American Journal of Computational Linguistics *Microfiche 32*

PROCEEDINGS

13TH ANNUAL MEETING

ASSOCIATION FOR COMPUTATIONAL LINGUISTICS

1 : LANGUAGE UNDERSTANDING SYSTEMS

Timothy C. Diller, Editor

Sperry-Univac
St. Paul, Minnesota 55101

PREFACE

The 13th annual ACL meeting was held at Boston, Massachusetts, October 30 - November 1, 1975, in conjunction with the 38th meeting of the American Society for Information Science. The ACL thanks the ASIS for its assistance in publicizing the conference and in handling registration.

This and the following four microfiches contain 27 of the 30 papers presented at the meeting. The breadth of the conference is evident in (a) the modes of communication investigated (speech, sign language, and written text), (b) the styles of communication (monologues, dialogues, and note making), and (c) the uses envisioned for the processing of language data (e.g., theoretical modeling, data collection and retrieval, game playing, story generation, idiolect characterization, and automatic indexing).

Topics considered include the development of language understanding systems, the integration and utilization of specific components of language, specifically syntax and semantics, the representation and use of discourse structure and general world knowledge, and the construction of text processing systems.

The program committee was solely responsible for selecting the talks to be given, and hence the papers to be published herein. (Regretfully, nearly half of those submitted

could not be accepted for lack of program time.) Members of the program committee were Jonathan Allen, Joyce Friedman, Bonnie Nash-Webber, and Chuck Rieger. A special word of appreciation is due Jonathan Allen, who also served as Local Arrangements Chairman. Working with him were Betty Brociner and Skip McAfee of the ASIS. Aravind Joshi, president of ACL, provided guidance in all areas of preparation.

The AJCL kindly provided advance publication of the accepted abstracts and now makes possible the publication of the entire proceedings. David Hays, editor of AJCL, provided guidance in publication format and each author provided final copy in accordance with requested specifications. The Center for Applied Linguistics (in particular, David Hoffman and Nancy Jokovich with guidance from Hood Roberts) contributed in a variety of ways, most notably in the preparation of meeting handbooks.

This microfiche contains the papers as submitted by their authors for five of the six talks touching on Language Understanding Systems. The paper detailing "Conceptual Grammar" by William Martin was too long for inclusion in this microfiche and will appear elsewhere. My thanks to Yorick Wilks for chairing the session.

--Timothy C. Diller

Program Committee Chairman

TABLE OF CONTENTS

Program Schedule 5

SESSION 1: LANGUAGE UNDERSTANDING SYSTEMS

PEDAGLOT and Understanding Natural Language Processing *William Fabens* 9

A General System for Semantic Analysis of English and its Use in Drawing Maps from Directions *Jerry R. Hobbs* . 21

An Adaptive Natural Language Parser *Perry L. Miller* . . . 42

Conceptual Grammar (abstract only) *William A. Martin* . . . 57

Semantic-based Parsing and a Natural-language Interface for Interactive Data Management *John F. Burger, Antonio Leal, and Arie Shoshani* 58

PHLIQA 1: Multilevel Semantics in Question Answering *P. Medema, W. J. Bronnenberg, H. C. Bunt, S. P. J. Landsbergen, R. J. H. Scha, W. J. Schoenmakers, and E. P. C. van Utteren* . . . 72

THIRTEENTH ANNUAL MEETING
THE ASSOCIATION FOR COMPUTATIONAL LINGUISTICS

Sheraton Boston Hotel
Boston, Massachusetts

October 30-November 1, 1975

Thursday, October 30, 1975

SESSION 1: LANGUAGE UNDERSTANDING SYSTEMS
Session Chairman: Yorick Wilks - University of Edinburgh

9:00 A.M. ***Greetings and Introductory Remarks***

9:15 A.M. ***PEDAGLOT and Understanding Natural Language Processing***
William Fabens - Rutgers University

9:40 A.M. ***A System for General Semantic Analysis And Its Use In Drawing Maps from Directions***
Jerry R. Hobbs - The City College of CUNY

10:05 A.M. ***An Adaptive Natural Language Parser***
Perry L. Miller - MIT.

10:30 A.M. COFFEE & DONUTS

11:00 A.M. ***Conceptual Grammar***
William A. Martin - MIT

11:30 A.M. ***Semantic-Based Parsing And A Natural-Language Interface For Interactive Data Management***
John F. Burger, Antonio Leal, and Arie Shoshani - System Development Corporation

12:00 NOON ***PHLIQA 1: Multilevel Semantics in Question Answering***
P. Medema, et. al - Philips Research Laboratories, The Netherlands

12:30 P.M. LUNCHEON BREAK

SESSION 2: LANGUAGE GENERATION SYSTEMS

Session Chairman: Martin Kay - Xerox Corporation

2:00 P.M. — ***A Framework for Writing Generation Grammars for Interactive Computer Programs***
David McDonald - M I T.

2:30 P.M. — ***Incremental Sentence Processing***
Rodger Knaus - Bureau of the Census

3:00 P.M. — ***A Lexical Process Model of Nominal Compounding In English***
J.R. Rhyne - University of Houston

3:30 P.M. — COFFEE & DONUTS

4:00 P.M. — ***Generation as Parsing from A Network into a Linear String***
Stuart C Shapiro - Indiana University

4:30 P.M. — ***Speech Generation from Semantic Nets***
Jonathan Slocum - Stanford Research Institute

5:00 P.M. — ***Using Planning Structures to Generate Stories***
Jim Meehan - Yale University

5:30 P.M. — DINNER BREAK

8:00 P.M. — WINE, CHEESE & COMPUTER DEMONSTRATIONS

Friday, October 31, 1975

SESSION 3: PARSING, SYNTAX, AND SEMANTICS

Session Chairman: Joyce Friedman - Stanford Research Institute

9:00 A.M. — ***Syntactic Processing in the BBN Speech Understanding System***
Madeline Bates - Bolt, Beranek & Newman, Inc

9:30 A.M. — ***System Integration and Control for Speech Understanding***
William H. Paxton and Ann E. Robinson
Stanford Research Institute

10:00 A.M. — ***A Tuneable Performance Grammar***
Jane J. Robinson - Stanford Research Institute

10:30 A.M. COFFEE & DONUTS

11:00 A.M. *Semantic Processing for Speech Understanding*
Gary G. Hendrix - Stanford Research Institute

11:30 A.M. *SPS: A Formalism for Semantic Interpretation and Its Use in Processing Prepositions that Reference Space*
Norman K. Sondheimer - Ohio State University

12:00 NOON *The Nature and Computational Use of a Meaning Representation for Word Concepts*
Nick Cercone - University of Alberta

12:30 P.M. LUNCHEON BREAK

SESSION 4: MODELING DISCOURSE AND WORLD KNOWLEDGE I

Session Chairman: Carl Hewitt - MIT

2:00 P.M. *Establishing Context in Task-Oriented Dialogs*
Barbara G. Deutsch - Stanford Research Institute

2:30 P.M. *Discourse Models and Language Comprehension*
Bertram C Bruce - Bolt, Beranek & Newman, Inc

3:00 P.M. *Judging the Coherency of Discourse (and Some Observations About Frames/Scripts)*
Brian Phillips - University of Illinois at Chicago Circle

3:30 P.M. COFFEE & DONUTS

4:00 P.M. *An Approach to the Organization of Mundane World Knowledge: the Generation and Management of Scripts*
R.E. Cullingford - Yale University

4:30 P.M. *The Conceptual Description of Physical Activities*
Norman Badler - University of Pennsylvania

5:00 P.M. *A Frame Analysis of American Sign Language*
Judy Kegl (MIT) and Nancy Chinchor (U. of Mass.)

5:30 P.M. ACL BUSINESS MEETING AND ELECTION OF OFFICERS

DINNER: ACL BANQUET

Saturday, November 1, 1975

SESSION 5A: MODELING DISCOURSE & WORLD KNOWLEDGE II

Session Chairman: Georgette Silva - System Development Corporation

9:00 A.M. *Cross-Sentential Reference Resolution*
David Klappholz and Abe Lockman - Columbia University

9:30 A.M. *How Does a System Know When to Stop Inferencing?*
Stan Rosenschein - University of Pennsylvania

10:00 A.M. COFFEE & DONUTS

SESSION 5B: TEXT ANALYSIS

10:30 A.M. *Developing a Computer System for Handling Inherently Variable Linguistic Data*
David Beckles, Lawrence Carrington, and
Gemma Warner - The University of the West Indies

11:00 A.M. *A Natural Language Processing Package*
David Brill and Beatrice T Oshika -
Speech Communications Research Laboratory

11:30 A.M. *On the Role of Words and Phrases in Automatic Text Analysis and Computation*
Gerard Salton - Cornell University

12:00 NOON *Grammatical Compression in Notes and Records: Analysis and Computation*
Barbara Anderson (University of New Brunswick),
Irwin Bross (Roswell Park Memorial Institute),
and Naomi Sager (New York University)

American Journal of Computational Linguistics *Microfiche 32 : 9*

P E D A G L O T AND UNDERSTANDING NATURAL LANGUAGE PROCESSING

WILLIAM FABENS

Computer Science Department
Rutgers University
New Brunswick, New Jersey 08903

ABSTRACT

PEDAGLOT is a programmable parser, a 'meta-parser.' To program it, one describes not just syntax and some semantics, but also--independently--its modes of behavior. The PEDAGLOT formulation of such modes of behavior follows a categorization of parsing processes into attention-control, discovery, prediction and construction. Within these overall types of activities, control can be specified covering a number of syntax-processing and semantics-processing operations. While it is not the only possible way of programming a meta-parser, the PEDAGLOT mode-specification technique is suggestive in itself of various new approaches to modeling and understanding some language processing activities besides parsing, such as generation and inference.

This work was sponsored by through NIH Grant #RR643.

Introduction

It is well known that to process natural language, one needs both a syntactic description of possible sentences, blended in some way with a semantic description of a certain domain of discourse, and a rather detailed description of the actual processes used in hearing or producing sentences.

An augmented transition network (Woods, 1970) is an example of the blending of syntactic and quasi-semantic descriptions. Here registers would be repositories of, or pointers to, semantics. When used in conjunction with a semantic network, an ATN can be used to parse or to generate (Simmons and Slocum, 1972) sentences. The issue of changing the description of the actual processes used in such systems has been touched on by Woods (in using a 'generation mode'), to some extent by Simmons and Slocum (using decision functions to control style of generation), and to a larger extent by Kaplan (1973), in his General Syntactic Processor, GSP. GSP indeed is one example of a system in which syntax, semantics and to some extent processes can each be usefully defined.

If we look at syntax, semantics and processes as three describable components, these systems just mentioned illustrate how thoroughly intertwined they can become--to the extent that theorists from time to time deny the existence or at least the importance of some one of them. Ignoring that dispute, I would like to concentrate on the question of being able to comprehensively describe one's theory of language in terms of its syntax, semantics and processes in a way that allows for their necessary and extensive intertwining connections, but at the same time allows one to describe them independently.

I came to the need for doing this while designing a 'relaxation parser,' a parser which can make grammatical relaxations if it is given an ill-formed string, so as to arrive at a 'closest' possible parse for the string. This problem involved describing a 'correct' grammar and then (in some way) describing a space of deviations

that might be allowed by the parser. Thus the syntax would be fixed and the way the parser uses it would separately have to be described. It was soon noticed that efficiency could be greatly enhanced if some rudimentary notion of semantic plausibility could also be used. It would have to be described in a way related to the correct syntax but still be usable by the parser. Thus, for my purposes, the descriptions had to be independent of one another.

One feature of a relaxation parser is that it can 'fill in the gaps' of a string that is missing various words. If one could, which my relaxation parser did not, specify the semantic context of a sentence, the generated sentence might be semantically rather plausible. In any case, the relaxation parser operates in various respects like an actual parser or like a generator, and it was this relationship between parsing and generating that became of interest.

Out of the design of the relaxation parser, the notation (independent of syntax) which to some extent describes various processes and choices of alternate ways of processing was developed. Thus, one may take a set of syntax and semantic descriptions and then through describing the processing 'modes' involved, define a processor which uses the particular algorithm that the individual processes together define. One may call the parser that is programmable in its processes a meta-parser, of which various existing parsers and generators appear to be special cases.

A closer examination of the parser I have developed (called PEDAGLOT*) may show some such aspects of meta-parsing, especially as regards the relationship between parsing and generating. I will describe the syntactic and semantic parts of the parser first by noting its resemblances to the parser of J. Earley (1970) and the ATN system of Woods. Then I will describe the process-type specifications that are available, and the use of meta-parsers as a basis for defining general language behaviors. Further detail can be found in the PEDAGLOT manual (Fabens, 1972 and 1973).

*for pedagogic polyglot

1. The Core of the Parser

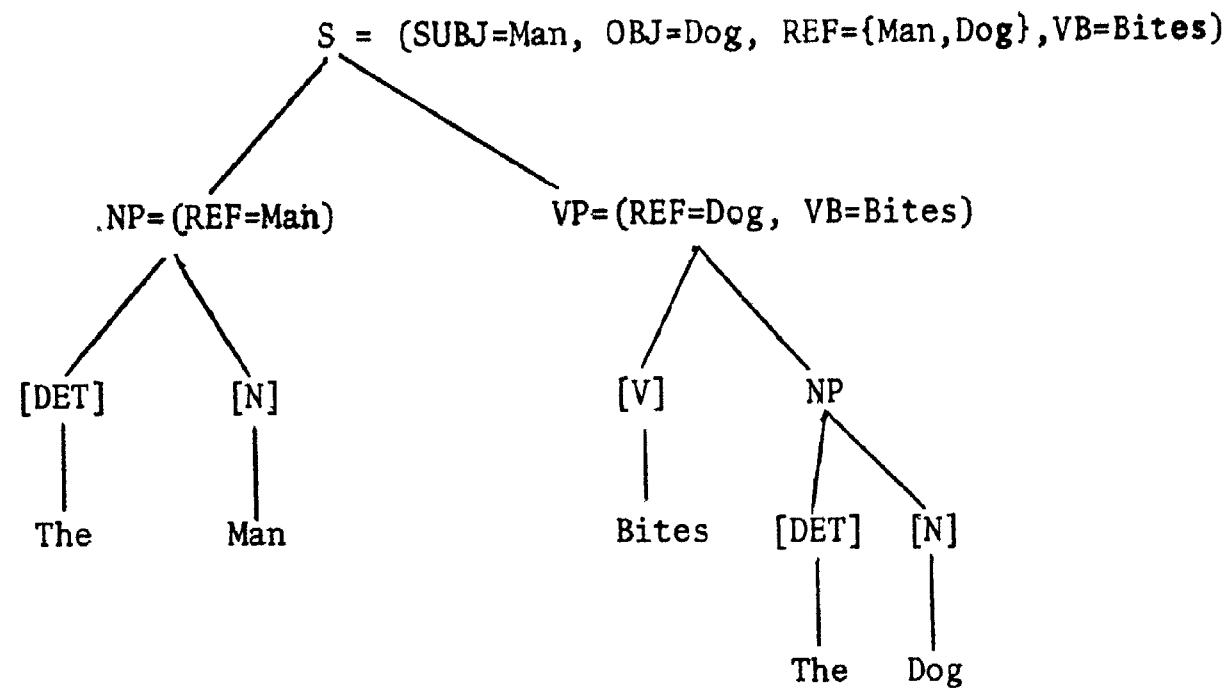
The fundamental operation of the parser is very similar to the operation of Earley's parser, with augmentations for recording the results of parses (e.g., their tre structure, and various of their attributes, which I call 'tags'). It is given a grammar as a set of context-free rules with various extensions, most important of which are that LISP functions may be used as predicates instead of terminals, and that each rule may be followed by operations that are defined in terms of the syntactic elements of the rule in question.

An example of this notation is as follows:

```
S → NP  VP
  => [AGREE [REF NP][VB VP]]
     [SUBJ = [REF NP]][OBJ = [REF VP][VB = [VB VP]]
S → NP [BE][VPASS] BY NP
  => [AGREE [REF NP][VB[BE]]]
     [SUBJ = [REF NP']][OBJ = REF NP]][VB = [VB[VPASS]]]
NP → [DET][N]
     => [REF = [N]]
VP → [V]NP
     => [VB = [V]][REF = [REF NP]]
```

Here, each bracketed symbol is the name of a recognition predicate (e.g., [N] recognizes nouns, [BE] recognizes forms of 'to be'). Following the => are the post-recognition functions. For instance [AGREE [REF NP][VB VP]] specifies a call to the AGREE function which is given, as arguments, the REF attribute (tag) of the sub-parse involved in that rule and the VB attribute of the VP part of the rule.

