Rule-Based Computations on Story Trees

Alfred Correira

NL-36 September, 1978

Technical Report NL36
Department of Computer Sciences
University of Texas at Austin 78712
September 1978

ABSTRACT

A theory of understanding (parsing) of texts as a process of collecting simple textual propositions into thematically and causally related units is described, based on the concept of macrostructures as proposed by Kintsch and van Dijk. These macrostructures are organized into tree hierarchies - the story tree - and their interrelationships are described in rule-based story grammars structurally derived from the Kowalski logic based on Horn clauses. A procedure for constructing such trees (a story generator) and for synthesizing them from texts derived from these same generated trees (a story parser) is detailed. The resulting program is capable of understanding and summarizing any story it can generate using the same basic control structure. This paper is related to ideas first presented in Simmons and Correira (1978).

1.0 INTRODUCTION

One of the most difficult tasks in the field of Computational Linguistics is that of processing (parsing or understanding) bodies of connected textual material from simple narratives, like fairytales and children's stories, to complex technical articles, like textbooks and encyclopedia articles. When effective parsers were created capable of processing single sentences (Woods, 1970) (Schank, 1975a), (Norman and Rumelhart, 1975), (Winograd, 1972), it was quickly realized that these same techniques were not in themselves adequate for the larger task of processing sequences of sentences. The understanding of such texts involved more and different knowledge from that necessary for understanding single sentences, and the resulting structures from a text parser need not look like the structures of the sentences parsed individually.

Experiments with processing texts lead to such procedural entities as frames (Minsky, 1975; Charniak, 1976; Winograd and Bobrow, 1977), scripts and plans (Schank and Abelson, 1977), focus spaces (Grosz, 1977), partitions(Hendrix, 1976) among others. These efforts involve conceptual structures consisting of large, cognitively unified sets of propositions, and they model understanding as a process of filling in or matching the slots in a particular structure with appropriate entities

derived from input text.

There have also been rule-based approaches to the text processing problem, most notably the template/paraplate notion of Wilkes(1976), and the story grammars of Rumelhart (1975). Although each approach (procedures and rules) has its merits, it is the rule-based approach which will be presented here.

This paper describes a rule-based computational model for text comprehension, patterned after the theory of macrostructures proposed by Kintsch and van Dijk (1975). The rules are notationally and conceptually derived from the Horn clause, especially as described by Kowalski (1974). Each rule consists of a set of thematically, causally, or temporally related propositions. The rules are organized into a network with the macrostructures becoming more generalized approaching the root. The resulting structure, called the Story Tree, represents a set of textual structures.

The process of generating from this network consists of choosing one of the rules to serve as the root of a particular story tree and recursively instantiating its descendents until terminal propositions are achieved (Simmmons and Correira, 1978). These propositions form the text of the generated story. Conversely, a text is understood if its input propositions can be mapped into rules and these rules recursively mapped into more abstract

rules until a single node (the root) is achieved.

2.0 MACROSTRUCTURES AND STORY GRAMMAR RULES

In this section the fundamental notion of macrostructure, as proposed and used by Kintsch and van Dijk, is presented and then analyzed from a computational, rather than a psychological, standpoint. An effective representation for macrostructures is described, derived from Horn clauses and organized into story trees.

2.1

Kintsch and van Dijk (1975) present a system for organizing an entire discourse hierarchically into macrostructures, which are essentially metapropositions. These macrostructures correspond to higher-order narrative macro-categories such as "complication", "resolution", etc. They introduce a number of rules for relating these macrostructures to sets of input textual propositions: information reduction (generalization), deletion (of less important propositions), integration (combining events with their pre- and postconditions), and construction (which relates complex propositions to their component sub-propositions).

There are two conditions that are always true regarding these macrostructures: a macrostructure must be "implied" by its subordinate propositions, and the macrostructures of a text collected together form a meaningful summary of the text. Kintsch and van Dijk believe that it is primarily macrostructures that are retained when a text is understood by a human reader and that the macrostructures are created as the text is being processed.

As evidence in support of their theory they present a number of psychological experiments in recall and summary with human subjects using as a text a 1600-word narrative taken from Boccaccio's Decameron (the Rufolo story). The psychological validity of this theory is not an issue here, only its utility as a model for a computational theory of text processing. As a computational entity, a macrostructure is a node in a story tree whose immediate descendents consist of the subordinate propositions by which the node is implied, and is itself a descendent of the macrostructure it (partially) implies. Every macrostructure in this tree is the root of a derivation tree whose terminals are simple propositions.

Each level of the tree shares the attribute of summarizability, i.e. a summary of the text may be extracted from any level of the tree, becoming less specific as the summary level approaches the root. The lowest level summary is the original text itself; the highest level (the

root) is a title for the text.

The ability to give meaningful (coherent) summaries for a text is one attribute of comprehension for that text, and any procedure yielding trees possessing the summary property can be said to understand the text. Furthermore, given an appropriate data base internalizing the relationships between a macrostructure and its subordinate macrostructures or simple propositions (microstructures) and a summary derived from a story tree, it is possible for a procedure to reconstruct to a certain degree of detail the original text from which the tree was derived. The degree of detail recovered is directly proportional to the relative distance from the nodes forming the summary to the original input propositions (the leaves) of the text tree.

2.2

Having defined an appropriate model for a computational structure for text processing the next question is that of finding a suitable rule structure for representing the relationship between a macrostructure and its subordinate propositions. One simple statement of this relationship is in the form of a rule

A <= B C D

meaning "you may assert the truth (presence) of

macrostructure A if you can find the (nearly) contiguous propositions B, C, D present in the input text." "Nearly" means that an allowable level of "noise," in the form of irrelevant side information, may be present between the specified propositions, a problem not addressed here.

This rule form closely resembles in structure and meaning the Horn clause notation. The general Horn clause has the format,

C1, ... , Cm <- A1, ... , An

where the set C1, ... , Cm is called the Consequent, and the set A1, ... , An is called the Antecedent. For this rule form the following interpretations are imposed

1. m = n = 0 is the null clause

2. m = 1, n = 0 is an assertion

3. m = 0, n > 0 is a goal statement

4. m = 1, n > 0 is an operator.

If the propositions in a clause contain the variables x1, ... , xi, then the clause has the interpretation

forall x1, ... , xi, A1, ... , An TRUE

implies C1, ... , Cm TRUE

The rule-base for the system described in this paper

contains only clauses (propositions) of types 2 and 4; clauses of type 2, written without the left-arrow, are called facts, and the type 4 operators are rules. Clauses of types 1 and 3 (without the left-arrow) are embedded in the procedures for processing texts.

There are several differences between the Kowalski logic, and the logic adopted here, but the most important has to do with the ordering of the antecedent propositions. In a true Horn clause, the ordering is irrelevant and A <= B C D is as good a rule as A <= C D B, etc., i.e. the antecedents can be proved in any order. The ordering in the system described here is governed by rules of coherence. For example, the rule

John built a house <=
    John purchased building materials
    John laid the foundation
    John built the walls
    John raised the roof

is a meaningful rule, whereas

John built a house <=
    John raised the roof
    John built the walls
    John laid the foundation
    John purchased building materials

is nonsense and impossible. The antecedent set of propositions is ordered by a principle of coherence. This principle can be one (or more) of several standards: forward causal connectedness (B causes C, which causes D), backward causal connectedness (D is the result of C, which is in turn the result of B), temporal ordering (B happens before C, which happens before D), etc.

The Horn clause rule is reversible. If we are given the proposition

John built a house

in a text, or a summary of a text, we may infer the chain of events

John purchased building materials
John laid the foundation
John built the walls
John raised the roof

and if given some of these events in order in a text we may infer that John was engaged in building a house.

The antecedent rule form is bidirectional and can be used in either the parsing or generating tasks. A parser can avail itself of antecedent rules to group sets of propositions in a text into units headed by macrostructures (which are the consequents of the rules). These can be

further grouped into more generalized macrostructures recursively to yield a story tree. A generator would proceed in the other direction, starting with the top node of a tree and expanding it and its constituents downward recursively (by using the antecedent rules of the constituents) to arrive at a tree whose terminals form a coherent text. The head and the outcome together summarize the macrostructure.

2.3

After several early experiments with this rule form on a simple text (the Margie story - Rumelhart, 1975) it was discovered that the simple antecedent rule form, although capable of handling the computations necessary for the text at hand, tends to gloss over several inherent attributes that macrostructures generally have in common. For example, in the previous housebuilding rule several categories can be distinguished. In order to build the house, John must have the proper building materials and the necessary knowledge of house construction to use them. Given these, John will follow a certain sequence of actions; he will lay the foundation, then the walls, and finally the roof. The result of this effort will be that John possesses a finished house (for personal habitation or perhaps for sale). Therefore, we can break the housebuilding rule into several natural groupings:

John builds a house <

John has building materials

John has construction training

>

John lays the foundation

John builds the walls

John raises the roof

>>

John possesses a house

Structurally, this rule form will be referred to as an "extended" Horn clause (EHC). The first part of the rule (before the left-arrow) is the Head of the rule, and represents the macrostructure pattern. The second part (between the left- and single right-arrows) is the Antecedent, forming the precondition set for the rule. The propositions in the antecedent are the conditions which must be true, or can be made true, before John can embark on an episode of "housebuilding." The third part (between the single and double right-arrows) is the Consequent, or expansion, of the rule. If John builds a house, then these are the (probable) actions he will take in doing so. The last part of the rule (after the double right-arrow) is the Outcome, or post-condition set, of the rule, which consists of the conditions that will become true upon the successful completion (expansion) of the rule.

The resulting rule form is related conceptually and historically to the notion of a script as developed by Schank and Abelson (see also Norman and Rumelhart, 1975). The antecedent sets the stage for the invocation of a rule. It describes the setting and the roles of the characters involved in the rule. The consequent consists of the actions normally taken during the invocation of the rule. The outcome is the result of the actions. When used in a script-like role, a rule is activated when its antecedent has been satisfied, and its consequent can then be sequentially instantiated.

