Syntactic Representation and Analysis of the Cognitive Structures Underlying Ritual Acts

Robert G. Hardin
Western Michigan University

Follow this and additional works at: http://scholarworks.wmich.edu/masters_theses

Part of the Computer Sciences Commons

Recommended Citation
http://scholarworks.wmich.edu/masters_theses/1337

This Masters Thesis-Open Access is brought to you for free and open access by the Graduate College at ScholarWorks at WMU. It has been accepted for inclusion in Master's Theses by an authorized administrator of ScholarWorks at WMU. For more information, please contact maira.bundza@wmich.edu.
SYNTACTIC REPRESENTATION AND ANALYSIS OF THE
COGNITIVE STRUCTURES UNDERLYING RITUAL ACTS

by

Robert G. Hardin

A Thesis
Submitted to the
Faculty of The Graduate College
in partial fulfillment of the
requirements for the
Degree of Master of Science
Department of Computer Science

Western Michigan University
Kalamazoo, Michigan
April 1987
A formal grammar was proposed by Lawson and McCauley (1980) to model the cognitive structures underlying an informant's representation of religious ritual acts. This study classifies the language generated by that grammar as context-sensitive, presents an LR(1) parser for the language, and specifies a computer program to implement that model. The system functions as an intelligent assistant using techniques involving rule-based systems, non-monotonic logic, and multiple levels of abstraction. Knowledge is represented in a parse tree, rules stored as patterns, and the inference engine uses a pattern matcher. The consequences of an act change over time and can change previous consequences. Abstraction ranges from lexical processing of characters through meta knowledge stored as transformations of parse trees. The parser and the intelligent system allow algorithmic exploration of the consequences of a theory which models the deep principles at work in a complex domain.
ACKNOWLEDGEMENTS

I wish to thank my thesis advisor, Ben Pinkowski, and committee member, Iyad Natour, along with Dionysios Kountanis and Ken Williams for their help with the content and direction of this project. Special thanks to Thomas Lawson who has assisted me in many ways, and to Robert McCauley, co-author of the theory of ritual syntax.

I also wish to thank the Computer Science Department of Western Michigan University, especially the Chairman, Don Nelson, for the continuing use of the research lab facilities by people working on this project including faculty from other departments; David Johnson, adjunct professor at W.M.U., for the use of an IBM AT and software; Fran Hardin for her editing; and my friends and family for their understanding and support.

Robert G. Hardin
INFORMATION TO USERS

This reproduction was made from a copy of a document sent to us for microfilming. While the most advanced technology has been used to photograph and reproduce this document, the quality of the reproduction is heavily dependent upon the quality of the material submitted.

The following explanation of techniques is provided to help clarify markings or notations which may appear on this reproduction.

1. The sign or “target” for pages apparently lacking from the document photographed is “Missing Page(s)”. If it was possible to obtain the missing page(s) or section, they are spliced into the film along with adjacent pages. This may have necessitated cutting through an image and duplicating adjacent pages to assure complete continuity.

2. When an image on the film is obliterated with a round black mark, it is an indication of either blurred copy because of movement during exposure, duplicate copy, or copyrighted materials that should not have been filmed. For blurred pages, a good image of the page can be found in the adjacent frame. If copyrighted materials were deleted, a target note will appear listing the pages in the adjacent frame.

3. When a map, drawing or chart, etc., is part of the material being photographed, a definite method of “sectioning” the material has been followed. It is customary to begin filming at the upper left hand corner of a large sheet and to continue from left to right in equal sections with small overlaps. If necessary, sectioning is continued again—beginning below the first row and continuing on until complete.

4. For illustrations that cannot be satisfactorily reproduced by xerographic means, photographic prints can be purchased at additional cost and inserted into your xerographic copy. These prints are available upon request from the Dissertations Customer Services Department.

5. Some pages in any document may have indistinct print. In all cases the best available copy has been filmed.
Syntactic representation and analysis of the cognitive structures underlying ritual acts

Hardin, Robert G., M.S.
WESTERN MICHIGAN UNIVERSITY, 1987
PLEASE NOTE:

In all cases this material has been filmed in the best possible way from the available copy. Problems encountered with this document have been identified here with a check mark √.