Following is a parse tree for 'The Man Bites the Dog' and values of tags after the parse.



The general flow of the parser is from top-down, and as the lowest components (symbols in the string) are found, the post-recognition functions that are associated with the rule that recognized them are applied. Tags become associated with sub-parses when the post-recognition operation uses the form [x = y] (in which the value referenced by y is stored as the x tag of the sub-parse). In the example, [DET] and [N] recognize 'The Man' and 'Man' is used as the REF attribute of the first NP. In the second S rule, the operation of [SUBJ = [REF NP']] would be to retrieve the REF tag of the second NP (thus the prime), and to store that as the SUBJ tag of the final parse.

As in most top-down parses, this parser begins with S and its two rules, since S is non-terminal. S is expanded into the two sequences of matches it should perform. This expansion results in various (in this case, two) predictions of what to find next. When the initial symbol in some rule is a terminal or a predicate, a discovery is called for (in which a match is performed, possibly involving the known values of the tags). When some complete sequence of elements is found (here, for instance, when NP → [DET][N] has matched the [N]). Construction invokes the post-recognition operations and then usually completes some earlier part of a rule (here, the 'NP' of S → NP VP) so further predictions (involving VP) or discoveries

are then specified.

I have broken up the parsing process into these three parts so as to similarly catalog the 'parsing modes,' turning this parser into a meta-parser. Before doing so, I should note that this parser stores each result under construction in a 'chart' as is done by Kaplan in his GSP, so that, for instance, the NP 'test' will only have to be evaluated once for each place one is wanted in the string.

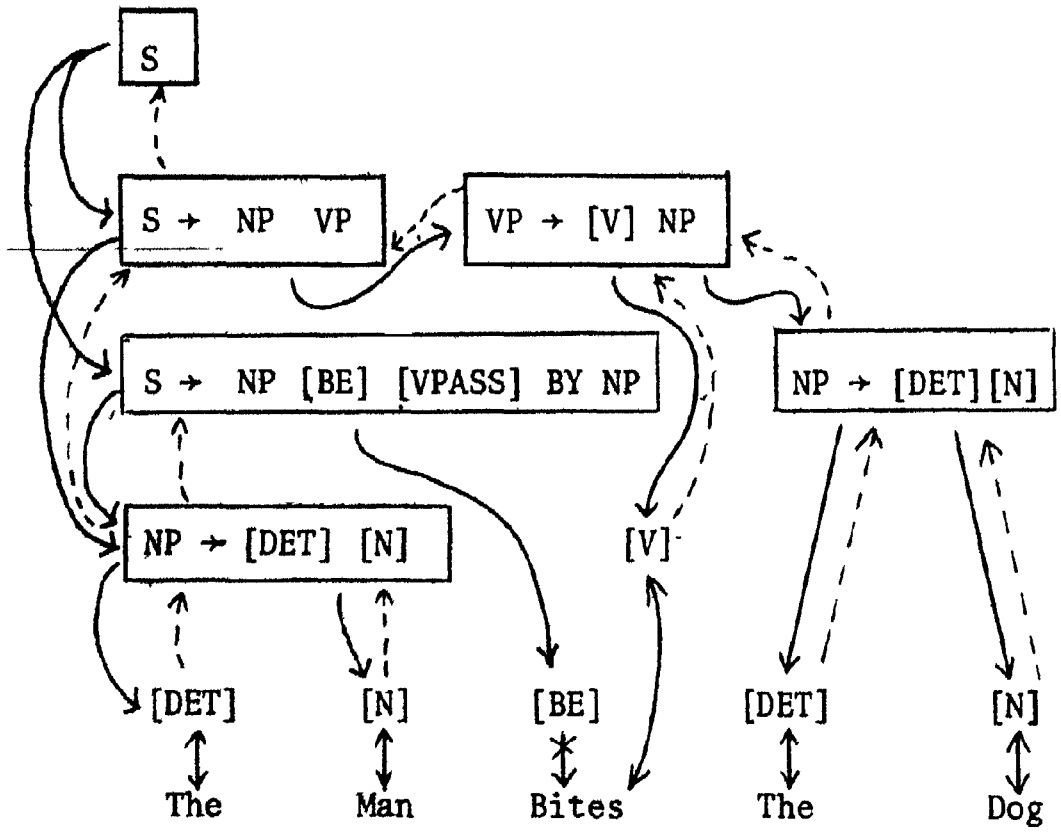


Illustration of PEDAGLOT's Parsing Chart

Simple Arrows indicate 'Predictions.'

Double Head Arrows indicate 'Discoveries.'

Dotted Arrows indicate "Construction.'

Also, for various well known reasons of efficiency, Earley's concept of independent processing of syntactic events is used (combined conceptually with the chart), so that a main controller can evaluate the individual syntactic 'tests' in almost any order, and not just in a backtracking sense (cf. Woods, 1975). The efficiency is realized here since many 'partial parses' (partially recognized forms)

are effectively abandoned if other results can complete the parse, or a sub-parse, first.

2. Meta-Parsing Modes

One can see that, except for the notational inefficiencies of the context-free formalism (as opposed to the augmented transition network form), this parser is very much like other standard parsers (especially ATN's). It differs in that there is a way of specifying how to proceed. Currently, this system has approximately a dozen 'modes' and I will present some of them here. Each mode specifies how to handle a certain part of the parsing process. They can be classified into four categories: attention control, prediction, discovery and construction.

a. Attention Control Modes:

Since the parser operates on a chart of independent events ('parsing questions'), one must give the parser a method of sequencing through them. Thus, one may specify 'breadth-first' or 'depth-first' and the appropriate mechanism will be invoked (this merely involves the way the processor stacks its jobs). A 'best-first' option is under development, which, when given an evaluation function to be applied to the set of currently active partial parses, allows the system to operate on the 'best' problem next. Experiments with this mode have so far been inconclusive.

One also can specify when to stop (i.e., at the first complete parse, or to wait until all other ambiguous parses have been discovered). The disambiguation routine (which is described as a part of the construction modes) defines which parse is 'best'. Further, one may specify a left-to-right or right-to-left mode of how to progress along the string.

b. Discovery Modes:

The starting point of building a relaxation parser is to specify what to do when an exact match is not made. If the parser is expecting one word and finds another it can look around the indicated place in the string to find

what it is looking for, or it can in certain other circumstances simply insert the expected word into the string. Thus, under discovery modes, there are various options: either the parser is allowed to attempt matches in out-of-sequence parts of the string, or not. And if not, or if no such match is found, the parser may or may not be allowed to make an insertion.

So in PEDAGLOT, there is an INSERT mode (and various restricted versions of it) and a 'where to look' mode which is used to control the degree to which the parser can try to find out-of-place matches. There are tags associated with these two specifications, the INSERT tag and the OMIT tag, which are associated with the parses involving insertions and omissions that contain the number of insertions made and the number of input symbols omitted in building the parse.

There is also a rearrangement mode. Thus, given certain constraints, the parser could be given 'The Bites Man Dog' and produce a parse for 'The Man Bites the Dog' since it would have found 'Man,' by temporarily omitting 'Bites,' but then it looks for and finds 'Bites' and finally, finding no second 'the,'the,' inserts one (or some other determiner because of the [DET] function]) and finds 'Dog. In a similar way it would try to produce a passive form (i.e., the Man Is Bitten By the Dog) but since this involves more insertions, etc. it would not be chosen.

These heuristics are controlled by recording numerical summary tags with each sub-parse that participate in, and are 'judged' by the disambiguation routines. Similar ideas are used by Lyon (1974).

c. Prediction Modes:

As Woods (1975) has pointed out, the extent to which a parser's prediction increases efficiency varies with the quality of the expected input. This fact affects greatly our discovery procedures, since, if insertions are to be made, one ought to be rather sure of one's predictions, or risk a combinatorial ex-

plosion. In PEDAGLOT, there is a programmable 'choice' function that controls predictions. Specifically, when the parser encounters a non-terminal symbol, that symbol is the left-hand side of various rules. An uncontrolled prediction (used by a canonical top-down parser) is to select each such rule as the expansion. Intuitively, however, people do not seem to do this. Instead, as in an ATN, they try one and only if that fails, go into the next. In PEDAGLOT, the choice of which rule to try can be defined as the result of the call to a 'choose' function (or it can be left uncontrolled). We have designed various approaches to such predictions (e.g., a limited key-word scan of the incoming string, and the use of 'language statistics' such as the set of rules which can generate the next symbol in the string as their left most symbol).

The prediction is currently made once for any given choice point; its outcomes are expected to be an ordered set of rules to try next.

d. Construction Modes:

The phase of parsing in which the parts of the parse tree and associated tag values are formed, is a place where most of the non-syntactic information (tags) about the string being parsed can come into play.

In the first place, new tags can be formed as functions of lower level parse tags through a process called melding. Thus, 'nonsense' can be discovered and pronoun references can sometimes be tied down. In the second place, it is a result of construction that ambiguity is discovered and dealt with.

Since these features of parsing deal primarily with semantics (and since, if anywhere, semantic representations of the string reside in the tags), most of the PEDAGLOT construction modes involve tags.

One may explicitly meld tag values by using post-recognition operators, or one may define an 'implicit' melding routine that is associated with the tag

names themselves instead of with individual rules. In our example we use this device to implicitly form a simple list of the two REF tags that become associated with the S rule. This implicit melding operation can also include a blocking function, or some reference to a data base. The tags that contain INSERT and OMIT information are used in this way to keep running totals of, and to minimize the number of such heuristics in the relaxation parsing modes. One may also associate a LIFT function which, when the partial parse becomes complete, specifies a transformation of that tag to be used as the tag of the next higher level parse.

Ambiguity is discovered when two parses from the same symbol, covering the same string segment are found. For this case, an AMBIG function is associated with tag names, and it makes a 'value judgement' of which tag is 'better, hence which interpretation to use. (Other types of criteria can also come into play here such as user interaction, (cf. Kay, 1973).

3. The Uses of Meta-Parsers

I have just catalogued some of the parsing modes available in PEDAGLOT. Others, such as Bottom-Up (instead of Top-Down) or Inside-Out (instead of Left-to-Right, etc.), are envisioned but not implemented. Since PEDAGLOT is an interactive program, the user can change modes at will, just as he can change syntax or introduce new tags. Thus, the obvious first use of meta-parsers is that one may use them to design language processors without having to tie oneself down from the start to say, a depth-first parser.

Meta-parsers also have a certain amount of tractibility that parsers that blend all activities into one huge network may not. One may see at a rather high level what is going to be happening (i.e., all tags of a certain name will meld together in a certain way, unless the grammar specifies otherwise). If one, however, wants certain forms of local behavior, one may use predicates or functions on individual rules. Further, if one wants to change the order in which predictions are evaluated,

one can program a 'choose' function which will make that global change. To a large extent, the language designer may specify much of the processor in broad terms and still be able to control local events where necessary.

In a more general sense, a meta-parser allows one to understand and build higher order theories about how people might represent and process language.

For instance, while it may be true that generating is the inverse of parsing, there is more than one way to do such inverting. One could start from a semantic network, using the choose function along with the INSERT mode to restrict means of expression consistent with the intended message, and using AMBIG functions to weed out all but reasonable messages from among the many the parser may produce or one might simply take from the semantic network a simple string of meaningful words, and then use a less tightly programmed 'relaxation parser' to rearrange these words to be syntactically correct. We are now considering using a crude 'backwards' mode which begins with the operation part of a rule and, by using predicates (e.g., AGREE) to yield inverses, specifies what the context-free pattern must produce. Thus there are many variations of how to generate using a meta-parser.

In the area of language inference, to take another example of language processing, PEDAGLOT suggests various differing ways of approaching the problem. First, one may use it as a 'relaxation-parser,' the 'parse tree' can be pattern-matched against the new sentence, and hypotheses can be formed. Or, one could place a more rudimentary inference system on the 'prediction' part of the processor itself, and using other controls, the predictions that are successful could be rewritten as a new grammar. These two learning paradigms could each be strengthened by way of the use of tags to contain (in a sense) the meaning of the sentences to be learned. Each of these paradigms can be modeled using a meta-parser like PEDAGLOT. Thus, a meta-parser can raise (and be prepared to answer) a number of interesting questions.

References

Earley, J. (1970), "An Efficient Context-Free Parsing Algorithm," Comm. ACM 13, number 2, (February 1970), pp. 94-102.

Fabens, W. (1972), PEDAGLOT Users Manual, Rutgers University CBM-TR-12, Oct. 1972.

Fabens, W. (1973), PEDAGLOT Users Manual: Part II, Rutgers University CBM-TR-23, Nov. 1973.

Kaplan, R.M. (1973), "A General Syntactic Processor," in R. Rustin (ed.) Natural Language Processing, New York: Algorithmics Press, (1973), pp. 193-242.

Kay, M. (1973), "The MIND System," in R. Rustin (ed.) Natural Language Processing, New York: Algorithmics Press, (1973), pp. 155-188.

Lyon, G. (1974), "Syntax-Directed Least-Errors Analysis for Context-Free Languages: A Practical Approach." Comm. ACM 17, number 1, (January 1974), pp. 3-13.

Simmons, R. and Slocum, J. (1972), "Generating English Discourse from Semantic Networks." Comm. ACM 15, number 10, (October 1972), pp. 891-905.

Woods, W.A. (1970), "Transition Network Grammars for Natural Language Analysis." Comm. ACM 13, number 10, (October 1970), pp. 591-606.

Woods, W.A., (1975), Syntax, Semantics, and Speech, BBN Report No. 3067, A.I. Report No. 27. Bolt Beranek and Newman Inc., to appear in D.R. Reddy (ed.) Speech Recognition, Academic Press (1975).

American Journal of Computational Linguistics *Microfiche 32 : 21*

A General System for Semantic Analysis of English and its Use in Drawing Maps from Directions

Jerry R. Hobbs

Department of Computer Science
The City College of the
City University of New York
Convent Avenue at 140th Street
New York, New York 10031

ABSTRACT

We describe a semantic processor we are constructing which is intended to be of general applicability. It is designed around semantic operations which work on a structured data base of world knowledge to draw the appropriate inferences and to identify the same entities in different parts of the text. The semantic operations capitalize on the high degree of redundancy exhibited by all texts. Described are the operations for interpreting higher predicates, for detecting some intersentential relations, and in particular detail, for finding the antecedents of definite noun phrases. The processor is applied to the problem of drawing maps from directions. We describe a lattice-like representation intermediate between the linguistic representation of directions and the visual representation of maps.

OVERVIEW[1,2]

We are trying to construct a semantic processor of some

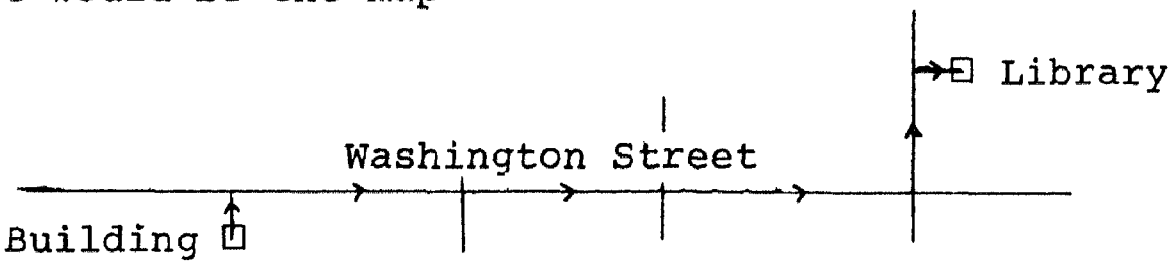
[1] This research was supported by the Research Foundation of the City University of New York under Faculty Grant No. 11233.
[2] The author would like to express his indebtedness to Harry Elam for many insights into the problems discussed here.

generality. We are using as our data base a set of facts involving spatial terms in English. To test the processor and to study the interfacing of semantic and task components, we are building a system which takes as input directions in English of how to get from one place to another and outputs a map, a map such as one might sketch for an unfamiliar region, hearing the directions over the phone.

A typical input might be the text

> "Upon leaving this building, turn right and follow Washington Street three blocks. Make a left. The library is on the right side of the street before the next corner." (1)

The output would be the map



To bypass syntactic problems, we are using as our input the output of the Linguistic String Project's transformational program (Grishman et al. 1973, Hobbs & Grishman), which is very close to a predicate-like notation. The semantic component is designed around general semantic operations which work on a structured data base of world knowledge to draw the appropriate inferences and to identify phrases in different parts of the text which refer to the same entity. The text, augmented and interrelated in this way, is then passed over to the task component, which makes arbitrary decisions when the map requires information not given by the directions and produces the map.

