Parsing Strategies and Semantic Interpretation Of Syntactic Parse Structures Of An English Subset

by

Nasser Abbasi

12 June 1988

Oakland University

CONTENTS

ABSTRACT

The purpose of this paper is to outline methods of parsing and sematically understanding an English sentence. Also this paper will outline an implementable interpreter that can be used to translate an english input into a formal representaion suitable for input to a system such as a data base query interpretor.

Grammans that can be Used for natural language understanding are discussed and classified in detailes.

A semantic Interpreter is analyzed and an implementable algorithm is outlined (using PL.I)

CHAPTER 1

INTRODUCTION

1.1 DIFFERENCE BETWEEN ENGLISH AND PROGRAMMING LANGUAGES

The principle difference between English and a Programming Language is that the grammatical and spelling rules for English are complicated and have many exceptions and ambiguities, whereas the corresponding rules for a programming language are concise are highly structured, have few (if any) special cases, and -if well designed- have no ambiguities.

There is a general understanding of the computational techniques for dealing with the syntax of English sentences , but no clear understanding of how to deal with their semantics.

This study will attempt to outline a method that provides a frame work for semantic interpretation of a subset of the English language.

1.2 USES OF NATURAL LANGUAGE PROCESSOR

A natural language processor can be used as a front-end for other computer systems that would otherwise require specialized command language to interact with , such as data base queries, and operating system user interface layer.

Also as translator to neutral representation layer within a translation system between two natural languages , where after understanding input from language X it will build an internal representation for this input as Y and then give an output in language Z from Y.

1.3 APPROACHES TO NATURAL LANGUAGE PROCESSING

1. noise disposal methods where results were sought in limited specific constrained domain, input sentences were restricted to simple declarative, and interrogative forms and were scanned by the programs for

re declared KEY words or patterns that indicated know objects and relationships.

If none were found then a previous subject will be picked and asked again.

domain-specific rules, called heuristic were used to derive the required response from the key word in the sentence and the knowledge in the data base.

2. text based approach: these systems essentially stored a representation of the text itself in the database using a variety of clever indexing schemes to retrieve material containing specific word or phrases.

these systems did not deal with the meaning of the sentence, i.e. they had no deductive powers.

3. limited logic systems: predicate calculus used to deduce semantic information from the input. for instance if the system were told that "tom is a man" and then told "all men are humans" then it will build an internal knowledge that "tom is a human".

deductions these systems carried were a limited subset of the full range of logical inference used in ordinary conversations.

4. Knowledge based systems: these systems use a large amount of information about the domain under discussion to help understand sentences- knowledge that is stored within the program using some knowledge representation scheme like logic, procedural semantics, semantic networks, or frames.

1.4 DOMAIN OF DISCOURSE

For a natural language the following questions need to be addressed :

- o What domain of discourse is rich enough to be a vehicle for studying the central issues yet simple enough to facilitate progress ?
- o What Representations are required for the sentence ?
- o What computations can be done and the constraints the can be applied ?
- o When do we conclude that something is understood ?

The first of these questions is the most critical for the following analysis as it outlines the constraints that can be used is the semantic analysis part of NLP , what we mean by this is to restrict the type of sentences that the processor will understand.

This means we need to restrict the grammar that acceptable sentences will adhere to.

For a natural Language we need a number of processing on a sentence to travel from the syntactic level reach the lowest level which is the semantic level .

For a programming Language this process is facilitated by the language Definition at hand when we build the parsing machine that will attempt this process from syntactic level to the semantic level.

In a natural Language such a definition either does not exist or does not apply to all sentences or is not formal enough to be used in the same manner as in Formal programming languages.

1.5 LEVELS OF PROCESSING

An example of the processing levels consider the statement " MARY WAS SICK YESTERDAY" call this statement M :

1. Sentence M contains 20 alphabets.
2. Sentence M contains 4 English words.
3. Sentence M contains one proper noun, one verb , one adjective, one adverb in that order.
4. Sentence M is an English sentence.
5. Sentence M contains one person's name, one linking verb, one adjective describing a potential health state of living, and one temporal adverb in that order.
6. The subject of sentence M is an individual probably female named MARY and the predicate is an ascription of ill health to the individual so indicated , on the day preceding the sentence utterance.
7. Sentence M asserts that the health of an individual named MARY was not good the day before yesterday.
8. Sentence M says that MARY was sick yesterday.

index thematic-role specialist
Level 1-4 will be carried by the Syntactic specialist
Level 5 will be carried by the Thematic-role specialist.
Level 6-8 will be carried by the Semantic specialist.