1. Glossy photographs or pages _____
2. Colored illustrations, paper or print _____
3. Photographs with dark background _____
4. Illustrations are poor copy _____
5. Pages with black marks, not original copy _____
6. Print shows through as there is text on both sides of page _____
7. Indistinct, broken or small print on several pages √
8. Print exceeds margin requirements _____
9. Tightly bound copy with print lost in spine _____
10. Computer printout pages with indistinct print _____
11. Page(s) ___________ lacking when material received, and not available from school or author.
12. Page(s) ___________ seem to be missing in numbering only as text follows.
13. Two pages numbered ______. Text follows.
14. Curling and wrinkled pages _____
15. Dissertation contains pages with print at a slant, filmed as received ______
16. Other___________________________

___________________________

University Microfilms International
TABLE OF CONTENTS

ACKNOWLEDGEMENTS .............................................. ii
LIST OF FIGURES ................................................ v

CHAPTER

I. INTRODUCTION ................................................. 1
   The Theory of Ritual Syntax ................................. 1
   Scientific Research in Artificial Intelligence .......... 2
   The Relation to Current Work in the Field ............... 5
   Implementation .............................................. 8

II. ANALYSIS OF THE RITUAL SYNTAX ......................... 10
   Development of Grammars .................................. 10
   Ritual Syntax Transformational Generative System .... 12
   Use of the Ritual Syntax ................................... 13
   Rewrite Rules and Qualities ............................... 14
   The Unwritten Rule ......................................... 17
   Lexicon ..................................................... 19
   Transformations ............................................. 20
      Object Agency Transformation .......................... 20
      Reversal Transformation ................................ 22
   Conclusion ................................................. 24

III. SPECIFICATION OF THE RITUAL STRUCTURES SYSTEM .... 25
   RSS Compared to a Rule-Based System ..................... 25
   Concepts Written Into the Program ....................... 27
      Action Structure with Qualities as Qualifiers ...... 27

iii
<table>
<thead>
<tr>
<th>CHAPTER</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transference of a Quality to the Object</td>
<td>28</td>
</tr>
<tr>
<td>Types of Users: Theorist, Expert and Researcher</td>
<td>30</td>
</tr>
<tr>
<td>Uses of the System</td>
<td>32</td>
</tr>
<tr>
<td>Structural Level</td>
<td>32</td>
</tr>
<tr>
<td>Ritual Level</td>
<td>35</td>
</tr>
<tr>
<td>Ceremonial Level</td>
<td>37</td>
</tr>
<tr>
<td>The Pattern Matcher</td>
<td>37</td>
</tr>
<tr>
<td>Going Through Changes</td>
<td>43</td>
</tr>
<tr>
<td>User Interface and Sub-Grammars</td>
<td>44</td>
</tr>
<tr>
<td>Defining Patterns</td>
<td>45</td>
</tr>
<tr>
<td>Defining Transformations</td>
<td>46</td>
</tr>
<tr>
<td>Describing Ceremonies</td>
<td>46</td>
</tr>
<tr>
<td>Detailed Specifications</td>
<td>47</td>
</tr>
<tr>
<td>Comments and Open Problems</td>
<td>50</td>
</tr>
<tr>
<td>APPENDICES</td>
<td>52</td>
</tr>
<tr>
<td>A. Parser for Original Grammar</td>
<td>53</td>
</tr>
<tr>
<td>B. Parser Incorporating Transformations</td>
<td>60</td>
</tr>
<tr>
<td>BIBLIOGRAPHY</td>
<td>70</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

1. Original Rules Rewritten ................................ 15
2. Example of Rules at Work ................................... 17
3. Nonsense Allowed by Grammar ................................ 18
4. Object Agency Transformation ................................ 21
5. Grammar Incorporating Object Agency Transformation .... 22
6. Object Agency Transformed .................................... 22
7. Example of Universal Pattern .................................. 34
8. The Pattern Matcher at Work ................................... 38
9. Terminal Session #1 ............................................. 39
10. Terminal Session #2 ............................................. 40
11. Terminal Session #3 ............................................. 41
12. Tree of Knowledge .............................................. 42
13. Multiple Lists in a Pattern .................................... 48
CHAPTER I

INTRODUCTION

"Some religious form is, like language, susceptible to rule-governed analysis. . . . A syntactic description of ritual action is possible" (Lawson, 1985, p.11). This thesis does two things: (1) language theory is used to analyze the formal grammar proposed in Ritual Structures: Aspects of a Theory of Ritual Syntax (Lawson & McCauley, 1980), and (2) the specifications are presented for a program that allows formal analysis of religious rituals in terms of the theory of ritual syntax.