ORGANIZATION OF TEXT AND WORLD KNOWLEDGE

The two problems of semantic analysis are to find, out of a potentially enormous collection of inferences, the appropriate inferences, and to find them quickly. Our solution to the first is in our semantic operations described below. Our approach to the second problem is in the organization of the data base.

The data in the semantic component is of two sorts:

1. The Text: the information which is explicitly in the text. In the course of semantic processing this is augmented by information which is only implicit in the text. The text consists of the set of entities $X_1, X_2, \ldots$, explicitly and implicitly referred to in the text, and structures of the form $p(X_1, X_2)$ representing the statements made or implied about these entities, e.g.

$\text{walk}(X_1)$ = X_1 walks,

$\text{building}(X_2)$ = X_2 is a building,

$\text{door}(X_3, X_2)$ = X_3 is a door of X_2.

2. The World Knowledge or the Lexicon: the system's knowledge of words and the world. Words are the boundary between the Text and the Lexicon. A word is viewed as a key indexing a large body of facts (Holzman, 1971).

Associated with each word are a number of facts or inferences which can be drawn from the occurrence of $p(X_1, \ldots, X_n)$ in the Text. The facts are expressed in terms of p's set of parameters $Y_1, \ldots, Y_k$, and a set of other lexical variables $z_1, \ldots, z_m$, standing for entities whose existence is also implied. A fact consists of enabling conditions and conclusions. When $p(X_1, \ldots X_n)$ occurs in the Text and the semantic operations determine a

particular inference appropriate, its enabling conditions are checked. If they hold, the conclusions are instantiated by creating a copy of them in the Text with the lexical variables replaced by Text entities.

Clusters. One way to state the "frames" problem (Minsky 1974) is "How should the data base be organized to guide, confine, and make efficient the searches which the semantic operations require?" We approach this by dividing the sets of inferences into clusters according to topic and salience in the particular application. In the searches, the clusters are probed in order of their salience. In our application, the top-level cluster concerns the one-dimensional aspects of objects and actions. For example, the fact about a block that it is the distance between two intersections is in the cluster. If "around the block" is encountered, less salient clusters will have to be accessed to find information about the two-dimensional nature of blocks. The most important fact about an apartment building is that it is a building, to be represented by a square on the map. But if the directions take us inside the building, up the elevator, and along the hallway, the cluster of facts about the interiors of buildings must be accessed.

A self-organizing list (Knuth 1973) of the clusters is maintained--when a fact in a cluster is used, it becomes the top-level cluster--on the assumption that the text will continue to talk about the same thing.

The "Truth Status" of Inferences. In natural language, unlike mathematics, one is not always free to draw certain

inferences. We tag our inferences always, normally, or sometimes. These notions are defined operationally. An always inference is one we are always free to draw, such as that a street is a path through space. A normally inference is one we can draw if it is not explicitly contradicted elsewhere, such as that buildings have windows. A sometimes inference may be drawn if reinforced elsewhere, such as the fact used below that a building is by a street. This classification of inferences cuts across the cluster structure of the Lexicon.

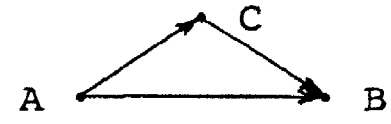
Lattices. A large number of statements in any natural language text, especially the texts this system analyzes, involve a transitive relation, or equivalently, say something about an underlying scale. For example, the word "walk" indicates a change of location along a path through space, or a distance scale; "turn" indicates a change along a scale of angular orientation.

In any particular type of text there are scales or transitive relations which are important enough to deserve a more economical representation than predicate notation. In this particular task, the important scales are a distance scale, a subscale of this indicating the path "you" will travel, and a scale representing angular orientation. This is the principal information used in constructing the map. For these scales we translate into a directed graph or lattice-like representation (Hobbs 1974).

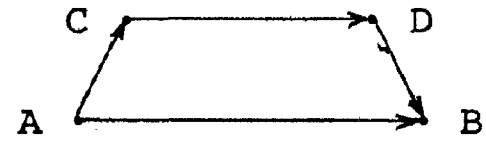
Some of the things which can be said about the structure of a scale are that some point is on the scale, that of two points on the scale one is closer to the positive end than the other,

and that a scale is a part of another scale. If a point B is closer to the positive end of the scale than point A, this fact is represented by

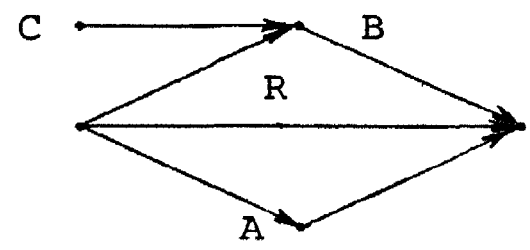
If point C lies in the interval from A to B the representation is



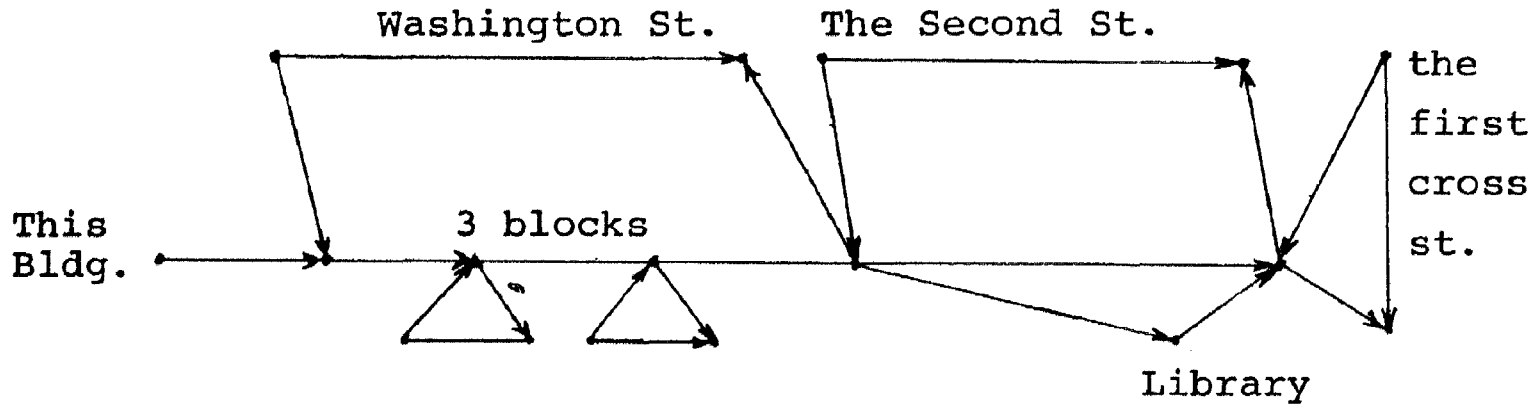
The diagram



means the scale from C to D is part of the scale from A to B. It is possible to represent incompleteness of information. For example, if it is known that points A and B both lie in a region R of a scale but their relative positions are not known and if it is known about C only that it precedes B this is represented by



The lattice for the distance scale for text (1) is as follows:



The lattices are intermediate between the linguistic representation of the directions and the visual representation of the maps. They are used at several points in the semantic and task

processes. They can be constructed for any transitive relation, and could be very useful, for example, in representing causal and enabling relations in a system translating descriptions of algorithms into flowcharts or programs.

SEMANTIC OPERATIONS

Basic Principle of Semantic Analysis. We believe the key to the first problem of semantic analysis, that of finding which inferences are appropriate, is Joos' Semantic Axiom Number One (Joos 1972), or what I will call the Principle of Knitting. Restated, this is, "The important facts in a text will be repeated, explicitly or implicity." That is, we capitalize on the very high degree of redundancy that characterizes all texts. Consider, for example, the simple sentence, "Walk out the door of this building." "Walk" implies motion from one place to another. "Out" implies motion from inside something to the outside. "Door" is something which permits motion from inside something to the outside or from the outside to the inside, or if closed, prevents this motion. "Building" is something whose purpose is for people to be in. Thus, all four content words of the sentence repeatedly key the same facts. Those inferences which should be drawn are those which are keyed by more than one element in the text.

This principle is used both formally and informally by the semantic operations. It is used formally in the interpretation of higher predicates and in finding antecedents. It is used more informally for deciding among competing plausible antecedents, resolving ambiguities, detecting intersentential relations, and knitting the text together in some minimal way. Here it is,

primarily the formal uses that will be described.

Interpretation of Higher Predicates. In "walk out", "walk slwoly", and "pleasant walk", the higher predicates "out", "slow" and "pleasant" all apply to "walk", but they narrow in on different aspects of walking. That is, each demands that a different inference be drawn from the statement that "X walks". "Out" and "slow" demand their arguments be motion from one place to another, forcing us to infer from "X walks" that "X goes from A to B". "Out" then adds information about the locations of A and B, while "slow" says something about the speed of this motion. "Pleasant", on the other hand, requires its argument to be an awareness, so we must infer from "X walks" that "X engages in a bodily activity he is aware of".

Stored in the Lexicon with each higher predicate is the inference which must be drawn from its argument and the information it adds to this inference. For example, "go(z_1,z_2,z_3)" must be inferred from the argument of "out". When the statement "out(walk(X_1))" is encountered in the Text, the higher predicate operation makes efforts to find a proof of "go(z_1,z_2,z_3)" from "walk(X_1)". The search for this inference is similar to the search procedure described below for finding antecedents. The facts in the resulting chain of inference are instantiated together with the information added by the higher predicate, and they are subsequently treated as though part of the explicit Text. It is usual for them to be useful in further processing, unless the modifier is simply gratuitous information.

Note that this operation allows considerable compression in

the number of senses that must be stored for each word. It allows us, for example, to define "slow" as something like "Find the most salient associated motion. Find the most specific Speed Scale for the object X of this motion. X's speed is on the lower end of this scale". This definition is adequate for such phrases as "walk slowly" (the most salient motion is the forward motion of the walking), "slow race" (the forward motion of the competitors), "slow horse" (its running at full speed, usually in a race), and "slow person". This last case is highly dependent on context, and could mean the person's physical acts in general, his mental processes, or the act he is engaged in at the moment.

This operation has a default feature. If a proof of the required inference can't be found, it is assumed anyway. This allows a text to be understood even if all the words aren't known. Suppose, for example, "veer right" is encountered, and the word "veer" isn't known, i.e. no inferences can be drawn from it. Since "right" requires a change in angular orientation as its argument, it is assumed this is what "veer" means. Only the information that the change is small is lost.

FIND ANTECEDENTS OF DEFINITE NOUN PHRASES

Entities referred to in a text may be arranged in a hierarchy according to their degree of specification:

1. proper names, including "you" and "I"
2. other noun phrases, including those with definite, indefinite, and demonstrative articles
3. third person pronouns
4. zeroed arguments and implied entities.

So far our work has concerned primarily definite noun phrases, but it is expected that many features of the definite noun phrase algorithm will carry over to other cases.

The definite noun phrase algorithm consists of four steps. First, "uniqueness conditions" are checked to determine whether an antecedent is required. If so, the Text and Lexicon are searched for plausible antecedents. Third, consistency checks are made on these. Finally if more than one plausible antecedent remains the Principle of Knitting is applied to decide between them.

Uniqueness Conditions. In the phrase "the end of the block", we know we must look back in the text for an explicitly or implicitly mentioned "block" (the search case), but we do not necessarily look for a previously mentioned "end" (the no-search case). Given a definite noun phrase the algorithm first tries to determine whether it belongs to the search or no-search case. This is done by checking two broad criteria. (These criteria were motivated by a large number of examples not only from sets of directions but also from technical and news articles.)

These criteria are checked by searching the Lexicon for certain features. However these searches are generally very shallow, in contrast to the potentially much deeper searches in the next step of the algorithm. Since by far the majority of definite noun phrases are in the no-search case, checking uniqueness conditions can result in great savings.

A caveat is in order. We state the criteria at a very high level of abstraction. We feel in fact that the algorithm can

work at that level of abstraction if the Lexicon is properly constructed. But how to construct a large Lexicon properly is a problem we have not yet tackled in detail. In any event, we give examples for each case, and the examples themselves form a reasonably exhaustive classification.

1. A definite entity is in the no-search case if it can be located precisely with respect to some framework. This includes the following conditions.

a. Objects which are located with respect to some identified point in space: "the building on the corner".

b. Plurals and mass nouns which are restricted to some identified region of space: "the trees in the park", "the water in the swimming pool". Here "the" indicates all such objects or substance.

c. Points and intervals in time which are fixed with respect to some identified event: "the minute you arrive", "the hour since you left".

d. Events in which at least some of the participants are identified and which can be recognized as occurring at a specific time: "the ride you took through the park yesterday".

e. Points or intervals on more abstract scales: "the end of the block", "the size of the building". The end is a specific point on the distance scale defined by the block. The size of the building is a specific point on the general size scale for objects, i.e. the volume scale.

f. Superlatives, ordinals, and related terms: "the largest house on the block", "the second house on the block", "the only

house on the block". If the set of comparison is identified, the superlative or ordinal indicates the scale of comparison and the place on that scale of the entity it describes. This is a subcase of (e).

All of these conditions can be checked in one operation if the facts in the Lexicon are expressed in terms of suitably abstract operators relating entities to scales. We simply ask if the definite entity is on or part of a scale or at a point on or along an interval of a scale, where the scale can be identified. However this requires that we take very seriously my suggestion in Hobbs (1974) that the lexicon for the entire language be built, insofar as possible, along the lines of a spatial metaphor. We have not yet had to face these problems since our only scales are physical -- our "at" and "on" are the locative "at" and "on".

Also checking this criterion presupposes a very sophisticated syntactic and semantic analysis. For example, (d) assumes that the times of events mentioned in tenseless constructions can be recovered.

2. A definite entity is in the no-search case if it is the dominant entity of that description. This divides into two sub-criteria:

a. Those entities which are unique or dominant by virtue of the properties which describe them: "the sun", "the wind". If the properties $p_1(X), p_2(X), \ldots$, are known about the definite entity X, the definitions of $p_1, p_2, \ldots$, are probed for the fact that the entity does not normally occur in the plural. Included under this heading are proper names beginning with "the", like

"the Empire State Building", and appositives, like "the city of Boston".

b. Those entities which are unique by virtue of the properties of an entity with which they are grammatically related: "the door of the building", "the Hudson River valley". "The door of the building" is represented in the Text as "$X_1|door(X_1,X_2|building(X_2))$" i.e. "the X_1 such that X_1 is the door of X_2 which is a building". The uniqueness or dominance of X_1 is not a property of "door" but of "building". Stored with "building" is the fact that a building has in its front surface a main door which does not normally occur in the plural. "The door of the building" is interpreted as this dominant door.

If the uniqueness conditions succeed, a pointer is set from the dominant lexical variable to the corresponding entity. If subsequently the same definite noun phrase occurs, the uniqueness check will discover this pointer and correctly identify the antecedent. Thus, we can handle the example

> "Walk up to the door of the building. Go through the door of the building."

Here the uniqueness check gives us a shortcut around the next step in the algorithm.

The Search for Plausible Antecedents. To illustrate the search for an antecedent, consider

> "Walk out the door of this building. Turn right. Walk to the end of the block."

What block? From "block" we follow a back pointer to the fact stored with "street" that "streets consist of blocks", and from

"street" the fact with "building" that "Buildings are by streets" Since a building is mentioned, we assume it is "the block of the street the building is on". The facts in the chain of inference leading to this are instantiated. An entity is introduced into the text for the "street" and the Text is augmented by the statements that "the building is on the street" and "the block is part of the street". This information turns out to be required for the map. Note that the fact that a building is on a street is a sometimes fact and that we are free to draw it only because "the block" occurs.

To conduct the search of the Lexicon, ideally we would like to send out a pulse from the word "block" which travels faster over more salient paths, and look for the first entity which the pulse reaches. The saliency is simulated by the cluster structure described above. The parallel process of the spreading signal is simulated by interleafing deeper probes from salient clusters with shallower probes from less salient clusters. For example, if "streets consist of blocks" is a cluster 1 fact, then we might probe for a cluster 1 fact involving streets and a cluster 2 fact involving blocks at roughly the same time. After one plausible antecedent is found in this way, the search is continued for possible antecedents which are nearly as plausible. If after a time no plausible antecedents are found, the search is discontinued.