CHAPTER 2

STRUCTURE OF NATURAL LANGUAGE PROCESSOR

2.1 BASIC DEFINITIONS

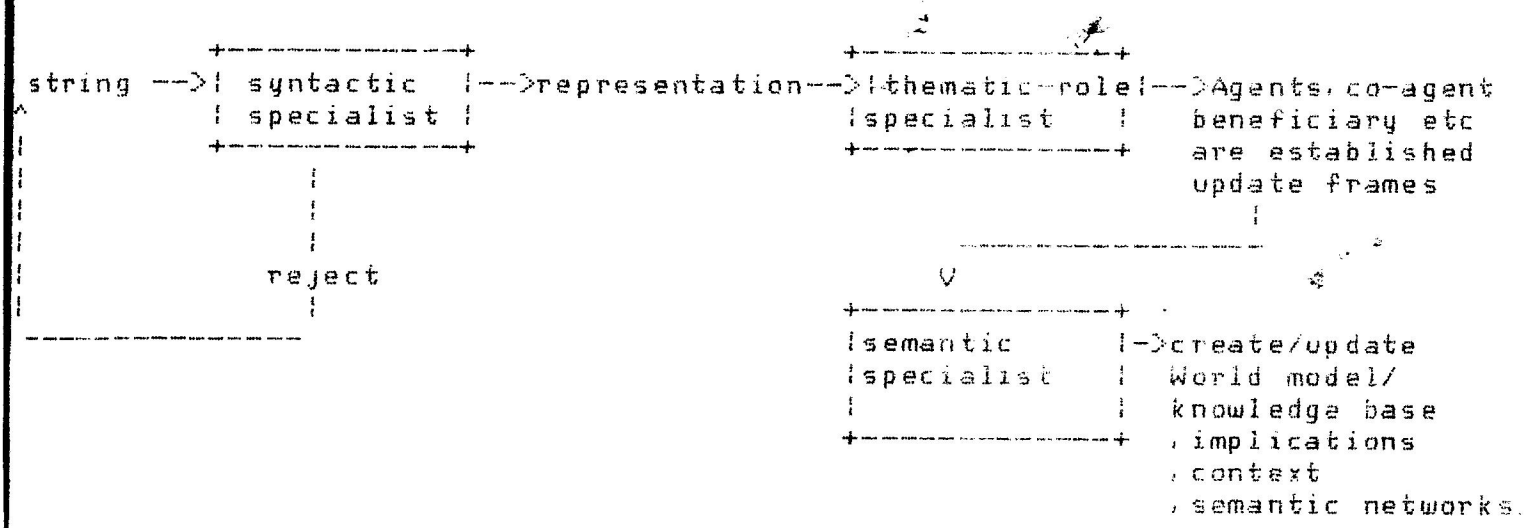
The set of all possible strings that can be made up from the alphabets of the language is the universe of discourse.

A language is a subset of the Universe of discourse such that every string in this set can be derived by defined rules (the grammar) of the language.

Formally a language L is subset of GAMMA* , where GAMMA is the terminal alphabet and GAMMA* = GAMMA+ U {epsilon} where epsilon is the empty string.

2.2 FLOW DIAGRAM OF NATURAL LANGUAGE PROCESSOR

the Overall structure on Natural language machine can represented as:



CHAPTER 3

SYNTACTIC SPECIALIST

3.1 DEFINITION OF PURSUER

Informally a grammar of a language is a scheme for specifying the sentences allowed in the language, indicating the syntactic rules for combining words into well-formed phrases and clauses.

the theory of generative grammar introduced by Chomsky (1957) radically influenced all linguistic research , including AI work in computational linguistics. the grammar is used to pick apart the sentence to its constituting parts to help understand it.

parsing is the process by which we determine that an input conforms to the grammar of the language and hence is part of the language, also it is the process of delineriazation of linguistic input to create a more complicated data structure , example a derivation tree, also this structure depicts some of the relations between words in the sentence ("this adjective modifies that noun, which is the object of a prepositional phrase..").

3.2 GRAMMARS FOR NATURAL LANGUAGE PROCESSING

The kind of grammars that are used for natural language understanding are:

- o phrase structure grammars
- o transformational grammars
- o case grammars
- o systemic grammars
- o semantic grammars

3.2.1 Phrase Structure Grammar

restrict language to a phrase-structured grammar (formal grammars):

A grammar is a four-tuple(N,&,P,S) where & is the terminal alphabet, N is the nonterminal alphabet, P is the set of productions of the form y->x where y and x are in (N U &*) and y contains at least one element of N such a grammar is called a phrase-structured grammar.

Chomsky [1965] distinguished four general classes of formal grammars.

-- type(0) unrestricted grammar	! increase	decrease
-- type(1) context sensitive	! restrictions	the size
-- type(2) context free	! on the	of the
-- type(3) right linear	! production	language
	v rules	produced

it turns out that type 2,3 are computational possible but do not represent all the forms of natural language strings.

for type(0) a language can be generated by this grammar if and only if it can be recognized by a turing machine.

for type(1) grammar the right hand side of the production rule contains at least as many symbols as the left hand side (context sensitive).