A rule can also be used as a plan. A plan is a data structure that suggests actions to be taken in pursuit of some goal, or outcome. This corresponds to activating a rule according to its outcome, i.e. employing a rule because its outcome is the desired effect. If one has the goal "possess a house" one might wish to employ the housebuilding rule to achieve it. In this case, a rule is invoked when a character has an active goal that matches one of the outcome propositions of the rule. The antecedents must then be checked to insure that the world at that point will allow the application of the rule. If this is the case, then the rule can be applied and the goal marked as being satisfied (made true).

This extension of the Horn clause structure serves two

purposes. First, by separating the propositions subsumed under a macrostructure into three parts, it renders it unnecessary to label the roles that the individual propositions play in the macrostructure. A macrostructure will usually have preconditions, expansion(s), and postconditions, which would have to be labeled (probably functionally) in a simple Horn clause system. Secondly, it serves as a means of separating those propositions which must be true before a rule can be invoked (antecedents) from those whose success or failure follow the invocation of the rule.

2.4

A rule may have several consequents attached to it, one for each expansion of the rule. Thus if a couple wants a child they could employ a rule - written in case-predicate notation (see the Notes on Appendices section for description of case-predicate notation and how to read it):

```
(POSSESS A COUPLE TH CHILD)
     > (OR (HAVE A COUPLE TH CHILD)
     (ADOPT A COUPLE TH CHILD)
     (STEAL A COUPLE TH CHILD)
     (BUY A COUPLE TH CHILD))
```

where each of the propositions, HAVE, ADOPT, STEAL, BUY are

complex rules of the same form as the POSSESS rule .

A rule may also have multiple antecedents, if there are several sets of circumstances under which the rule can be invoked. Thus a rule for watching a drive-in movie could have the form

```
(WATCH A PERSON TH DRIVE-IN-MOVIE)
     <   (OR   ((PERSON   IN   CAR)   (CAR   IN
     DRIVE-IN-MOVIE-LOT))
     ((SEE A PERSON TH  DRIVE-IN-MOVIE-SCREEN)  (CAN  A
     PERSON TH (READ A PERSON TH LIPS))))
```

A rule can have but a single outcome, composed of simple propositions, since by definition the outcome is the set of propositions that are true if the rule succeeds. If an outcome could contain a rule proposition, then that proposition could fail independently of the rule for which it is part of the outcome, since it could have its own antecedents - thus defeating the purpose of the outcome in the first place.

Syntactically, the Extended Horn Clause can be represented

```
HEAD < (OR (P1) (P2) ...  (Pi))
     > (OR (P1) (P2) ...  (Pj))
     >> (AND (PROP1) (PROP2) ...  (PROPk))
```

where each of the Ps are an ANDed set of PROPositions. In the actual implementation, the ORs and ANDs are implicit, and the structure becomes simply

HEAD < (P1) < (P2) < ... < (Pi)
> (P1) > (P2) > ... > (Pj)
>> ((PROP1) (PROP2) ... (PROPk)).

2.5

The rules are stored in memory in a semantic network. However, unlike the usual semantic networks for case predicates, where the individual nodes (tokens) are connected by case relations (Simmons, 1977), the semantic links in the EHC rule network are based on the story tree concept.

Each rule node (or instantiation of one) in the network may have any of the following arcs: ANTE, CONSE, and OUTCOME. The value attached to the arc depends on whether the node represents a rule or an instantiation of a rule. If the node is a rule, then the value of ANTE is a list of the antecedent sets for the rule, the value of CONSE is a list of the consequent sets for the rule, and OUTCOME is the set of outcome propositions for the rule. If the node is an instantiation of a rule, then the value of ANTE is a list of the instantiated nodes in its antecedent, the value of CONSE

is a list of instantiated nodes in its consequent, and the value of OUTCOME is a list of instantiated nodes in its outcome. Thus, the system is able to distinguish between rules and instances in the database.

All the case relations are kept in a single list called PATT (for PATTern) as arc-value pairs. Each node in the ANTE, CONSE, and OUTCOME lists (whether the node is a rule or an instance of a rule), and each case value in PATT has a USEDIN arc pointing back to the node in which they are used.

Three other arcs are used. TVAL marks terminal nodes whose truth value is false as, for example, on the token for KISS in the proposition (KISS A JOHN TH MARY), if John did not kiss Mary. If this negated proposition were to be printed, it would appear as (NOT OF (KISS A JOHN TH MARY)).

CTXT is a context marker. Each node in a tree belongs to the context of that tree, i.e. to the context of the story being generated or parsed. Each story is processed under its own CTXT value. Thus, several stories may be coresident in memory without conflict, since the propositons in each will be marked by a unique CTXT.

The last arc label, NXT, is a special arc used only with class objects in the database. In the semantic networks based on case-predicates, every class object points

to all of the tokens subsumed by it in the network. For example, the class object representing DOG would have pointers to each of its tokens, DOG1, DOG2, DOG4, etc., which represent individual objects of the class DOG. In the EHC semantic network, this information is kept implicitly via the NXT arc. The class objects are marked with a zero token (DOG0) and NXT has as value the number of tokens plus one in the database for that class. For example, if DOG0 has a NXT of 5, then the network (probably) contains the tokens DOG1, DOG2, DOG3, and DOG4, not all of which, however, need be in the same context. If a token no longer exists in the network, like DOG3 in the first example, this fact will be evident because DOG3 will have no attributes (the value of its PATT arc will be NIL), since all extant objects in the network must have at least one attribute (usually an ISA case relation).

The database retrieval functions utilize the NXT attribute, coupled with "fuzzy" (partial) matching, to retrieve potential rules to be applied to a proposition. At any point where a rule can be applied, the class object for the head of the proposition is queried for its tokens (via the NXT arc). Each token in turn has its PATT arc value pattern-matched against the arc-value pairs in the proposition.

For each candidate rule, the pattern-matching proceeds

as follows. For each case relation in the proposition a search is made of the PATTern in the candidate rule for that same case relation. If it is found, then a check is made to ensure that any restrictions on the value slot for the PATT are satisfied by the value in the proposition, i.e. if the value slot in the PATT calls for a person, then the corresponding slot in the proposition cannot be filled by a dog, etc. If such a contradiction is found the rule is rejected. If no case arc is found in PATT then the next arc in the proposition is examined. At least half the arcs in the proposition must be found in the PATT of the candidate rule before the rule can be used.

Partial matching allows the rule-writer to combine rules that only differ in one or two minor cases, but which all share the same major cases; he need only write one rule specifying the major case relations and ignoring the minor ones. The second benefit is in the generator. Partial matching also allows the generator to bring more operators to bear at a given point in the story construction. However, the parser pays the price for this flexibility by having to examine more alternatives at each point in its parsing where rules apply.

The ASK function, which queries the database for the existence of facts (instantiated propositions without variables), uses a complete pattern-matching algorithm (all

case arguments must match the entire candidate instance PATTern), since "John ate at Mary's place last night" is not deemed a sufficient answer to the question "Who ate all the cake last night at Mary's place?".

3.0 SUCCESS AND FAILURE IN EHC RULES

The idea of rule success or failure in the EHC rule form is tied to the domain the rules are attempting to model - real life motivated behavior (actions) where things can, and do, go wrong and fail to achieve their goals. In real life, John's decision to build a house does not guarantee that the house will be built. For any given rule, one of three conditions may arise. First the rule's antecedents may fail to be true and the rule cannot be activated. In the building rule, if John cannot procure the building materials necessary for constructing the house or does not possess the appropriate knowledge needed to build it, the house cannot be built despite John's intentions. A rule is not invoked until one of its antecedent sets has been successfully processed.

Once this condition is satisfied then the rule can be invoked and John can start building his house. At each step of expanding the macrostructure, i.e as John performs each step in the consequent of the rule, he is subject to success or failure. He may succeed in laying the foundation, or he

may fail, and the same is true for constructing the walls and the roof. If he does manage to successfully perform all the acts in a consequent, then the rule is said to be successful, and its outcome propositions can be asserted as being true. In the housebuilding rule, an assertion that John possesses the house he just constructed would be built.

But if a consequent fails, a different logic applies. If John has laid his foundation, and built his walls, but the walls collapse, then the last consequent proposition, raising the roof, cannot be performed. The rule is said to have failed, and the outcome predicates are negated. In the housebuilding rule, an assertion that John does not possess the house he was attempting to construct would be built.

Rule failure is an important concept with regard to narratives. Many narratives consist of a series of episodes in pursuit of a goal; often each episode, except the last, represents an instance of a failed script, since, if any rule prior to the last succeeded, then the goal would have been achieved and no further episodes would have been forthcoming, (unless a new goal were chosen). The mechanism of rule failure is a natural one for analyzing this narrative pattern.

4.0 GENERATION

In this section the procedure for using macrostructures, embodied in the EHC rule-form, to generate stories, as well as a program, TELLTALE, for implementing the procedure, will be discussed.

4.1

The paradigm used by TELLTALE for story generation is similar to that used by Meehan (1976) in his TALESPIN program, which wrote "metanovels" concerning the interrelationships, both physical and mental, of the motives and actions of anthropomorphized animals possessing simple behavior patterns. It was Meehan's contention that a story was an exposition of events that occur when a motivated creature attempts to achieve some goal or fulfill some purpose, such as satisfying a need or desire, solving a problem, doing a job, etc. TALESPIN was essentially a problem-solver that operated on a database consisting of living entities, inanimate objects, and a set of assertions (rules), typifying the relationships among them, and descibing operations for changing those relationships. The rules were organized into plans based on conceptual relatedness, which aided in retrieval of specific rules.

Meehan emphasized the use of plans as a mechanism for generating stories. Plan activation was a process of

assigning a goal to a character and then calling up any relevant plans that were indexed on this goal.

