Rule Forms for Verse, Sentences and Story Trees

R. F. Simmons & A. Correira

Technical Report NL35

Department of Computer Sciences

University of Texas at Austin 78712

January 1978

Abstract

Rule forms and their interpreters are described for deriving sensible and nonsensical verse, for analyzing sentences into case structures, for generating sentences from case structures, and for generating story trees. A system of inference rules and assertions in the form of Horn clauses and their interpreter are presented as a computational method for generating narrative story trees which have the property that their terminal propositions form the story, while nodes closer to the root provide summaries. The story trees and their generator are proposed as a promising computational model for the macrostructure theorized by Kintsch and Van Dijk to account for a human reader's memory and understanding of narrative text.

1.0 INTRODUCTION*

A most significant problem in computational linguistics is to develop a formal definition of a computational structure that represents a human's understanding of a narrative or any other discourse. If such a structure is defined for a given text, it becomes possible to produce a computational model in the form of a grammar and an algorithm that applies the grammar to the input text to compute the representation of understanding. This assertion is more fully developed and supported in later sections of this paper.

A series of psychological studies (typified by Kintsch (1974) van Dijk (1975), Meyer (1975), Thorndike (1977), and Crothers (1972)), offers strong evidence for the hypothesis that a person, after reading a text, has developed a macrostructure that organizes the propositions of the text in a hierarchical form. The propositions highest in the hierarchy are more general than those that occur at the lower levels. When tested for recall, what a person remembers from reading a text are propositions at the higher levels. If asked to produce a summary, the person reports a set of high level propositions also. Significantly, what a person remembers after a few days is a set of propositions very similar to those produced by subjects asked to write summaries.

This research is partially supported by NSF Grant MCS 77-0135 and RADC Contract F-30602-78-c-0132.

From a computational viewpoint this hypothesized macrostructure can be represented as a tree where nodes closest to the root are the most important events -- i.e. episodes in a narrative, or content categories in an expository text-- and nodes closest to the leaves are concerned with details of the episodes or description. What a person remembers after a period of days, or what a person reports as a summary is an organized set of nodes closest to the root -- a small subset of all the nodes in the tree.

These findings lead us to the hypothesis that one effective computational model for a human's understanding of a discourse is the computation of such a tree so organized that the nodes closest to the terminals form a summary approaching the length of the text. The terminal nodes of this tree are the sentences or phrases of the text itself.

In purely linguistic terms a similar hypothesis -- that the content organization of folk tales could be described as a phrase structure tree or story grammar -- was published by Propp in 1927. In the last decade Propp's work has received wide recognition and has been a seminal influence on a whole subdiscipline of text linguists mainly in Europe, but now represented in the United States by Grimes (1975), Klein (1965) and Rumelhart (1975) among others. Earlier computational linguists including Klein et. al. (1963) and Harper and Su (1969) studied the application of text grammars to computing stories and representations of story content, respectively, with limited success. Psychologists cited above adopted the notion of text grammars as a technique for representing the structure of discourse as a basis for measuring what human readers understood about a text.

In artificial intelligence work, comparable notions of macrostructure -- frames, scripts, partitions etc. -- are found to be essential to provide representations that maintain connectivity among sentences in systems that use the organized content of dialogue or discourse for some computational purpose. Schank (1975), Lehnert (1977), and Meehan (1976) have published demonstrations of the power of their scripting approach, and Bobrow and Winograd (1977) showed applications of frames to understanding dialogue. A related idea of partitioned semantic networks and focus spaces was shown to useful in computing representations of dialogue by Hendrix (1976), by Grosz (1977) and others. A selected survey of text understanding research by Young (1977) outlines much the work in the psychological, linguistic and artificial intelligence research in this area. Books by Charniak and Wilks (1976) and by Schank and Abelson (1977) develop more fully the notions of frames and scripts.

In an attempt to generalize the purely linguistic approach to include methods from artificial intelligence work, the next two sections of this paper describe some uses of grammars for generating and analyzing English sentences using the derivation tree to direct the flow of control in generation and parsing procedures. Subsequent sections develop the idea of story trees and present an organized system of inference rules and assertions for generating simple stories and their summaries.

A necessary first step in computational understanding of a text is to define the structure that represents that understanding. By developing a system for generating story trees we believe we have accomplished that first step by defining a precise computational model for the macrostructure hypothesized by psychologists to account for human understanding of text.

2.0

Pogen for Sense and Nonsense

By 1960, Yngve had published a procedural description for a method of generating (i.e. computing) syntactically well-formed sentences from a grammar. This formed a basis for some early work by Klein et. al. (1963) for generating coherent discourse where coherence was controlled by a dependency structure among words that was derived from recording their usages from actual texts. The generation algorithm has proved useful over the years in teaching computational linguistics by showing students how a text can be analyzed into a grammar form that can be used by a computer to generate that text, and with appropriate vocabulary classes, variations on it.

Let us first consider a phrase structure grammar for a simple sentence, S1.

S1) The little train goes toot, toot.

```
                  S
                /     \
           NP              VP
          /  \            /   \
     ART    NP1       V        NP2
     /      /   \     |        /  \
    /     ADJ    N    |       N    N
   /      /      |    |      /      \
THE   LITTLE  TRAIN GOES  TOOT    TOOT
```

After constructing the above immediate constituent phrase structure tree, we can start with the root, S, and record the grammar for the sentence as follows:

S = NP + VP	ART = THE
NP = ART + NP1	ADJ = LITTLE
NP1 = ADJ + N	N = TRAIN
VP = V + NP2	N = TOOT
NP = NP2	V = GOES
NP2 = N + N	

A sentence generation algorithm starts with the symbol, S, selects a rule beginning with S from the grammar, rewrites the rule as its right side and recurses until it reaches elements such as THE, TOOT, etc. which have no rules associated with them and thus are terminal nodes. It lists these terminal nodes in the order which they are achieved and thus outputs a sentence.

```
For example,  (GEN S) = (GEN NP) (GEN VP)
              (GEN NP) = (GEN N) (GEN N)
              (GEN N) = (GEN TRAIN)       = TRAIN
              (GEN N) = (GEN TOOT)        = TOOT
              (GEN VP) = (GEN V) (GEN NP)
              (GEN V) = (GEN GOES)        = GOES
              (GEN NP) = (GEN ART) (GEN NP1)
              (GEN ART) = (GEN THE)       = THE
              (GEN NP1) = (GEN ADJ) (GEN N)
              (GEN ADJ) = (GEN LITTLE)    = LITTLE
              (GEN N) = (GEN TOOT)        = TOOT
```

The result is on the rightmost column, "TRAIN TOOT GOES THE LITTLE TOOT", a syntactically well-formed nonsense sentence.

POGEN is a realization of this procedure as a LISP function shown in Figure 1. This function expects each left part of a rule to have a list of right parts, as it's VALue, and provides a random selection device, CHOOSE, to select one. If the left part also has the property

REPLACE, a new right part is formed as a VALue with only the previously selected member of the left part. This procedural feature provides for controlled repetitions as illustrated in the next example.

```
(DEF
 (GEN (SYMB)
      (PROG (J)
            (COND ((NULL SYMB) (RETURN))
                  ((ATOM SYMB) (GO A)))
            (RETURN (APPEND (GEN (CAR SYMB)) (GEN (CDR SYMB))))
          A (COND ((SETQ J (GET SYMB "VAL))
                   (SETQ J (CHOOSE J))
                   (COND ((GET SYMB "REPLACE)
                          (PUT SYMB "VAL (LIST J))))
                   (RETURN (APPEND (GEN (CAR J)) (GEN (CDR J))))))
            (RETURN (LIST SYMB)))))
```

Figure 1. LISP FUNCTION FOR POGEN

The procedure can be used to control the coherence and sense of a text as well as to provide syntactic well-formedness. The last two lines of Keats' "Ode to a Grecian Urn" are:

"Beauty is truth, truth beauty,
 That's all ye know on earth,
  and all ye need to know."

To obtain semantically controlled variations on this verse, we first form substitution classes named with arbitrary variables as following,

```
Beauty is truth, truth beauty,
   X   is   Y,    Y     X,
 That's all    you    know    on earth
   T            P       K        L
 That's all     you   need to                   know
   T            P     N                         K
```

We can then take each substitution class and augment it with items semantically similar to those of the original under the constraint that each new item maintain the syntax and the sense (to whatever extent desirable) of the original passage. One such set follows:

CLASS	ORIGINAL	SUBSTITUTES
X	beauty	life, this, knowledge, wisdom, love,
Y	truth	honor, all, joy, rapture, love,
T	(that's all)	(that's what), (it's all), (it's what)
P	you	Ye, I, we, some,
K	know	have, get, see, sense, meet
L	(on earth)	('til heaven), ('til hell), (for living),
N	(need to)	(want to) (have to) (ought to)

Since classes, X,Y,T,P and K are all repeated in the verse, they are marked with the property REPLACE. Then the phrase structure grammar is formed as follows:

```
  LINE1 = X IS Y, Y X,
  LINE2 = T P K L,
  LINE3 = T P N K,
  VERSE = L1 L2 L3
```

The contents of the substitution classes are the terminal elements of the grammar. These data are set up as property list structures and