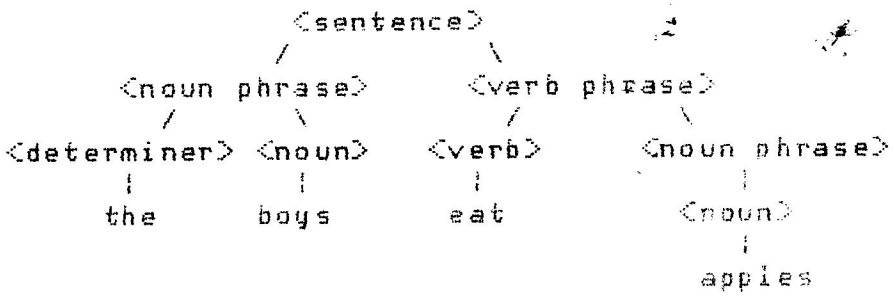
for type(2) grammar (context free grammar) each production must have only one non-terminal in the left hand side of the rule.

context free grammar is the one that would be used if this option is taken .

an example of context free grammar that might be used to generate some sentences in natural language is the following:

```
<sentence>        -> <noun phrase> <verb phrase>
<noun phrase>     -> <determiner> <noun>
                   |  <noun>
<verb phrase>     -> <verb> <noun phrase>
<determiner>      -> the
<noun>            -> boys
                   |  apples
<verb>            -> eat
```

using this derivation the sentence "the boys eat apples" have the following derivation tree:



In this grammar every production has the form x->y where x member of N and y is ant string in (N U)*, note y can be an empty string.

That is any CFG with rule x->epsilon is not a context sensitive.

3.2.1.1 Bottom Up Parser -

A Bottom Up parser always can be constructed for context free grammar. two broad classes of determinestic bottom up parsers are the precedence parser and the LR(K) which includes all the LL(K) grammars (top-down parsers).

A determinestic top-down parser can not be constructed for every context free grammar.

That is there are context free languages that cant be recognized by such a machine. such top-down parsers include LL(K), and predicative parsers (recursive descent).

Bottom up parser starts from the left side of the input and tries to reduce the handler to a right hand side symbol of a rule.

if it can reduce further (after a new handle is identified) then further reduction is carried.

if no such reduction is possible then it will shift to the right to retrieve another input token and the stack handle is adjust accordingly.

if at the end of the input the parser manages to reduce the handle to the goal symbol then it accepts the sentence else failure is announced.

In Bottom up parsing we use the production rules "backward".

for example given "he ate a frog ." as input and a rule PRONOUN-> he then we match r.h.s. of rule with "he" we find a match we replace "he" in the sentence by l.h.s. to become "PRONOUN ate a frog" and we start again until we end up with only the Goal symbol on the stack with no more input.

hence we accept the sentence.

3.2.1.2 Top Down Parser -

in deriving a syntactic structure the parser can operate from the goal.

it starts by looking at the right hand side symbols of the goal, then tries to find the rules for each one of these symbols, this is carried until a complete sentence structure is built up.

or if the sentence do not belong to the language then it will announce it failed.

In Top-down parsing we start with the goal and continue replacing it by r.h.s. of the rules finally replacing a r.h.s. with a terminal.

in Top down we need to be ready to backup and try another path. hence bottom up are considered more powerful.

3.2.2 Transition Networks

Transition Networks are a common representation of abstract recognition devices.

Transition networks are an alternative to formal grammar for defining formal languages (i.e. set of strings). Whereas a formal grammar defines a language in terms of combinations of types of strings, a transition network for a language is a kind of abstract machine for recognizing whether strings are in the language.

The machine can be in one of a set of states, and the transition state to state corresponds to processing successive symbols in the string to be recognized.

Certain special states correspond to acceptance or recognition of the string. if the machine finishes processing when in such a state, the string is accepted as being in the language recognized by the machine.

If the machine does not end in such a state the string is not accepted.

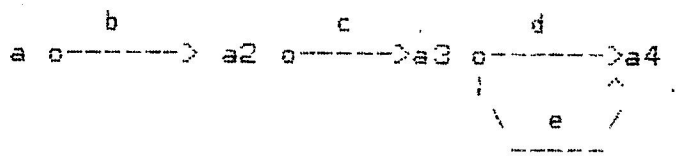
3.2.2.1 Transition Net Grammar - transition-net grammar. a version of top-down parsing the basic notation used here is nodes and arcs to connect nodes for example

a ->bcd
a ->bce

in the above representation the parser will throw out the result of the first production if it failed and try the second.

a better method would be to combine the two rules into one a->bc(d/e).

an alternative notation to this rule is using transition-network



the name of the first node or state a indicates that this is for parsing a, while the arcs indicate the next thing to be found first b then c, etc.

these should be thought of as test that need to pass to reach next state. from state a3 the parser can follow either d or e.

if one fails then parser goes back to a3 state and try other path to reach a4.

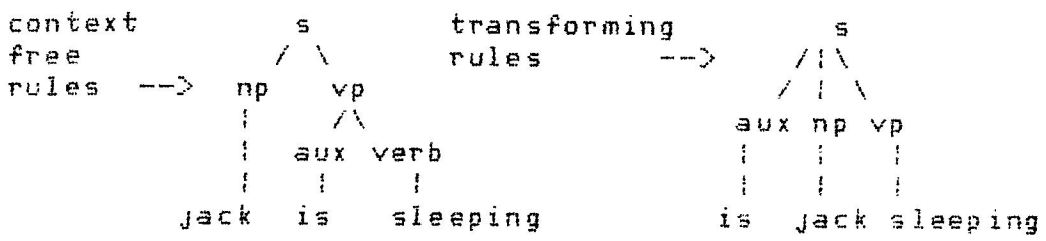
note that any language that can be described by context free grammar can be described by transition network.