In the EHC rule the antecedent governs the ability of a rule to be invoked. If a rule is considered to be a plan in the Meehan sense above, then instead of indexing the rule by its outcome propositions, it can be indexed by its antecedent if the antecedent contains information relating to motivation. For example, the rule,

```
(MARRY-RULE A X TH Y)
                < ((X ISA MAN) (Y ISA WOMAN)  (WANT A X TH
                (MARRY A Y TH X)))
                > ((GO A X TO Y) (ASK A X TH Y IF (MARRY A  Y
                TH X)) (ACCEPT A Y TH X))
                >> ((MARRY A Y TH X))
```

is a rule that can be activated if some X, who must be a man, WANTs to marry some Y, who must be a woman. Every rule should contain some propositions in its antecedents that restrict the set of objects that can be inserted into the variables of the rule.

The last proposition in the antecedent, the WANT, is a goal statement; it states that the MARRY-RULE can be invoked when one person wants to marry another. In the process of generating a story, if a point is reached where a man develops the goal of wanting to marry a woman, the

MARRY-RULE can be invoked in order to satisfy that want.

4.2

Since Propp it has been common linguistic practice to describe a story tree in a phrase-structure grammar. An example of this is the work of Rumelhart (1975). Rumelhart describes a simple set of rules for analyzing stories into subordinate units. For example,

STORY -> SETTING + EPISODE

SETTING -> (STATES)

EPISODE -> EVENT + REACTION

is a description of the highest level of the story structure. A story consists of a setting followed by an episode, where the setting is a collection of states, and the episode consists of an event followed by some reaction. Rumelhart also outlined semantic interpretations for the rules, to the effect that the setting ALLOWS the episode, and that an event INITIATES a reaction. The semantic constructs of the Rumelhart system are not necessary here, since the causal connections are to be found in the antecedents of the rules and need not be explicitly stated. The corresponding EHC rules would be

(STORY) > ((SETTING) (EPISODE))

(SETTING) > ((STATE))

> ((STATE) (SETTING))

(EPISODE) > ((EVENT) (REACTION))

The actual syntactic rules used in the TELLTALE program were slightly different, and are shown as part of the Rufolo story rules in Appendix [D]. A subset of these syntactic rules was used in the simpler fairytale story in Appendix [A].

4.3

TELLTALE generates stories (sequences of propositions) based on such rules, contained in its database, either under user control or directed by the program itself. The database is a semantic network of nodes created by building a network of the case-predicate rules supplied by the user. The input to TELLTALE is the root of a story tree, which informs TELLTALE of the type of story which is to be generated. The output from TELLTALE is an instantiated story tree whose terminals are the propositions of the text. The program SUMMARIZE computes summaries from the story tree. An example rule base can be found in Appendix [A] for generating a set of fairytales. A sequence of propositions for a story generated from that rulebase is shown in Appendix [B]. Two summaries of the story are shown in Appendix [C].

The procedure for TELLTALE proceeds as follows. For any given node (proposition), if that node has no rules concerning it in the database, then it either SUCCEEDs or FAILs. If story control is in user mode, he is queried as to whether he wishes the node to succeed or not, otherwise the program decides. If node suceeds, an instantiated copy is returned as value, otherwise the negation of the node (in the form (NOT OF NODE)) is instantiated and returned.

If there is a rule form for this node in the rule-base, then it is expanded. The node and the HEAD of the rule are unified (by standard resolution techniques), and the resulting variable bindings are substituted throughout the antecedents, consequents and the outcome of the rule. The antecedents are then stripped from the rule and each is examined in turn until one of them succeeds. If all antecedents fail, the rule cannot be applied and another rule is tried, if one exists, or else NIL is returned. Each proposition in an antecedent is examined by first ASKing if it occurs in the database (i.e. has already been created prior to the invocation of this rule, by some previously instantiated rule). If this is not the case, then the procedure recurs on the proposition, expanding it in the same manner described above for its father. As each proposition of an antecedent is EXPANDed, if a successful instantiation is returned, then the resulting bindings are recorded and passed on to the remainder of the antecedent.

If an entire antecedent succeeds, then the consequents are stripped from the rule and examined. The same basic procedure as outlined above is followed, with the following exceptions: no node that is examined is ASKed, since consequents always represent new actions being performed by the program, and cannot already exist in the database. The second exception stems from the nature of a consequent proposition as opposed to an antecedent proposition. The failure of an antecedent proposition means that the rule cannot be applied at all, whereas the failure of a consequent proposition determines whether the rule returns success or failure (a positive or negative outcome).

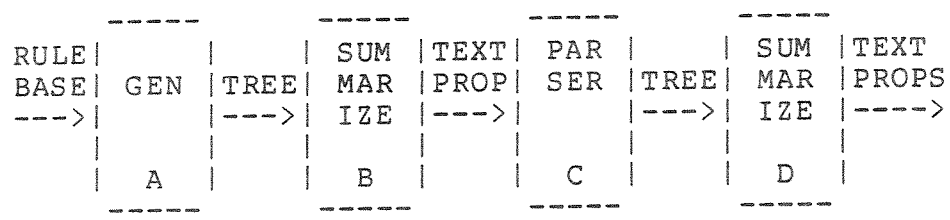
If a consequent succeeds (none of its constituent propositions fail) then the outcome is stripped off and bound, and each of its propositions is asserted to the database. The node is then rebound with the bindings recorded from the antecedent, consequent and outcome that succeeded and returned. If one of the propositions in a consequent fails, then the remainder of the propositions in that consequent are skipped, and another consequent is tried. If no further consequents exist, then the outcome is stripped from the rule and bound, and then each of its propositions negated and asserted. The node is then rebound, then negated and returned.

Starting with the root, the program will execute until

no more nodes remain to be expanded, because all have been reduced to terminal propositions. A traversal of the terminals of the resulting story tree will yield the propositional text of the story; any higher level traversal will yield a summary of the story.

5.0 PARSING

This section describes a procedure, BUILDTALE, for using macrostructures to parse stories. The goal of the generator-parser system is to create a program capable of understanding those texts that it can generate from a rule-base. Diagrammatically



The tree that results from the story generator should yield the same bottom-level terminal propositions as the tree that results from the parsing of the terminal propositions of the story, i.e. the output from part B in the diagram above should equal the output from part D. The story trees may or may not be identical, depending on whether the story grammar is ambiguous.

5.1

The philosophy of the understander can be summarized by recalling a concluding statement by Rumelhart (1975)

> "It is my suspicion that any automatic 'story parser' would require... 'top-down' global structures..., but would to a large degree discover them, in any given story, by the procedures developed by Schank [1975]."

The procedures developed by Schank emphasize the "bottom-up" approach to story parsing. Starting with the raw input data, emphasis is placed on integrating each sentence into the developing narrative structure. The structures used are scripts (and plans), and parsing integrates propositions into scripts. The scripts form an implicit episodic structure, but the higher order relationships among the scripts are not generally made specific (i.e. what structures formed episodes, setting information, complications, resolutions, etc).

BUILDTALE is a program that combines the top-down and bottom-up approaches. As the parse progresses, BUILDTALE attempts to integrate the terminal propositions into the context of a story tree. It manipulates the terminals bottom-up, integrating them into macrostructures, while it

is building story structure nodes (STORY, SETTING, EPISODE, etc.) top-down. If BUILDTALE successfully parses the text these two processes will "meet in the middle", forming a complete story tree.

The BUILDTALE procedure proceeds as follows. A root node is given to the BUILD function to be constructed. Initially this node is the root of the story tree (STORY). A global variable, STREAM, points to the next input proposition that has not yet been integrated into the tree (has not been marked as USED); initially this is the first proposition in the text. If this proposition is to be a node in the story tree a path from it to the root node (STORY) must exist in the rule-base, i.e. the proposition must be used in a rule that is itself used in a rule, etc., until a rule is reached that is used in the root node. Each of the paths found with this property will be tried until one succeeds in constructing an instance of the root node.

The paths are ordered by the size of the rule closest to the terminal proposition in the path, i.e. the rule with the largest number of propositions in its body (taking the largest antecedent and the largest consequent set plus the outcome set) is tried first. An attempt is made to map the chosen rule into the text starting at the left-most unmarked proposition.

The rule body is examined in the order: antecedents, consequents, and outcome. Each proposition in an antecedent is ASKed sequentially in the database. If it is found (unnegated) then mapping continues, otherwise the left-most unmarked proposition is matched against the antecedent proposition. If matching succeeds, the mapping of the antecedent continues, otherwise the antecedent is rejected, any propositions that were marked are unmarked, and the next antecedent is tried. If no more exist then the rule being tried cannot be applied, and another rule is examined. If there are no more alternatives, then the root node fails and NIL is returned.

An antecedent may also be rejected if one of its propositions is logically negated, i.e. a proposition of the form (HD ...) is found in the database instantiated as (NOT OF (HD ...)), or vice versa. In such a case, the logic of the previous paragraph also applies.

As each proposition in an antecedent matches, the resulting bindings are recorded (i.e. variables are bound to the constants in the text propositions) and bound through the remainder of the antecedent. When an antecedent succeeds in being matched the total binding record is bound through the consequent set. Processing continues with the consequents (in order by length) in the same manner as with the antecedent, with two exceptions. First, consequent

propositions are not ASKed in the database, since they represent new actions and cannot yet exist. Instead each is matched against the left-most unmarked proposition in the text, and the matched text proposition is marked as USED when a successful match occurs.

Second, if a logically negated proposition is found in the text stream, then a failed instance of the rule under examination is built. The remainder of that consequent is skipped, but the consequent propositions already matched are recorded, and another consequent is tried (any consequents in the rule that could not be applied previously are added back to the list of candidate consequents). If no more consequents remain to be tried, then the outcome propositions are negated and matched against the text stream, and a failed instance of the rule is built.

If all the propositions in a consequent match successfully the recorded bindings are passed to the outcome propositions, which are also matched, and the rule being expanded is built (the head is bound and instantiated, and linked to its instantiated body). If this head finishes the path to the root node, then it is returned as value; otherwise, we must build up the next node in the path from the terminals to the root.

At this point a problem arises. Even though a rule has

been built, the procedure may still have chosen the wrong path to the root, and the effects of the build may have to be undone later when this condition is recognized. BUILDTALE has to save the context of the build; it does this by always descending in the LISP stack whenever a build is made that does not terminate a root path. The procedure takes the rule head just built and makes it, effectively, the left-most unmarked proposition in the input text, and then calls itself recursively with the same root as its argument. If, at a later point in the parse, BUILDTALE cannot match further, it will back up to its last build point and undo it, and attempt another path if one exists.