Published works in the field of religion contain a rich diversity of opinion concerning the concepts of religion, ritual, myth, or culturally postulated superhumans. Religious ritual acts are defined in the theory of ritual syntax. This is a scientific theory because it describes consequences arising from that definition which can be falsified (Popper, 1959; Popper, 1963). Neither Lawson and McCauley's paper nor this thesis is a critique or justification of any religion or ritual act.

The Theory of Ritual Syntax

The theory of ritual syntax "attends primarily to the ritual acts that religious people characteristically perform as opposed to the beliefs they hold" (Lawson & McCauley, 1980, p. 3).
The structure of ritual acts, in a specific religious tradition, is what the ritual syntax is about. It is a "theory of ritual competence, . . . a formal system which is calculated to capture the cognitive structure of ritual practitioners regarding the efficacy of ritual performance" (Lawson & McCauley, 1980, p. 5). The transformational generative theory proposed by Lawson and McCauley is a system of rules which will "generate results which cohere with the verdicts of informants where they overwhelmingly agree. In those grey areas of disagreement, the theory itself will serve as an instrument of adjudication (as they do in every other scientific context)" (p. 6).

The ritual syntax is based on the transformational generative grammars originally proposed by Chomsky in 1957, explained later in this thesis. In the ritual syntax, descriptions of rituals, gathered from informants or experts, are the input strings and the output is an analysis of those ritual descriptions in the form of a parse tree.

Scientific Research in Artificial Intelligence

Searle, one of the harshest critics of AI [Artificial Intelligence], has no qualms about accepting AI as an aid in conceptualizing and testing theories of human intelligence (Hofstadter, 1981; Gardner, 1985). One of the most basic and least disputed definitions of AI research involves the computer modeling of theories of cognition. The ritual syntax is a theory of human intelligence. This study is concerned with the computer science part of the
development of this theory and computer programs to test it. Therefore, this study is AI research.

Is this study scientific research? AI has been accused of being simply a practical trial-and-error pursuit not likely to come up with a scientific theory which has explanatory power—one that can stand up to the standards of the "hard" sciences. This criticism first came from the transformational linguists but is now also held by active workers in the field of AI such as: Bobrow, Brown, Dennett, Greeno, McCarthy, Waldrop and Winograd (Gardner, 1985).

Most current AI work fails to meet traditional criteria of scientific theories. . . . The realization that one's chosen discipline may not have been operating in a scientifically impeccable manner. . . is an important step in the maturing of a discipline. A second valuable trend. . . has been the growing number of workers who seek systems reflecting the deep principles at work in a particular domain of knowledge. Spurning both the search of the most general properties of problem solving. . . and the interest in those expert systems that perform well simply by drawing. . . on massive amounts of stored knowledge, these researchers take seriously the principles that appear to be at work in the only other intelligent system we know—the human being. (Gardner, 1985, pp. 178)

Transformational linguists have concentrated on "deep principles" and on defining what constitutes a scientific theory with explanatory power. The ritual syntax theory is based on their work because it represents the most successful scientific approach to the science of the mind to date (Lawson & McCauley, 1980; Gardner, 1985). The ritual syntax is presented as a principled account of the "deep principles at work" in the domain of ritual acts. While discussing scientific AI research methods, Longuet-Higgins (cited in Gardner, 1985) states: "The task of the theoretician is to formulate hypotheses and to elicit their logical implications as
carefully as [possible]. . . . The duty of the experimenter is to confront the predictions of a theory with firm and relevant observations" (p. 180). This thesis confronts the predictions of the theory presented by Lawson and McCauley (1980).

The maturing process has involved a recognition that the practice of AI involves deep philosophical issues which cannot be ignored or minimized. . . . It is important to have genuine demonstrations and not just verbal descriptions of possible programs: this insistence has been among the greatest contributions of Newell and Simon. But it is equally important that these demonstrations reflect robust principles and are not just fragile constructions. (Gardner, 1985, pp. 179)

The ritual syntax is a theory which addresses some of the "philosophical issues" involved in the study of the mind. The parser presented in this thesis is a "working demonstration" and reflects the "robust principles" of mathematical linguistics as defined in Hopcroft and Ullman's book Introduction to Automata Theory, Languages, and Computation (1979). Chapter II of this thesis applies the models listed in their title to the analysis of the ritual syntax: (a) The language generated by the grammar is classified using the Chomsky hierarchy of languages, (b) computation theory is used to discuss the existence of an algorithm to accept or reject strings from that language, and (c) computing machine models are used to classify a parser for that language. The parsers presented in this thesis are constructed using well proven algorithms which have been encoded into automated tools used for compiler construction (Aho, Sethi, & Ullman, 1986).
The Relation to Current Work in the Field

Chapter III of this thesis reports on the work to create an automated research tool for use in the study of religious ritual acts. The system allows the storage, analysis, and manipulation of several levels of knowledge about ritual acts, all in terms of the theory of ritual syntax. The internal functioning of the program is a direct reflection of the theoretical principles proposed by Lawson and McCauley (1980).