3.2.3 Transformation Grammar

Chomsky(1957) states that an adequate grammar of a language like English must be a GENERATIVE grammar. this kind of grammar is concerned with competence as opposed to performance of the system.

Transformations can specify symbol arrangement, as when the declarative sentence "the silly robot has moved the red pyramid" becomes the interrogative sentence " has the silly robot moved the red pyramid?". context free grammars that uses transformations , are called transformation grammars.

an example of transformation carried on a context free grammar generated derivation tree is the following:
input is "jack is sleeping"



One logical inadequacy of transformation grammar arises in the deep structure (parse tree) assigned to negative questions containing quantifiers.
for example :
the question "Doesn't some American Airlines flights leave Detroit ? "
is not the question equivalent to the statement :
"some American airlines flights doesn't leave Detroit"

3.2.4 Augmented Transition Grammar

ATN parsing methods has its earliest roots in works by [thorne68, bobrow69] however it was woods [Woods 72] who popularized the idea.

3.2.4.1 Preliminary Theoretical Concepts -

a finite state transition diagram (FSTD) is a simple device consisting of a set of states (nodes) and arcs leading one state to another.

one state is the start state and one is the final state.

arcs are labeled with the terminals of the grammar

if the device reaches a final state from the start state when starting from the beginning of the sentence and reaches the final state at the end of the sentence (input) then we say that the device accepted the sentence.

FSTD can recognize only type(3) grammars. That is it is impossible to build a FSTD that will dependably distinguish the sentence in even a context free language.

However regular grammars are inadequate for dealing with natural languages, so a natural extension to FSTD is to provide recursion mechanism that increases their power to handle the context free languages.

these modified FSTD are called recursive transition networks RTNs.

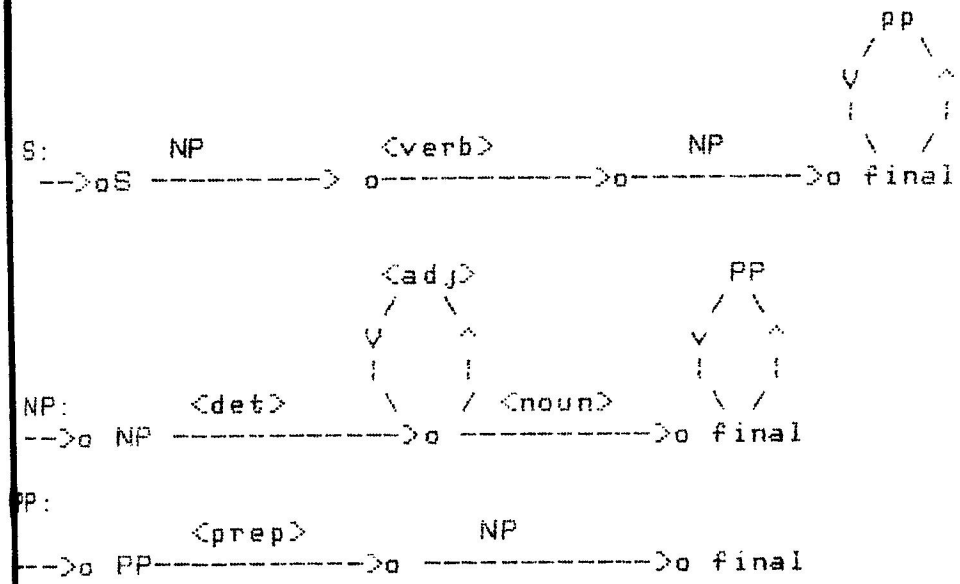
also an arc in RTN can be traveled using a nonterminal as well as a terminal as in the case of FSTD .

an example of an RTN is let NP represent a noun phrase
PP prepositional phrase
det determiner
prep preposition
adj adjective
final is the accepting node.

the following RTN will parse the following sentence
" The little boy in the swimsuit kicked the red ball"
NP : the little boy in the swimsuit
PP : in the swimsuit
NP : the swimsuit
Verb: kicked
Np : the red ball

notice that any sub network may also call another sub network including itself. in the below example , for instance, the prepositional phrase contains a noun phrase.

also note that RTN may be nondeterminstic in nature, that is, there may exist more than more arc to follow at one time, this is solved by parallel processing of the various arcs, or by trying each one and backtracking if it fails (similar to top down parsing).



Context free grammar however are still insufficient to handle natural languages so RTNs are augmented even more to increase their power and this leads to Augmented transition networks.

3.2.4.2 Difference Between RTN And ATN -
In ATN the augmentation is in three ways from RTN's :

1. A set of registers has been added, to store informations about partially formed derivation trees , between jumps to different networks.
2. Arcs, aside from being labeled by terminals and nonterminals have tests on them, these tests have to successed before transition from one state to another is employed.