BUILD can also be called at any point in the matching of an antecedent or consequent proposition, when that proposition does not exist either in the database (for antecedents) or in the text stream (for either), but does have rules that can be applied to it. For example, in the rule

(STORY) > ((SETTING) (EPISODE))

after the SETTING has been built, an attempt to match EPISODE in the input stream will (probably) fail, and BUILD will call itself with (EPISODE) as argument. If the build succeeds a derivation tree with an instantiation of EPISODE for root will be returned, and the STORY node can then be built. An outcome proposition cannot avail itself of this

capability, since outcomes cannot contain rules.

Starting with the root of a story tree (STORY) and an unmarked string of text propositions, BUILDTALE executes until one of three conditions occurs:

1) if the procedure exhausts the proposition list before the STORY node has been built, then it fails.
2) if the procedure builds an instance of STORY but there still remain unmarked propositions, then it fails.
3) if an instance of STORY is built and the text stream is empty, the the procedure returns the instantiated root.

A terminal-level traversal of the resulting tree will yield the original input text proposition stream; higher level traversals will yield summaries of the text.

6.0 THE RELATIONSHIP BETWEEN GENERATING AND PARSING

There are several major differences between the BUILDTALE and TELLTALE procedures. First, whereas the parser is restricted to propositions from the input text for its terminals, the generator is free to build a terminal at

any time. Second, the generator is free at each step to choose any rule that will match a proposition it is trying to expand, and also to use any of a rule's antecedents or consequents in any order. The parser must be careful in its choice of rules and in how it examines the antecedents and consequents in a rule, always examining them in order of decreasing length, to insure that it does not build a story tree and find left-over unmarked text propositions when done.

The generator builds a proposition by first instantiating its antecedent, consequent, and outcome, and then attaching them to the instantiated head. The generator knows at all times the father of the proposition it is instantiating; in fact it knows every ancestor of that proposition between it and the root, since it operates strictly top-down.

The parser operates primarily as a left-corner bottom-up procedure, with overall direction supplied by some top-down processing. Therefore, when the parser builds a structure, it cannot be sure at that time that the structure is indeed the correct one to be integrated into the tree at that point, i.e. it does not yet know the correct path to the root. The parser must, therefore, save the information of when, where and in what context it makes its build decisions, so that they can be undone (or at least ignored)

if they are later found to be in error.

Some previous parsers solved this problem by resorting to higher-level languages like CONNIVER and PLANNER, paying the price in higher computational costs. A conscious effort was made in this program to avoid the expense of resorting to a higher-level language by having LISP perform the necessary bookkeeping to handle the backtracking involved in undoing an incorrect choice (build). In BUILDTALE, the bookkeeping is accomplished by pushing context information onto the LISP control stack; usually, when a build is performed , instead of returning (popping the LISP stack), a further descent is made in order to construct the next proposition. If a build is later found to be in error, then the failing function automatically causes LISP to back up in its stack to the point where the build was made and undo it, since all the information that was around when the first decision was made to build is still present on the stack.

These differences should not obscure the very real similarites between the two processes. TELLTALE and BUILDTALE use the same functions to analyze the antecedents, consequents and the outcome of a rule. In fact, the "basic" control structure of TELLTALE is a special case of the control structure of BUILDTALE. The difference between the two occurs at build time. In BUILDTALE, when a node in the tree is built, a check is made to see if this node matches

the root of the derivation tree being built. This may not be the case since the node may be many levels lower in the tree than the root in question, and these levels will need to be built before the derivation tree is complete. Of course, if the node should match the root, then it is returned.

TELLTALE, on the other hand, never descends more than a single level in the tree at a time. When a build is performed, it will always be the root of the derivation tree being processed. The node and the root always match, and the node is returned. At build time, when BUILDTALE decides whether to recur (call itself to add the next higher level in the tree) or to pop the stack (returning the derivation tree root), TELLTALE will always pop the stack.

This implies that generation is a special case of parsing. This fact was borne out during the implementation of the system. TELLTALE was the first procedure to be written, and the eventual TELLTALE/BUILDTALE control structure for processing antecedents, consequents, and outcomes was debugged and tested by generating many story trees. BUILDTALE grew out of TELLTALE by adding the build-time recursion/backup mechanism to the control structure.

The relationship between generation and parsing is one

of the most significant features to grow out of the EHC rule system.

7.0 EXTRACTING SUMMARIES FROM STORY TREES

One of the principle reasons for the choice of the Kintsch and van Dijk macrostructure theory was the resulting property of summarizability; the ability to produce coherent summaries is one mark of intelligence in a parser. The summarizer produces various strings of propositions from the story tree. One such string is composed of the terminals and represents the complete story. Any sequence of propositions output by the summarizer is a well-formed input to the parser. The system is thus able to parse all proposition sequences it can generate.

Since the summary feature is inherent in the trees as they are given to the summarizer, a simple level traversal algorithm would have been sufficient to generate useful output. However, this would have resulted in needless repetition of propositions (since some are checked repeatedly by the antecedents of the rules and have pointers to them at many places in the tree). Therefore, the summarizer remembers what it has already output in the summary, so as never to repeat itself.

Another area of repetition is in the outputting of attributes for objects in the story. To avoid repeating an object's attributes, the summarizer keeps a list of objects that have already appeared in at least one proposition, and whenever it encounters in a new proposition an object not on this list, it outputs it with all of its properties and then places it on the list of expanded objects. Since no time markers are put on an object's properties, they are all printed out at once, even if some of those properties are not attached to the object until much later in the story; this is a weakness in the procedure that can be corrected by the introduction of time-markers.

One attribute of story trees is that, at their higher nodes, one can read off the structure of the story. For example some story consists of a setting followed by three episodes. As a summary, "setting plus three episodes" is not very illuminating; therefore the summarizer has the ability to recognize and not output these story structure nodes in the final summary. Story structure nodes are those nodes whose proposition head (not to be confused with rule head) are marked with a number sign (see examples in Appendix [A]). The nodes are treated like all other nodes to the tree building procedures, but the summarizer uses the number sign cue to always descend below the nodes so marked to print their descendents, no matter what level summary is being computed.

There is also the question of summarizing the macrostructures, which are nodes marked with double asterisks (see examples in Appendices). By definition, these nodes are expandable, i.e., they have a rule for expanding them in the rule-base). Macrostructures are not marked with a NOT if they fail; only simple propositions - terminals - are. However, whether a script achieved its goal or not is vital information to be included in any reasonable summary produced from the tree. Therefore, when summarizing a macrostructure, the summarizer outputs both the head (the macrostructure pattern) and its outcome. If the script failed to achieve its goal, the outcome will be negated.

Finally, the summarizer outputs only those propositions it recognizes as "actives" (those whose tokens are marked with a terminal asterisk); it nevers outputs "statives" (nodes without the asterisk or number sign markers). The reason for this is that a stative always describes the attribute(s) of some object, and can therefore be outputted the first time that that object appears in an active proposition.

The summarizer is an algorithm that is given a story tree and a level indicator, scans the nodes of the tree at that level, and applies the following rules to each node:

1) if the node is marked with a number sign, then

summarize its sons.

2) if the node has already been outputted, then skip it.

3) if the node is marked as a script, output its head followed by its outcome propositions.

4) if the node is a stative, then skip it.

For each object in a proposition to be output, the following rule is applied:

5) if the object has not appeared in a previously outputted proposition print it and all its attributes; otherwise, just print the node.

Some example summaries appear in Appendices [C, E].

An initial version of an English language generator has been written that applies a set of rules to the output of the summarizer to produce well-formed English texts (Hare and Correira, 1978). This generator uses the rule forms and the nature of the summarizer output to produce English texts displaying the attributes of: reasonable paragraphing and sentence connectivity, elision of noun groups and verbs, and pronominalization of nouns.

8.0 DISCUSSION AND CONCLUSIONS

The task of text processing requires solutions to several important problems. The computational theory for macrostructures using Extended Horn Clauses was designed with the following goals in mind. A computational model should have a degree of psychological validity, both to provide a humanly useful representation of textual content and organization and to insure that the task of rule-writing is as natural as possible for the human grammar-producer. It should be conceptually simple, in both design and implementation, taking advantage of similarities between generation and parsing, and it should offer a rigorous data structure that is uniform and avoids the growth of ad hoc large-scale structures.

The computational macrostructures realized by the EHC notation succeed in many ways in satisfying these goals. They are based on a theory of macrostructures that, as Kintsch and van Dijk have given evidence for, appears to resemble the processing performed by human readers above the sentential level for texts. The resulting grammars are not dictated or constrained by the syntax of the EHC so much as they are by the experiential bias of the rule-writer.

The Story Tree is proposed as a logical way to organize these macrostructures, with the terminals of a particular story tree comprising the actual textual propositions, and the interior nodes containing the instantiated heads of

rules (corresponding to macrostructures). The story tree has the summary property; if the tree is truncated at any level, then a "meaningful" (coherent) summary of the original text can be read off directly. The generality of the macrostructure propositions increases as one nears the level of the root (going from the level of specific actions to the scripts that contain them, to the story categories that contain these scripts), which can be considered as the title for the text at its terminals.

The concept of rule failure takes the EHC out of the strictly logical system of the normal (Kowalski-type) Horn clause logic, since failure in a normal logic system means something different from failure here. In narratives, failure should be recorded, since it is one source of "interest" in the resulting story; striving, failing, and striving again is a common attribute of simple narratives. These failure points, and their consequences, have to be recorded in the story tree (whereas, in normal logic systems, failure points are invisible to the final result) and, furthermore, they restrict the choice of paths that can reasonably be expected to emanate from the failure.