Besides the definition given earlier, AI is also defined as the creation of programs which function or behave intelligently. If the observable behavior of a program is "intelligent," one group defines it as AI. Another group defines AI in terms of programming techniques or "approaches" used by the program internally. The strongest of these definitions requires a program both to exhibit intelligent behavior and to function internally according to certain favored techniques (Rich, 1983; Nandhakumar & Aggarwal, 1985). This thesis is not concerned with developing new programming techniques or new AI approaches. However, the ritual structures system does use an AI approach to techniques from several branches of computer science in novel ways.

The system is capable of reasoning about what it knows "within what might be called classical AI. . . [where] the information contained in an intelligent system is embodied in data structures that explicitly represent the propositions that the system knows or believes" (Moore, 1986, p. 81). The data structures for facts and rules are graphs built out of syntactically correct parse trees.
Also, "queries to this knowledge base are handled by applying inference rules in a way that amounts to searching for a derivation of an answer to the query" (Moore, 1986, p. 81).

The inference engine works the same way as a traditional rule-based system. This approach is an attempt to "face the problem head on, reasoning explicitly in a first-order way, treating propositions as objects and knowing as a binary relation" (Moore, 1986, p. 81). Moore (1986) goes on to describe other attempts "to axiomatize within first-order logic the possible-world semantics of some variant of Hintikka's modal epistemic logic... or the idea of semantic attachment to simulate other agents' reasoning from their knowledge" (p. 81).

The explicit route was chosen for two reasons: (1) expert-systems using this form of logic are well understood and developed so the resultant system can not easily be criticized as a "fragile construction of programming tricks," and (2) one of the basic assumptions of the transformational linguist is the need to study syntax separately from semantics. This "research strategy" reduces the complexity of the domain, but still allows "basic structures" to be discovered (Chomsky, 1957; Gardner, 1985).

Traditional rule-based systems store rules as IF-THEN statements. However, any syntax can be used for the rules as long as they "express as conditional with an antecedent and a consequent component" (Hayes-Roth, 1985, p. 923). Rule-based systems have a separate component of facts and "data in working memory which adheres to the syntactic conventions of facts." In the ritual
structures system, the syntax for the rules and the facts is based on a grammar. The primary output of the program is a parse tree representation of what is known. This reflects the syntactic approach taken by Lawson and McCauley (1980).

Hayes-Roth (1985) defines the architecture of a rule-based system as including several elements: (a) rules which "are data that generally conform to highly specialized grammars capable of using symbolic expressions to define conditions and actions" (p. 924); (b) the rule interpreter which "matches a rule component to working memory data. Generally this requires pattern matching" (p. 924); (c) translations of the rules to different forms for communication with the user; and (d) explanations which require that "a history of working memory changes and their causes be kept that can be searched as needed" (p. 924).

Pattern matching, in the ritual structures system, is straightforward because input, output, rules, facts and data all use the same underlying syntax and are represented as parse trees. The parse tree is the desired output and LR(1) parsers are extremely efficient at producing one (Aho, Sethi, & Ullman, 1986). After a parse tree is produced the pattern matching is a simple process (Pu, 1982). Future work could make good use of AI approaches to syntactic pattern matching in the deduction of rules from descriptions. This is currently done by the user with some system help. Since the descriptions are already in a tree form (as are the generalizations of those descriptions called patterns), the structures exist for
sophisticated automated learning (Fu, 1982; Nandhakumar & Aggarwal, 1985).

"Religious phenomena can be formally represented, theoretically analyzed and scientifically explained" This is the basic claim made by Lawson and McCauley (1980, p.3). The ritual syntax is a formal representation which allows theoretical analysis in a scientific manner. The work presented in this paper is concerned with the computability of the formal system proposed by Lawson and McCauley and the production of programs which allow the use of that system.

Implementation

Several criteria were identified which influenced the choice of languages and computers used: (a) easy access to the software and hardware during development, (b) availability of the finished system to as large a group as possible, and (c) making use of high level tools, instead of writing program code, to allow modification and experimentation. There are many languages and expert system tools which undoubtedly are better than the ones chosen, but access to them was a limiting factor.