POGEN is then called to generate a set of verses. The following results from this grammar:

```
(VARIATIONS ON TRUTH IS BEAUTY FROM KEATS)

(LOVE IS JOY , JOY LOVE)
(IT'S WHAT WE SEE FOR LIVING)
(IT'S WHAT WE HAVE TO SEE)

(KNOWLEDGE IS JOY , JOY KNOWLEDGE)
(IT'S ALL YOU SEE 'TIL HEAVEN)
(IT'S ALL YOU HAVE TO SEE)

(KNOWLEDGE IS HONOR , HONOR KNOWLEDGE)
(IT'S WHAT YE HAVE FOR LIVING)
(IT'S WHAT YE NEED TO HAVE)

(LOVE IS JOY , JOY LOVE)
(IT'S WHAT YE KNOW 'TIL HEAVEN)
(IT'S WHAT YE NEED TO KNOW)

(LOVE IS TRUTH , TRUTH LOVE)
(THAT'S WHAT WE GET ON EARTH)
(THAT'S WHAT WE NEED TO GET)

(WISDOM IS JOY , JOY WISDOM)
(IT'S WHAT WE MEET 'TIL HEAVEN)
(IT'S WHAT WE OUGHT TO MEET)

(WISDOM IS TRUTH , TRUTH WISDOM)
(THAT'S WHAT I HAVE ON EARTH)
(THAT'S WHAT I WANT TO HAVE)

(KNOWLEDGE IS TRUTH , TRUTH KNOWLEDGE)
(THAT'S ALL YE MEET ON EARTH)
(THAT'S ALL YE HAVE TO MEET)

(BEAUTY IS HONOR , HONOR BEAUTY)
(THAT'S ALL YE SEE ON EARTH)
(THAT'S ALL YE NEED TO SEE)

(LIFE IS TRUTH , TRUTH LIFE)
(THAT'S WHAT SOME SENSE ON EARTH)
(THAT'S WHAT SOME OUGHT TO SENSE)
```

The technique described above is well known in one form or another and it has been used to generate bales of computer verse. With minor changes to the procedure it is possible to control rhyme and meter as well. Judith Merriam (in a seminar assignment) provided one modification to produce the following:

VARIATIONS ON PRUFROCK

THE SULLEN HAZE THAT STREWS HER LUST UPON THE HURRIED STREETS,
THE SULLEN AIR THAT STREWS HER POISON ON THE HURRIED STREETS,
SLIPPED HER HEAD INTO THE CURRENT OF THE TWILIGHT,
SAUNTERED BENEATH THE EVES THAT SHADE DEFEATS,
LET REST AGAINST HER MOUTH THE GRIT THAT RESTS IN STREETS,
PROWLED BY THE FOUNTAIN, LOOSED A MOCKING CRY,
AND SENSING THAT IT WAS SHRILL CICADA NIGHT,
SQUATTED LOW ALONG THE WALKS AND PACED THE SKY.

3.0 FROM SENTENCE TO NETWORK TO SENTENCE

Pogen is designed to accept a context-free phrase structure grammar and follow its flow of control to generate its terminal elements. Unfortunately, a pure context-free grammar is of little use for analyzing ordinary English text into underlying structures . Our goal in modelling macrostructures of discourse is to define methods for representing texts as structures suitable for providing summaries, have developed independently with experimental study of semantic networks. The arcs are binary semantic relations that make explicit the ideas of agent or actor, theme or object of an act, instrument of an act, location, time, frequency, duration etc. that are encoded in English mainly by morphology and word order (in German augmented by case endings).

The importance of semantic networks -- or the equivalent linear form S2, called case relations -- is to provide a canonical representation of a sentence meaning in terms of explicit semantic relations between pairs of unambiguous conceptual elements. A given case relation may have several equivalent surface representations. In contrast to Schank's conceptual dependencies formulated at a level of primitive constructs, the case relations use word-senses as primitive concepts. This approach proves convenient for translation from surface to canonical structure and back to surface form and also offers promise for translation between languages by providing for the mapping of sense meanings onto variant character strings depending on choice of target language. (The question of what conceptual depth is most desirable for what purposes, however, is still open.)

To obtain the structure S2 from the example sentence an all-paths parser -- a version of the Cocke-Kasami-Younger algorithm described by Pratt (1975) -- applies a grammar whose rules are of the following form:

```
(PHRASENAME (LIST OF SYNTACTIC CONSTITUENTS)
```

retrieval capabilities and translations. For this purpose we must consider more complicated grammatical forms.

A twenty year preoccupation of natural language researchers has been to develop useful natural language question answering systems. This led, among other approaches, to the development of quantified semantic networks as a useful method for classifying and representing factual data for sentence content or for data bases. A sentence such as "Rufolo loaded his ship and sailed to Cyprus" can be represented as follows:

```
S2)    (LOAD1 TENSE PAST AGT RUFOLO TH(SHIP1 OWNBY RUFOLO)*AND SAIL1)
       (SAIL1 TENSE PAST AGT RUFOLO TH SHIP1 *TO CYPRUS1)
```

This can also be shown as the following graph:

```
LOAD1----TENSE---PAST
  |
  |------AGENT---RUFOLO1------|--------|
  |                           |        |
  |-------TH-----SHIP1-----OWNBY       |
  |                 |                  |
*AND-----SAIL1----TH                   |
           |                           |
           |------AGENT----------------|
           |
           |------*TO----CYPRUS1
           |
           |---TENSE-----PAST
```

The arcs, AGent, THeme, INSTRUMENT, *FROM, *TO, *AND, and others* are adapted originally from Fillmore's (1968) theory of case structure but

----
* The * prefix distinguishes case relations such as *TO, indicating a goal relation, from the word, to.

```
(CORRESPONDING LIST OF SEMANTIC CONSTRAINTS)
(TRANSFORMATION)     )
```

Example:

```
(S(NP VP)(ANIM ACT)(2 AGT 1))
```

This rule applies to two constituents, an NP followed by a VP. If the NP has the semantic feature ANIMate and the VP has the feature, ACT, then a new constituent is formed by combining the VP (i.e. the 2nd constituent) and the NP (the 1st) connected by the semantic relation, AGT, and the new constituent is labelled S. Thus if the two constituents appear as follows:

```
NP = (RUFOLO ANIM T SING T)
VP = (SAIL PAST T ACT T ... )
```

the new constituent S is formed,

```
S = (SAIL PAST T ACT T ... AGT (RUFOLO ANIM T SING T))
```

The symbol "..." indicates additional data in the constituent that is not relevant to the example.

Lexical rules follow the same form so,

```
(N(SHIP) T (1 SING T POBJ T ))
(N(CYPRUS) T (1 SING T PLACE T NAME T))
(V(SHIP) T (1 3PS T ACT T ))
(N(RUFOLO) T (1 SING T ANIM T NAME T))
(V(SAILED) T (SAIL PAST T ACT T))
(PREP(TO) T (1))
(POSSPRON(HIS) T (1 MASC T SING T POSS T))
```

The example lexicon accounts for the words of the sentence "Rufolo sailed his ship to Cyprus" and the following grammar rules apply:

```
(NP(N) T (1))

(VP(V) T (1))

(VP(VP NP) (ACT POBJ) (1 TH 2))

(VP(VP PP) (ACT (PLACE PREP TO)) (1 *TO 2))

(PP(PREP NP) T(2 PREP 1))

(NP(POSSPRON NP) (OK POBJ) (2 OWNBY 1))

(S(NP VP)(ANIM ACT)(2 AGT 1))
```

It can be noticed that the lexical rules contain semantic features which characterize the particular word sense, and that these features are included as the constituent is embedded in higher level constituents.

The complete result of the parse for "Rufolo sailed his ship to Cyprus" is as follows:

```
S3)   (SAIL PAST T ACT T

           TH (SHIP SING T POBJ T OWNBY (HIS SING T MASC T

                                                 POSS T))

           *TO (CYPRUS SING T PLACE T NAME T PREP TO)

           AGT (RUFOLO SING T ANIM T NAME T))
```

The feature notation, SING T, ANIM T, etc. is analogous to the linguistic feature markers, + and - where T is + and F is -. After parsing, the words of the structure are assigned subscripts and the semantic features may be discarded -- unless needed for some additional linguistic processing as for the case of translation to another natural language.

Discarding the semantic features in this example the resulting semantic relation is:

```
S4)  (SAIL PAST T TH(SHIP SING T OWNBY(HIS SING T))
                  AGT (RUFOLO SING T)
                 *TO(CYPRUS SING T PREP TO))
```

This form is suitable for conversion to a knowledge base entry by subscripting the words to give SAIL1, SHIP1, etc., and fitting it into the discourse network by discovering preceding referents for definite noun phrases and pronouns. It is appropriate to mention that this process of finding referents is still only partially understood and requires a human editor to assist the programs.

Once in the knowledge base the concepts of the semantic relation are augmented by the network of general knowledge. Thus RUFOLO is a MAN, SHIP is a VESSEL and SAIL implies MOVE; this particular SAILing followed a LOADing of the SHIP, etc. The relation of a word to the classification network is available through the lexical entry, and this serves as the primary basis for finding answers to questions. The process of answering questions is described elsewhere, (Simmons and Chester 1977). This semantic representation system is easily able to encode causal and coherency relations among sentences, but a single sentence parser cannot easily compute them.

Let us now suppose that the above relation, S4, has been retrieved as an answer to some question such as "Where did Rufolo Go?" We now wish to transform S4 into English. A generation procedure resembling Pogen but using a grammar sensitive to the case markers accomplishes the task.