Searches for antecedents are conducted not only for entities but also for definite noun phrases that the nominalization transformations of the syntactic component have turned into statements

--e.g. "The walk was tiring". Here we look back for a statement whose predicate is "walk" or from which a statement involving "walk" can be inferred. There are cases in which the required inference is in fact a summary of an entire paragraph--e.g. "These actions surprised..."--although of course we cannot handle these cases.

Consistency. Each of the plausible antecedents is checked for consistency. Suppose X_1 is the definite entity which prompted the search and its properties are

$$p(X_1), q_1(X_1), \ldots, q_m(X_1)$$

and X_2 is the proposed antecedent with properties

$$p(X_2), r_1(X_2), \ldots, r_n(X_2)$$

We must cycle through the q's and the r's to ensure they are consistent properties. Of course, to prove two properties q(X) and r(X) inconsistent can be an indefinitely long process with no assurance of termination. One admittedly ad hoc way we get around this is by placing into a special cluster those facts we feel are likely to lead quickly to a contradiction. The second tool we use for deriving inconsistencies may turn out to be quite significant.

In the course of processing, the lattice described above is constructed for several predicates. They contain information which can be useful in deriving an inconsistency. Suppose we have a text in which "the block" occurs explicitly several times. Toward the end of it, we encounter

> "Turn right onto Adams Street. The library is at the end of the block".

The search algorithm looks first for explicit mentions of "block" and finds them. Yet none of these entities is the one we want. Intuitively, the reason we know this is our almost visual feeling that we are already beyond those points.

The lattice consistency check corresponds precisely to this feeling. If a definite entity X_1 is a point or interval in a lattice or at a point or along an interval, we ask if the proposed antecedent X_2 is or can be related to a portion of the lattice. If so, then since the lattice represents a transitive relation, we need only ask if there is a path in the lattice from X_2 to X_1. If there is, they cannot be the same entity.

Many cases which pass for applications of the supposed recency principle--"Pick the most recent plausible antecedent"--are in reality examples of this consistency check. The earlier plausible antecedent is rejected because of lattice considerations.

As the text is processed, the whole structure of the discourse is built up. When a definite noun phrase is encountered, this discourse structure is known and it is this knowledge that is used to determine the antecedent rather than the linear ordering of the words on the page.

Competition among Remaining Plausible Antecedents. Even after the consistency checks, several plausible antecedents may remain, forcing us to decide among them on less certain criteria. To do this, we appeal to the Principle of Knitting again and make the choice that will maximize the redundancy in the simplest possible way.

A probe is sent out from the definite entity and from each plausible antecedent. Each plausible antecedent is searched for properties it has in common with the definite entity. Common properties count most if they are already in the Text, and within the Lexicon, common properties count more if they are within more salient clusters or they result from shorter chains of inference.

Default. Like the higher predicate algorithm, the definite noun phrase algorithm has a default feature. If the uniqueness conditions fail and the search turns up no antecedent, we simply introduce a new entity. In fact, in the directions texts there are a disproportionately large number of default cases, for "the object" may simply be the object you will see when you reach that point in following the directions.

Other Anaphora. We have not yet implemented routines for handling other anaphora. However, we believe they are very similar to the definite noun phrase routine, with certain differences. For entities tagged with demonstrative articles, we do not check uniqueness conditions, and the search will be narrower since the antecedent must be an entity or statement actually occurring in the text. For pronouns also, no uniqueness conditions are checked. The search will turn up more consistent plausible antecedents, and a correspondingly greater burden will be placed on the competition routine.

INTERSENTENTIAL CONNECTIVES

We detect unstated inter-sentence connectives by matching two successive sentences S_1 S_2 with a small number of common

patterns. In the directions texts the patterns are usually few and simple. The most common are

1. S_1 asserts a change whose final state is asserted or presupposed by S_2.

2. S_1 asserts or presupposes a state which is the initial state of a change asserted by S_2.

(These are likely very common patterns in all narratives.) For example, in the text

"Walk out the door of this building. Turn right.
Walk to the end of the block".

pattern(1) joins the first two sentences, where the state is "You at X". Pattern(2) joins the last two sentences, where again the state is "You at X". Note moreover that the sentences are interlocked by a second application of the two patterns: The first sentence assumes an angular orientation which is the initial state of the change asserted in the second sentence. The final state of this change is assumed by the third sentence.

In addition to providing the discourse with structure, this operation is one of the principal means by which implied entities in one sentence, like X above, are identified with those in another.

When pattern (2) is applied, we delete the independent occurrence of the state in the Text, so that subsequently it exists only as one intermediate state in a larger event. Changes across time are handled in this way.

TASK PERFORMANCE COMPONENT

Arbitrary Decisions. The semantic operations are quite

general and can be used for any application. The augmented and interrelated Text is then handed over to the task performance component, which of course is specific to the application.

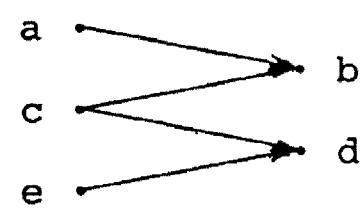
Our task component first makes arbitrary decisions required by the map but not given in the text. Both natural language directions and sketched maps allow information to be incomplete and imprecise, but in different ways. For example, in

"Turn right at the third street or the second stoplight".

we must decide whether to put the first stoplight at the first or second street.

The lattice representing the path "you" take must be complete in the sense that it is continuous, begins at the initial location, and ends at the desired goal, and that the relative locations of all points on the path are known. The lattice is complete if and only if there is a directed path passing through every point in the lattice at least once. If it is not complete, it is completed by supplying the fewest possible new links.

Geometrizing the Lattices. The second task operation is to convert the topological lattice representation into the geometric representation required by the maps. First we assign directions to all the points in the angular orientation lattice. In the simplest case we may have something like



where "a ⟶ b" means direction b results from a clockwise rotation of direction a. If no explicit directional information

is present, we simply assume a, c, and e are the same direction, and b and d are the same, and then assume the two directions are at right angles. Then in the distance lattice, contiguous or overlapping paths which share the same orientation are assumed to be parts of the same path and are mapped into a straight line. Information about names is accessed and assigned to the streets and buildings and the map is drawn.

Specific Systems with a General Semantic Component. We are aiming not so much at the construction of a general natural language processing system, which still seems reasonably far off but at an easier way of constructing specific systems. The case of syntax is instructive. It would be foolish for one who is building a natural language processing system to build his syntactic component from scratch. Large general grammars and parsers for them exist (e.g. Grishman et al. 1973, Sager & Grishman 1975). It is easier by several orders of magnitude to begin with a general grammar and specialize it, by weeding out the rules for constructions that don't occur in the texts one is dealing with, and by adding a few rules for constructions and constraints peculiar to one's application.

We are trying to make a similar facility available for the most common kinds of semantic processing. Specializing the general semantic component would consist of several relatively easy steps. First the Lexicon would be organized into a cluster structure appropriate to the task. At worst, this would mean specifying the necessary knowledge in a fairly simple format. If a very large Lexicon were available, this could mean no more

than designating for each fact the cluster it should appear in. Certain inferences could be made obligatory while others which are irrelevant to the task could be left out of the special Lexicon altogether. Second a Task Component would be built which would take, as ours does, the semantically processed Text, and use it to perform the task. We are demonstrating the usefulness of this approach in performing a task involving a visual representation. It is likely to be useful in other sorts of tasks also.

BIBLIOGRAPHY

Grishman, R., Sager, N., Raze, C., & Bookchin, B., "The Linguistic String Parser," Proc. NCC, AFIPS Press, Montvale, N.J. 1973.

Hobbs, J., "A Model for Natural Language Semantics, Part I: The Model," Yale Univ. Dept. Comp. Sci. Res. Rep. 36, Nov. 1974.

Hobbs, J., and Grishman, R., "The Automatic Transformational Analysis of English Sentences: An Implementation," Submitted to International Journal of Computer Mathematics.

Holzman, M., "Ellipsis in Discourse: Implications for Linguistic Analysis by Computer, The Child's Acquisition of Language, and Semantic Theory," Language and Speech (1971, 86-98.

Joos, M., "Semantic Axiom Number One," Language (1972) 257-265.

Knuth, D. The Art of Computer Programming, 3, Addison-Wesley, Reading, Mass., 1973.

Minsky, M., "A Framework for Representing Knowledge," MIT AI Memo 306, June 1974.

Sager, N., and Grishman, R., "The Restriction Language for Computer Grammars of Natural Language," CACM 18, 7 (7/75) 390-400.

American Journal of Computational Linguistics *Microfiche 32:* 42

An Adaptive Natural Language Parser

Perry L. Miller
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

ABSTRACT

When a user interacts with a natural language system, he may well use words and expressions which were not anticipated by the system designers. This paper describes a system which can play TIC-TAC-TOE, and discuss the game while it is in progress. If the system encounters new words, new expressions, or inadvertent ungrammaticalities, it attempts to understand what was meant, through contextual inference, and by asking intelligent clarifying questions of the user. The system then records the meaning of any new words or expressions, thus augmenting its linguistic knowledge in the course of user interaction.

1. INTRODUCTION

A number of systems are being developed which communicate with users in a natural language such as English. The ultimate purpose of such systems is to provide easy computer access to a technically unsophisticated person. When such a person interacts with a natural language system, however, he is quite likely to use words and expressions which were not anticipated. To provide truly natural interaction, the system should be able to respond intelligently when this happens.

Most current systems, such as those of Winograd [10] and Woods [11], are not designed to cope with such "linguistic input uncertainty." Their parsers fail completely if an input sentence does not use a specific, built-in syntax and vocabulary. At the other extreme, systems like ELIZA [9] and PARRY [2] allow the user to type anything, but make no attempt to fully understand the sentence. The present work explores the middle ground between these extremes: developing a system which has a great deal of knowledge about a particular subject area, and which can use this knowledge to make language interaction a flexible, adaptive, learning medium.

In pursuing this goal, the present work is most closely related to work being done in the various speech recognition efforts [5, 7, 8, 12] which are studying how linguistic and semantic constraints can help deal with the ACOUSTIC error and uncertainty of speech. The adaptive system, however, is designed to deal with a much more LINGUISTIC type of uncertainty.

When people use unfamiliar words or expressions in conversation, we can usually deduce from context what is meant, and if not, we can at least ask intelligent clarifying questions. To allow the machine to do the same, there must be a very flexible interaction of syntax and semantics in the parsing/understanding process. There must be a

different parser organization, and a more flexible use of linguistic and semantic constraints, than is present in current natural language systems.

The adaptive system is a step towards this goal The current implementation is a prototype, designed to illustrate many of these ideas, and to tie them together in a restricted system that is complete but not too complex. The system's domain of discourse is TIC-TAC-TOE: it plays a game and discusses the game in progress. It has a (currently limited) set of semantic concepts, and attempts to interpret the user's sentences in terms of these. The set of semantic concepts is FIXED. The system does not attempt to learn new concepts, merely new ways of describing concepts that are already familiar.

When the system encounters a new word or a new phrasing, it uses contextual information to determine the possible meaning, and to ask clarifying questions which are as intelligent as possible. It then remembers the meaning of the new expression, and will use it itself when it talks.

2. SAMPLE SESSION

The best way to introduce the system is to show it in operation. In the sample session that follows, user input is preceded by "U:", machine output by "M:", and comments added by the author are in small type, enclosed in brackets.

(MACHINE: I PLACE AN X IN THE TOP LEFT SQUARE)

```
(X - -)
(- - -)
(- - -)
```

[The machine makes its opening move.]

(USER: PLAY A ZERO IN THE CENTER SQUARE)

(M: IS ZERO A NOUN MEANING 0)

(U: YES)
(M: OK)

```
(X - -)
(- O -)
(- - -)
```

[The new word "zero" is added to the machine's vocabulary.]

(M: I PLACE AN X IN THE BOTTOM RIGHT SQUARE)

(X - -)
(- O -)
(- - X)

(U: SLAP AN O IN THE BOTTOM LEFT CORNER)

(M: IS (TO SLAP SOMETHING SOMEWHERE) A WAY OF SAYING (TO PLACE SOMETHING SOMEWHERE))

(U: YES)
(M: OK)

(X - -)
(- O -)
(O - X)

[A new surface frame (as described in section 4) is added to the set of surface frames which express the concept of making a move.]

(M: I SLAP AN X IN THE TOP RIGHT SQUARE)

(X - X)
(- O -)
(O - X)

[In this sample session, the machine uses its newly acquired vocabulary at every opportunity.]

(U: HOW COME YOU PLACED AN X IN THE TOP RIGHT SQUARE)

(M: IS (HOW COME SOMEONE DOES SOMETHING) AN EXPRESSION MEANING (WHY DOES SOMEONE DO SOMETHING))

(U: YES)
(M: OK)

(M: TO BEAT YOU)

[A new surface frame is added to those expressing the concept 'WHY'. Syntactic features are associated with each such frame indicating such things as whether subject-verb inversion is to be performed.]

(U: CAN YOU CURRENTLY BEAT ME)

(M: IS CURRENTLY AN ADVERB REFERRING TO THE TIME OF A MOVE)

[These are the only adverbs the system knows about. In fact, at present, the system ignores adverbs in its semantic processing. It asks the question because it wants to account for all the words in the sentence.]

(U: YES)
(M: OK)

(M: YES)

(U: AFTER YOU PLAYED AN X IN THE TOP WEST SQUARE COULD I BEAT YOU IF I PLAYED AN O IN THE BOTTOM RIGHT SQUARE)
(M: IS WEST AN ADJECTIVE MEANING)
EITHER
(1: CORNER)
(2: LEFT)

(U: 2)
(M: OK)

(M: NO)

[Notice that two top squares contain an x. However, only the left one was played while the bottom right square was free. Therefore the system concluded that the top left square was being described. This example illustrates how contextual information can be used to ask as intelligent a clarifying question as possible.]

(U: HOW COME YOU PLACED AN X IN THE TOP EAST SQUARE)

(M: IS EAST AN ADJECTIVE MEANING)
EITHER
(1: RIGHT)
(2: WEST)

(U: 1)
(U: OK)

(M: TO BEAT YOU)

[Here, either top square might have been referred to.]

(U: WHEN I PLAYED AN O IN THE BOTTOM LEFT SQUARE COULD YOU BEAT ME)

(M: IS (WHEN SOMEONE DOES SOMETHING) AN EXPRESSION MEANING (AFTER SOMEONE DOES SOMETHING))

(U: YES)
(M: OK)

(M: YES)

[In the domain of this TIC-TAC-TOE discussion, "when" and "after" have the same meaning.]

[The next two sentences each contain two problems for the system to resolve.]

(U: WHEN YOU PLAYED AN X IN THE TOP RIGHT CELL COULD I KEEP YOU FROM BEATING ME)

(M: IS CELL A NOUN MEANING SQUARE)

(U: YES)
(M: OK)

(M: IS (TO KEEP SOMEONE FROM DOING SOMETHING) A WAY OF SAYING)
EITHER
(1: (TO PREVENT SOMEONE FROM DOING SOMETHING))
(2: (TO ALLOW SOMEONE TO DO SOMETHING))
(3: (TO FORCE SOMEONE TO DO SOMETHING))

(U: 1)
(M: OK)

(M: NO)

(U: IF I LAY AN O ON THE RIGHT SIDE SQUARE HOW CAN YOU POLISH ME OFF)

(M: IS (TO LAY SOMETHING SOMEWHERE) A WAY OF SAYING (TO SLAP SOMETHING SOMEWHERE))

(U: YES)
(M: OK)

(M: IS (TO POLISH SOMEONE OFF) A WAY OF SAYING)
EITHER
(1: (TO BEAT SOMEONE))
(2: (TO LOSE TO SOMEONE))
(3: (TO TIE SOMEONE))

(U: 1)
(M: OK)

(M: BY LAYING AN X ON THE TOP CENTER CELL)

[If this same session were subsequently repeated, of course, the system would not need to ask any questions of the user.]

2.1 Comments on Current Limitations

There are a number of limitations to the adaptive system as it now stands. Some of these may be apparent in the sample session, but an introduction to the system is not complete without discussing them explicitly.

(1) The number of concepts available to the system at present is very small. This, in fact, is why the system's first guess is usually the correct one. If the sentence is at all within the system's comprehension, the options as to its meaning are currently quite limited.

(2) The range of expressive devices presently recognized is quite limited as well. For instance, the system does not recognize relative clauses, conjunctions, or pronouns (except for I and you).