The concepts of data representation and pattern matching were developed using PC Scheme. This implementation of the Scheme standard runs on the IBM PC and compatibles. A lexically scoped dialect of LISP, Scheme is an excellent AI tool (Abelson, Sussman & Sussman, 1985). The language allows use of the "major programming paradigms in use today, including imperative, functional, and message passing styles" (Texas Instruments, PC Scheme User's Guide, 1985, p.3).
The need to make the system widely available led to the use of the language C because of the high level tools which allowed the program to be developed in the UNIX environment and then be compiled and run on MSDOS machines. Micro computers running MSDOS are the most popular and widely available computer systems in the United States.

The parser used in the system was generated by the UNIX tool YACC [Yet Another Compiler Compiler] (AT&T Information Systems, 1986). This program produces an efficient parser from an LR(1) grammar with C statements associated with each reduction. The parser uses a function, generated by the companion tool LEX, to break the input stream into tokens. YACC allows easy experimentation with various aspects of the model. The appendices have examples of the parsing functions at work.

The storage and retrieval of information, the organization of the information, and memory management are all taken care of by a data base management system which provides a C function library based on the network data base model. These functions are very efficient and were designed for micro computers. Instead of keeping the data structures in memory, they are stored in the data base.

The specifications and the programs described in this thesis are in the public domain. For further information contact E. Thomas Lawson, Chair, Religion Department, Western Michigan University, Kalamazoo, Michigan 49008. UNIX is a registered trademark of AT&T. XENIX and MSDOS are registered trademarks of Microsoft Corporation. PC Scheme is a registered trademark of Texas Instruments.
CHAPTER II

ANALYSIS OF THE RITUAL SYNTAX

The ritual syntax proposed by Lawson and McCauley (1980) contains three parts: (1) the lexicon, (2) the rewrite rules, and (3) the transformations. In this chapter these three parts plus a critical underlying assumption in the ritual syntax are analyzed. Automata theory and examples of other systems are used to show that the language generated by their system is context-sensitive but that it can be parsed with a program that is no more complex than the parsers used for programming languages. This allows the use of a modified context-free grammar (Aho, 1967; Fu, 1982) as a model for the ritual system instead of a general grammar as is necessary for a natural language.

Development of Grammars

The rewrite rules in the ritual syntax are also known as "phrase structure rules" and "production rules." Sowa (1984) gives a concise history of their development and cites the classic works:

Systems of "production rules" were first developed by Thue (1914) for transforming strings of characters. They were used as a basis for computation by Post (1943) and Markov (1954) and were adapted by Chomsky (1957) to the formal description of grammars. About the same time that Chomsky was developing formal grammars for English, John Backus, who was the manager of the first FORTRAN project, independently developed a similar notation for defining programming languages. The main difference between the two forms is that Backus limited his notation to the
"context-free grammars," while Chomsky also defined the more general "context-sensitive" and "general-rewrite" grammars. (pp. 391)

Hopcroft and Ullman (1979) describe general and context-sensitive grammars:

The largest family of grammars in the Chomsky hierarchy permits productions of the form $a \rightarrow b$ where $a$ and $b$ are arbitrary strings of grammar symbols, with $a$ not equal $b$. These grammars are known as semi-Thue, type 0, phrase structure, or unrestricted grammars. . . . Suppose we place the restriction on productions $a \rightarrow b$ of a phrase structure grammar that $b$ be at least as long as $a$. Then we call the resulting grammar context-sensitive and its language a context-sensitive language (CSG and CSL). The term "context-sensitive" comes from a normal form for these grammars, where each production is in the form $aAb \rightarrow aBb$ with $B$ not null. Productions of the latter form look almost like context-free productions, but they permit replacement of variable $A$ by string $B$ only in the "context" $a - b$. (pp. 220-224)

Chomsky proposed transformational generative grammars to generate strings in the natural languages. Chomsky (cited in Harris, 1985) proved the natural languages were context-sensitive and defined a generative grammar as:

A rules system formalized with mathematical precision that generates, without drawing upon any information that is not represented explicitly in the system, the grammatical sentences of the language that it describes and assigns to each sentence a structural description, or grammatical analysis. (pp. 117)