One convenient form for the generation grammar can be seen below.

```
S = AGT:NP + VP                  VP = VSTRNG + COMPL
NP = DET:NP + NP                 VPAS = BE + VP
NP = MOD:NP + NP                 VSTRNG = AUX + VSTRNG
NP = OWNBY:POSSPRON + NP         VSTRNG = *
NP = PREP:PP + NP                AUX = PAST
NP = *                           AUX = *
PP = PREP + NP                   COMPL = TH:NP + COMPL
PREP = *                         COMPL = INSTR:PP + COMPL
S = INSTR:NP + VP                COMPL = *TO:PP + COMPL
S = TH:NP + VPAS                 COMPL = #
```

In this grammar the three symbols, :,*, and # have the following effect on the generation algorithm:

ARCNAME : PHRASENAME	indicates that the phrase is to be generated from this relation only if the arc name is present
*	indicates that the head of the relation is to be realized as an English word,
#	nothing need be generated; used to terminate a sequence of complements or PPs.

Applying the above grammar to the relation,

(SAIL1 PAST T TH(SHIP1 OWNBY RUFOLO) AGT(RUFOLO SING T)

*TO (CPRUS1 SING T))

the following steps result:

S = AGT:NP + VP	
NP = *	RUFOLO
VP = VSTRNG + COMPL	
VSTRNG = AUX + VSTRNG	
AUX = PAST	PAST
VSTRNG = *	SAIL
COMPL = TH:NP + COMPL	
NP = OWNBY:POSSPRON	
POSSPRON = HIS	HIS
NP = *	SHIP
COMPL = *TO:PP + COMPL	
PP = PREP + NP	
PREP = *	TO
NP = *	CYPRUS
COMPL = #	

When a word is to be generated, morphological routines are called to examine the SING/PLURAL markers, decide on the desirability of constructing a pronoun or definite noun phrase, and to realize the English words. The result is shown in the rightmost column, "RUFOLO PAST SAIL HIS SHIP TO CYPRUS." A last pass on the sentence tenses the verb and adjusts agreements of subject, verb etc.

It should be noticed that neither the analyzer nor the generator constructs a derivation tree. Instead, as in Pogen, the tree of derivation from the grammar is used only to direct the flow of control through the procedure. If the syntactic tree were needed for some purpose, a minor change in these procedures would record it.

Although the grammar shown above is a sufficient form for transforming semantic relations into single sentences, it offers no provision for constructing connected paragraphs to form an organized text. Nor does the recognition grammar provide capability beyond the single sentence level; it too cannot develop the organization of sentences in a paragraph or discourse. This limitation is one of the reasons that it cannot always find referents for pronouns or definite noun phrases, and in this form it does not record from sentence to sentence even what is being talked about.

A grammar relating sentences to each other must provide the connective tissue between the propositional elements of the text in order to remove these limitations of sentence analysis. It is our belief that if we can find a satisfactory structure for story trees we will be able to write text grammars in the 4-tuple form described above that can be used to analyze text organization as well as sentence structure.

4.0 STORY TREES

Limitations of the single sentence grammar approach shown in the preceding section reveal much of the motivation underlying the invention of such procedural notions as scripts, implicational molecules, plans etc. (Schank and Abelson 1977) partitions, focus spaces etc. (Hendrix 1976) Grosz (1977) frames (Minsky 1975, Bobrow and Winograd 1977, and Charniak and Wilks 1976). The latter book also describes a rulebased approach of templates and paraplates by Wilks. Our own bias is toward a rule-based approach to the recognition and generation of English discourse and we have sought for systems of rules that can more formally describe the organizations imposed by scripts, frames etc.

Story grammars provide one basis for the approach. Generally, a Story can be analyzed into a Title and a Setting followed by Episodes, and an Episode into Actions and Results; and this analysis can be recorded as a phrase structure grammar as shown in Figure 2.

```
--------------------------------------------------------------
THE HOLY GRAIL                TITLE-----------------------------|
ONCE LONG AGO                 TIME---|                          |
IN CAMELOT                    PLACE--|                          |
PARSIFAL WAS A KNIGHT                |                          |
  OF THE ROUND TABLE          CHAR---|                          |
HE WAS A DEVOUT CHRISTIAN     MOTIVE-|-----------------SETTING--|
AND SOUGHT THE HOLY GRAIL     ACT--|                            |
HE WANDERED THE WORLD         ACT--|--ACT--|                    |
BUT COULD NOT FIND IT         RESULT-------|-EPISODE--|-EPISODE-|
AFTER YEARS OF SEEKING        ACT-----|               |         |
HE FOUND IT IN HIS HEART      RESULT--|--EPISODE------|      STORY

===========================================

STORY = TITLE + SETTING + EPISODE
TITLE = THE HOLY GRAIL
SETTING = TIME + PLACE + CHARACTERS + MOTIVE
EPISODE = ACT + RESULT
EPISODE = EPISODE + EPISODE
ACT = ACT + ACT
ACT = PROPOSITION
RESULT = PROPOSITION

=========================================
```

TIME, PLACE, CHARACTERS, and MOTIVE are PROPOSITIONS. A PROPOSITION is an English phrase or sentence and thus a terminal element in the grammar.

Figure 2. A Story and its Grammar

About all that this grammar shows, however, is that it is possible to organize the events of a story into SETTING, ACTS, RESULTS and EPISODES. What has been gained is the conception that phrases and sentences at the story level can be "syntactically" classified in a manner analogous to words and phrases at the sentence level of analysis. But the resulting tree appears to be lacking in the critical property required for the Kintsch-Van Dijk macrostructure that the nodes closest to the root should form a summary. So not just any grammar for a story will form a story tree, i.e. a tree with the desired properties.

```
--------------------------------------------------------------------------
1 STORY ----- 2  TITLE ------------------------------- 3 THE HOLY GRAIL
    |
    |------- 2 SETTING ----- 3 CHAR ----------- 4 PARSEFAL WAS A KNIGHT
    |              |              |
    |              |              |---- 4 MOD ----- 5 OF THE ROUND TABLE
    |              |
    |              |---- 3 CIRCUM ----- 4 TIME ------------- 5 LONG AGO
    |                        |
    |                        |---- 4 PLACE --------------- 5 IN CAMELOT
    |                        |
    |                        |---- 4 ATTRIBS -------- 5 HE WAS A DEVOUT
    |                                                        CHRISTIAN
    |
    |------- 2 EPISODE ----- 3 ACT --------- 4 HE SOUGHT THE HOLY GRAIL
                 |               |
                 |             4 ACT------------ 5 HE WANDERED THE WORLD
                 |
                 |--- 3 RESULT ---------------- 4 BUT COULDN'T FIND IT
                        |
                      2 EPISODE-- 3 ACT------- 4 AFTER YEARS OF SEEKING
                          |
                          |---- 3 RESULT---- 4 HE FOUND IT IN HIS HEART
                                     |
                                     |------------------- 4 THE END

==================================================
STORY = TITLE + SETTING + EPISODE
TITLE = P
SETTING = CHAR + CIRCUM
CHAR = P + MOD
MOD = P
CIRCUM = TIME + PLACE + ATTRIBS
EPISODE = ACT + RESULT
ACT = P + ACT
ACT = P
RESULT = P + EPISODE
RESULT = P + THE END
==================================================
NOTES:
P means proposition, and TIME, PLACE, and ATTRIBS are rewritten
as P's.
Numbers on the tree are depth counters, the shortest summary can
be seen by reading the propositions marked 4 or less. The entire
story emerges at level 5.

--------------------------------------------------------------------------
```

Figure 3. A Story-Tree and its Grammar

Figure 3 shows a story tree analysis and its grammar. The nodes of this tree are numbered to show their depth; when an episode is the rightmost member of a result rule, its level is assigned the same number as its ancestral episode. This feature insures that a sequence of episodes forms what is effectively a shallow tree despite the deeply nested form of the grammar. If we read the propositions whose numbers are not greater than 3 from the tree, the only candidate is the title, THE HOLY GRAIL. At level not greater than 4, we read; THE HOLY GRAIL: PARSEFAL WAS A KNIGHT, HE SOUGHT THE HOLY GRAIL, HE COULDN'T FIND IT, AFTER YEARS OF SEEKING HE FOUND IT IN HIS HEART, THE END. At level 5, the whole story can be read:

THE HOLY GRAIL: PARSEFAL WAS KNIGHT, OF THE ROUND TABLE,
LONG AGO, IN CAMELOT, HE WAS A DEVOUT CHRISTIAN, HE SOUGHT
THE HOLY GRAIL, HE WANDERED THE WORLD, HE COULDN'T FIND IT,
AFTER YEARS OF SEEKING, HE FOUND IT IN HIS HEART, THE END.

Figures 2 and 3 show that if we want a story tree with the property that its higher nodes produce a summary, we must design a grammar that will place certain propositions at higher nodes in the tree than others. A better approach that will be developed in Section 6 is to include transformations at each node in the tree in such a manner that each node will provide a summary of the complex structure that lies below. These example story grammars also reveal the prominent weakness of the context free phrase structure approach; it is suitable for generating a sequence of propositions with such a system as Pogen, but it does not provide the semantic conditions or transformations required to derive a story tree from a sequence of propositions.

5.0 COMPUTATIONAL ASPECTS OF SEMANTIC NETWORKS AND STORY TREES

We believe Meehan's (1976) generation of stories about the interactions of the motives of talking animals may prove to be an excellent model for the human story writing process. In his view a story is the description of the events that occurred when a character attempted to achieve a goal or solve a problem. If we look at his work as an abstract problem solving program, we can see that his approach was based on the use of a set of objects, i.e. bears, canaries, worms, trees, etc. each characterized as appropriate by conceptual dependency assertions of motives, personality characteristics, locations, etc. and a set of rules in the form of plans, that could transform properties of these objects. The rules were organized in plans and planboxes for easy access by appropriate goals. The action of a story developed as an exploration of paths through the rule tree as a character sought to satisfy a goal. The system for generating the story is a problem solver specialized to the use of indexed plans and conceptual dependency structures.