(3) The system currently deals only with TOTALLY UNFAMILIAR words and expressions in this adaptive fashion. It will not correctly handle familiar words which are used in new ways (such as a noun used as a verb, as in "zero the center square").

(4) The system tries to map the meaning of new words and expressions into its specified set of underlying concepts. It then displays its hypotheses to the user, giving him only the option of saying yes or no. The user cannot say "no, not quite, it means ...". (Thus concepts like "the 'northeast' square" or "the 'topmost' square" would be confusing and not correctly understood.)

The present simple system has been developed with two goals in mind: (1) to explore the techniques required to achieve adaptive behavior, and (2) to help formulate the issues which will have to be faced when incorporating these techniques into a much broader natural language system.

3. OVERVIEW

Fig. 1 shows the various stages that the Adaptive System goes through in understanding a sentence. In this section, we shall watch while the system processes the sentence "How come you placed an x in the top right square."

(1) Local Syntactic Processing:
In this first stage, the system scans the entire sentence looking for local constituents. These include "simple" noun phrases (NPs) and prepositional phrases (PPs), ("simple" meaning "up to the head noun but not including any modifying clauses or phrases"), and verb groups (VGs) consisting of verbs together with any adjoining modals, auxilliaries, and adverbs. In this instance, the system finds the two NPs, "you" and "an x", the PP "in the top right square", and the VG "placed".

(2) Semantic Clustering:
At this stage, the clause-level processing starts. Unlike most systems, this clause-level processing is driven by SEMANTIC relationships, rather than by syntactic form. It uses a semantics-first "clustering", with a secondary use of syntax for comments and confirmation. In this example, all the local constituents found can be clustered into a description of a single concept: that of making a move. Section 4 describes the mechanics of this stage in more detail.

(3) Cluster Expansion and Connection:
During this stage an attempt is made to account for each word in the sentence by expanding the concept clusters, and if there is more than one, by joining them together to form an entire multiclausal sentence. In this case, the concept cluster might be expanded in two ways.
a) One possibility might be that it is a "HOW" type question, and that "come" is some sort of adverb. However this possibility violates a semantic constraint, since the system is not set up to answer how a move is made; only how to win, how to prevent someone from winning, etc. Therefore this possibility is ignored.
b) The other possibility is that "how come" is a new way of describing some other clause function.

(4) Contextual Inference; Clarification; and Response:
During this final stage, any contextual information available is brought to bear on areas of uncertainty, any necessary clarifying questions are asked, and the system responds to the sentence. In this example, the only uncertainty is the meaning of "how come". Since this is the main

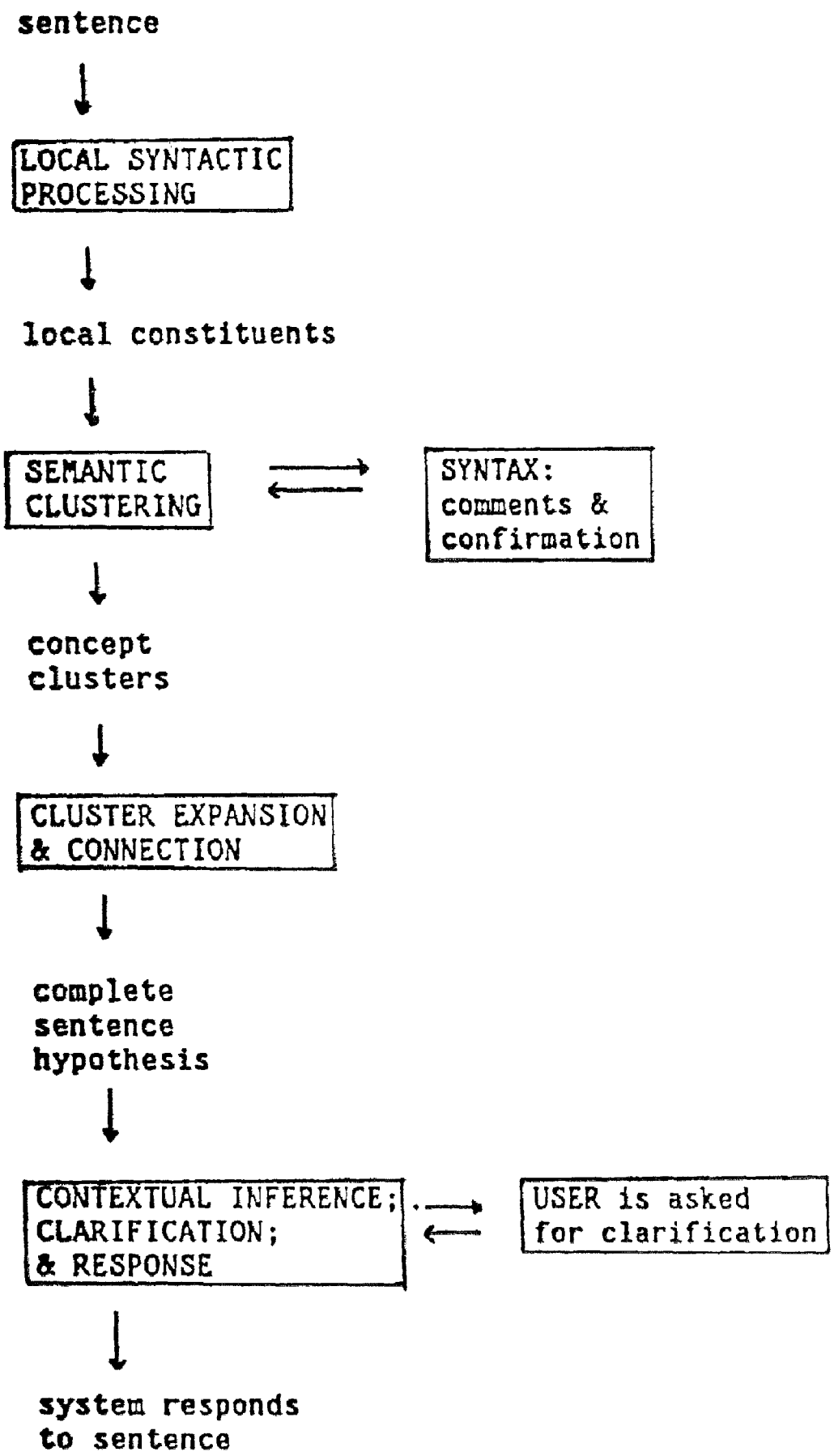


Fig. 1: Adaptive System Overview

clause of the sentence, the possibility of its being an "if" or "after" clause are discarded. The remaining possibilities are "imperative", "how", "why", and "can". The system does not answer "how" and "can" questions in relation to making moves. Similarly, "imperative" does not make sense since the action described is a previously made move. Therefore the system asks if "How come someone does something" means "Why does someone do something". The user answers "yes", so the system stores this new way of asking "why", and proceeds to answer the question.

4. SEMANTICS-FIRST CLAUSE-LEVEL PROCESSING

One of the major differences between this approach to parsing and that of a top-down, syntax-driven system (such as Woods' or Winograd's) is the order in which syntactic and semantic processing is done at the clause level.

In a top-down system, a sentence must exactly match the built-in syntax before semantics can even be called and given the various constituents of a clause. This is clearly undesirable when one is dealing with input uncertainty, since one cannot be sure exactly how the user will phrase his sentence. One would prefer to let semantics operate first on any local consituents present, so that it can make a reasonable guess as to what is being discussed.

As semantically-related clusters of local constituents are found, syntax can be consulted and asked to comment on the relative grammaticality of the various clusters. If there are two competing semantic interpretations of one part of a sentence, and syntax likes one much better than the other, then the "syntactically pleasing" interpretation can be pursued first. Later, if this does not pan out, the syntactically irregular possibility can be looked at as well. In this way, syntax can help guide the system, but is not placed in a totally controlling position.

A by-product advantage of this semantics-first approach is that the system can handle mildly ungrammatical input without any extra work. In addition, the semantics-first clustering approach lends itself quite naturally to handling sentence fragments.

In the remainder of this section, we describe how the adaptive system organizes its linguistic knowledge to implement this semantics-first approach. As we shall see, there are three components of this knowledge.

(a) The local recognizers which initially find local constituents. These recognizers are represented in Augmented Transition Network [11] form, are quite simple, and are not described further in this paper.
(b) Clause-level knowledge of how actions and clause-functions are described. This knowledge is expressed in a descriptive fashion which makes it easily manipulable, and easy to add to.
(c) Clause-level syntactic knowledge which is expressed in a domain-independent form.

4.1 Knowledge of how Actions are Described

Figure 2 illustrates how the system stores its knowledge of how actions (or events) are described. This knowledge is stored at two levels: the conceptual level, and the surface (or expressive) level

As shown in Fig. 2, the concept PLACE represents the act of making a TIC-TAC-TOE move.

(a) On the CONCEPTUAL level, there are three "conceptual slots" indicating the actors which are involved in the action: a player, a mark, and a square.

(b) On the SURFACE, or expressive, level there is a list of surface frames each indicating one possible way that the concept can be expressed. Each surface frame consists of a verb plus a set of syntactic case frames to be filled by the actors.

(Notice that neither the conceptual slots nor the surface frames indicate explicitly the order in which the various constituents are to appear in a sentence.)

When the system processes a sentence, it fills the conceptual slots with local constituents found in the sentence If it has found a familiar verb, then it also gets any surface frame(s) associated with that verb. At this point it calls syntax, asking for comments.

For instance, if the input sentence is "I place an x in the corner", then all the conceptual slots of #PLACE would be filled, and the system would pass the following string to syntax "agent verb obj pp". As a result, clause-level syntax does not see the actual constituents of the sentence, only the labels specified in the surface case frame, plus information indicating number, tense, etc.

An interesting aspect of this approach is that the clause-level syntax is entirely domain-independent. It knows nothing about TIC-TAC-TOE, or even about the words used to talk about TIC-TAC-TOE. The surface frames allow semantics to talk to syntax purely in terms of syntactic labels. As a result, one could write a single syntactic module, and then insert it unchanged into many domains.

4.1.1 Using this Information

In this section, we describe in more detail how this knowledge can be used when processing a sentence.

(1) If the verb and constituents are familiar:

If there is no uncertainty in a clause, then each constituent can be put into one of the conceptual slots, and any surface frames associated with the verb can be examined The frame indicates the case (agent, object, etc.) associated with each constituent when that verb is used. The frame is used to create a string of case labels that are sent to syntax for comments.

For instance, if the sentence is "I place an x in the center

CONCEPT: PLACE

CONCEPTUAL SLOTS:

P: player
M: mark
S: square

SURFACE FRAMES:

VERB: place (as in:
AGENT: P "I place an x in the center")
OBJ: M
in: S

VERB: play (as in:
AGENT: P "I play an x in the center")
OBJ: M
in: S

VERB: play (as in:
AGENT: P "I play the center")
OBJ: S

Fig. 2: Linguistic Knowledge about Actions

square", the string passed to syntax is "agent verb obj pp". Syntax replies that the sentence follows normal order. Had the string been "verb obj pp", syntax would reply that the subject had been deleted. If the string was "do agent verb obj pp", syntax would reply that subject-verb inversion had taken place. Given "agent obj verb pp", syntax would reply that the object was out of position.

Thus syntax is set up to notice both grammatical and ungrammatical permutations in constituent order, and to comment appropriately. The system must then decide how to interpret these comments.

For instance, if syntax replies that the object is out of position in the clause, or that there is incorrect agreement in number between subject and verb, the system may decide that the user has made a minor grammatical error, and allow the sentence to be processed anyway, especially if there is no better interpretation of the sentence. In this way, clause-level syntax plays an assisting role rather than a controlling role in the analysis of a sentence.

(2) If a constituent is unknown:

If an unknown constituent is present, then both the frame and slot information can be used to help resolve its meaning. For instance, suppose the sentence is "I place a cross in the center square", and the word "cross" is unfamiliar.

Here, during the semantic clustering, the conceptual slots for a player and a square can be filled by "I" and "in the center square", but the slot for a mark is unfilled. In addition, there is the unknown constituent "a cross".

A natural hypothesis, therefore, is that the unknown constituent refers to a type of mark. Since the verb is familiar, a surface frame is available. Next, assuming the unknown constituent is a mark, the string "agent verb obj pp" can be passed to syntax. When syntax approves, this offers additional confirmation that the hypothesis is probably right.

Subsequent evaluation of this hypothesis indicates that the sentence makes sense only if the mark referred to is an x, so the system asks if "cross" is a noun meaning "x".

(3) If the verb is unknown:

If an unfamiliar verb is used, then there is no surface frame available to help guide the analysis. Instead, syntax must be used in a different mode to propose what the surface frame should be.

Suppose the sentence is "I plunk an x in the center square". Here, all the constituents can be clustered into the concept #PLACE, but there is an unknown word, and no verb. The logical hypothesis is that the new word is a verb. A special syntactic module is therefore passed the following string "NP(P) verb(plunk) NP(M) PP(in,S)". This module examines the string and produces a new frame:

VERB: plunk
AGENT: P
OBJ: M
in: S

The system can then ask if "to plunk something somewhere" means "to place something somewhere", and upon getting an affirmative reply, can add the new frame to those associated with the concept PLACE.

Since the system uses the surface frames to generate its own replies, it can now use this new frame itself when it talks. When the system wants to generate a clause, it passes a selected frame, the constituents, and a list of syntactic features to a clause generator which outputs the specified form. (Thus, clause-level syntax can be used by the system in three different modes: (1) to comment on the grammaticality of a string of case markers, (2) to construct a new surface frame, and (3) to generate clauses when the system itself replies.)

4.2 Knowledge of how Clause-Functions are Described

As illustrated in Fig. 3, knowledge of how clause-function concepts are described is also expressed as two levels.

```
CONCEPT: #WHY

CONCEPTUAL SLOTS:

        ACTION: #PLACE

SURFACE FRAMES:

        Why ACTION(SVINV) (as in:
                                "Why does someone do something")

        How come ACTION() (as in:
                                "How come someone does something")
```

Fig. 3: Linguistic Knowledge about Clause Functions

Each clause function has a conceptual slot indicating what types of action can be used with that clause type (in this case, the action #PLACE), and a list of surface frames indicating different ways in which the concept can be expressed.

A clause-type frame currently includes any special words which introduce the clause (ie. "why" or "how come"), together with a list of syntactic properties which should be present in the clause. This list of syntactic properties might include SVINV, "subject-verb inversion" (as in "why does someone do something"), or "subject deletion", "ING form", and "use of a particular preposition" (as in "from doing something").

These syntactic features, however, need not be inflexible rules. Sentence understanding can still proceed even if the syntactic features

found by syntax do not exactly match those specified by the clause-function frame. Thus, an inadvertent ungrammaticality can readily be recognized as such, and processing can continue.

4.2.1 Using the Clause Function Knowledge

In this section we examine how this clause function knowledge can be used.

(1) With no uncertainty:

If the input sentence is "Why did you place an x in the center square", then during the semantic clustering the string "do agent verb obj pp" is passed to syntax, which replies that subject-verb inversion has taken place.

When examining the whole clause, the system sees that it exactly matches one of the surface frames for a #WHY-type question, since it starts with the word "why" and contains subject-verb inversion.

Suppose, however, the sentence had been "Why you place an x in the center square", or "How come did you place an x in the center square". Each of these sentences matches a surface frame for a #WHY-type question, except that in both cases subject-verb inversion is incorrect. In such a case, the system can, if it chooses, decide that the user has made a minor error, and allow the sentence to be processed anyway. The locally-driven semantics-first approach lets this happen in a natural way.

(2) A new surface frame:

Another problem arises when a new clause introducer is encountered, as in: "Wherefore did you place an x in the center square". Here, as described in section 3, the system hypothesizes that this may be a new way of asking a #WHY-type question. Since syntax reports that subject-verb inversion has taken place, the system can therefore create a new surface frame:

Wherefore ACTION(SVINV)

to be added to the frames associated with #WHY.

4.3 Comments

In summary, the adaptive system stores its linguistic knowledge in a very accessible form. It is not embedded in the parsing logic. Knowledge of how actions and clause-functions are described is represented in a descriptive, manipulable format. Syntax is domain independent, and is used only to make comments, with semantics playing the guiding role. This organization allows the parsing/understanding process to proceed in a flexible fashion.

5. CONCLUSION

Language communication is an inherently adaptive medium. One sees this clearly if one takes a problem to a lawyer and spends time trying to assimilate the related "legalese". One also sees it in any conversation where a person is trying to convey a complicated idea, expressed in his own mental terms, to someone else. The listener must try to relate the words he hears to his own set of concepts. Language has, presumably, evolved to facilitate this sort of interaction. Therefore it is reasonable to expect that a good deal of the structure of language is in some sense set up to assist in this adaptive process. By the same token, studying language from an adaptive standpoint should provide a fresh perspective on how the various levels of linguistic structure interact.