The "structural description" assigned to each sentence is called a phrase marker and is usually represented as a tree structure. The "rules system" is composed of three parts: (1) the lexicon, (2) the phrase structure rules or rewrite rules, and (3) the transformation rules or transformations. The lexicon does lexical processing of strings.
The transformations consist of two components: the structural description (SD) and the structural change (SC). The structural description list the constituents which must be present in a sentence in order for the sentence to be transformed by that particular rule. For example, the SD for the passive rule could be written: P, Aux, Vt, NP, (Adv) which would be indexed as: X1, X2, X3, X4, X5 to indicate the position of each constituent within the sentence. The SC would show the final position of each constituent by giving the index numbers: X1 X2 X3 X4 X5 -> X4 - X2 - be + en + X3 - by + X1 - X5. Thus the sentence: The car will hit that tree soon, would be transformed by this rule to: That tree will be + en + hit by the car soon. Another transformational rule would resolve be + en + hit into be hit: That tree will be hit by the car soon. (Harris, 1985, pp. 131-32)

Harris (1985) expresses a widely held belief that "this type of grammar is not symmetrical, in that the rules which work for generation of sentences cannot be reversed to apply to the analysis of sentences" (p. 142). A transformational generative grammar works roughly as follows. Using strings from the lexicon, the rewrite rules generate sentences at the base level. These sentences are then transformed to generate grammatical sentences at the surface level along with a phrase marker or structural description of the sentence generated.

Ritual Syntax Transformational Generative System

The primary use of the ritual syntax is the analysis of descriptions of religious ritual actions. The transformational generative paradigm was chosen for both theoretical as well as practical reasons. In the ritual syntax, the rewrite rules are context-free and are therefore less complex than Chomsky's (cited in Harris, 1985) rewrite rules which are context-sensitive. The transformations, on the other hand, use a more complex form than those
presented above. Chomsky uses an index for the elements on the left side which are rearranged on the right. The ritual syntax uses variables from the grammar on both sides so that the transformation above might be written: NP1 Aux Vt NP2 Adv --> NP2 Aux be en Vt by NP1 Adv. Transformations make the ritual syntax context-sensitive.

Lawson and McCauley (1980) present two transformations. These transformations, discussed below, are examples of the transformations which may be necessary to incorporate the rituals of a given religion. This leaves open the possibility that transformations will be defined which make descriptions of ritual structures as complex as a natural language.

Use of the Ritual Syntax

The use of the ritual syntax involves more than just generation of sentences from the base up. The ritual system takes strings as input and produces as output an analysis of those strings. The strings describe religious ritual actions and the parse tree produced shows the structure of those actions. The user enters a string at the base level. The grammar parses the string producing a parse tree. The transformations are then performed and the final parse tree is available to the user.

The analysis of an input string instead of generating strings, means that the system must be symmetrical, i.e., it must work both ways. As stated above, transformational generative grammars are not symmetrical and don't normally parse input strings. This chapter shows that the language proposed by Lawson and McCauley (1980) can
be recognized by a parser similar to parsers used by programming language compilers. This means an efficient LR(1) parser created by one of the many parser generation programs can be used.

Rewrite Rules and Qualities

The rewrite rules proposed by Lawson and McCauley (1980) are in a constant state of flux. The different versions, however, do not substantially change the language recognized by the grammar. The version of the rules chosen for this thesis is very close to the rules proposed with slight modifications. Figure 1 presents the original rules rewritten in the style of input to the UNIX tool YACC. In this notation "|" is the "exclusive or" symbol, "::" is the "replaced by" symbol, and "/* */" enclose a comment. Furthermore, for ease of reading, upper case letters are variables, and lowercase are terminals. Also, changes have been made to allow easier parsing of patterns and eventual placement of and/or nodes in the parse tree generated by the parser. The original grammar had the productions RP-> PT Q | 0 and 0 -> OBJ OQ. The production 0 -> OBJ OQ was removed so RP -> PT Q | OBJ OQ.
The grammar used for the rewrite rules is context-free and can be parsed with a LR(1) parser. There is an underlying assumption, not obvious from these productions, which could move the language that Lawson and McCauley (1980) want the rewrite rules to accept into the class of context-sensitive. As discussed later, this may not be true for the rewrite rules, but is certainly a prerequisite for the proper implementation of the reversal transformation.

The ritual syntax rewrite rules are based on the familiar three-part linguistic form of agent action object. Each ritual (R) is composed of three parts (RP RA RP). The first ritual participant (RP) is the agent of the action or the performer of the act. The

Figure 1. Original Rules Rewritten
ritual action (RA) is the act being performed. The last ritual participant (RP) is the object of that action.

The ritual syntax distinguishes between objects and participants since this is one of the areas where ritual actions differ from normal actions. In a religious ritual system things normally thought of as inanimate objects, can perform significant actions and are classified as members of the participant (PT) class. Things not capable of action are classed as objects (OBJ).