One significant aspect of Meehan's study was that although his system was based on conceptual dependency networks, he found no necessity to draw graphs or deal explicitly with the network in his exposition. His system of plans and planboxes was adequately described in propositional form; his network was implicit in the chaining of subgoals leading to the accomplishment or failure of a plan. The interrelations among concepts, e.g. "bears like honey", "bears live in caves", etc. were similarly implicit in such common coreferential terms as "bear". In fact, the conceptual dependency system is an explicitly indexed network.

In previous papers (Simmons 1977, Simmons and Chester 1977) semantic networks are defined as sets of nodes that represent verbal concepts, connected by arcs that represent semantic case relations between pairs of concepts. A single semantic relation is a Node-Arc-Node triple. An English sentence or proposition is represented by a head node and a set of arc-node pairs. We call this a case predicate and usually write it as an n-tuple,

(HD ARC1 VALUE1 ARC2 VALUE2 ...ARCn VALUEn)

where the values refer to other n-tuples, i.e. nodes. One natural computational form for this structure is an atom associated with a set of incoming arcs and a set of outgoing arcs. If we consider these n-tuples as relations where the head is a predicate name and each arc identifies an argument by name then it is apparent that in comparing two relations, each arc has a value which is a pointer to the location of the value of a relation's argument and no searching of the data base is required to find that value. The naming of arguments also allows the case predicate to list its arguments in any order and to represent only the arguments that are specified. Thus the named arcs in a semantic network serve two important purposes; first, they provide indexing for directly accessing data, and second, they relax the constraint that a relation be represented as a strict n-tuple with an explicit position for every argument. Both of these properties are of significant value for dealing with large amounts of data derived from natural language statements.

But an explicit semantic network is not the only way to represent the interrelations among case predicates; a simple list of the predicates provides exactly the information needed to compile the actual semantic network. We think of such a list as one that implies a network, i.e. as a virtual network, and have found it to be most convenient for representing rules containing free variables which in explicit representation could require infinite storage and computation. The connectivity of the virtual network is signified by the correspondence of values of arguments with names of other case predicates. This correspondence is discovered by searching the list of case predicates to find one that matches a given value. In other words, we can trade the memory required to represent the connectivity of the network for an increased amount of computation to explicate it by searching through the case predicates. We can also delete the case names and impose the constraint that every argument position be represented in the n-tuple thus transforming our semantic case predicates to ordinary logical relations of the form,

(HD ARG1 ARG2... ARGn)

whose elements are English terms (i.e. words and variable names). The connectivity of the virtual network remains the same and other indexing schemes can be provided to minimize the amount of computation required to explicate the connective properties.

In subsequent sections of this chapter we will use the strict n-tuples of English terms to represent the propositions and rules used in computing story trees. Excepting only the retrieval and direct matching steps, the computational procedures are the same as would be required for explicit semantic network representations. We sacrifice some of the indexing properties of the network with the consequence of increased computation time and decreased memory requirements. What we hope to gain is easier readability for the rules and propositions, a more direct correspondence with a system of logic, and a simpler exposition of the essential steps in generating story trees.

From Meehan we understand that problem solving is an appropriate basis for developing a story, but we have been interested in a method different from the use of conceptual dependency plans and planboxes. Kowalski in his monograph, "Logic for Problem Solving" (1974) presents an attractive approach that allows the statement of problems, rules and assertions as ordered sets of propositions. Although he interprets his logic in terms of resolution theory, various procedures are available to achieve flexible and efficient problem solving. Further relations of this system to semantic networks are developed in a later paper by Deliyanni and Kowalski (1977). Using Kowalski's logic with strict n-tuple English notation we are able to define sets of propositions and inference rules to describe a virtual network that enables us to develop story trees and compute stories in a very general fashion.

The Kowalski logic for problem solving is based on Horn clauses. A Horn clause is a set of logical predicates

B0,...Bm<--A0...An,   where m and n are equal or greater than zero.

Kowalski distinguishes four types of Horn Clauses:

1 m=0, n=0 is the <u>null clause</u>

2 m=1, n=0, B<-- a Horn clause with no antecedents
is an <u>assertion</u>

3 m=0, n not=0, <--A1,...An A Horn clause with no
consequent is a <u>Goal statement</u>

4 m=1, n not=0, B<-- A1,...An Every other Horn clause
with a single element on the left
is an <u>operator</u>

The clauses Bi...Bm can be called Consequents, and Ai...An, antecedents. If some clause, Bi...Bm <-- Ai...An contains variables such as x, y, z, then the clause should be read:

For all x, y, z, Bi...Bm is True if Ai...An is True.

A sentence, Ai...An or Bi...Bm is read as the conjunction of its clauses Ai and Aj and Ak etc.

Kowalski gives a complete development of this system as a resolution based logic and he shows its application to solving problems.

For example, a Horn clause procedure for doing sums in successor arithmetic is as follows:

```
    (SUM 0 X X)<--
    (SUM (SUC X) Y Z)<--(SUM X (SUC Y) Z)
    <--(SUM (SUC(SUC 0)) (SUC 0) U)
```

The first clause says that the sum of zero and any y is y. This is an <u>assertion</u>. The second statement is an <u>operator</u>; it states that the sum of the successor of any x and any y is z if the sum of x and the successor of that y is z. The third clause states the <u>goal</u>; find the sum of the successor of the successor of 0 and the successor of 0 and its value will be the value of u.

The goal statement is matched to the elements that have an arrow to their right, i.e. consequents; it fails to match the assertion but it matches the operator. The values of argument positions in the goal are bound to those in the consequent and through into the antecedent. The consequent is detached and the antecedent is taken as a new goal until finally the goal is

```
  <--(SUM 0 (SUC(SUC(SUC 0))) Z)
```

which matches the assertion. The variable Z at that point takes on the value (SUC(SUC(SUC 0))) or 3. This brief outline of the problem solving use of Horn clauses is sufficient only if if the reader is acquainted with the use of such logical systems . Kowalski's description and exploration is recommended for a more complete and clear explanation.

Kowalski´s logic is of interest here to show some relationships between problem solving and story generation. The basic method that Meehan used for story generation could be illustrated in this formalism but instead of using his complex world of motivated animals, let us use the familiar MIT robot and block world.

Suppose we command Winograd´s (1972) robot,
"PUT THE GREEN PYRAMID ON THE RED BLOCK".
This translates into a parenthesized Horn clause as follows:

```
a)  <--(PUT ROB GP ON RB)
```

Variables are recognized by being members of a special list, "S T U V W X Y Z" and other terms are constants. The robot´s problem solving system is encoded as follows:

```
b)  (PUT ROB X ON Y)<--(Y IS CLEAR)  (ROB PUTS X ON Y)
c)  (Y IS CLEAR)<--(Z IS ON Y)(ROB REMOVES Z FROM Y)
          (ROB PUTS Z ON TABLE)
d)  (ROB REMOVES X FROM Y)<--(ROB PICKS UP X)(A*(Y IS CLEAR))
e)  (ROB PUTS U ON W)<--(ROB HOLDS U)(W IS CLEAR)
          (A*(U IS ON W))
f)  (ROB PICKS UP Z)<--(Z IS CLEAR)(A*(ROB HOLDS Z))
g)  (ROB HOLDS X)<--(ROB PICKS UP X)
h)  (RB IS ON TABLE)<--
i)  (GB IS ON RB)<--
j)  (GB IS CLEAR)<--
k)  (GP IS ON TABLE)<--
```

l) (GP IS CLEAR)<--

m) (TABLE IS CLEAR)<--

The symbols GP, GB, RB, and ROB stand for GREEN PYRAMID, GREEN BLOCK, RED BLOCK and ROBOT, respectively. Symbol A* is the name of the function ASSERT. When (X ON Y) is asserted, any (Z ON Y) where Z not equal X, and Y not equal TABLE are deleted. Similarly for (Y IS CLEAR). The initial assertions that the GREEN BLOCK IS ON THE RED BLOCK, THE RED BLOCK IS ON THE TABLE, THE GREEN PYRAMID IS ON THE TABLE etc. are shown as h through m. The robot accepts the command, (PUT U GP ON RB) and proves it in the following steps:

1) Command (PUT U GP ON RB) matches the consequent of b,
   so (PUT ROB GP ON RB) is true if (RB IS CLEAR) and
   (ROB PUTS GP ON RB).

2) (RB IS CLEAR) matches c.

3) c is true if (Z IS ON RB)(ROB REMOVES Z FROM RB) and
   (ROB PUTS Z ON TABLE)

4) (Z IS ON RB) matches assertion i,(GB IS ON RB) so

5) (ROB REMOVES GB FROM RB) is tried and found to match d

6) which is true if (ROB PICKS UP GB) which matches f
   and is true if 7 and 9,

7) (GB IS CLEAR)

8) (GB IS CLEAR) by j, so

9) (ROB HOLDS GB) is asserted by f.

10) Statement 5 asserts (RB IS CLEAR) and is now true.

11) Statement 3 still requires that (ROB PUTS GB ON TABLE)

12) which matches e, whose antecedents require,

13) (ROB HOLDS GB), true by 9,

14) (TABLE IS CLEAR), true by m,

15) and asserts (GB IS ON TABLE).