Tests on arcs in ATN :

test	success condition	success side effect
(CATEGORY terminal symbol)	next token of input is a terminal	remove next token from input stream
(PARSE non-terminal)	the sub-network for the non-terminal completes to a final state	None
(PEEK terminal symbol)	next token is a terminal symbol	None
(WORD "abc")	next token of input is the word "abc"	remove next token from input stream

3. Actions may be attached to an arc , to executed whenever the arc is taken.

so in summary , FSTD = state + terminals
RTN = FSTD + nonterminals
ATN = RTN + set of registers + actions + tests

ATNs are capable of building deep structures descriptions of a sentence in an efficient manner.

The test and the actions on the arcs make ATNs well suited to handle Transformational grammars.

the ability to place arbitrary conditions on the arcs provide the context sensitivity.

ATNs have been successfully applied to question answering in a limited (closed) domains.

one of the limitation of the ATNs is the heavy dependence on syntax restricts the ability to handle ungrammatical (although meaningful) utterance.

2.5 WASP Grammer

Wait-and-see grammar WASP substitute looking ahead for backing up.

3 WHY PHRASE STRUCTURE INADEQUATE

English is neither a regular nor a context free language.

the reason is that these restricted grammars can not generate certain ommon constructions in every day English such as :

"tom, bill, dave, and charlie are the husbands of jane, MARY, sue, and oan respectively"

it was not determined if a context sensitive grammar could be written o generate precisely the sentence of English. rather such a grammar was ejected for the following reasons:

— it made the description of English unnecessarily clumsy and complex, for example in the treatment required for conjunction, auxiliary verbs, and passive sentences.
— Assigns identical structures(derivation trees) to sentences that are understood differently e.g.
" the picture was painted by a new technique"
" the picture was painted by a new artist"
— it provided no basis for identifying as similar the sentences that have different surface structures but much of their meaning in common e.g.
" john ate an apple"
" did john eat an apple?"
" what did john eat ?"
"who ate an apple?"

3.4 SUMMERY OF PARSING SYSTEMS

Actual parsing systems are the following

1. template matching
2. simple phrase-structure grammar parsers
3. transformational grammar parsers
4. extended grammar parsers (ATN, and CHARTS methods) which extends the concept of phrase-structure rules and derivations by adding mechanisms for more complex representations and manipulations of sentences.
5. semantic grammar parsers
   similar to ATN's but the non-terminals used are not <noun>, <verb> etc, but classes that are motivated by the concepts involved . e.g. for a flight reservation system we can have nonterminals such as <destinations>, <flight-time> etc.
6. grammerless parsers: some natural languages processing systems have abanded the traditional use of grammars for linguistic analysis and used ad-hoc procedures centered around individual words.

3.5 PROLOG AND NATURAL LANGUAGE PROCESSING

Of all procedural and declarative languages , Prolog is most suitable for use in this area since we can directly represent production rules (rewrite) and the semantic rules that we will do the interpretation against as a definite-claues .

Prolog also facilitates the latter stages of matching the syntactic tree portion we which to interpret against the semantic rules in the data base which could be more than one for the particular tree root.

An example of all this will be explained in chapter 5 where the algorithm is explained and in chapter 6 where the algorithm is presented.

3.5.1 Prolog Syntax For Tree Representations

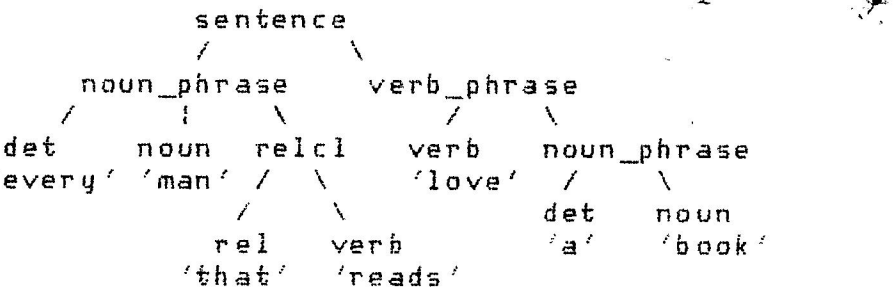
this section will show how to represent in prolog a tree structure.
given the following as the rules

```
sentence - > noun_phrase verb_phrase
noun_phrase -> det noun relational_clause
det ->
    !determiner
relational_clause ->
                  ! relative  verb_phrase
verb_phrase -> verb
             ! noun_phrase
```

then given sentence "every man that reads love a book" can be as

```
object=sentence(noun_phrase(det("every")
        ,"man"
        ,relational_clause("that",verb("reads")))
        ,verb("love",noun_phrase(det("a"),"book",none)))
```

and the parse tree will look like

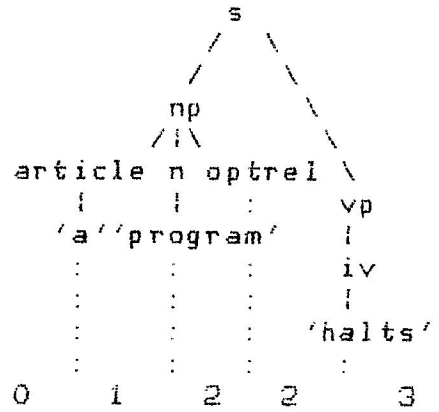


3.5.2 Axiomatizing Parse Tree For Prolog Representation

to be able to represent the production rules in definite-claues from in prolog we need to axiomatize the rules. for example gives the following rules in context free grammar (please see appendix for explanation of notations)

```
s-> np vp
np->article n optrel
np->pn
optrel ->
        !'that' vp
vp-> tv np
vp-> iv
pn->'terry'
   !'naser'
det->'a'
n->'program'
iv->'halts'
tv->'writes'
```

and given the following sentence "a program halts" the parse tree is



the non terminals can now be seen as a binary relations on the positions. for example n(1,2) and vp(2,3) etc.

