translation,* Vol. 5, No. 1, April 1958, pp. 25-41.

8. Copi, I. M., *Symbolic Logic,* Macmillan, New York, 1958, p. 140, example 1.
9. Copi, I. M., *Introduction to Logic,* Macmillan, New York, 1955, p. 275, example 4.
10. Wang, H., "Toward mechanical mathematics," IBM J. *Res. Develop.,* Vol. 4, No. 1, January 1960, pp. 2-22.
11. Davis, M. and Putnam, H., "A computing procedure for quantification theory," JACM, Vol. 7, No. 3, July 1960, pp. 201-215.
12. Guard, J. R., "Automated logic for semi-automated mathematics," Contract No. AF19(628)-3250, Project No. 5632, Task No. 563205, Scientific Report No. 1, Applied Logic Corporation, Princeton, New Jersey, March 30, 1964.
13. Copi, I. M., *Symbolic Logic,* Macmillan, New York, 1958, p. 140, example 4.
14. *Ibid.,* example 6.
15. *Ibid.,* example 8.
16. *Ibid.,* p. 158, example 1.
17. Guard, J. R., *op. cit.,* pp. 23-24.
18. Quine, W. V., *Methods of Logic,* Holt, Rinehart, and Winston, New York, 1959, pp. 26 ff.