16) The remaining antecedent of statement 1 is (ROB PUTS GB ON RB)

17) which matches e, whose antecedents require
(ROB HOLDS GP)(RB IS CLEAR) and the assertion (GP IS ON RB);

18) (ROB HOLDS GP) matches g, which is true if

19) (ROB PICKS UP GP) which matches f, and is true since its
antecedents,

20) (GP IS CLEAR) is true by 1, and

21) (ROB HOLDS GP) is then true by assertion.

22) Statement 17 still requires (RB IS CLEAR) which is true by
the assertion in 10, and asserts

23) (GP IS ON RB) thus completing the procedure.

The proof tree for establishing the goal, "PUT ROB GP ON RB" is shown in Figure 4 as the tree, BLOKSTAK.

```
                 BLOKSTAK
             /             \
           /                  \
     SETTING                  GOAL
       |                       |
*GP IS CLEAR                   |
*TABLE IS CLEAR                |
*GB IS ON RB            PUT ROB GP ON RB
*RB IS ON TABLE            /        \
*GP IS ON TABLE           /         (1)
*GB IS CLEAR       RB IS CLEAR -------------------|
                     /       \                    |
         *GB IS ON RB   ROB REMOVES     ROB PUTS GB ON TABLE
                        GB FROM RB           /           \
                         /       \      *ROB HOLDS GB    *GB IS ON TABLE
          ROB PICKS UP GB    *RB IS CLEAR
            /        \
   *GB IS CLEAR   *ROB HOLDS GB              (1)

                                          ROB PUTS GP ON RB
                                           /               \
                                  ROB PICKS UP GP        *GP IS ON RB
                                    /          \
                            *GP IS CLEAR    *ROB HOLDS GP
```

Figure 4. The Derivation Tree for BLOKSTAK

Assertions are marked with the symbol, *, while procedural clauses are left unmarked. The nodes, Setting and Goal are simply added as labels to suggest a story tree. Actually, the Setting shows only the initial assertions; for some other purpose it might include the whole inference system. Several summaries can be read from BLOKSTAK; the highest of these is the GOAL dominated node which could be stated as, THE ROBOT IS COMMANDED TO PUT THE GP ON THE RB. Next, can be read the combination of two nodes, THE ROBOT CLEARS THE RB THEN PUTS THE GP ON THE RB. The entire story in terms primarily of changes in world states is readable from the terminal leaves. Winograd took advantage of this hierarchical ordering to program his robot's reporting at successively deeper levels of what was done and why. A similar proof tree underlies the organization of each of Meehan's stories. A related approach is shown by Chester (1976) for translating from trees representing mathematical proofs to English paragraphs. And it appears that some variation on this proof tree is what we are striving to compute when we attempt to understand a text.

An English description of what the robot did might take the following form:

1. Robot removed the green block from the red block.

2. He put the green block on the table,

3. picked up the green pyramid

4. and put it on the red block.

These sentences suggest the scope of the text understanding problem. From about twenty nodes on the derivation tree, four are selected to describe the series of events. The underlying motivation, that the robot wants to put the GP on the RB is not explicitly stated. As readers, our knowledge of the Blocks World gives us a context and the still inadequate theory of story trees gives us a basis for approaching the problem. But until we can develop a theory of story or text grammars that approaches the completeness of our understanding of sentence structures and proof trees we can only make informed guesses at what the underlying derivation tree might be for a given text.

Knowing the derivation tree for this example and following the constraints of story grammars we can construct the following story tree:

```
                        BLOKSTAK
                         /      \
         |---------------|       |----------|
   SETTING                                 EPISODE
      |                                   /      \
GB ON RB             (rob, put gb on rb) ACT1   RESULT (gb on rb)
RB ON TABLE                           /      \
GP ON TABLE         (clear rb)   EPISODE      ACT3 (put gp on rb)
GP IS CLEAR                     /     \         \
                (remove gb  ACT2        RESULT   (1)
                from rb)     /  \           \
                           /      \       (rb is clear)
                   *PICK UP GB   *PUT GB
                                  ON TABLE
                                             (1)
                                           /     \
                                (get gp) ACT4   *PUT GP ON RB
                                         /
                                   *PICK UP GP
```

Figure 5. A Story Tree for BLOKSTAK

ACT1 corresponds to the derivation tree's top goal, ACT2 to the goal of clearing RB, ACT3 to the goal of putting the GP on the RB. The parenthesized lower case expressions show what might be expressed as a description at each node in the tree and suggest the use of special comments for constructing summaries. The next section will show how useful rule systems can be prepared to generate story trees and summaries.

6.0 SOME RULES FOR STORY TREES

A general form for narrative grammars can be described in phrase structure rules as follows:

```
STORY =  SETTING + EPI
SETTING = ACT
EPI = ACT + RESULT
ACT = ACT + ACT
ACT = T +...T
RESULT = TEST + EPI
RESULT = THE  END
```

T+...T stands for a sequence of terminal assertions. Finer subdivisions in the grammar are possible and often desirable as shown previously in Figure 3. The notion of an episode being composed of an act and a result, -- often referred to as an act followed by a complication -- is the main control branch of the story grammar. The result or complication is defined as a test to determine whether the story should continue with additional episodes or terminate at this point.

The phrase structure rules merely show the form of the story tree and reflect the flow of control in its generation. If we are actually to generate story trees from rules we require the capability to test whether a rule applies at a given point and the ability to transfer the values of variables bound in one rule for use in another. We can use a phrase structure organization of Horn clauses to satisfy these requirements and to accomplish the generation.

In this form the first three rules for a story grammar to generate a rather trite story, OLD WEST SAGA, appear as follows:

```
(OLD WEST SAGA)<--(SETTING GG BG L MOT OUTC)
                  (EPI GG BG L MOT OUTC)

(SETTING GG BT L MOT OUTC)<--(THE SHERIF IS GG) (THE BAD GUY
     IS BG) (THE PLACE IS L) (SET MOT (PEACEFUL TOWN))
     (GG LIVES IN L) (BG RIDES INTO L)

(EPI GG BG L MOT OUTC)<--(ACT GG BG L MOT OUTC)
                         (RESULT GG BG L MOT OUTC)
```

The title, OLD WEST SAGA, stands for the node, STORY and it is composed of a setting and an episode each of which is characterized by the variables, GG, BG, L, MOT, OUTC. The setting is composed of a set of clauses-- sentences containing variables. Corresponding to some of these clauses the grammar contains the following assertions, (the implication mark, <--, is deleted for convenience):

```
(THE SHERIFF IS DESTRY)
(THE BAD GUY IS BLACKBART)
(THE PLACE IS DODGECITY)
(SET X X)
```

When the rule for SETTING is evaluated, each of its antecedents is matched against the database of rules and assertions. When (THE SHERIFF IS GG) is found to match (THE SHERIFF IS DESTRY), the variable GG is bound to DESTRY for subsequent references within the rule. The assertion (SET X X) which assigns a value to a variable matches (SET MOT (PEACEFUL TOWN)), so MOTivation is bound to (PEACEFUL TOWN). The result of evaluating the rule, (SETTING...) gives the following:

```
(SETTING DESTRY BLACKBART DODGECITY (PEACEFUL TOWN) OUTC)
    (THE SHERIFF IS DESTRY)(THE BAD GUY IS BLACKBART)
    (THE PLACE IS DODGECITY)(SET (PEACEFUL TOWN)(PEACEFUL TOWN))
    (DESTRY LIVES IN DODGECITY)(BLACKBART RIDES INTO DODGECITY)
```

This evaluation was a substep in evaluating the first rule, (OLD WEST SAGA) and its values are bound back up into that rule as follows:

```
(OLD WEST SAGA) (SETTING DESTRY BLACKBART DODGECITY
                      (PEACEFUL TOWN) OUTC)
          (EPI DESTRY BLACKBART DODGECITY (PEACEFUL TOWN) OUTC)
```

So the values of the variables have been established by the SETTING rule and communicated through the antecedents of (OLD WEST SAGA) into the EPI clause. The EPI clause is now evaluated and discovered to match the EPI rule, so the variables are bound through the EPI rule and it appears as follows:

```
(EPI DESTRY BLACKBART DODGECITY (PEACEFUL TOWN) OUTC)
      (ACT DESTRY BLACKBART DODGECITY (PEACEFUL TOWN) OUTC)
      (RESULT DESTRY BLACKBART DODGECITY (PEACEFUL TOWN) OUTC)
```

The ACT antecedent is now to be evaluated. In the database the following rules are found to match the ACT antecedent:

```
     (ACT GG BG L (PEACEFUL TOWN) (BG ROBS THE BANK))
     (ACT GG BG L MOT ( PEACEFUL TOWN))
          (GG DISCOVERS BANKROBBERY) (FIND GG BG)
```

These rules are in the above ordering and both are found to fit the ACT antecedent under the conditions that a variable matches a variable or a constant and equal strings match. So DESTRY=GG, BLACKBART=BG, DODGECITY=L, in both rules; (PEACEFUL TOWN)=(PEACEFUL TOWN) and

OUTC=(BG ROBS THE BANK) in the first rule; and (PEACEFUL TOWN)=MOT and OUTC=(PEACEFUL TOWN) in the second. The first rule is selected. This rule is an assertion with the constant outcome that (BG ROBS THE BANK) so it is true and its OUTC value is bound back into the ACT above it and the RESULT rule is now evaluated with its OUTC=(BG ROBS THE BANK). In the database there are two result rules as follows:

```
(RESULT GG BG L MOT OUTC)<--(EQ* MOT OUTC) (THE END)

(RESULT GG BG L MOT OUTC)<--(NEQ* MOT OUTC)
                            (EPI GG BG L MOT MOT)
```

Both are retrieved and the first one is tried. EQ* and NEQ* are functions for testing equality and nonequality, respectively. As a result of the last ACT, MOT=PEACEFUL TOWN and OUTC=(BLACKBART ROBS THE BANK). Since MOT is not equal OUTC, the first RESULT rule fails and the second succeeds, causing a new episode whose outcome is bound to the value of MOT, i.e. (PEACEFUL TOWN). And so the story will continue until the value of the MOT and OUTC arguments match at which point, (THE END) will be asserted.