It can be seen then that the statement s can be expressed as
np(0,X) & vp(X,3) ==> S

an extension to this yields V1(x1,x2)..&..&Vi(Xi-1,Xi) ==> S
this is why now we can express production rules in clause form in prolog syntax. for example the above production rules expressed in BNF can now be expressed in prolog (I use turbo prolog here) as :

```
s(p0,p) :- np(p0,p1),vp(p1,p).    (note "," in prolog means logical AND)
np(p0,p) :- det(p0,p1),n(p1,p2),optrel(p2,p).
np(p0,p) :- pn(p0,p).
vp(p0,p) :-tv(p0,p1),np(p1,p).
vp(p0,p):- iv(p0,p).
optrel(p0,p) :- connects(that,p0,p1),vp(p1,p).
pn(p0,p) :- connects(terry,p0,p1).
pn(p0,p) :-connects(naser,p0,p).
iv(p0,p) :- connects(halts,p0,p).
det(p0,p) :- connects(a,p0,p).
n(p0,p) :- connects(program,p0,p).
tv(p0,p) :- connects(writes,p0,p).
```

note that literal connects(Terminal,Position1,Position2) is used to mean that terminal symbol Terminal lies between Position1 and Position2.

CHAPTER 4

SEMANTIC ITERPRETER SPECIALIST

4.1 GOAL OF INTERPRETER

The goal is to translate the syntactic representation of the English sentence into an expression if formal language (query language) that could in turn be passed into a formal language processor to do the action specified by the expression such as output a piece of information or update a data base.

That is given a complete and correct parsing of the input (example using transformation grammar) it will apply a formal procedure to come up with a formal intermediate meaning.

4.2 BASIC DEFINITIONS

4.2.1 Definition Of Meaning

We need to give a more rigors definition of what is meant by finding the meaning (semantic) of a sentence as this is the goal of the interpreter.

the meaning of a word such as dog consist of an operational procedure for determining weather a given object is actually a dog.

that is the meaning of dog consist of determining if the predicate dog(x) is TRUE or FALSE.

likewise the meaning of verbs, adjectives, and prepositions consists of operational procedures for determining the truth or falsity of the corresponding predicates.

This operational procedure is what we are seeking .

We also need the procedure to tell us that a proposition has no meaning if the arguments are inappropriate to the predicate.

No meaning here means that the sentence did not have a corresponding semantic rules in the data base to carry the process of the interpretation.

4.2.2 Predicate Definition

Like a function a predicate takes a specific number of arguments. but unlike a function , a predicate does not have a value, instead it deliver a meaning, That is it "say something" about its arguments.

Thus if P is a predicate which takes 3 arguments, then p(x,y,z) is a message which says something about the objects x,y,z.

For example a predicate PRIM may be defined such that PRIM(x) has the same meaning as the statement "x is a prime number".

Or a predicate Between can be defined such that between(x,y,z) has the same meaning as "x<y<z".

A predicate which takes more than one argument is called a relation.

A proposition is a message formed by a predicate name together with designators for its arguments.

4.3 HOW INTERPRETER WORKS

The basic algorithm can be outlined initially as the following :

1. Read user input sentence.
2. Obtained the parse tree of the sentence . (example using using transformation grammar on context free parse tree)
3. Interpret the parse tree meaning and obtain the meaning in the form of a query or command the can be passed to commands function to carry the commands or the query and return the result to the user.

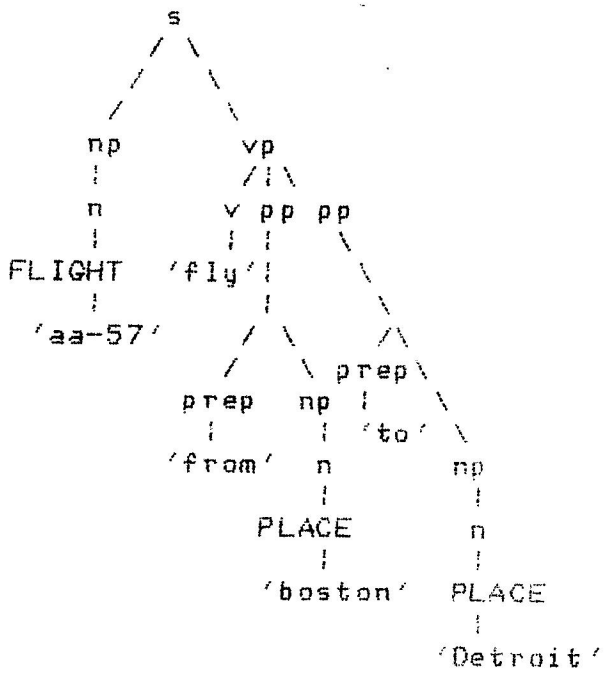
   once the sentence has been interrupted, its truth with respect to data base of the system (where all semantic rules reside) can be evaluated by determining whether the interpretation is one of the propositions (beliefs) of the system, or if the interpretation is deductible from them using the semantic deduction rules and predicate calculus .

   the interpretation of the sentence here is a clause form . it is the job of the interpreter to transform an parse tree to this form and pass the form to the commands interpreter to determine its truth and if so process it and return the result back to the user.