This failure mechanism is based on a rather simplistic model of human behavior, where individuals act on single motives, pursued sequentially. A more comprehensive model, and one necessary for processing longer text structures such

as novels would include facilities for complex and interacting motivations, as well as rules for ordering them into a hierarchy. It would also allow for shifting the narrative focus from one character's motivation structure to another's. Finally, the failure mechanism is tailored to the purpose of narratives involving entities exhibiting motivated behavior. Other text forms, such as technical or encyclopedia articles, would probably not require the failure mechanism to be parsed or generated.

As a result of programming experience the relationship between the processes of generating and parsing text was noted. The TELLTALE/BUILDTALE procedures form a single program with a decision switch for choosing between modes. The major computational difference between them stems from the nature of the two modes, TELLTALE being entirely a top-down process, and BUILDTALE being mostly bottom-up, which requires that BUILDTALE incorporate a back-up facility in case of error. This back-up capability was successfully embedded into LISP, making it unnecessary to resort to a higher-level language, saving time, space and complexity in the program.

The underlying approach in the program is that of a problem-solver, as was also true of Meehan's story-writer. A rule-base, organized as a Story Tree, is used to generate a particular, instantiated, story tree by an augmented (via

the failure mechanism) inference procedure. Each instantiated tree is treated as a context, consisting of the events, objects, and their relationships, relating to a particular story. The facts and rules in the rule-base serve as a model for the possible states in the microworld formed by that story tree. These states are categorized using standard linguistic entities, such as SETTING, EPISODE, COMPLICATION, and RESOLUTION.

The problem-solving approach, coupled with the story grammar concept, is a natural one for processing most forms of narratives. Analogous systems of rules could be used for processing other large text forms, although the categories involved would be different. For example, one template for an encyclopedia article can be described by the rule

(ARTICLE) > ((INTRODUCTION)(OUTLINE)(EXPANSION)(SUMMARY)).

By changing the rules describing the overall syntactic structure of the text, and adding the necessary specific knowledge, the same techniques could be used to generate and parse encyclopedia articles.

A summarizing algorithm was also described for the story tree. Rather than simply reading off left-to-right the propositions at a level in the tree, extra rules were added to allow the summarizer to avoid clumsiness in its output. By avoiding the repetition of antecedent

propositions that are constantly being checked, the summarizer streamlines its summaries and also aids in the TELLTALE-to-BUILDTALE processing chain by making it unnecessary to check for redundant information when BUILDTALE is making its tree. The summarization of a macrostructure consists of the head of the rule embodying it, plus the outcome of the rule. This enables the reader to determine whether the rule succeeded or failed in achieving its goal.

The result is a procedure capable of understanding wexts that it can generate. One way of improving the system is to allow partial, or "fuzzy" matching of rules to input text proposition sequences, since it is frequently the case the macrostructures are incompletely realized in real life, and it is also the case that human story-writers will omit some sequences of propositions, provided they are extremely familiar to the expected reader and need not be specified. This capability would require that the parser be able to distinguish those propositions in a rule that are critical to its presence being detected in a piece of text, and those that can be missing without effecting the presence of a rule.

One prominant feature of the generating and parsing procedures is the amount of variable binding that takes place. Variables are bound forward when a rule is invoked;

each proposition in the antecedent, consequent and outcome of a matching rule have to be bound forward, and also contribute bindings of their own to the ongoing matching process. Finally, the results of all this matching are bound back into the rule head before it can be returned. In fact, calculating bindings and passing them around occupied a goodly percentage of the total computational effort in a program run. The problem (such as it is) stems from the use of purely local variables in all rules in the rule-base. Some sort of global variable feature, such as the register notion in the ATN parser could cut processing time considerably. Registers could also reduce the amount of redundant testing of simple conditions in the antecedents of the rules.

A reasonable extension to BUILDTALE would be to enable it to parse texts that it cannot generate (at least not prior to first encountering the text). One solution path for this goal is to give the BUILDTALE procedure the capability of making up its own rules. This is primarily a problem of deciding coherence, i.e. when a proposition fits into a growing rule structure (a potential macrostructure), as well as when a rule begins and ends. This task remains to be solved before a truly general text parser can be achieved.

NOTES ON THE APPENDICES

Several facts concerning the Appendices should be noted. The rules are all written in a case predicate notation (Simmons [1977]). The general form for such a predicate is

(HEAD ARC1 VALUE1 ... ARCn VALUEn)

The HEAD of a case predicate is either a verb form or an object; because no formal lexicon was maintained for the TELLTALE/BUILDTALE program, verb forms were marked with an asterisk and objects were left unmarked. The ARCs are standard case relations, such as Agent, THeme, LOCation, INSTance, etc., although no attempt was made to be strictly correct linguistically with their every use, and a few relations were created for convenience sake. The VALUE can be either a verb form, an object, or another case predicate.

The case predicates used in the program were written to enhance readability. For example, in the fairytale story (Appendix [B]), the case predicate

(WANT*1 TO POSSESS1 A GEORGE1 TH MARY1)

can be rendered into English as "George wants to possess Mary". The sequence

(GO*3 A GEORGE1 TH IRVING1)

(SLAY*1 A GEORGE1 TH IRVING1)

(RESCUE*1 A GEORGE1 TH IRVING1)

can be rendered as "George goes to Irving. George slays Irving. George rescues Mary.".

In some instances, information was added to a predicate because it was necessary for the English language text generator (Hare and Correira). For example, the initial predicate in the fairytale story,

(LIVE*2 A (GEORGE ISA KNIGHT1 SEX MALE2 PERSON T MOD BRAVE1) LOC (CAMELOT1 ISA PLACE1) DURING (ONCEUPONATIME1 ISA TIME1))

can be rendered into English as "Once upon a time in a place called Camelot there lived a brave knight named George." The case arcs PERSON and SEX were used in the English generator to determine pronouns and capitalization. These markers do not occur in the Rufolo story (Appendices [D, E]) because it was never run through the English generator.

Two other markers were also used with the verb forms. The first, the number sign, was used by the summarizer and its effect is described in section 7; the number sign was treated like an asterisk (verb form) by the program. The second notation, the double asterisk, was a device used by the author to denote macrostructures (propositions that were expandable) from the simple text propositions (marked by the single asterisk); they were treated as verb forms by the

program.

Finally, in several places, there occur variables marked with an exclamation point (!). This notation was used in rules containing more than one instance of objects from a single class, where the marked objects had to be replaced by objects different from any other object used to fill the other variable positions for that class in the rule.

APPENDIX A

RULE-BASE FOR FAIRYTALE

```
((FAIRYTALE*) > ((FAIRYSTORY** A G0241 TH G0242 )))
((FAIRYSTORY** A G0241 TH G0242 )
>
((SETTING  A G0241)
(EPISODE  A G0241 TH G0242))
>> ((LIVE* A G0241 TH G0242 MANNER HAPPILYEVERAFTER)))
((SETTING  A G0244) <  ((LIVE*  A  G0244  LOC  G0248  DURING
G0249)))
((LIVE* A G0245 LOC  G0246  DURING  G0247)  <  ((CHAR   INST
G0245)  (G0246 ISA PLACE)  (G0247 ISA TIME)))
((CHAR  INST G0248)
<
((G0248 ISA KNIGHT SEX MALE PERSON T))
<
((G0248 ISA PRINCE SEX MALE PERSON T))
>
((G0248 MOD BRAVE))
>
((G0248 MOD HANDSOME)))
((EPISODE  A G0249 TH G0250) < ((MOTIVE  A G0249  TH  G0250)
(ACTION  A G0249 TH G0250)))
((MOTIVE  A G0251 TH G0252) < ((DESIRE* A G0251 TH G0252)) >
((WANT* TO* POSSESS A G0251 TH G0252)))
((DESIRE* A G0253 TH G0254)
<
((CHAR  INST G0253) (G0254 ISA PRINCESS SEX FEMALE PERSON T)
(G0254 MOD BEAUTIFUL))
<
((CHAR  INST G0253) (G0254 ISA HOLYOBJECT POBJ T) (G0254 MOD
LOST)))
((ACTION  A G0255 TH G0256)
>
((ASK** TO* MARRY A G0255 TH G0256 ))
>
((RESCUE** A G0255 TH G0256 FROM G0262 ))
>
((QUEST** A G0255 TH G0256 ))
>
((PRAY** PART FOR A G0255 TH G0256 )))
((ASK** TO* MARRY A G0257 TH G0258 )
```

```
<
((WANT* TO* POSSESS A G0257 TH G0258)  (G0258  ISA  PRINCESS
SEX FEMALE PERSON T))
>
((GO* PART TO A G0257 TH G0258)
(ASK* A G0257 TH G0258 IF (MARRY* A G0258 TH G0257))
(ACCEPT* A G0258 TH G0257))
>> ((MARRY*!  A G0258 TH G0257)))
((RESCUE** A G0260 TH G0261 FROM G0262 )
<
((WANT* TO* POSSESS A G0260 TH G0261) (G0261  ISA  PRINCESS)
(THREATEN** A G0262 TH G0261))
>
((GO* PART TO A G0260 TH G0262)
(SLAY* A G0260 TH G0262)
(RESCUE* A G0260 TH G0261))
>> ((MARRY* A G0261 TH G0260)))
((THREATEN** A G0264 TH G0265)
<
((G0264 ISA DRAGON ANIMATE T) (G0264 MOD  EVIL)  (G0265  ISA
PRINCESS) (WANT* TO* POSSESS A G0264 TH G0265))
>
((CARRY** PART OFF A G0264 TH G0265 TO G0273)))
((CARRY** PART OFF A G0266 TH G0267 TO G0285)
<
((G0266 ISA DRAGON) (G0267 ISA PRINCESS)(G0285 ISA DEN  POBJ
T))
>
((GO* PART TO A G0266 TH G0267) (CAPTURE* A G0266 TH  G0267)
(FLY* A G0266 TH G0267 PREP (TO!  TH G0285))))
((QUEST** A G0268 TH G0269 )
<
((CHAR  INST G0268) (G0269 ISA HOLYOBJECT MOD LOST))
>
((GO* PART TO A G0268 TH ORACLE)
(REVEAL* A ORACLE TH PLACE OF G0269)
(G0269 LOC PLACE)
(GO* PART TO A G0268 TH PLACE)
(FIND* A G0268 TH G0269))
>> ((POSSESS* A G0268 TH G0269)))
((PRAY** PART FOR A G0271 TH G0272 )
<
((CHAR  INST G0271) (WANT* TO* POSSESS  A  G0271  TH  G0272)
(G0283 ISA GOD PERSON T SEX MALE)
(G0282 ISA CHURCH POBJ T))
>
((GO* PART TO A G0271 TH G0273)
(KNEEL* A G0271 PREP (IN TH (FRONT PREP (OF TH ALTER))))
(PRAY* A G0271 PREP (TO TH G0283) PREP (FOR TH G0272))
(GRANT* A G0283 TH (PRAYER POSSBY G0271)))
>
( (G0286 ISA PRIEST SEX MALE PERSON T)
(GO* PART TO A G0271 TH G0273)
(PAY* A G0271 TH G0286 EXPECT (INTERCEDE* A G0286 PREP (WITH
TH G0283) PREP (FOR TH G0271)))
```

```
(PRAY* A G0286 TO G0283 FOR G0272)
(GRANT* A G0283 TH (PRAYER POSSBY G0286)))
>> ((POSSESS* A G0271 TH G0272)))
(JOHN ISA PRINCE SEX MALE PERSON T)
(GEORGE ISA KNIGHT SEX MALE PERSON T)
(PRINCECHARMING ISA PRINCE SEX MALE PERSON T)
(LANCELOT ISA KNIGHT SEX MALE PERSON T)
(PARSIFAL ISA KNIGHT SEX MALE PERSON T).
(MARY ISA PRINCESS SEX FEMALE PERSON T)
(GUENEVIERE ISA PRINCESS SEX FEMALE PERSON T)
(HOLYGRAIL ISA HOLYOBJECT POBJ T)
(SACREDCROSS ISA HOLYOBJECT MOD PIECE POBJ T)
(CAMELOT ISA PLACE)
(MONTSALVAT ISA PLACE)
(ONCEUPONATIME ISA TIME)
(IRVING ISA DRAGON ANIMATE T)
(HERMAN ISA DRAGON ANIMATE T)
(CARMEN ISA DRAGON ANIMATE T)
```