The interpreter that evaluates these Horn clauses has two modes, QUERY and ASSERT. In QUERY mode it is a problem solver that must show that all antecedents of a rule are eventually derivable from assertions in the data base. In this mode it can be used to solve the Block's world problem shown earlier. In ASSERT mode it is a tree generator and if an antecedent is not present in the data base, it can be asserted. Only functions, e.g. EQ* and NEQ*, can cause an antecedent to fail and so reject a rule whose consequent has been matched. The ASSERT mode is used in the above example so such statements as, (DESTRY LIVES IN DODGECITY) and (BLACKBART RIDES INTO DODGECITY) were asserted although not found in the database. In general the ASSERT mode allows an author to place terminal assertions that he wishes to appear in his story directly in the antecedents of the rules that order the events of the story.

At this point we can introduce one last important mechanism, the TF antecedent. This is a transformation statement that can be associated with each node to form a summary of what is happening. The TF antecedent allows the author to formulate such comments as were shown associated with nodes in the Blokstak example of Figure 5. The last antecedent in a statement may be a transformation and its arguments in the form, (TF (   ) (   )...). Let us add a TF antecedent to the SETTING rule.

```
(SETTING GG BG L MOT OUTC)
        ...(TF (SHERIFF GG LIVES IN L A MOT UNTIL 7))
```

After binding, the whole expression appears as follows:

```
(SETTING DESTRY BLACKBART DODGECITY  (PEACEFUL TOWN) OUTC)
   (THE SHERIFF IS DESTRY)(THE BAD GUY IS BLACKBART)
   (THE PLACE IS DODGECITY) (SET(PEACEFUL TOWN)(PEACEFUL TOWN))
   (DESTRY LIVES IN DODGECITY) (BLACKBART RIDES INTO DODGECITY)
   (TF(SHERIFF DESTRY LIVES IN DODGECITY A
                               (PEACEFUL TOWN) UNTIL 7))
```

Notice that the TF expression had variables, GG L and MOT which were bound to values, DESTRY, DODGECITY, and (PEACEFUL TOWN) when the expression was evaluated. It also contained the number 7 which remains. This number refers to the rule's seventh clause, (BLACKBART RIDES INTO DODGECITY). TF is a special interpreter function which will substitute the referenced clause for the number when it is evaluated. The TF expression is treated as an ordinary clause when the rules are evaluated, but when the story tree has been computed, the function (SAY PROOF N) evaluates the TF functions at level N in the tree to form a summary or complete story.*

A complete story grammar for OLD WEST SAGA is shown in Appendix A. The function RAND* is used in one rule; it returns True or Nil with a .5 probability to allow the author to include variations in his story. A story and its summaries derived from this grammar are shown in Figure 6. The grammar provides a second story which is shown with its summaries in Figure 7. The complete story tree corresponding to the story in Figure 7 is included in Appendix A. A careful study of the grammar and its corresponding story tree will show the style of programming that is required to compute stories.

----

* Programmed by M. Kavanagh Smith.

```
(STORY OLD WEST SAGA)
*VALUE:
(OLD WEST SAGA (IN (THE DAYS OF THE OLD WEST) (THE SHERIFF IS DESTRY)
AND (THE BAD GUY IS BLACKBART) ; (DESTRY LIVES IN DODGECITY) HAPPILY
ENOUGH UNTIL (BLACKBART RIDES INTO TOWN)) (BLACKBART RIDES INTO
DODGECITY AND (BLACKBART ROBS THE BANK) (DESTRY DISCOVERS THE HEIST
AND (DESTRY SWINGS INTO THE SADDLE AND (GALLOPS AFTER BLACKBART) (
BLACKBART HIDES IN GREAT ROCK CANYON) (DESTRY RIDES INTO GREAT ROCK
CANYON) (DESTRY HEARS THE WHINNY OF A HORSE) (DESTRY DISMOUNTS AND
TRACKS BLACKBART) (DESTRY GETS THE DROP ON BLACKBART) THEN (BLACKBART
SURRENDERS) SO (DESTRY SHOOTS HIM) AND THERE IS A (PEACEFUL TOWN)) (
THE END))))
*TIME: 5821
(SAY PROOF 2)
*VALUE:
(OLD WEST SAGA (DESTRY KEEPS A (PEACEFUL TOWN) UNTIL BLACKBART ARRIVES
) (BLACKBART ROBS THE BANK (DESTRY HUNTS BLACKBART (THE END))))
*TIME: 265
(SAY PROOF 3)
*VALUE:
(OLD WEST SAGA (IN (THE DAYS OF THE OLD WEST) (THE SHERIFF IS DESTRY)
AND (THE BAD GUY IS BLACKBART) ; (DESTRY LIVES IN DODGECITY) HAPPILY
ENOUGH UNTIL (BLACKBART RIDES INTO TOWN)) (BLACKBART RIDES INTO
DODGECITY AND (BLACKBART ROBS THE BANK) (DESTRY DISCOVERS THE HEIST
AND (DESTRY SHOOTS BLACKBART TO MAKE A (PEACEFUL TOWN)) (THE END))))
*TIME: 329
(SAY PROOF 4)
*VALUE:
(OLD WEST SAGA (IN (THE DAYS OF THE OLD WEST) (THE SHERIFF IS DESTRY)
AND (THE BAD GUY IS BLACKBART) ; (DESTRY LIVES IN DODGECITY) HAPPILY
ENOUGH UNTIL (BLACKBART RIDES INTO TOWN)) (BLACKBART RIDES INTO
DODGECITY AND (BLACKBART ROBS THE BANK) (DESTRY DISCOVERS THE HEIST
AND (BLACKBART HIDES IN A CANYON (DESTRY RIDES INTO GREAT ROCK CANYON)
 (DESTRY HEARS THE WHINNY OF A HORSE) (DESTRY DISMOUNTS AND TRACKS
BLACKBART) (DESTRY GETS THE DROP ON BLACKBART) THEN (BLACKBART
SURRENDERS) SO (DESTRY SHOOTS HIM) AND THERE IS A (PEACEFUL TOWN)) (
THE END))))
*TIME: 459   (Note: Times are CP time in milliseconds.)
```

Figure 6. Old West Saga and Summaries

```
(STORY OLD WEST SAGA)
*VALUE:
(OLD WEST SAGA (IN (THE DAYS OF THE OLD WEST) (THE SHERIFF IS DESTRY)
AND (THE BAD GUY IS BLACKBART) ; (DESTRY LIVES IN DODGECITY) HAPPILY
ENOUGH UNTIL (BLACKBART RIDES INTO TOWN)) (BLACKBART RIDES INTO
DODGECITY AND (BLACKBART ROBS THE BANK) (DESTRY DISCOVERS THE HEIST
AND (DESTRY SWINGS INTO THE SADDLE AND (GALLOPS AFTER BLACKBART) (
BLACKBART HIDES IN GREAT ROCK CANYON) (DESTRY RIDES INTO GREAT ROCK
CANYON) (BLACKBART BUSHWHACKS DESTRY) AND (DESTRY DIES) AND BLACKBART
CONTROLS A (PEACEFUL TOWN)) (THE END))))
*TIME: 5453
(SAY PROOF 2)
*VALUE:
(OLD WEST SAGA (DESTRY KEEPS A (PEACEFUL TOWN) UNTIL BLACKBART ARRIVES
) (BLACKBART ROBS THE BANK (DESTRY HUNTS BLACKBART (THE END))))
*TIME: 230
(SAY PROOF 3)
*VALUE:
(OLD WEST SAGA (IN (THE DAYS OF THE OLD WEST) (THE SHERIFF IS DESTRY)
AND (THE BAD GUY IS BLACKBART) ; (DESTRY LIVES IN DODGECITY) HAPPILY
ENOUGH UNTIL (BLACKBART RIDES INTO TOWN)) (BLACKBART RIDES INTO
DODGECITY AND (BLACKBART ROBS THE BANK) (DESTRY DISCOVERS THE HEIST
AND (BLACKBART AMBUSHES DESTRY) (THE END))))
*TIME: 320
```

Figure 7. Variant Blackbart Story and Summaries

Although the example stories are brief and uncomplicated, the problem solving system using Horn clauses that include functions is theoretically strong enough to support the generation of arbitrarily complex stories. In QUERY mode it can solve a large class of problems and describe the solution path with the aid of the TF antecedents. The study of more complicated applications is an ongoing line of experimentation with the system.

7.0 AN INTERPRETER FOR STORY GRAMMARS

Systems of antecedent-consequent rules have been a favored tool for problem solving in artificial intelligence research beginning with GPS. Fischer Black in 1964 used a system of consequent-antecedent rules patterned partly after GPS to demonstrate the efficacy of answering questions by proving theorems. He showed that his system mapped into Smullian's logic. We found his algorithm to be one of the most natural for answering questions in quantified semantic networks (Simmons and Chester 1977) and it applies as well to the interpretation of Horn clause representations of story grammars.