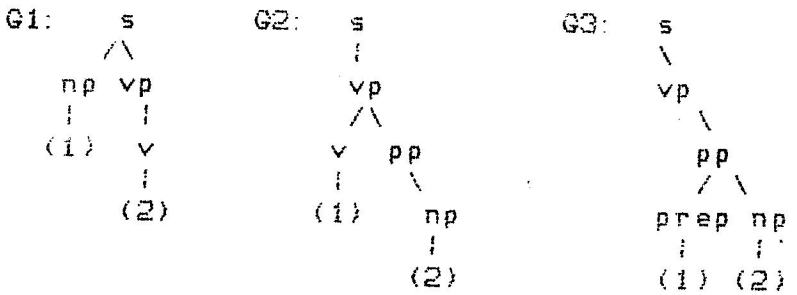
4. Loop back to 1

4.3.1 Algorithm Explained With Example

I'll outline the algorithm against the following sentence
"aa-57 flies from Boston to Detroit"
first we obtain the parse tree :



The grammatical relations among elements of phrase structures are defined by partial tree structures which matches subtrees of the parse tree. three sub trees are extracted from the above tree are :



each above structure can be represented by the following logical expression
(G1: FLIGHT(1) AND (2)='fly')
(G2: (1)='from' AND PLACE(2))
(G3: (1)='to' AND PLACE(2))

a semantic interpretation for the sentence is obtained by LOGICALLY anding all the semantic rules found from the subtrees. for example for the above parse tree we make the flowing rule and add the rule to the rules data base:

IF (G1: FLIGHT(1) AND (2)='fly')

```
    AND (G2: (1)='from' AND PLACE(2))
    AND (G3: (1)='to'   AND PLACE(2))
THEN
    CONNECT(1-1,2-2,3-2)

            ^
            |
    this is the semantic interpretation of the given input sentence.

    1-1 means the terminal under node 1 in subtrees 1
    2-2 means the terminal under node 2 in subtrees 2
    3-2 means the terminal under node 2 in subtrees 3

    so the final interpretation is
    CONNECT('aa-57','Boston','Detroit')
```

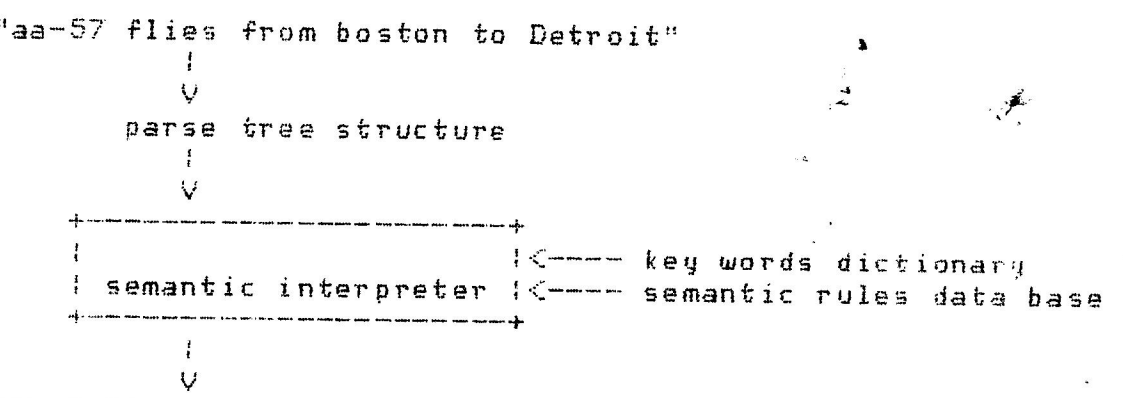
the semantic rules (G1,G2, etc.) are rules that we have to write down and add to a semantic rules data base. we number each rule. we can have many rules for our system (flight reservation system) example of some of the rules :

```
rules #1
IF      (G1: FLIGHT(1) AND (2)='fly')
    AND (G2: (1)='from' AND PLACE(2))
    AND (G3: (1)='to'   AND PLACE(2))
THEN
    CONNECT(1-1,2-2,3-2)

     e.g. "aa-57 flies from boston to chicage"

rule #2
IF      (G1: FLIGHT(1) AND ( (2)='fly' OR (2)='leave' OR (2)='go')  )

    AND (G3: (1)='from' AND PLACE(2))
THEN
    DEPART(1-1,2-2)

    e.g. "aa-57 departs from boston"

rule #3
IF      (G1: FLIGHT(1) AND ( (2)='fly' OR (2)='go')  )
    AND (G3: (1)='to' AND PLACE(2))
THEN
    ARRIVE(1-1,2-2)

    e.g. "aa-57 flies to Detroit"
```

many other rules can be constructed that will cover the domain of possible sentences within the domain of our system.