APPENDIX B

TEXT OF FAIRYTALE

```
(FAIRYTALE*1)

((LIVE*2 A (GEORGE1 ISA KNIGHT1 SEX MALE2 PERSON T MOD
BRAVE1)
LOC (CAMELOT ISA PLACE1) DURING (ONCEUPONATIME1 ISA TIME1))
(DESIRE*2 A GEORGE1 TH (MARY1 ISA PRINCESS1 SEX FEMALE1
PERSON T
MOD BEAUTIFUL1))
(WANT*1 TO POSSESS1 A GEORGE1 TH MARY1)
(GO*1 PART TO1 A GEORGE1 TH MARY1))
(ASK*1 A GEORGE1 TH MARY1 IF (MARRY*1 A MARY1 TH GEORGE1))
(NOT OF (ACCEPT*1 A MARY1 TH GEORGE1))
(NOT OF (MARRY*2 A MARY1 TH GEORGE1))
(WANT*2 TO POSSESS2 A (IRVING ISA DRAGON1 ANIMATE T MOD
EVIL1) TH MARY1)
(GO*2 PART TO3 A IRVING1 TH MARY1)
(CAPTURE*1 A IRVING1 TH MARY1)
(FLY*1 A IRVING1 TH MARY1 PREP (TO TH (DEN1 ISA DEN2 POBJ
T)))
(GO*3 PART TO3 A GEORGE1 TH IRVING1)
(SLAY*1 A GEORGE1 TH IRVING1)
(RESCUE*1 A GEORGE1 TH MARY1)
(MARRY*4 A MARY1 TH GEORGE1)
(LIVE*3 A GEORGE1 TH MARY1 MANNER HAPPILYEVERAFTER))
```

APPENDIX C

SUMMARIES OF FAIRYTALE

```
(FAIRYTALE*1)

((LIVE*2 A (GEORGE1 ISA KNIGHT1 SEX MALE2 PERSON T MOD
BRAVE1)
LOC (CAMELOT1 ISA PLACE1) DURING (ONCEUPONATIME1 ISA TIME1))
(DESIRE*2 A GEORGE1 TH (MARY1 ISA PRINCESS1 SEX FEMALE1
PERSON T
MOD BEAUTIFUL1))
(WANT*1 TO POSSESS*1 A GEORGE1 TH MARY1)
(GO*1 PART TO1 A GEORGE1 TH MARY1)
(ASK*1 A GEORGE1 TH MARY1 IF (MARRY*1 A MARY1 TH GEORGE1))
(NOT OF (ACCEPT*1 A MARY1 TH GEORGE1))
(NOT OF (MARRY*2 A MARY1 TH GEORGE1))
(WANT* TO POSSESS2 A (IRVING1 ISA DRAGON1 MOD EVIL1) TH
MARY1)
(CARRY**2 PART OFF1 A IRVING1 TH MARY1 TO DEN1)
(GO*3 PART TO3 A GEORGE1 TH IRVING1)
(SLAY*1 A GEORGE1 TH IRVING1)
(RESCUE*1 A GEORGE1 TH MARY1)
(MARRY*4 A MARY1 TH GEORGE1)
(LIVE*3 A GEORGE1 TH MARY1 MANNER HAPPILYEVERAFTER1))


(FAIRYTALE*1)

((LIVE*2 A (GEORGE1 ISA KNIGHT1 SEX MALE2 PERSON T MOD
BRAVE1)
LOC (CAMELOT1 ISA PLACE1) DURING (ONCEUPONATIME1 ISA TIME1))
(DESIRE*2 A GEORGE1 TH (MARY1 ISA PRINCESS1 SEX FEMALE1
PERSON T
MOD BEAUTIFUL1))
(WANT*1 TO POSSESS1 A GEORGE1 TH MARY1)
(ASK**2 TO MARRY1 A GEORGE1 TH MARY1)
(NOT OF (MARRY*2 A MARY1 TH GEORGE1))
(RESCUE**2 A GEORGE1 TH MARY1 FROM (IRVING1 ISA DRAGON1
ANIMATE T
MOD EVIL1))
(MARRY*4 A MARY1 TH GEORGE1)
(LIVE*3 A GEORGE1 TH MARY1 MANNER HAPPILYEVERAFTER1))
```