The interpreter is written to offer two modes of operation, QUERY and ASSERT. In Query mode it is an ordinary problem solver that requires that a problem or theorem be reduced to some combination of assertions and theorems in the data base. In Assert mode it is a generator, so if a theorem cannot be proved, it is asserted. This latter mode makes the procedure comparable to POGEN (of Section II) in flow of control, but it binds variables in a different manner and computes a syntactic derivation structure, the story tree.

The two basic functions GEN and GENLST are shown in LISP 1.5 in Figure 8. They incorporate the following steps:

```
(DEF
 (GEN (SYM)
      (PROG (Z ANS V A Q P V1)
            (RETURN
             (COND ((NULL SYM) NIL)
                   ((ATOM SYM) (GEN (LIST SYM)))
                   ((GET (CAR SYM) "FN)
                    (AND (SETQ Z (EVAL SYM)) (LIST Z Z)))
                   ((NOT (SETQ V (MATCHV SYM RULES)))
                    (AND (NULL QUERY) (LIST SYM SYM)))
                   ((FOR1 "V1
                          V
                          "(COND ((NOT (SETQ Q (MEMBER "< V1)))
                                  (SETQ ANS
                                        (BIND SYM (CAR V1) (CAR V1)))
                                  (SETQ SYM (BIND ANS SYM SYM))
                                  (LIST SYM SYM))
                                 ((SETQ A
                                        (GENLST
                                        (BIND SYM (CAR V1) (CDR Q))))
                                  (SETQ P
                                        (BIND (CAR A)
                                        (CDR Q)
                                        (CAR V1)))
                                  (SETQ P (BIND P SYM SYM))
                                  (LIST P (CONS P (CADR A)))))))))))


[DEF (GENLST(LST)(PROG(R R2)(RETURN
 (COND((NULL LST)NIL)
      ((SETQ R(GEN(CAR LST)))
         (SETQ R2 (GENLST(BIND (CAR R) (CAR LST)(CDR LST))) )
         (OR (AND (NULL R2) (LIST (CONS (CAR R) NIL) (CONS (CADR R))))
          (LIST (CONS (CAR R) (CAR R2))
               (CONS (CADR R) (CADR R2))  ) )      )
       (T NIL) )]
```

Figure 8. The LISP functions GEN and GENLST

```
To Generate a Relation;  SYM;

1.  If SYM is null, return NIL,

2.  If SYM is a function, return the value of the evaluated
    function

3.  Match SYM to the database of rules and assertions:

    1.  If there is no match,

        1.  if in Query mode, return nil,

        2.  if in Assert mode, Assert SYM to database, Return SYM.

    2.  If SYM matches a database element, R;

        1.  if there is no right half to R--i.e. R is an
            assertion--Bind the constants of R through the
            variables of SYM and return SYM.

        2.  there is a right half -- R is a rule--

            1.  bind constants of SYM through R,

            2.  detach the first element of R -- it is proved if
                the remainder are proved,

            3.  for all the remaining elements Rm=R1...Rn, Generate
                Ri;  as each Ri is proved, Bind its values through
                Ri+1...Rn and reiterate from step 3.2.2.2 until Rn
                is proved,

            4.  Bind the values of variables in R for the variables
                of SYM and return SYM.
```

The tree of instantiated rules form the proof, and in the LISP procedure SYM is consed to this tree to return both the answering relation and its proof.

The function, BIND, takes three arguments, a relation, R, the first element A of the rule it matches, and the remainder of the rule, C. Since R matches A, the variables in A are replaced by the constants in R. Since rules are written so that variables in A must stand for the same values as they do in C; where a variable in A has been assigned a value, the same value is assigned to that variable wherever it occurs in C, and C is returned. When C has been proved , values of variables in C are bound backward through A to R. The match function uses the following logic:

1. a variable matches a constant

2. a constant matches a variable

3. two identical strings match

4. two expressions match if every element in one matches the corresponding element in the other.

The algorithm is programmed in LISP in a nesting (the Forl "v... function) such that when matching fails the LISP interpreter automatically tries the next candidate with no explicit instructions needed for backup.

The procedure described above is a simplified first path version of the question answering algorithm published in Simmons and Chester (1977). It uses a list of rules and assertions as its database. In the more complete version it consults a semantically organized network with significant savings in computation time. We have tested the semantically organized versions of this procedure for question answering and for several of the problems that Kowalski presents (Kowalski 1974), and it is our belief that it is a suitable interpreter for experimenting with many applications of predicate logic expressed in Horn clauses.

8.0 DISCUSSION AND CONCLUSIONS

The problem of most concern in computational linguistics is to compute a representation of the understanding of ordinary natural language text. Several psychologists, but particularly Kintsch and Van Dijk, have accumulated evidence that human readers construct a macrostructure above the level of the sentences that they read. This metastructure appears to organize the propositions of the text in a fashion similar to the way we might construct an outline. It is this macrostructure and its contents that appears best to account for the way human subjects remember, summarize and forget what they read.

In developing a computational model of this theory we first presented a study of generating sense and nonsense from texts modelled in phrase structure grammars with more or less random selection of terminal elements. A considerable degree of control over the coherence of the resulting text was demonstrated by the ability to generate reasonable variations on an author's verses.

A method for using more complex grammar rules was then described for computing deep case structures from sentences and a corresponding rule-based method for reconstructing English sentences was shown. It is expected that these semantically based grammars will ultimately prove adaptable to computing macrostructures that organize sentences into larger units of discourse.

We then defined a story tree as a phrase structure organization of text propositions that provided the property that the most general propositions were closest to the root. The test of "story-treeness" is that when the tree is truncated at a given depth less than that of its leaves, the resulting terminals produce a summary. The leaves of the complete tree are the propositions of the entire story.

Meehan's use of Plans and Planboxes to write stories was next considered as a model of the thought process that underlies the construction of a story. As a much simpler example, we examined a story of Winograd's robot stacking a green pyramid on a red block. We used Horn clauses following Kowalski to model the robot's problem solving for this task and then showed that the resulting derivation tree had the story tree's property of providing summaries at levels approaching the root. A similar tree ordered according to a story grammar was constructed from four sentences describing the robot's task.

This form of grammar, augmented with transformation rules, was then used to organize Horn clauses so as to produce some simple stories and to demonstrate that the resulting story trees provided summaries. An interpreter for the story grammars was then described in terms of a problem solving system with an Assert mode that makes it a tree generator. In Query mode it can also prove ordinary AI examples written in Horn clauses.

What has been accomplished in this study is the definition of a structure called a story tree that is generated by assertions and inference rules in such a manner that its terminals provide the sequence of propositions comprising a story while higher nodes contain transformations that describe summaries of the nodes below. As a result, truncation of the story tree at successively higher levels provides briefer and briefer summaries. This structure for representing narratives is presented as an initial computational model for the macrostructure theorized by Kintsch and Van Dijk to account for aspects of human understanding and memory for stories.

Following Meehan, we used a problem solving approach, but in the form of Kowalski's logic of Horn clauses, to represent models of microworlds. A story tree is generated by using the inference procedure to generate a particular subtree of instantiated events in the microworld. All possible events in the microworld are modelled by a set of assertions and inference rules which can imply an infinitely large network.

The microworlds that we use for defining narratives are comprised of such events as Episodes, Acts, Results, TF-transformations, and Propositions. These are obviously linguistic entities that form models of possible text constituents in contrast to microworlds that model motivated characters and sequences of their problem solving actions. Our analysis of the blocks problem as a story tree indicates that problem solving in "blocks world" or "talking animals world" can also be organized to generate story trees.

Defining story trees by generating them is an initial step toward the goal of computing from English texts a representation of their meaning. If we could compute story trees from any arbitrary sequence of propositions from a narrative, we could claim a model of human understanding. Having established one form that has properties that can model a human reader's understanding, our next step is to apply the inference network to infer the story trees that account for a sequence of propositions that comprise a story. In this recognition or parsing task in order to minimize the multiple interpretations that could possibly account for a sequence of propositions, the inference rules will require many more antecedents to specify tight semantic constraints on the realization of a story tree.

Although it is clear that much work remains to be accomplished, we conclude that this approach to generating story trees provides a promising initial computational model for the macrostructure that appears to underly human understanding and memory for narratives.