REFERENCES

[1] Davies, D.J.M., and Isard, S.D., 'Utterances as Programs,' presented at the 7th International Machine Intelligence Workshop, Edinburg, June 1972.
[2] Enea, H., and Colby, K.M., 'Ideolectic Language Analysis for Understanding Doctor-Patient Dialogs', Proceedings of the 3rd IJCAI, Stanford, August 1973.
[3] Fillmore, C.J., 'The Case for Case', in 'Universals in Linguistic Theory', Bach and Harms (Eds.), Holt, Rinehart, and Winston, Inc., Chicago 1968.
[4] Joshi, A.K., and Weischedel, R.M., 'Some Frills for the Modal TIC-TAC-TOE of Isard and Davies: Semantics of Predicate Complement Constructions,' Proceedings of the 3rd IJCAI, Stanford, August 1973.
[5] Miller, P.L., 'A Locally Organized Parser for Spoken Input', Comm. ACM 17, 11 (Nov. 1974), 621-630.
[6] Miller, P.L., 'An Adaptive System: for Natural Language Understanding and Assimilation', RLE Natural Language Memo No. 25, MIT, February 1974.
[7] Reddy, D.R., Erman, L.D., Fennell, R.D., and Neeley, R.B., 'The HEARSAY Speech Understanding System', Proceedings of the 3rd IJCAI, Stanford, August 1973.
[8] Walker, D.E., 'Speech Understanding through Syntactic and Semantic Analysis', Proceedings of the 3rd IJCAI, Stanford, August 1973.
[9] Weizenbaum, J., 'Eliza- a Computer Program for the Study of Natural Communication between Man and Machine', CACM 9, 1972.
[10] Winograd, T. Procedures as a Representation of Knowledge in a Computer Program for Understanding Natural Language, MAC-TR-84, Project MAC, MIT, Cambridge, Mass., February 1971.
[11] Woods, W.A., and Kaplan, R.M., 'The Lunar Sciences Natural Language Information System', BBN Report No. 2265, Bolt, Beranek, and Newman Inc. September 1971,
[12] Woods, W.A., and Makhoul, J., 'Mechanical Inference Problems in Continuous Speech Understanding , Proceedings of the 3rd IJCAI, Stanford, August 1973.

CONCEPTUAL GRAMMAR

WILLIAM A. MARTIN
Massachusetts Institute of Technology

In OWL, an implementation of conceptual grammar, the two types of data items are symbols and concepts and the two basic data composition operations are specialization and restriction.

A symbol is an alphanumeric string headed by ". Symbols correspond to words, suffixes, prefixes, and word stems in English and the programmer can introduce them at will.

OWL concepts correspond to the meanings of English words and phrases. They are constructed using the specialization operation, comparable to CONS in LISP. (A B) is the specialization of A, a concept, by B, a concept or symbol. OWL forms a branching tree under specialization, with SOMETHING at the top.

Concepts are given properties by restriction, which puts a concept on the reference list of another concept (compare property lists and S-expressions in LISP). A/B is the restriction of A by B.

The categories in the specialization tree are semantic, but we use them also for the purposes usually assigned to syntactic categories.

A predication is a double specification of a model such as present tense or can. Examples are

The pool is full of water. ((PRES-TNS (BE (FULL WATER))) POOL/THE)
The cookie can be in the jar. ((CAN (BE (IN JAR/THE))) COOKIE/THE)
Bob is the father of Sam. ((PRES-TNS (BE (FATHER SAM)/THE)) BOB)
Bob hits the ball. ((PRES-TNS (HIT BALL/THE)) BOB)
Bob is hitting the ball. ((PRES-TNS (BE (-ING (HIT BALL/THE))))BOB)

Starting from this base we will discuss a number of issues such as nominalization incorporation, and deep vs surface cases.

American Journal of Computational Linguistics *Microfiche 32 : 58*

SEMANTIC-BASED PARSING AND A NATURAL-LANGUAGE INTERFACE FOR INTERACTIVE DATA MANAGEMENT

JOHN F. BURGER, ANTONIO LEAL, AND ARIE SHOSHANI

System Development Corporation
Santa Monica, California 90406

ABSTRACT

We describe a natural-language recognition system having both applied and theoretical relevance. At the applications level, the program will give a natural communications interface facility to users of existing interactive data management systems. At the theoretical level, our work shows that the useful information in a natural-language expression (its "meaning") can be obtained by an algorithm that uses no formal description of syntax. The construction of the parsing tree is controlled primarily by semantics in the form of an abstraction of the "micro-world" of the DMS's functional capabilities and the organization and semantic relations of the data base content material. A prototype is currently implemented in LISP 1.5 on the IBM 370/145 computer at System Development Corporation.

INTRODUCTION

In a recent article in Scientific American, Dr. Alphonse Chapanis says, "If truly interactive computer systems are ever to be created, they will somehow have to cope with the...errors and violations of format that are the rule rather than the exception in normal human communication"[1]. An example

dialogue produced by two persons interacting with each other by teletype-writer to solve a problem assigned to them by experimenters showed that "not one grammatically correct sentence appears in the entire protocol."

Many existing language processors (Woods, Kellogg, Thompson, etc.) [2,3,4] are limited to what Chapanis calls "immaculate prose," that is, "the sentences that are fed into the computer are parsed in one way or another so that the meaning of the ensemble can be inferred from conventional rules of syntax," which are a formal description of the language. In effect, users are required to interact with these systems in some formal language, or at least in a language that has a formal representation in the computer system that a user's expression must conform to (we are thinking, in the latter instance, of Thompson's REL, which has an extensible formal representation facility). In addition, most natural-language question-answering systems, including all referenced above, require that a user's data be restructured and reorganized according to the particular data base requirements of the natural-language system to be used.

At the level of artificial intelligence research [5,6,7], there is some interest in systems that recognize meaning in natural-language expressions by methods that do not require compiler-like syntactic analysis of an expression prior to semantic interpretation. We believe it is possible, practical, and feasible, using new linguistic processing strategies, to design a natural-language <u>interface</u> system that will permit flexible, intuitive communication with information management systems and other computer programs already in existence. This interface is open-ended in that it has

no prejudice about the user's system functions and can be joined to almost any such system with relatively little effort. It is, in addition, able to infer the meaning of free-form English expressions, as they pertain to the host system, without requiring any formal description or representation of English.

THE SEMANTIC INTERFACE ALTERNATIVE

The syntactic inflexibility of existing natural-language processors limits their usefulness in interactive man-machine tasks. Our approach does not use a collection of syntax rules or equations as they are normally defined. Instead, we construct a dictionary in which we define words in terms of their possible meanings with respect to the particular data base and data management system (DMS) we want to use and according to the possible relations that can exist between data-base and DMS elements (e.g., an averaging function on a group of numbers) in the limited "micro-world" of this precisely organized data collection. Words appearing in a user's expression that are not explicitly defined are ignored by the system in processing the expression; an example would be the word "the," which is usually not meaningful in a data management environment. We thus avoid the expressive rigidity that formal syntactic methods impose on the user and the excessive time and resource consumption that results from the combinatorial explosions usually produced by such methods.

We distinguish in their definitions between two types of words: content words and function words (or "operators"). Content words are words whose

"meanings" are the objects, events, and concepts that make up the subjects being referred to by users. More precisely, for data management systems, these meanings (or "concepts") are the field names and entry identifiers for the data base and the names for available DMS operations such as averaging, summing, sorting, comparing, etc. Function words serve as connectors of content words. Their use in natural language is to indicate the manner in which neighboring content words are intended to relate to one another. In the example "the salary of the secretary," used below, "salary" and "secretary," are content words, and "of" is a function word used to connect them.

Many content words are context sensitive. In a particular data base, for instance, the word "salary" may refer to the data-base field name SECSAL if the salary is "of a secretary," but may also indicate the field name CLKSAL if it is a "salary of a clerk." In recognition of this we therefore define each content word by a set of one or more pairs of the form

((X1 Y1) (X2 Y2) ... (Xn Yn))

where the Xi and Yi are "concepts" (that is, field names, etc.) as described above. This expression may be interpreted as, "if the word so defined is contextually related in a sentence to X1, its particular meaning in this context is Y1, if it is so related to X2, it means Y2, and so forth." This particular contextual meaning of the word is called its <u>sense</u>. Two content words are considered to be semantically related if the intersection of the Xi's from the definition of one word with the Yi's from the definition of the other is not empty.

To get a more intuitive understanding of this process, suppose, again, that a data base contains entries for both secretaries and clerks with salaries for each. Suppose "Suzie" is an instance of a secretary and "Tom" is an instance of a clerk. We then have three words defined as follows:

Suzie	((SUZIE SECY))
Tom	((TOM CLK))
Salary	((SECY SECSAL) (CLK CLKSAL))

Processing the phrase "Suzie's salary" would intersect the Yi ("(SECY)") from the definition of "Suzie" with the Xi's ("SECY" and "CLK") from the definition of "salary." The intersection is non-empty ("(SECY)"), and, in discovering the semantic relationship, the sense "SECSAL" is assigned to the word "salary." Similarly, "Tom's salary" assigns the sense "CLKSAL" to "salary."

A particular implementation of the natural-language interface processor operates for a particular DMS/data-base target system. It contains a particular dictionary created for that target system. For a particular dictionary, the set of all lists of pairs as described above, therefore, constitutes the equivalent of a concept graph or network for the particular data base analogous to those used by many of the more conventional parsers for semantic analysis following (or during) the syntactic phase of parsing.

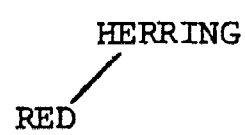
In the analysis of a particular input by our system, two words in context are tested using the "intersection" method described above and, if they are found to be semantically related, they are considered candidates for "connection" as described below. Two words so connected form a phrase.

Function words are defined as operators or processors that perform this semantic test. The definition of one function word differs from that of another according to its slope (see below) and also in that the operational definition of a function word can reject a connection even though the two words may be semantically related. In the operational definition of the function word may be a list of acceptable concepts or a rejection list of unacceptable concepts. In most conceivable data bases, the phrase "salary in the secretary" would be thus rejected by the function word "in."

As the analysis of an input expression proceeds, a "clumping" of word and phrase meanings is more and more explicitly connected until, normally, the processing of the entire sentence results in a tree structure made up of the connected senses of all the content words from the sentence. This result we term the sentence graph even though the input expression may not be a grammatically complete sentence. This sentence graph will be translated into a formal DMS statement.

We recognize that the linear ordering of the words in an input expression is not entirely random and that certain aspects of the function of syntax must be taken into account. This is done by means of a new and powerful algorithm based on what we call the syntactic-semantic slope. Linguists generally recognize that whenever two units of meaning are combined, one is semantically dominant and the other subordinate, as a modifier is subordinate to the modified word. After combination, the dominant word may be used in most cases to refer to the conjoined pair. Thus, a "red herring" is a "herring" (not a "red"), and the "salary of the secretary" is a

"salary." If this relationship of dominance is represented vertically on a rectangular graph (i.e., dominance on the Y-axis), and if the linear ordering of the words in the expression is represented on the X-axis in normal left-to-right order, then the connection of an adjacent pair of content words or phrases will describe a linear slope on the graph. The slope is positive or negative as the dominating sub-unit is, respectively, to the right or to the left of the subordinate sub-unit. For example, the phrase "red herring" makes a positive slope, thus:



and "the salary of the secretary" makes a negative slope:

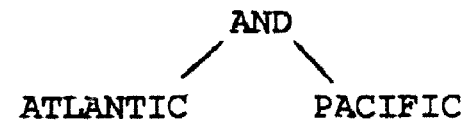
Thus, the operational meanings of function words operate on the meanings of nearby content words. Dominance is assigned, semantic relationships are verified, and the relationships so discovered are accepted or rejected. If accepted, the two word-meanings are connected, and the acceptable sense is assigned to the dominant word.

Function words may connect content words in "positive," "negative," or "peak" connections. The following are examples of each manner of connection:

1. "Of" is a negative operator, as in "the salary of the secretary":

2. "'s" is a positive operator, as in "the secretary's salary":

3. "And" is a peak operator, as in "Atlantic and Pacific." In contrast with positive and negative operators, peak operators add a representation of their own semantics into the structures they build:



4. Between any two adjacent content words there is an implicit "empty" operator that is a positive operator, as in "red herring":

In general, all prepositions are defined as negative operators. This is equivalent to the rule

NP → NP PREP NP

used by syntactic processors. The positive empty operator is equivalent to the rule

NP → ADJ NP

and others, while verbs and conjunctions are defined as peak operators, giving our statement of rules such as

S → NP VP NP

and

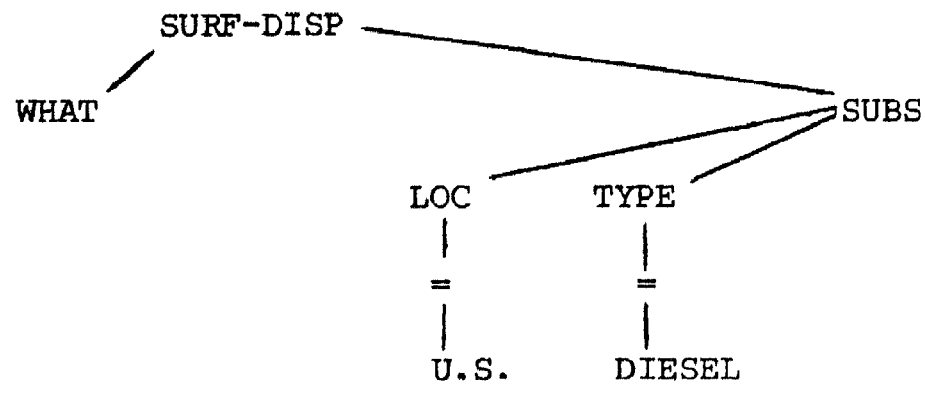
NP → NP CONJ NP.

Each operator has the facility to accept or reject any semantic relation according to the precise definition of the function word for the host data management system.

Progressive connection of word meanings and previously connected groups or "phrase meanings" results in a tree graph that we call the sentence graph. For example, the question "What is the surface displacement of U.S. diesel submarines?" could, for a particular data base, produce from the dictionary a string of content-word and function-word definitions that might be represented typographically like this:

((SUB SURF-DISC)) <OF> ((U.S. LOC)) ((DIESEL TYPE)) ((LOC SUBS) (TYPE SUBS))

As a result of processing, these will assemble into a tree structured (using the senses of the words) like this:



Even though this tree, or sentence graph, is created as a result of semantic relationships instead of formal rules of grammar, it still closely resembles the "parse tree" produced by most conventional syntactic language processors. With respect to the user's target data management system, the sentence graph is precise and unambiguous and contains enough information for a

straightforward translation into the formal query language of the DMS. In SDC's DS/3 language, for example, the above question would be expressed as

PRINT SURF-DISP WHERE TYPE EQ DIESEL AND LOC EQ U.S.

The response to the user's question will thus be the response from his DMS to the formal query statement.

The user's input in this hypothetical example is proper in form and grammar. However, it need not have been. The request

OBTAIN SURFACE DISP FOR US SUBS SUCH AS HAS TYPE EQ DIESEL.

would produce exactly the same sentence graph and therefore, exactly the same formal query statement with the same response from the DMS.

It is not likely that a syntax-based parser would have anticipated the odd language-use and grammar of this last request. Without a syntax rule that would allow for the phrase "such as has" such a parser would not look at the semantics involved and would be unable to interpret the request. Our syntax algorithm gets the same results that would be expected from the application of syntax rules without the need to anticipate each grammatical construct expected from the user.

In overview, the parsing algorithm makes a series of positive, negative, and peak connections based on the operational meanings of the function words (including the "empty" operator) and on the relations between meanings of the content words. The algorithm adheres to the following rules:

Rule 1: Connections between content words are possible only if the result of the intersection test described above is non-empty

and if this result is not rejected by the operation of the function word performing the test. The function word definition also determines which word supplies its X's and which its Y's for the test. It thus controls which word has its sense determined if the test is successful. Most often (though there are exceptions), positive operators use the X's from the word to the right and the Y's from the word to the left of the operator. Positive operators, therefore, determine the sense of the word to the right. This is illustrated using, again, the secretary and her salary. Consider the definition of "Suzie" and "salary" as shown on page 5. The phrase "Suzie's salary" has two content words, "Suzie" and "salary," separated by the function word, "'s." This function word is a positive operator and, hence, applies the intersection test to the Xi from the definition of "salary" with the Yi from the definition of "Suzie." These values are, respectively, "(SECY CLK)" and "(SECY)." The intersection yields "(SECY)," which is acceptable to the "'s" operator, and the connection is made with "salary" as the dominant word. The sense of "salary" is the Yi associated with "SECY" in the definition of "salary," hence, "SECSAL." This selection process is reversed for negative operators, while peak operators employ both kinds of tests, one on each side of the peak.