APPENDIX D

RUFOLO STORY

```
(LIVE*2 A (RUFOLO1 ISA PERSON1 FIRSTNAME1 (LANDOLFO2 ISA
     FIRSTNAME1)) LOC (RAVELLO1 CONT (AND1 OF (MERCHANT1 ISA
     MERCHANT2 MOD (AND2 OF (RIC H1) (ENTERPRISING1)) NBR
     MANY2 LOC RAVELLO1) (GARDEN1 ISA GARDEN2 NBR MANY3 LOC
     RAVELLO1) (FOUNTAIN1 ISA FOUNTAIN2 NBR MANY4 LOC
     RAVELLO1)) PARTOF (TOWN1 ISA TOWN2 NBR MANY1 MOD SMALL1
     LOC (COAST1 ISA COAST2 MO D BEAUTIFUL1 BETWEEN (AND3 OF
     (REGGIO1) (GAETA1)) HASPRT (REGION1 LOC (SALERNO1 PROX
     NEAR1) NAME AMALFI1) LOC ITALY1)) HOMEOF RUFOLO1) DURIN
     G (TIME1 ISA TIME2))
(POSSESS*1 A RUFOLO1 TH (WEALTH1 ISA WEALTH2 VAL
     CERTAINAMOUNT1))
(SATISFY*1 A RUFOLO1 TH WEALTH1 MODAL NOT1)
(WANT*1 A RUFOLO1 TH (DOUBLE*1 A RUFOLO1 TH WEALTH1))
(MAKE*1 A RUFOLO1 TH (CALCULATIONS1 MOD MERCHANTS1 TYPE
     USUAL1))
(BUY*1 A RUFOLO1 TH (SHIP1 ISA SHIP2))
(BUY*2 A RUFOLO1 TH (GOODS1 ISA GOODS2))
(LOAD*1 A RUFOLO1 TH SHIP1 WITH GOODS1)
(SAIL*1 A RUFOLO1 TH SHIP1 TO (CYPRUS2 ISA ISLAND1))
(DISCOVER*1 A RUFOLO1 TH (SHIP3 NBR MANY5 MOD DOCKED1 POSSBY
     (MERCHANT 3 NBR SOME1 MOD OTHER1)))
(CARRY*1 INSTR SHIP3 TH (GOODS3 SAMEAS GOODS1))
(NOT OF (SELL*1 A RUFOLO1 TH GOODS1 FOR (PRICE1 ISA PRICE2
     MOD GOOD1)) )
(NOT OF (MAKE*2 A RUFOLO1 TH (PROFIT1 MOD GREAT1)))
(BRING*1 INSTR COMPETITION1 TH RUFOLO1 TO (RUIN1 MOD
     VERGE1))
(REDUCED*1 TH RUFOLO1 TO SEMI-POVERTY1)
(DISTRESSED*1 TH RUFOLO1)
(DECIDE*1 A RUFOLO1 TH (OR0 OF (MAKEGOOD*1 A RUFOLO1 TH
     LOSSES1 MEANS PRIVATEERING1) (DIE*1 TH RUFOLO1)))
(SETOUT*1 A RUFOLO1 MANNER RICH2 FROM RAVELLO1 CAUSE (WANT*2
     A RUFOLO1 TH (RETURN*1 A RUFOLO1 TO RAVELLO1 MANNER
     POOR1 MODAL NOT2)))
(SELL*2 A RUFOLO1 TH SHIP1)
(BUY*3 A RUFOLO1 TH (SHIP4 ISA SHIP5 TYPE PIRATE1 MOD
     LIGHT1))
(FITOUT*1 A RUFOLO1 TH SHIP4 WITH (EQUIPMENT1 MOD
     BESTSUITED1))
```

```
(SEIZE*1 A RUFOLO1 TH (SHIP6 NBR MANY6 POSSBY (TURK2 ISA
     NATIONALITY1) ))
(DOUBLE*2 A RUFOLO1 TH WEALTH1)
(POSSESS*2 A RUFOLO1 TH (WEALTH3 ISA WEALTH4 VAL (TWICE*1 R1
     WEALTH3 R 2 WEALTH1)))
(PERSUADE*1 A RUFOLO1 TH (SATISFIED*1 A RUFOLO1 TH WEALTH3))
(DECIDE*2 A RUFOLO1 TH (GOHOME*1 A RUFOLO1 TH WEALTH3))
(SAIL*2 A RUFOLO1 TH WEALTH3 FOR RAVELLO1 MEANS SHIP4)
(PROPEL*1 A (ROWERS1 MOD VIGOROUS1) TH SHIP4)
(BLOW*1 INSTR GALE1 DURING EVENING1 LOC ARCHIPELAGO1)
(PUTINTO*1 A RUFOLO1 TH SHIP4 LOC (BAY1 ISA BAY2 LOC
     (ISLAND2 MOD SMAL L2)))
(PUTINTO*2 A (GENOESE1 ISA GENOESE2 NBR SEVERAL1 MOD (AND4
     OF (RAPACIO US1) (MONEYGRABBING1)) FROM
     CONSTANTINOPLE1) TH (CARRACK1 ISA CARRACK2 NBR TWO1
     POSSBY GENOESE1) LOC BAY1)
(BLOCK*1 A GENOESE1 TH (PATH1 MOD ESCAPE1 OF RUFOLO1))
(DESIRE*1 A GENOESE1 TH (AND5 OF (RUFOLO1 ISA PERSON1
     FIRSTNAME1 LANDO LFO2) (SHIP4 ISA SHIP5 TYPE PIRATE1
     MOD LIGHT1)) RESULTOF (REALIZE*1 A GENOESE1 THAT
     (POSSESS*3 A RUFOLO1 LOC SHIP4 TH (LOOT1 MOD MUCH1))))
(SURROUND*1 A (PARTY1 COMMANDBY GENOESE1) TH SHIP4)
(POSSESS*4 A PARTY1 TH (AND6 OF (CROSS-BOWS1) (WEAPONS1 MOD
     DEFENSE1)) )
(LAUNCH*1 A GENOESE1 TH (CUTTER1 NBR SEVERAL2))
(DRAW*1 INSTR CURRENT1 TH CUTTER1 TO SHIP4)
(CAPTURE*1 A (MEN1 LOC CUTTER1) TH SHIP4 AIDBY (ROWER1 NBR
     MANY7 OF SH IP4) RESULT (ESCAPE*1 TH NOONE1))
(PUTABOARD*1 A MEN1 TH RUFOLO1 LOC (CARRACK3 MEMBEROF
     CARRACK1 CONT RU FOLO1))
(ROB*1 A MEN1 TH SHIP4 MOD FULLY1)
(SINK*1 A MEN1 TH SHIP4)
(WANT*3 A RUFOLO1 TH (ESCAPE*2 A RUFOLO1 FROM GENOESE1))
(SAIL*3 A GENOESE1 TH RUFOLO1 FOR DESTINATION1 MEANS
     CARRACK1)
(BLOW*3 INSTR GALE3 DURING EVENING3)
(SEPARATE*1 INSTR GALE3 TH CARRACK1)
(THROW*1 INSTR GALE3 TH CARRACK3 ONTO (SANDBANK1 LOC
     CEPHALONIA1))
(CRASH*1 TH CARRACK3 LIKE (THROW*2 TH BOTTLE1 LOC WALL1))
(STREWABOUT*1 INSTR GALE3 TH (AND7 OF (PLANKS1) (CHESTS1)
     (SPARS1)))
(TRY*1 TO (AND10 OF (GRAB1) (CLINGTO1)) A (AND11 OF
     (EVERYONE1) (RUFOL O1 ISA PERSON1 FIRSTNAME1
     LANDOLFO2)) TH WRECKAGE1)
(FRIGHTEN*1 TH RUFOLO1)
(CLING*1 A RUFOLO1 TH SPAR1)
(WANT*4 A RUFOLO1 TH (RESCUE*1 A GOD1 TH RUFOLO1 MANNER
     SOMEHOW1))
(SET*1 A RUFOLO1 TH SPAR1 MANNER ASTRIDE1)
(TOSS*1 A (AND12 OF (SEAS1) (WINDS1)) TH RUFOLO1 MANNER
     (AND13 OF (HIT HER1) (THITHER1)) UNTIL DAYBREAK1)
(SEE*1 A RUFOLO1 TH (AND14 OF (WATER1) (CLOUDS1) (CHEST2
     PROPERTYOF RU FOLO1)))
(FLOAT*1 TH (CHEST2 PROPERTYOF RUFOLO1) LOC (VICINITY1 OF
```

```
     RUFOLO1) MAN NER EVERYSOOFTEN1)
(FEAR*1 A RUFOLO1 THAT (AND15 OF (COLLIDE*1 TH CHEST2 WITH
     RUFOLO1) (H URT*1 INSTR CHEST2 TH RUFOLO1)) SO (PUSH*1
     A RUFOLO1 TH CHEST2 AWAYFRO M RUFOLO1))
(SEND*1 A (SQUALL1 MOD SUDDEN1) TH CHEST2 INTO RUFOLO1
     MANNER HARD1)
(OVERTURN*1 TH SPAR1 RESULT (GOUNDER*1 TH RUFOLO1))
(SWIM*1 A RUFOLO1 TH CHEST2 REASON (TOOFAR*1 TH SPAR1))
(DRAG*1 A RUFOLO1 TH RUFOLO1 ONTO CHEST2)
(SPRAWLACROSS*1 A RUFOLO1 TH CHEST2 RESULTOF (HOLD*1 A
     RUFOLO1 TH CHES T2 MANNER STEADY1 MEANS (ARMS1 POSSBY
     RUFOLO1)))
(SURVIVE*1 A RUFOLO1 DURING (AND16 OF (DAY1 MOD WHOLE1)
     (NIGHT1 MOD FO LLOWING1)))
(STARVE*1 TH RUFOLO1)
(THIRST*1 TH RUFOLO1)
(ARRIVE*2 A RUFOLO1 LOC (BEACH2 LOC (ISLAND4 NAME CORFU2))
     RESULTOF (O R1 OF (GOD2) (POWER1 OF WIND1)))
(CLEAN*1 A (WOMAN1 ISA WOMAN2) TH (POT1 NBR MANY10 POSSBY
     WOMAN1) MEAN S (AND17 OF (SAND1) (SALTWATER1)))
(SEE*2 A WOMAN1 TH RUFOLO1 RESULT (ALARMED*1 TH WOMAN1))
(RECOGNIZE*1 A WOMAN1 THAT (BE*1 A RUFOLO1 TH MAN1))
(WADE*1 A WOMAN1 INTO SEA1)
(DRAG*2 A WOMAN1 TH RUFOLO1 TO SHORE1 MEANS (HAIR1 POSSBY
     RUFOLO1))
(PLACE*1 A WOMAN1 TH CHEST2 ONTO (HEAD1 POSSBY (DAUGHTER1 OF
     WOMAN1)))
(CARRY*2 A WOMAN1 TH RUFOLO1 TO VILLAGE1 MANNER GENTLY1)
(PUT*1 A WOMAN1 TH RUFOLO1 INTO (BATH1 MOD HOT1))
(RUB*1 A WOMAN1 TH RUFOLO1 WITH (WATER2 MOD HOT1))
(THAWOUT*1 TH RUFOLO1)
(TAKE*1 A WOMAN1 TH RUFOLO1 OUTOF BATH1)
(RECOVER*1 A RUFOLO1 TH (STRENGTH1 MOD SOME2))
(GIVE*1 A WOMAN1 TO RUFOLO1 TH (AND20 OF (WINE1) (FOOD1 MOD
     SWEET1)))
(NURSE*1 A WOMAN1 TH RUFOLO1 DURING (DAY2 NBR SEVERAL3))
(RECOVER**1 A RUFOLO1 TH (STRENGTH2 MOD REMAINDER1))
(GIVE*2 A WOMAN1 TH CHEST2 TO RUFOLO1)
(TELL*1 A WOMAN1 TH RUFOLO1 THAT (LEAVE*1 TH RUFOLO1 MODAL
     SHOULD1))
(REMEMBER*1 A RUFOLO1 TH CHEST2 MODAL NOT3 BUT (ACCEPT*1 A
     RUFOLO1 TH CHEST2))
(HOPE*1 A RUFOLO1 TH (CONTAIN*1 LOC CHEST2 TH (VALUABLE1 MOD
     SOMEWHAT1 )))
(DISCOVER*2 A RUFOLO1 TH (BE*2 TH CHEST2 WEIGHT LIGHT2)
     RESULT (DISAPP OINTED*1 TH RUFOLO1))
(FORCE*1 A RUFOLO1 TH CHEST2 RESULT (OPEN*1 TH CHEST2)
     DURING (ATHOME* 1 TH WOMAN1 MODAL NOT4))
(DISCOVER*3 A RUFOLO1 TH (CONTAIN*2 LOC CHEST2 TH STONE2))
(HAVE*1 A RUFOLO1 TH (KNOWLEDGE1 OF JEWELS1))
(REALIZE*2 A RUFOLO1 TH (VALUE1 OF STONE2))
(PRAISE*1 A RUFOLO1 TH (GOD3 ISA GOD4) FOR (COME*1 A GOD3 TH
     RESCUE*2 OF RUFOLO1))
(WANT*5 A RUFOLO1 TH (RETURN*2 A RUFOLO1 TO RAVELLO1 WITH
     STONE2))
```

```
(WRAP*1 A RUFOLO1 TH VALUABLE1 IN (RAGS1 MOD OLD1))
(EXCHANGE*1 A RUFOLO1 TH CHEST2 WITH WOMAN1 FOR SACK1)
(THANK*1 A RUFOLO1 TH WOMAN1 FOR SERVICES1)
(TAKE*2 A RUFOLO1 TH BOAT1 TO BRINDISI1)
(GOALONG*1 A RUFOLO1 TH COAST3 TO TRANI1)
(MEET*1 A RUFOLO1 TH (MERCHANTS2 MOD CLOTH1 FROM RAVELLO1))
(TELL*2 A RUFOLO1 TH MERCHANTS3 ABOUT (ADVENTURES1 OF
     RUFOLO1 BUT (MEN TION*1 A RUFOLO1 TH CHEST2 MODAL
     NOT5)))
(FEEL*1 A MERCHANTS3 TH RUFOLO1 MANNER SORRY1)
(GIVE*3 A MERCHANTS3 TO RUFOLO1 TH CLOTHES1)
(LEND*1 A MERCHANTS3 TH HORSE1 TO RUFOLO1)
(SEND*2 A MERCHANTS3 TH RUFOLO1 TO RAVELLO1 WITH COMPANY1)
(ARRIVE*3 A RUFOLO1 TH RAVELLO1)
(INSPECT*1 A RUFOLO1 TH (CONTENT1 OF SACK1))
(REALIZE*3 A RUFOLO1 IF (SELL*3 A RUFOLO1 TH VALUABLE1 THAT
     (DOUBLE*3 A RUFOLO1 TH WEALTH1)))
(SELL*4 A RUFOLO1 TH VALUABLE1 FOR WEALTH2)
(SEND*3 A RUFOLO1 TH (SUM1 MOD TIDY1 PARTOF WEALTH2) TO
     WOMAN1 REASON (REWARD*1 A RUFOLO1 TH WOMAN1))
(SEND*4 A RUFOLO1 TH (SUM2 PARTOF WEALTH2) TO MERCHANTS3
     REASON (REWAR D*2 A RUFOLO1 TH MERCHANTS3))
(KEEP*1 A RUFOLO1 TH (REMAINDER2 OF WEALTH2))
(INTEREST*1 A RUFOLO1 TH COMMERCE1 MODAL NOLONGER1)
(LIVE*3 A RUFOLO1 MANNER SPLENDOR1 DURING (REMAINDER3 OF
     (DAY3 OF RUFO LO1)))
```