Appendix A. Old West Saga Grammar and Story Tree

The Blackbart grammar is shown below.

```
(GENPRINT RULES)
((OLD WEST SAGA) < (SETTING GG BG L MOT OUTC) (EPI GG BG L MOT OUTC) (
TF (1 2 3)))

((SETTING GG BG L MOT OUTC) < (THE SHERIFF IS GG) (THE BAD GUY IS BG)
(THE PLACE IS L) (THE TIME IS Z) (GG LIVES IN L) (BG RIDES INTO TOWN)
(SET MOT (PEACEFUL TOWN)) (TF (IN Z 2 AND 3 ; 6 HAPPILY ENOUGH UNTIL 7
) (GG KEEPS A MOT UNTIL BG ARRIVES)))

((EPI GG BG L MOT OUTC) < (ACT GG BG L MOT OUTC) (RESULT GG BG L MOT
OUTC) (TF (2 3) (OUTC)))

((RESULT GG BG L MOT OUTC) < (EQ* MOT OUTC) (THE END) (TF (3)))

((RESULT GG BG L MOT OUTC) < (NEQ* MOT OUTC) (EPI GG BG L MOT MOT) (TF
 (3)))

((ACT GG BG L (PEACEFUL TOWN) (BG ROBS THE BANK)) < (BG RIDES INTO L)
(SET OUTC (BG ROBS THE BANK)) (SET MOT (PEACEFUL TOWN)) (TF (2 AND
OUTC) (OUTC)))

((ACT GG BG L MOT (PEACEFUL TOWN)) < (GG DISCOVERS THE HEIST) (FIND GG
 BG) (TF (2 AND 3) (GG HUNTS BG)))

((ACT GG BG) < (GG SWINGS INTO THE SADDLE) (GALLOPS AFTER BG) (BG
HIDES IN GREAT ROCK CANYON) (TF (2 AND 3 4) (BG HIDES IN A CANYON)))

((FIND GG BG) < (RAND*) (ACT GG BG) (GG RIDES INTO GREAT ROCK CANYON)
(BG BUSHWHACKS GG) (GG DIES) (SET OUTC (PEACEFUL TOWN)) (TF (3 4 5 AND
 6 AND BG CONTROLS A OUTC) (BG AMBUSHES GG)))

((FIND GG BG) < (ACT GG BG) (GG RIDES INTO GREAT ROCK CANYON) (GG
HEARS THE WHINNY OF A HORSE) (GG DISMOUNTS AND TRACKS BG) (GG GETS THE
 DROP ON BG) (BG SURRENDERS) (GG SHOOTS HIM) (SET OUTC (PEACEFUL TOWN)
) (TF (2 3 4 5 6 THEN 7 SO 8 AND THERE IS A OUTC) (GG SHOOTS BG TO
MAKE A OUTC)))

((THE SHERIFF IS DESTRY))

((THE BAD GUY IS BLACKBART))

((THE PLACE IS DODGECITY))

((THE TIME IS (THE DAYS OF THE OLD WEST)))
```

```
((SET X X))
     The Blackbart storytree starts below.

(PPRINT PROOF)
((OLD WEST SAGA)
 ((SETTING DESTRY BLACKBART DODGECITY (PEACEFUL TOWN) OUTC)
  (THE SHERIFF IS DESTRY)
  (THE BAD GUY IS BLACKBART)
  (THE PLACE IS DODGECITY)
  (THE TIME IS (THE DAYS OF THE OLD WEST))
  (DESTRY LIVES IN DODGECITY)
  (BLACKBART RIDES INTO TOWN)
  (SET (PEACEFUL TOWN) (PEACEFUL TOWN))
  (TF
   (IN (THE DAYS OF THE OLD WEST) 2 AND 3 ; 6 HAPPILY ENOUGH UNTIL 7)
   (DESTRY KEEPS A (PEACEFUL TOWN) UNTIL BLACKBART ARRIVES)))
 ((EPI DESTRY
       BLACKBART
       DODGECITY
       (PEACEFUL TOWN)
       (BLACKBART ROBS THE BANK))
  ((ACT DESTRY
        BLACKBART
        DODGECITY
        (PEACEFUL TOWN)
        (BLACKBART ROBS THE BANK))
   (BLACKBART RIDES INTO DODGECITY)
   (SET (BLACKBART ROBS THE BANK) (BLACKBART ROBS THE BANK))
   (SET (PEACEFUL TOWN) (PEACEFUL TOWN))
   (TF (2 AND (BLACKBART ROBS THE BANK)) ((BLACKBART ROBS THE BANK))))
  ((RESULT DESTRY
           BLACKBART
           DODGECITY
           (PEACEFUL TOWN)
           (BLACKBART ROBS THE BANK))
   *T*
   ((EPI DESTRY BLACKBART DODGECITY (PEACEFUL TOWN) (PEACEFUL TOWN))
    ((ACT DESTRY BLACKBART DODGECITY (PEACEFUL TOWN) (PEACEFUL TOWN))
     (DESTRY DISCOVERS THE HEIST)
     ((FIND DESTRY BLACKBART)
      *T*
      ((ACT DESTRY BLACKBART)
       (DESTRY SWINGS INTO THE SADDLE)
       (GALLOPS AFTER BLACKBART)
       (BLACKBART HIDES IN GREAT ROCK CANYON)
       (TF (2 AND 3 4) (BLACKBART HIDES IN A CANYON)))
      (DESTRY RIDES INTO GREAT ROCK CANYON)
      (BLACKBART BUSHWHACKS DESTRY)
      (DESTRY DIES)
      (SET (PEACEFUL TOWN) (PEACEFUL TOWN))
      (TF (3 4 5 AND 6 AND BLACKBART CONTROLS A (PEACEFUL TOWN))
          (BLACKBART AMBUSHES DESTRY)))
     (TF (2 AND 3) (DESTRY HUNTS BLACKBART)))
    ((RESULT DESTRY
             BLACKBART
             DODGECITY
             (PEACEFUL TOWN)
             (PEACEFUL TOWN))
     *T*
     (THE END)
     (TF (3)))
    (TF (2 3) ((PEACEFUL TOWN))))
   (TF (3)))
  (TF (2 3) ((BLACKBART ROBS THE BANK))))
 (TF (1 2 3)))
```

References

Black, F. 1969. "A deductive question answering system." In _Semantic Information Processing_. M. Minsky, _ed_. Boston: MIT Press.

Bobrow, D. G. and Collins, A. 1975. _Representation and Understanding_. New York: Academic Press.

Bobrow, D. and Winograd, T. 1977. "An overview of KRL, a knowledge representation language." _Cognitive Science_, Vol. 1, No. 1, January, 1977, pp. 3-46.

Charniak, E. and Wilks, Y. 1976. _Computational Semantics_. New York: North-Holland Press.

Chester, D. 1976. "The translation of formal proofs into English." _Artificial Intelligence_, Vol. 7, No. 3, pp. 261-278.

Crothers, E. 1972. "Memory structure and recall of discourse." In _Language Comprehension and the Acquisition of knowledge_. Freedle, R. and Carrol, J. _eds_. New York: Wiley.

Deliyanni, A. and Kowalski, R. 1977. "Logic and semantic networks." London, England: mss. Imperial College, University of London.

Fillmore, C. 1968. "The case for case." in _Universals in Linguistic Theory_. Bach, E. and Harms, R. _eds_. New York: Holt, Rinehart and Winston.

Grimes, J. 1975. _The Thread of Discourse_. The Hague, Netherlands: Mouton Publishers.

Grosz, B. 1977. "The representation and use of focus in dialogue understanding." Technical Note No. 151. Menlo Park, California: Stanford Research Institute.

Harper, K. and Su, S. 1969. "A directed random paragraph generator." Rand Manuscript No. RM6053-PR. Santa Monica, California: Rand Corporation.

Hendrix, G. G. 1976. "Partitioned networks for modelling natural language semantics." Dissertation. Austin, Texas: Department of Computer Sciences, The University of Texas.

Kintsch, W. 1974. _Representation of Meaning in Memory_. New York: Wiley Press.

Kintsch, W. and Van Dijk T. 1975. "Recalling and summarizing stories." unpublished mss. from the authors, May, 1975.

Klein, S. 1965. "Automatic paraphrasing in essay format." Mechanical Translation, Vol. 8, Nos. 3 and 4, pp. 68-83.

Klein, S. and Simmons, R. 1963. "Syntactic dependence and the computer generation of coherent discourse." Mechanical Translation, Vol. 7, No. 2, pp. 50-61.

Kowalski, R. 1974. "Logic for problem solving." Memo 75. Edinburgh, Scotland: Department of Computational Logic, University of Edinburgh.

Lehnert, W. 1977. "Question answering in a story understanding system." Cognitive Science, Vol. 1, No. 1, pp. 47-73.

Meehan, J. 1976. "The metanovel: writing stories by computer." Dissertation. New Haven, Connecticut: Department of Computer Sciences, Yale University.

Meyer, B. 1975. The Organization of Prose and its Effects on Recall. Amsterdam, The Netherlands: North Holland Press.

Minsky, M. 1975. "A framework for representing knowledge." In the Psychology of Computer Vision. P. Winston ed. New York: McGraw-Hill.

Pratt, V. R. 1975. "Lingol - a progress report." Proc. 4IJCAI.

Rumelhart, D. E. 1975. "Notes on a schema for stories." Bobrow, D. and Collins, A. eds.

Schank, R. C. 1975. Conceptual Information Processing. New York: North-Holland Press.

Schank, R. C. 1975. "The structure of episodes in memory." Bobrow, D. and Collins, A. eds.

Schank, R. and Abelson, R. 1977. Scripts, Plans, Goals and Understanding. New York: Wiley Press.

Simmons, R. F. 1978. "Rulebased computations on English." in Pattern-Directed Inference Systems. Hayes-Roth, R. and Waterman, D. eds. New York: Academic Press. (in press).

Simmons, R. and Chester, D. 1977. "Inferences in quantified semantic networks." Proc. 5IJCAI.

Thorndike, P. 1977. "Cognitive structures in comprehension and memory of narrative discourse." Cognitive Psychology, Vol. 9, No. 1, pp. 77-110.

Van Dijk, T. A. 1975. "Recalling and summarizing complex discourse." unpublished mss. from the authors. Amsterdam, The Netherlands: Department of General Literary Studies, University of Amsterdam.

Winograd, T. 1972. Understanding Natural Language. New York: Academic Press.

Yngve, V. 1960. "A model and an hypothesis for language structure." Proc. American Philosophical Society, No. 104, pp. 444:466.

Young, R. 1977. "Text understanding: a survey." Natural Languages Report No. 33. Austin, Texas: Department of Computer Sciences, The University of Texas. (in press, American Journal of Computational Linguistics).