Rule 2: No node in a sentence graph may have more than one dominating node. That is to say, all connections must result in trees. This is a common assumption consistent with conventional syntax-driven parsers.

Rule 3: Given a subtree, a constituent on its left has the possibility of connection only to nodes of the subtree's positive adjacent slope, and a constituent on the right can connect only to the nodes in the adjacent negative slope. Intuitively, this means that if the nodes of a subtree are connected by "lines" that are "opaque barriers," then a constituent on either side of the subtree may connect to it only on those nodes that it can "see." It may not connect to nodes on the "inside" or the "far side" of the subtree. This is a powerful heuristic rule that eliminates the need to try connections to many syntactically impossible portions of the subtree. In effect this one rule, together with the definitions of the function words, replaces all the syntax rules used by most conventional parsers.

Rule 4: In order to minimize disconnection of existing subtree structures (backup) and still consider all possible connections, the system should, whenever possible, construct subtrees starting from the top and make new connections from below. This rule leads to the following algorithm: Scan the constituents from left to right making negative connections, then scan from right to left making positive connections. Scan thus back and forth until no more connections can be made. Then make any possible peak connections and repeat the algorithm. Continue this process until all constituents have been connected into a single tree.

We have observed that if ambiguities exist under these conditions, they will be semantic and, in all probability, not resolvable by any further processing

or analysis of the expression. Therefore, there is no need to carry along temporary multiple construction possibilities. The algorithm may either query the user at this point for disambiguation or abort the processing and inform the user of the reason.

REFERENCES

1. Chapanis, Alphonse. Interactive human communication. Scientific American, May, 1975.

2. Woods, W. A. Transition network grammars for natural language analysis. Communications of the ACM, October 13, 1970.

3. Kellogg, C. H., et al. The CONVERSE natural language data management system: current status and plans. ACM Symposium on Information Storage and Retrieval, University of Maryland, 1971.

4. Thompson, F. B.; Lockman, P. C.; Dostert, B.; Deverill, R. S. REL: a rapidly extensible language. Proceedings of 24th National Conference, ACM, New York, 1969, 399-417.

5. Riesbeck, C. K. Computational understanding. Theoretical Issues in Natural Language Processing: Proceedings of an Interdisciplinary Workshop in Computational Linguistics, Psychology, Linguistics and Artificial Intelligence. Cambridge, Massachusetts, June 10-13, 1975.

6. Waltz, D. L. On understanding poetry. Theoretical Issues in Natural Language Processing, Proceedings of an Interdisciplinary Workshop in Computational Linguistics, Psychology, Linguistics and Artificial Intelligence. Cambridge, Massachusetts, June 10-13, 1975.

7. Schank, Roger, and Tesler, L. G. A Conceptual Parser for Natural Language. Stanford Artificial Intelligence Project. Memo No. AI-76, January, 1969.

American Journal of Computational Linguistics *Microfiche 32 : 72*

P H L I Q A 1 : MULTILEVEL SEMANTICS IN QUESTION ANSWERING

P. MEDEMA, W. J. BRONNENBERG, H. C. BUNT, S. P. J. LANDSBERGEN,
R. J. H. SCHA, W. J. SCHOENMAKERS, AND E. P. C. VAN UTTEREN

Philips Research Laboratories
Eindhoven, The Netherlands

ABSTRACT

This paper outlines a recently implemented question answering system , called PHLIQA 1 , which answers English questions about a data base .

Unlike other existing systems , that directly translate a syntactic deep structure into a program to be executed , PHLIQA 1 leads a question through several intermediate stages of semantic analysis . In every stage the question is represented as an expression of a formal language. The paper describes some features of the languages that are successively used during the analysis process : the English–oriented Formal Language , the World Model Language and the Data Base Language . Next , we show the separate conversion steps that can be distinguished in the process. We indicate the problems that are handled by these conversions , and that are often neglected in other systems .

1. Introduction

PHLIQA 1 is an experimental system for answering isolated English questions about a data base . We have singled this out as the central problem of question answering , and therefore postponed the treatment of declaratives and imperatives , as well as the analysis of discourse until a later version of the system . The data base is about computer installations in Europe and their users . At the moment , it is small and resides in core – but its structure and content are those of a realistic Codasyl format data base on disk (CODASYL Data Base Task Group [1971]) .

Only one module of the system , the "evaluation component" , would have to be changed in order to handle a "real" data base .

2. PHLIQA 1 's top level design

Like other recent QA systems (e.g. Petrick [1973] , Plath [1973] , Winograd [1972] , Woods [1972]) , the PHLIQA 1 system can , on the most global level , be divided into 3 parts (see fig. 1) :

– Understanding the question : Translating the question into a formal expression which represents its meaning with respect to the world model of the system.

– Computing the answer : Elaborating this expression , thereby finding the answer, as it is represented in the system' s internal formalism.

– Formulating the answer : Translating this answer into a form that can be more readily understood .

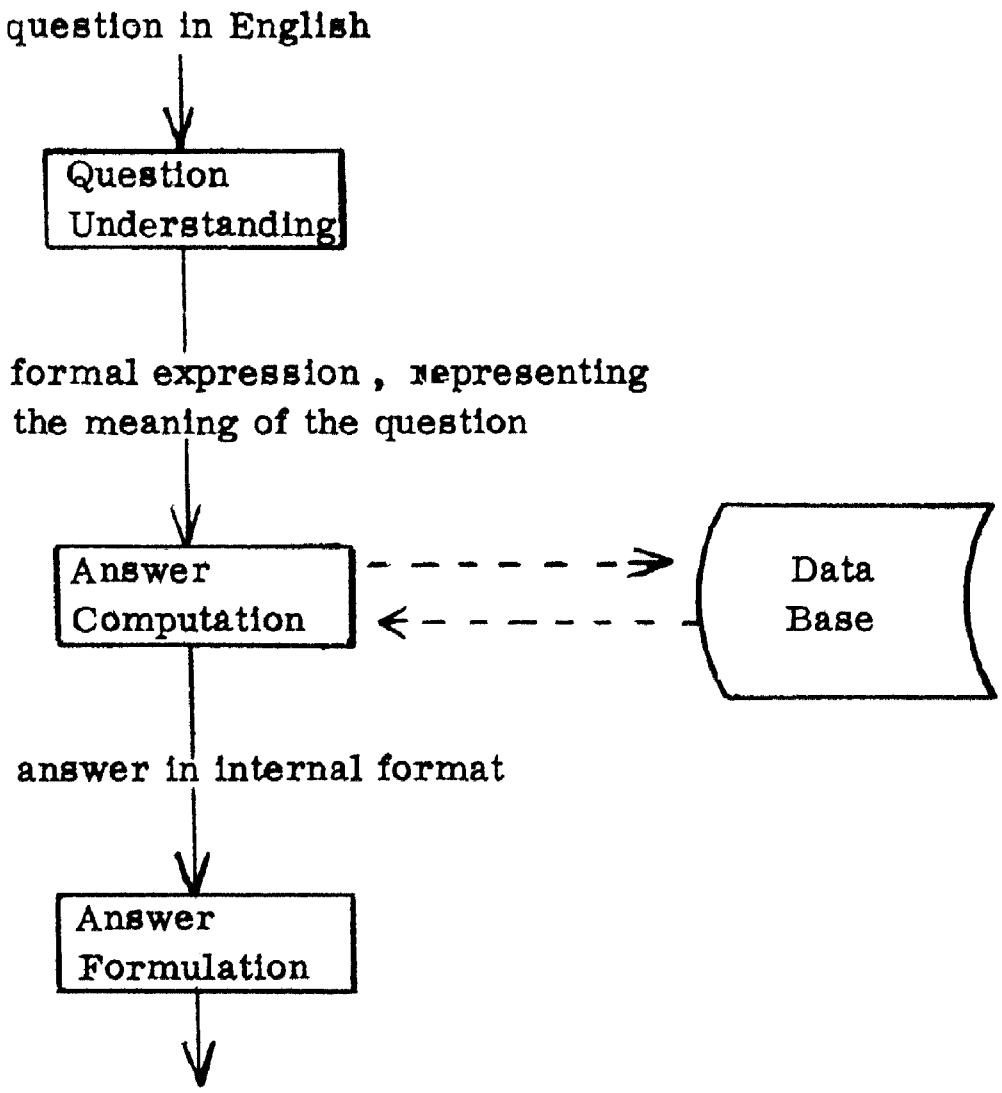


Fig . 1. Global subdivision of PHLIQA 1.

The interface between the Question Understanding component and the Answer Computation component is a formal language , called the World Model Language (WML) . Expressions of this language represent the meaning of questions with respect to the world model of the system. Its constants correspond to the concepts that constitute the universe of discourse . The language is independent of the input language that is used (in this case English) , and also independent of the storage structure of the data base .

If we now look at a further subdivision of the components , the difference between PHLIQA 1 and other systems becomes apparent . Both above and below the World Model level , there is an intermediate stage of analysis , characterized by a formal language , resp :

- The English–oriented Formal Language (EFL) , which contains constants that correspond to the terms of English . This language is used to represent the " semantic deep structure " of the question . That divides the Question Understanding component into two successive subcomponents :
  a. Constructing an EFL expression , using only linguistic knowledge .
  b. Translating the EFL expression into a WML expression , by taking knowledge about the structure of the world into account .
- The Data Base Language (DBL) , which contains constants that correspond to data base primitives . (The World Model constants do not correspond to data base primitives , because we want to handle a " realistic " data base : one that was designed to be stored efficiently , rather than to reflect neatly the structure of the world .)
  This splits the Answer Computation component into two successive subcomponents :
  a. Translating a WML expression into a DBL expression taking knowledge about the data base structure into account.
  b. Evaluating the DBL expression .

The set–up of the system that one arrives at in this way , is shown in fig. 2. In section 3 , we say something more about PHLIQA' s formal languages in general . How the three successive translation modules are further divided into smaller modules , called "convertors" , is discussed in the sections 4 , 5 and 6. Section 7 treats the evaluation component . The Answer Formulation component is very primitive , and will not be considered further .

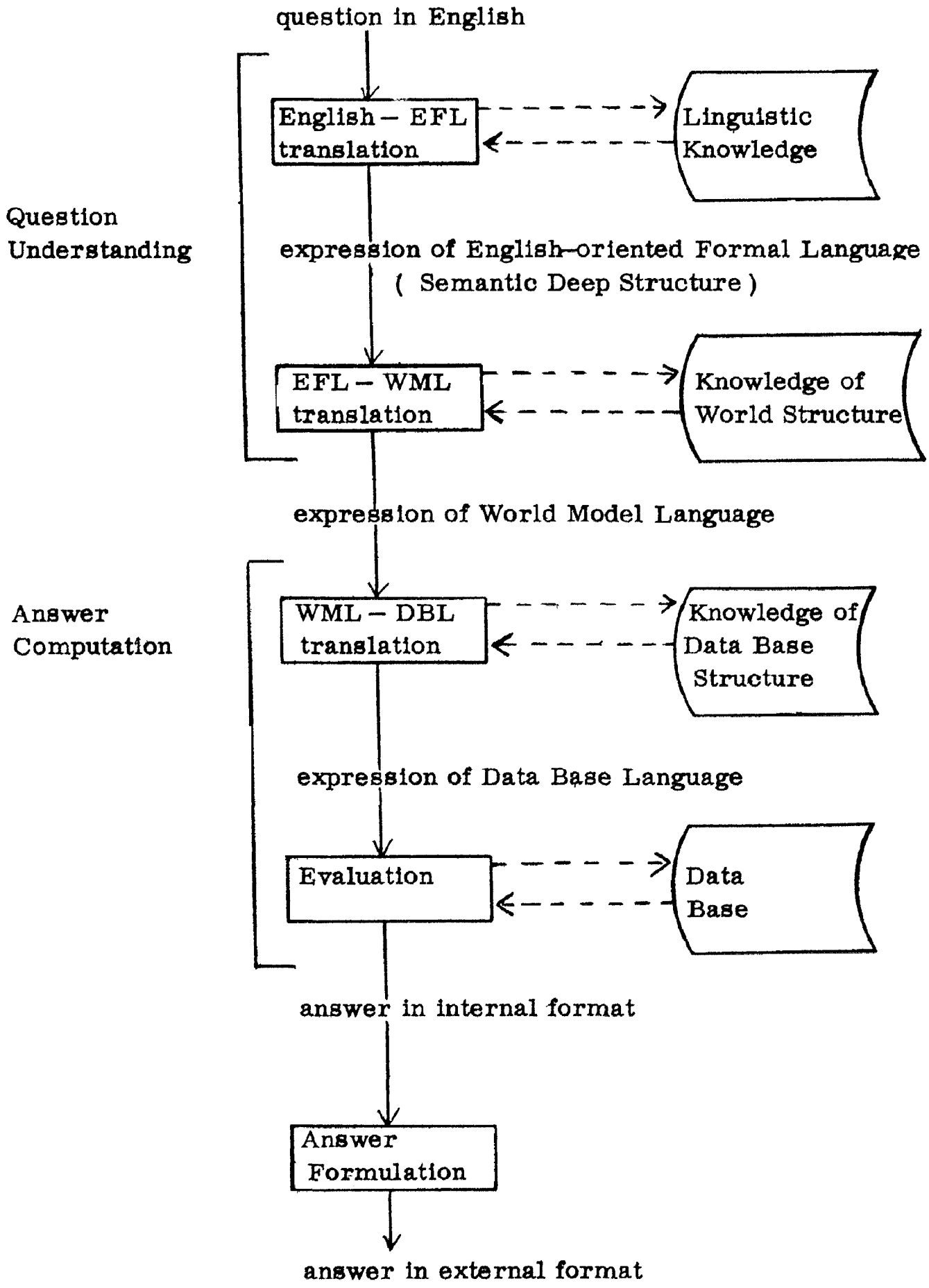


Fig. 2. PHLIQA 1 's main components .

3. PHLIQA 1's formal languages

3.1. Syntax

The three PHLIQA languages (the English-oriented Formal Language , the World Model Language and the Data Base Language) have largely identical syntactic definitions . As pointed out already , their most important difference is in the constants they contain . They share most , but not all , syntactic constructions .

PHLIQA expressions are " trees " that consists of terminal nodes (constants and variables) and syntactic constructions . A syntactic construction is an unordered collection of labeled branches , departing from one node .

The branches of a PHLIQA " tree " can converge to a common subtree .

Using a system of semantic types , the syntax of a PHLIQA language defines how expressions can be combined to form a larger expression . For every syntactic construction , there is a rule which specifies :

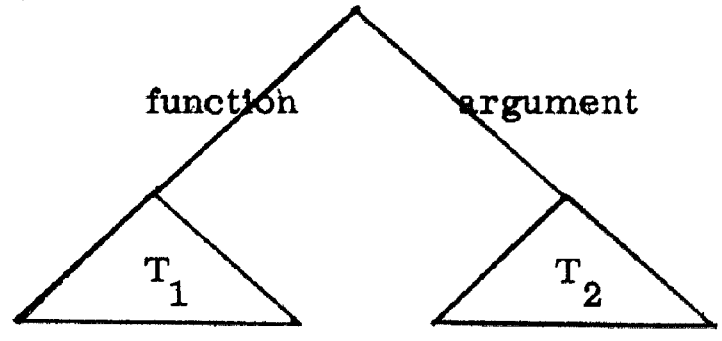
- What the semantic types of its immediate sub-expressions are allowed to be .
  (There is never a restriction on the syntactic form of the sub-expressions .)
- How the semantic type of the resulting expression is derived from the semantic types of the immediate sub-expressions .

Given the types of the elementary expressions (the constants and variables) , this defines the language . (Sources of inspiration for the syntax of our formal languages were the Vienna Definition Language (Wegner [1972]) , and a formulation of Higher – Order Logic by J.A. Robinson [1969] .)

Some simple examples of semantic types are the following :

A constant representing a single object has a simple type . E.g. , " 6 " has the type " integer " . A constant representing a collection of objects of type α has a type of the form $\langle\alpha\rangle$. E.g. , " companies " has the type "<company>", " integers " has the type "<integer>" .