APPENDIX E

A SUMMARY OF THE RUFOLO STORY

```
(LIVE*2 A (RUFOLO1 ISA PERSON1 FIRSTNAME1 (LANDOLFO2 ISA
     FIRSTNAME1)) LOC (RAVELLO1 CONT (AND1 OF (MERCHANT1 ISA
     MERCHANT2 MOD (AND2 OF (RIC H1) (ENTERPRISING1)) NBR
     MANY2 LOC RAVELLO1) (GARDEN1 ISA GARDEN2 NBR MANY3 LOC
     RAVELLO1) (FOUNTAIN1 ISA FOUNTAIN2 NBR MANY4 LOC
     RAVELLO1)) PARTOF (TOWN1 ISA TOWN2 NBR MANY1 MOD SMALL1
     LOC (COAST1 ISA COAST2 MO D BEAUTIFUL1 BETWEEN (AND3 OF
     (REGGIO1) (GAETA1)) HASPRT (REGION1 LOC (SALERNO1 PROX
     NEAR1) NAME AMALFI1) LOC ITALY1)) HOMEOF RUFOLO1) DURIN
     G (TIME1 ISA TIME2))
(POSSESS*1 A RUFOLO1 TH (WEALTH1 ISA WEALTH2 VAL
     CERTAINAMOUNT1))
(SATISFY*1 A RUFOLO1 TH WEALTH1 MODAL NOT1)
(WANT*1 A RUFOLO1 TH (DOUBLE*1 A RUFOLO1 TH WEALTH1))
(TRADINGVOYAGE**2 A RUFOLO1 TH (GOODS1 ISA GOODS2) TO
     (CYPRUS2 ISA ISL AND1))
(NOT OF (DOUBLE*1 A RUFOLO1 TH WEALTH1))
(BRING*1 INSTR COMPETITION1 TH RUFOLO1 TO (RUIN1 MOD
     VERGE1))
(REDUCED*1 TH RUFOLO1 TO SEMI-POVERTY1)
(DISTRESSED*1 TH RUFOLO1)
(DECIDE*1 A RUFOLO1 TH (OR0 OF (MAKEGOOD*1 A RUFOLO1 TH
     LOSSES1 MEANS PRIVATEERING1) (DIE*1 TH RUFOLO1)))
(SETOUT*1 A RUFOLO1 MANNER RICH2 FROM RAVELLO1 CAUSE (WANT*2
     A RUFOLO1 TH (RETURN*1 A RUFOLO1 TO RAVELLO1 MANNER
     POOR1 MODAL NOT2)))
(PIRATEVOYAGE**2 A RUFOLO1 TH WEALTH1 MEANS (SHIP4 ISA SHIP5
     TYPE PIRA TE1 MOD LIGHT1))
(DOUBLE*2 A RUFOLO1 TH WEALTH1)
(POSSESS*2 A RUFOLO1 TH (WEALTH3 ISA WEALTH4 VAL (TWICE*1 R1
     WEALTH3 R 2 WEALTH1)))
(POSSESS**5 A RUFOLO1 TH SHIP4)
(PERSUADE*1 A RUFOLO1 TH (SATISFIED*1 A RUFOLO1 TH WEALTH3))
(DECIDE*2 A RUFOLO1 TH (GOHOME*1 A RUFOLO1 TH WEALTH3))
(SAIL*2 A RUFOLO1 TH WEALTH3 FOR RAVELLO1 MEANS SHIP4)
(PROPEL*1 A (ROWERS1 MOD VIGOROUS1) TH SHIP4)
(STORM**3 R1 RUFOLO1 R2 SHIP4)
(BLOCK*1 A (GENOESE1 ISA GENOESE2 NBR SEVERAL1 MOD (AND4 OF
     (RAPACIOUS 1) (MONEYGRABBING1)) FROM CONSTANTINOPLE1)
     TH (PATH1 MOD ESCAPE1 OF RU FOLO1))
```

```
(CAPTURE**2 A GENOESE1 TH RUFOLO1)
(WANT*3 A RUFOLO1 TH (ESCAPE*2 A RUFOLO1 FROM GENOESE1))
(SAIL*3 A GENOESE1 TH RUFOLO1 FOR DESTINATION1 MEANS
     (CARRACK1 ISA CAR RACK2 NBR TWO1 POSSBY GENOESE1))
(STORM**4 R1 CARRACK1 R2 RUFOLO1)
(CLING*1 A RUFOLO1 TH SPAR1)
(SET*1 A RUFOLO1 TH SPAR1 MANNER ASTRIDE1)
(TOSS*1 A (AND12 OF (SEAS1) (WINDS1)) TH RUFOLO1 MANNER
     (AND13 OF (HIT HER1) (THITHER1)) UNTIL DAYBREAK1)
(SEE*1 A RUFOLO1 TH (AND14 OF (WATER1) (CLOUDS1) (CHEST2
     PROPERTYOF RU FOLO1)))
(FLOAT*1 TH (CHEST2 PROPERTYOF RUFOLO1) LOC (VICINITY1 OF
     RUFOLO1) MAN NER EVERYSOOFTEN1)
(FEAR*1 A RUFOLO1 THAT (AND15 OF (COLLIDE*1 TH CHEST2 WITH
     RUFOLO1) (H URT*1 INSTR CHEST2 TH RUFOLO1)) SO (PUSH*1
     A RUFOLO1 TH CHEST2 AWAYFRO M RUFOLO1))
(SQUALL**2 TH (AND21 OF (RUFOLO1 ISA PERSON1 FIRSTNAME1
     LANDOLFO2) (SP AR1) (CHEST2 PROPERTYOF RUFOLO1)))
(SURVIVE*1 A RUFOLO1 DURING (AND16 OF (DAY1 MOD WHOLE1)
     (NIGHT1 MOD FO LLOWING1)))
(STARVE*1 TH RUFOLO1)
(THIRST*1 TH RUFOLO1)
(WANT*4 A RUFOLO1 TH (RESCUE*1 A GOD1 TH RUFOLO1 MANNER
     SOMEHOW1))
(RESCUE**2 A (WOMAN1 ISA WOMAN2) TH RUFOLO1)
(LUCKYFIND**2 A RUFOLO1 IN CHEST2)
(WANT*5 A RUFOLO1 TH (RETURN*2 A RUFOLO1 TO RAVELLO1 WITH
     STONE2))
(GOHOME**2 A RUFOLO1 TH STONE2 TO RAVELLO1)
(INTEREST*1 A RUFOLO1 TH COMMERCE1 MODAL NOLONGER1)
(LIVE*3 A RUFOLO1 MANNER SPLENDOR1 DURING (REMAINDER3 OF
     (DAY3 OF RUFO LO1)))
```

BIBLIOGRAPHY

Bobrow, D. G. and Collins, A. 1975. Representation and Understanding. New York: Academic Press.

Bobrow, D. G. and Raphael, B. 1974. "New programming languages for AI research". Computing Surveys, Vol. 6, No. 3, pp. 155-174.

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 Wilkes, Y. 1976. Computational Semantics. New York: North-Holland Press.

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

Hare, D. and Correira, A. 1978. "Generating connected natural language from case predicate story trees." unpublished manuscript, May, 1978.

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

Kintsch, W. and van Dijk, T. 1975. "Recalling and summarizing stories." unpublished manuscript, May, 1975.

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, 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.

Norman, D. A. and Rumelhart, D. E. 1975. Explorations

in Cognition. San Francisco: W. H. Freeman and Company.

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

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

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

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

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

Simmons, R. F. and Correira, A. 1978. "Rule forms for verse, sentences and story trees." in Associational Networks. (in press).

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

van Dijk, T. A. 1975. "Recalling and summarizing complex discourse." unpublished manuscript. Amsterdam, The Netherlands: Department of General Literary Studies, University of Amsterdam.

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

Woods, W. A. 1970. "Transition networks grammars for natural language analysis". CACM, Vol. 13, pp. 591-602.

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).