pg

if we can find more than rule that we match against then the final sentence semantic interpretation is the adjunction of all the action part of the rules. for example our sentence example will have subtrees that will match rule 1,2 and 3 above so the final semantic interpretation of the sentence is



the dictionary shown above as another input is a file used to tell the interpreter the possible rule numbers that can match against for a parse tree with a particular word as its root. the record format for this file can be like this :

<terminal>/<syntactic category>/<rule number><rule number>

for example the following are some of the records that can be searched for out example

'fly'/NOUN/1,3
'to'/PREPOSITION
'a'/DETERMINER
'aa-57'/NOUN/FLIGHT
etc ..

4.3.1.1 Review Of The Algorithm -
The semantic interpreter will look up the dictionary to the find all the semantic rules that it needs to search.
if search fails the it announce that sentence can not be understood.
if a match found the it will read all the numbers of the rules.
it will then read all the rules themselves from the rules data base using their numbers as an index.
take each rule in turn and see if the subtrees at hand will match the CONDITION part of the rule . if it does then add the ACTION part of the rule the meaning string. this string is initialized at the start to empty.
try all the rules in turn and every time a rule (e.g. rule#2 above) is matched do the following
meaning_string := meaning_string + 'AND' + action_part_of_current_rule.
at the end of looking all the rules if the meaning_string is empty announce failure (sentence in grammatically not part of our sub English) either

add rules to the data base to be able to understand it or re sub
sentence if form that can be understood by the semantic interpr
if semantic_string is not empty the its content is passed to a
sentence interpreter to act on by the interpretation.

for example the output can be the formal language for a DATA BA
language processor that can understand the output of the semant

in this way a user will input an English sentence and the sema
does a transformation on the sentence and output another senten
a query command string to the data base query subsystem.

4.3.1.2 Presentation Of Algorithm -
PL I program of the main line algorithm

```
main_line : proc ;

loop:


  semantic_buffer = '' ;
  get (user_sentence);

  call parse(user_sentence);

  root = get_tree_root();

  list_of_rules = look_up_rules_for(root,number_of_rules);

  i=0;
  do while (number_of_rules ^= 0 ) ;
     i=i+1;
     current_rule = list_of_rules(i);
     number_of_rules = number_of_rules-1;

     condition_part_of_rule = get_condition_part_of_rule(current_rule);

     if match(condition_part_of_rule , tree_with_root(root)) then
       do;
          rule_interpretaion = action_part_of_rule(current_rule);
          semantic_buffer = semantic_buffer !! 'AND' !! rule_interpetation;
       end;
     else
     ;
  end; /* while */

  if semantic_buffer = '' then
       put ('SENTENCE not understood');
  else
    call QUERY_LANGUAGE_PROCESSOR(semantic_buffer);

 goto loop;

end main_line;
```

CHAPTER 5

CONCLUSIONS AND SUMMARY

A study into natural language processing is presented in this paper. many aspects of this filed of AI is outlined.

A formal method based on pre-defined semantic rules and a key words dictionary is presented and is practical for implementation as a question/answer system .

natural language processing can be practically used within a restriction of the domain it will be used on . the method outlined will not be suitable if the domain covered is very large since then the rules will be many and the process of pattern matching will take a long time

The above algorithm can be best implemented using the language PROLOG.

APPENDIX A

ABBREVIATES TABLES

Tables 1

symbol	abbreviates
s	statement
np	noun phrase
vp	verb phrase
iv	Intransitive verb
tv	Transitive verb
pn	proper noun
det	determiner
n	noun
optrel	optional relative clause

APPENDIX B

REFERENCES

- William Barrett, Rodney M. Bates, David A.Gustafson , John D. Couch
  Compiler Construction , Theory and practice
  SRA , second edition, 1986

- Patrich Henry Winstion
  Artificial Intelligence
  Addison Wesley, Second edition , 1984

- Eugene Charniak , Drew Mcdernmot
  Introduction to Artificial Intelligince
  Addison Wesley , 1985

- Edward A. Feigenbaum , Avron Barr
  The Handbook Of Artificial Intelligence Volume One
  Heuris tech press, 1981

- Schildt
  Artificial Intlligince Using C.
  McGraw Hill, 1987

- Thorne , J Bartley
  "Syntactic analysis of English by machine" in machine intellegence 3 ed.
  D.michi, American Elsevier, New York (1968)

- Bobrow, Daniel g and fraser, Bruce
  "An augmented state network analysis procedure"
  Proc. Ijcai ipp. 557-569 (1969)

- Woods, William A .
  "An experimental Parsing System For Transition Network Grammers"
  pp 113-154 in natural language processing, ed R. Rustin, Algorithmics press
  new york (1972)

- Prolog and natural language analysis
  Fernando C.N. Pereira
  Stuart M. Schieber
  Center for the study of languages and information, 1987

- Linguistic Theory and Computer applications
  P.Whitelock , Editor
  Academic Press 1987

- Syntax analysis and software tools
  K. john Gough
  International computer science series , 1988

INDEX