A constant representing a function that can have arguments of type α and values of type β has the type $\alpha \rightarrow \beta$. E.g. , the function " f–company–sites " has the type " company $\rightarrow$ ⟨site⟩", the function " f–sum " has the type " ⟨integer⟩ $\rightarrow$ integer " .
The syntactic rule for the construction " function – application " could state that the expression



is well – formed if T_2 is a well–formed expression of type α and T_1 is a well – formed expression of type $\alpha \rightarrow \beta$, where α and β may be any type ; the whole expression then has the type β
The PHLIQA languages contain a wide variety of syntactic constructions , e.g. constructions for different kinds of quantification , for selecting elements from a list , for reordering a list , etc .

3. 2. Semantics

The PHLIQA languages have a formal semantics which recursively defines the values of the expressions. This definition assumes as primitive notions the denotations of the constants of the language : function – constants denote procedures , and the other constants denote value – expressions . This means that if we know the denotations of the constants occurring in an expression , the value of the expression is defined by the semantic rules of the language . For the Data Base Language , we indeed know the denotations of the constants ; what we call the data base is nothing but the implementation of the " primitive procedures ", i.e. : the procedures corresponding to DBL functions , and the procedures for finding the value – expressions of the other DBL constants .

Therefore , the DBL expressions are actually evaluable .
For the World Model Language and the English–oriented Formal Language , such a data base does not exist , but one could be imagined . We express this by saying that the WML and EFL expressions are " evaluable with respect to a virtual data base " .

4. Construction of the semantic deep structure of a question .

As we have seen , the English–oriented Formal Language differs from the other two languages in two respects :

1. It has different constants , of which the most important are :
   a. names of sets corresponding to nouns (e.g. " computers ") , to verbs (" buy – situations ") and to some of the prepositions (" in – place – situations ") .
   b. grammatical functions : subject , object , etc .
2. It has some different constructions . Here the most striking difference is that EFL constructions contain semantic and syntactic features . The semantic features influence the formal semantics of the constructions (e.g. the definiteness or indefiniteness of a noun phrase influences the choice of the kind of quantification for that noun phrase) . The syntactic features only play a role during the transformation process from English to EFL .

It should be noted that in general two synonymous sentences need not be represented by the same semantic deep structure in EFL . For example , the synonymy of ' A buys B from C ' and ' C sells B to A ' is not accounted for at this level . However , at the level of the World Model Language synonymous sentences are mapped onto equivalent (not necessarily identical) WML expressions .
The construction of the semantic deep structure in EFL consists of three main phases :

phase 1: a lexicon , providing for each word one or more interpretations , represented by pairs (CAT_i , SEM_i) , where CAT_i is a syntactic category and SEM_i an EFL expression .

phase 2: a set of rules that enables to combine the sequence of pairs (CAT_i , SEM_i) , corresponding to the original sequence of words , into higher level categories and more complex structures , until we have ultimately the pair (SENTENCE , SEM_s) , where SEM_s is the EFL expression for the complete sentence .
A rule of phase 2 is a combination of a context free rule and a set of rules on EFL expressions , that show when and how a sequence of pairs

$$(CAT_1 , SEM_1) , \ldots\ldots , (CAT_k , SEM_k)$$

can be reduced to a pair (CAT_R , SEM_R) .
The general format of these rules is :

– context free reduction rule :

$$CAT_1 + \ldots\ldots\ldots + CAT_k \rightarrow CAT_R$$

– EFL rules :

$$\begin{bmatrix} COND_1 : \ldots\ldots\ldots \\ ACTION_1 : \ldots\ldots\ldots \end{bmatrix}$$

. . . . . .

. . . . . .

$$\begin{bmatrix} COND_n : \ldots\ldots\ldots \\ ACTION_n : \ldots\ldots\ldots \end{bmatrix}$$

The CONDi's are conditions on the EFL expressions SEM_1 , ... , SEM_k .
The $ACTION_i$'s show how a new EFL expression SEM_R can be constructed with the help of SEM_1 , ... , SEM_k . The rule is applicable if at least one of the conditions $COND_i$ is true . Then SEM_R is constructed according to $ACTION_i$ and the sequence of pairs is reduced to (CAT_R , SEM_R) . If more than one of the $COND_i$ is true , we have a local ambiguity .

phase 3: transformation rules that transform the semantic surface structure into an EFL expression that is called the semantic deep structure . These transformation rules handle aspects of meaning that could not be resolved locally , during phase 2. This applies for instance to anaphoric references and elliptic clauses in comparative constructions .

A simpler example is the specification of the subject in a clause like ' to use a computer ' . The semantic surface structure of this clause means: ' there is a use-situation , with some computer as its object , and an unspecified subject ' . Phase 2 can be said to ' disambiguate ' this expression in a context like ' when did Shell start to use a computer ? ' .

A transformation specifies the subject of the use-situation as ' Shell ' . This transformation would not apply if we had the verb ' propose ' instead of ' start ' .

The conditions of phase 2 and phase 3 contain a "shortcut" to the world model: the semantic types of the world model interpretations of the EFL constants are inspected in order to avoid the construction of semantic deep structures that have no interpretation in the world model . This blocks many unfruitful parsing paths .

5 . Translation from semantic deep structure to unambiguous World Model Language expression

The translation from a semantic deep structure (EFL expression) into an unambiguous World Model Language expression proceeds in 3 phases:

phase 1: Translation from EFL expression into ambiguous WML expression. In this phase , transformations are applied which replace expressions containing EFL constants by expressions containing WML constants . Their most conspicuous effect is the elimination of "situations" and "grammatical functions" . It is

important to note that the resulting expression often contains several "ambiguous constants". These arise from polysemous terms in English : words that have a "range" of possible meanings . Such terms lead now to expressions with ambiguous constants: constants that stand for a whole class of possible "instances". An expression containing such constants , stands for the class of well-formed expressions that can be generated by "instantiating" the ambiguous constants .

phase 2: Disambiguation of quantifications .

Many sentences are ambiguous with respect to quantification .

E .g . "Were the largest 3 computers bought by 2 French companies ?" can either ask whether there are 2 French companies such that they both bought each of these computers , or , perhaps more plausibly , it can ask whether there are 2 French companies such that together they bought these computers .

Until this stage in the process , the representation of such questions contains constructions which stand for both interpretations at once . But now that the system' s assumptions about the structure of the world are reflected in the expression , some such interpretations may be ruled out as implausible , because they would lead to the same answer , independent of what the state of affairs in the world is . E .g ., the first interpretation of the above example question has the value "false" , independently of the values of the constants in the expression . (Because the assumption that a computer can only be bought by one company was introduced by a previous transformation) . Therefore , the second interpretation is chosen .

phase 3: Disambiguation of WML constants .

The ambiguous WML constants can be instantiated in a very efficient manner by using the semantic type system: The possible interpretations of an ambiguous constant are severely restricted by the semantic types of the other constants that appear in its context.

6. Translation from World Model Language expression to Data Base Language expression

In the World Model Language , constants correspond to the concepts of the universe of discourse . In the Data Base Language , constants correspond to primitive logical and arithmetical procedures and to primitives of the data base . The choice of these primitives was governed by considerations of efficiency , rather than by the wish to represent neatly the structure of the universe of discourse . Therefore , WML and DB contain different constants .

The translation from a WML expression to the DBL expression that will be evaluated , proceeds in three stages :

1. Paraphrase of the WML expression , in order to eliminate " infinite notions ".
   WML contains constants representing infinite sets or infinite continua , like " integers " , " money-amounts " and " time ". Such constants can not be directly or indirectly represented in the data base , and hence have no DBL-translation . By paraphrasing the expression , the infinite notions can often be eliminated .
2. Translation of expressions containing WML constants into expressions containing DBL constants .
   This translation is required by phenomena like the following :
   - it is possible that a class of objects is not represented explicitly in the data base , while properties of its elements are represented indirectly , as properties of other , related objects . (E.g. , cities do not occur in the PHLIQA 1 data base , but their names are represented as the city-names of sites .)
     A special case of this phenomenon is the representation of a continuum by a class of discrete objects (E.g. , " core " is represented by " core memories ") :
   - objects may be represented more than once in the data base . E.g. , in the PHLIQA 1 data base , the file of computer users and the file of manufacturers

can contain records that represent one and the same firm .

– the data base is more limited than the world model . Some questions that can be expressed in WML can be answered only partially or not at all : the WML expression has no DBL translation . The present convertor detects such expressions and can generate a message which specifies what information is lacking .

Examples of this case are : the set " integers " (if the attempt of the previous convertor to eliminate it has been unsuccessful) , and the " date–of–taking–out–of–use " of a computer (which happens to be not in the data base).

3. Paraphrase of the DBL expression , in order to improve the efficiency of its evaluation .

The DBL expression produced by the previous convertor can already be evaluated , but it may be possible to paraphrase it in such a way , that the evaluation of the paraphrase expression is more efficient . This conversion is worthwhile because , even with our small data base , the evaluation is often the most time–consuming part of the whole process ; compared to this , the time that transformations take is negligible .

7. The evaluation of a Data Base Language expression

The value of a Data Base Language expression is completely defined by the semantic rules of the Data Base Language (see section 3 . 2 .) , and one could conceive of an algorithm that corresponds exactly to these rules . For reasons of efficiency , the actual algorithm differs from such an algorithm in some major respects :

– in evaluating quantifications over sets , it does not evaluate more elements of the set than is necessary for determining the value of the quantification .

– if (e.g. during the evaluation of a quantification) , a variable assumes a new value , this does not cause the re–evaluation of any subexpressions that don' t contain this variable .

Currently , evaluation occurs with respect to a small data base in core . To handle a real data base on disk , only the evaluation of constants would have to change .

8. PHLIQA 1 's Control Structure

The sections 4 through 7 sketched what the basic modules of the system (the " convertors ") do . We shall now make some very general remarks about the way they were implemented . These remarks apply to all convertors except the parser , which is described in some detail by Medema [1975] .

The convertors can be viewed as functions which map an input expression into a set of zero or more output expressions . Such a function is defined by a collection of transformations , acting on subexpressions of the input expression . Each transformation consists of a condition and an action . The action is applied to a subexpression if the condition holds for it . The action can either be a procedure transforming a subexpression to its " lower level equivalent " or it can be the decision " this subexpression cannot be translated to the next lower level " .

All convertors are implemented as procedures which operate on the tree that represents the whole question . The procedures cooperate in a " depth-first " manner : a conversion procedure finds successively all interpretations that the input expression has on the next lower level . For each of these interpretations , as soon as it is found , the next convertor is called. If no interpretation can be found , a message giving the reason for this " dead end " is buffered , and control is returned to the calling convertor .

If the answer is found , it is displayed . If requested , the system can continue its search for more interpretations . If the answer level is not reached , it displays the buffered message from the " lowest " convertor that was reached .

Colophon

The PHLIQA 1 program was written in SPL (a PL/I dialect) , and runs under the MDS time sharing system on the Philips P1400 computer of the Philips Research Laboratories at Eindhoven .
The quantification–disambiguation phase of the EFL–WML translation , the efficiency–conversion (step 3) in the WML–DBL translation , as well as some parts of the grammar , are not yet part of the running system , though the convertors are completely coded and the grammar is elaborately specified .
During the design of PHLIQA 1 , the PHLIQA project was coordinated by Piet Medema . He and Eric van Utteren designed the algorithmic structure of the system and made decisions about many general aspects of implementation .
The formal languages and related transformation rules were designed by Harry Bunt . Jan Landsbergen and Remko Scha . Wijnand Schoenmakers designed the evaluation component . Jan Landsbergen wrote a grammar for an extensive subset of English.
All authors were involved in the implementation of the system .
During the design of PHLIQA 1 , extensive discussions with members of the SRI Speech Understanding team have helped us in making our ideas more explicit .

References

CODASYL Data Base Task Group
April 71 report . ACM , New York , 1971 .

P. Medema
A control structure for a question answering system .
Proceedings of the 4th International Joint Conference on Artificial Intelligence . Tbilisi , USSR , 1975. Vol. 2 .

S. R. Petrick
Semantic Interpretation in the REQUEST system .
Proceedings of the International Conference on Computational Linguistics , Vol. 1 , Pisa , 1973 .

W. J. Plath
Transformational Grammar and Transformational Parsing in the REQUEST system .
Proceedings of the International Conference on Computational Linguistics , Vol. 2 , Pisa , 1973 .

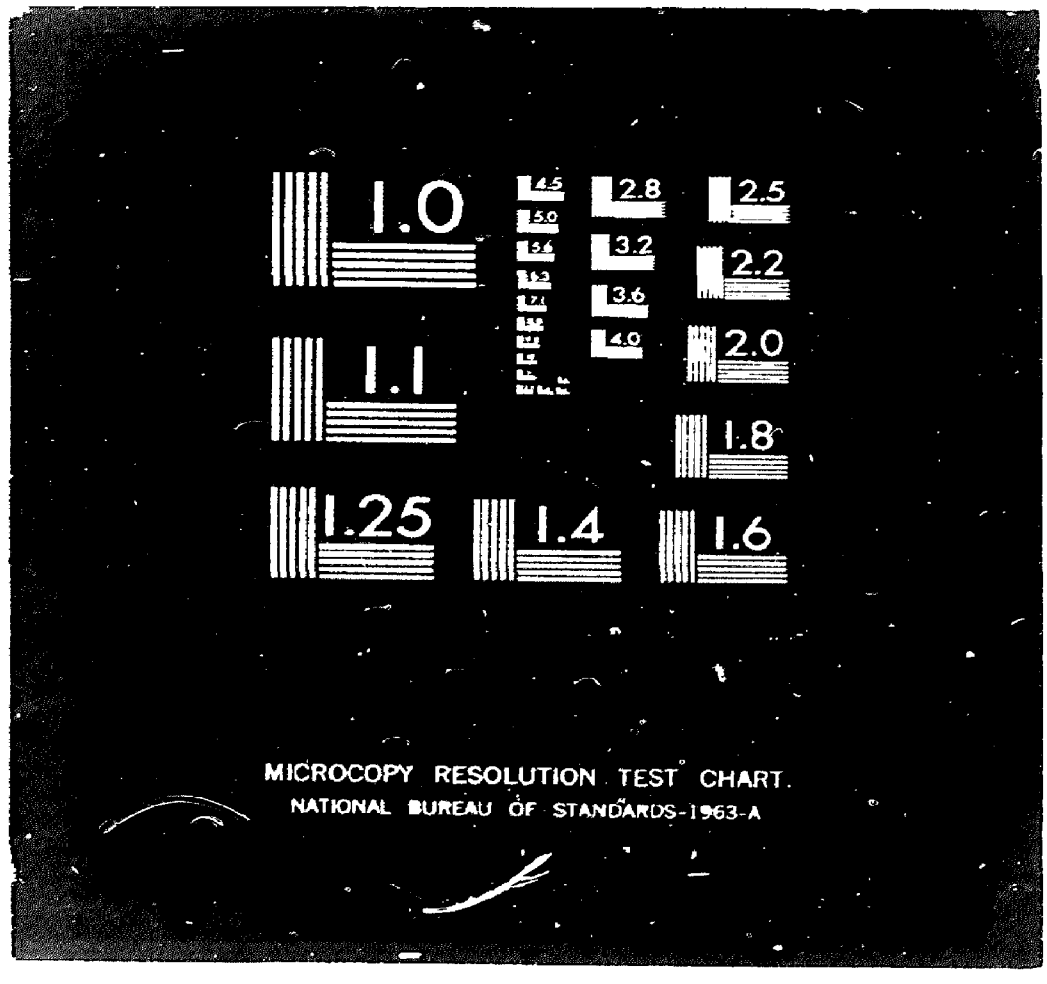
J. A. Robinson
Mechanizing Higher–Order Logic .
In : B. Meltzer and D. Michie (eds.) ,
Machine Intelligence 4 , Edinburgh University Press , 1969.

P. Wegner
The Vienna Definition Language .
Computing Surveys , Vol. 4 , no. 1 , 1972 .

T. Winograd
Understanding Natural Language .
Cognitive Psychology , Vol. 3 , no. 1 , 1972 .

W. A. Woods , R. M. Kaplan and B. Nash–Webber
The Lunar Sciences Natural Language Information System : Final Report . BBN , Cambridge , Mass. 1972 .

END


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
MICROCOPY RESOLUTION TEST CHART.
NATIONAL BUREAU OF STANDARDS-1963-A