A ritual action may be either an act alone or an act done with an object, such as a minister baptizing, with water, a baby. The phrase with water is the action condition (AC) where with is the (C) and "water" is the object (OBJ). Figure 2 is a simple example of a string accepted by the rewrite rules. Note that this string does not represent the way any member of a religion would phrase this ritual. The theory represents a way to discover and organize the cognitive structure underlying the understanding a religious practitioner has about why a ritual action is effective.

There is a quality (Q) associated with an object or participant. The efficacy of a given ritual is traced using the qualities of a participant. Part of the ritual structure of a religion might be that a Priest can bless water if the Priest has been ordained by the Church which has a ritually bestowed quality which gives it the power to ordain and so forth (see Figure 2). Qualities are given the name of the action performed, so the bless ritual might be written: Priest ordain bless water unspecified. Where ordain refers to
a ritual performed on the Priest so the Priest has ordain as a quality.

Figure 2. Example of Rules at Work

The Unwritten Rule

In Figure 2, it is important to note that the Priest is on the receiving side of the ordain ritual which thus becomes a quality of the Priest. In order for the Church to gain the quality institute it was the object of the institute ritual. This illustrates the unwritten assumption of the ritual syntax. In order for X to have a quality imparted by a ritual, X must be the object of that ritual. It does not make sense to say X has the quality bless imparted by the ritual: Y (superhuman) bless Z unspecified. However, the grammar would allow such a thing.
In Figure 3, X has the quality bless imparted by Y to Z. To paraphrase: X is blessed because Y blessed Z. This clearly does not make sense. The normal human understanding of these rituals would be that Z has been blessed, not X. The rewrite rules would have to be much more complex to force the object of the act bless to be the same as the participant which has the ritual bless as a quality; in other words to force the Z in the bless ritual to be X instead. Then one could say: X is blessed because Y blessed X. This is a case where the rewrite rules need external processing. A Pascal compiler uses similar processing to enforce the definition of variables before they are used and to handle other "context-sensitive" cases.

The argument could be made that the grammar does not have to enforce this unwritten rule. The end user must enter rules for ritual performance that make sense in other ways, so why make this a special case? The grammar can allow nonsense to be entered, the sense being enforced by the human using the system. However, in
order to perform the reversal transformation the unwritten rule must be enforced by the system.

Lexicon

Information stored in the ritual syntax lexicon includes an association of a string with a terminal symbol and semantic information not included in the rewrite rules. The lexicon could easily complicate the system but an analysis of Lawson and McCauley's (1980) paper shows this is not the case.

From conversations with Lawson and McCauley it is apparent that the association of a string with a specific grammar symbol is straightforward. There is no need for any specific word to have different meanings in different contexts. A string can be recognized by the lexical analyzer as a specific terminal symbol and only that symbol. The lexical processor can thus be a regular grammar and be implemented using a deterministic finite state machine with output. The program generated by the UNIX tool LEX can handle all the lexical processing necessary. The input to LEX is a regular expression, and the output is a program which uses an optimized deterministic finite state machine, with actions, based on tables produced by LEX.

The lexicon can also contain semantic information about a particular string. All of the semantic uses of the lexicon proposed in the ritual syntax are covered in the discussions of transformations.
Transformations

An important part of transformational generative grammars and an important part of the ritual structures theory is the transformational component. The rewrite rules show the underlying relationships between the various parts which make up a ritual, the transformations apply to whole rituals. While the rewrite rules may be general enough to cover all religions, the transformations may need to be changed from one religion to another. Each transformation proposed by Lawson and McCauley (1980) can be incorporated into the grammar, and the language generated by the complete system can be parsed by an LR(1) parser.

Object Agency Transformation

Lawson and McCauley (1980) give an illustration of the use of the object agency transformation. A Christian parishioner enters a church and "having dipped his finger into . . . water, makes the sign of the cross on himself with the water" (p. 14). The water has purified the parishioner. The water is able to do this due to embedded rituals eventually invoking superhuman agency. Objects like water are not thought of as performing an action, so the object agency transformation "constrains" the ritual action so that it does not violate this "basic action structure," and causes the action to be restated as the parishioner purifying himself with water. The formal statement of the object agency transformation is given in Figure 4. The transformation symbol "⇒" separates the left side from what it is transformed to on the right. The PT-1 Q-1 and PT-2
Q-2 on the right side allow different or, in this case, the same participant (with matching quality) at each node in the tree.

\[ \text{OBJ1 Q1 ACT PT1 Q1} \Rightarrow \text{PT-1 Q-1 ACT C OBJ1 PT-2 Q-2} \]

Example:

(Water Purify) Purify (Parishioner Ins)

\[ (\text{OBJ1 Q1}) \text{ ACT (PT1 Q1)} \Rightarrow (\text{PT1 Q1}) \text{ ACT C (OBJ1 Q1) (PT1 Q1)} \]

(Parishioner Ins) Purify with (water purify) (Parishioner Ins)

Figure 4. Object Agency Transformation

By changing the production \( R : \text{RP RA RP} \) to \( R : \text{P RA RP} \) and adding the production \( P : \text{PT Q} \), the object agency transformation is incorporated into the grammar (see Figure 5). Now a parser based on the new grammar will produce the parse tree in Figure 6. A ritual is given the name of the action in the ACT node of the ritual. A pattern which references an embedded ritual as a quality just puts the name of the ritual there. Here the superhuman quality is through the embedded ritual purify a quality of water in the AC node of the ritual. The language accepted by these new rewrite rules is still context-free.
R : P RA RP
P : PT Q
RP : PT Q | OBJ OQ
RA : ACT | ACT AC
Q : lps | lns | luns | R
OQ : lns | luns | R
AC : C RP
PT : lpt
ACT : lact
OBJ : lobj
C : lc

Figure 5. Grammar Incorporating Object Agency Transformation

Reversal Transformation

Rituals such as divorce reverse the action of other rituals such as marriage. The reversal transformation uses information stored in the lexicon to force the object of a reversing action to have previously been the object of the action being reversed. One must have been married before one can be divorced. Each act which participates in a reversal has this information stored as part of
the semantic knowledge in the lexical entry for that act. How the semantic information in the lexicon and the reversal transformation can be incorporated into a parsing system is presented below.

Each entry in the lexicon includes reversal information where necessary. Lexical Entries for Confirm and Excommunicate: (confirm, + ACT, . . . , R-excommunicate), the act confirm is reversed by excommunicate, (excommunicate, + ACT, . . . , REV ACT confirm), the act excommunicate reverses confirm.

A formal statement of the reversal transformation is: RP ACT2 PT1 Q => RP ACT2 PT1 (RP ACT1 PT1 Q), where ACT2 reverses ACT1. The transformation acts as a constraint forcing the participant PT1 to already have the quality ACT1 before it can be the object of ACT2. The participant symbolized by PT1 must be the same string in every place it appears. This fact forces the system to enforce the unwritten assumption discussed in the section about the rewrite rules.

There are two things the parser must do with external processes. First, each ritual used as a quality must have the owner of that quality as the object of that ritual. Second, an act which reverses another must only be performed on a ritual participant with the quality which is being reversed.

A parser which proceeds from left to right across a string, like the LR(1) parser produced by YACC, can easily assure that a ritual used as a quality has as its object whatever is using it as a quality. When the parser knows what string is the object of an action this string can be stored on the value stack. Then when the
parser is reducing the ritual to a quality it can check that the string stored matches whatever is associated with that quality.

Proper use of reversal actions can be enforced in a way similar to qualities. When the parser knows what the object of a reversing action is, the parser can check the associated quality and make sure it is a ritual containing the action being reversed. By placing the action involved with each ritual on the value stack along with the object of that action, this information can be used for enforcing the reversal transformation later in the parsing process.

Conclusion

The transformations described as examples by Lawson and McCauley (1980) can be incorporated into a LR(1) parser using techniques similar to the ones used by compilers. This information classifies the complexity of the ritual syntax in terms of some well-known problems. Appendix B contains the source code for LEX and YACC as well as sample runs. The resultant parser enforces the transformations and the unwritten rule described earlier.

The ritual syntax can be stated as a problem with a yes/no answer and instances of the problem can be encoded as strings. Ritual structures of a religion can be described in a string which is accepted or rejected by a computing machine. This implies several things: (a) the encoded language is recursive; (b) there exists an algorithm which can decide, for all possible instances, if a ritual description is legal or not; and (c) the problem is computable.
CHAPTER III

SPECIFICATION OF THE RITUAL STRUCTURES SYSTEM

The previous chapter showed that it was possible to parse the language generated by the ritual syntax. This chapter presents the specifications for a computer program, called RSS, which allows experimentation with the ritual syntax formalism proposed in Ritual Structures (Lawson & McCauley, 1980).

RSS Compared to a Rule-Based System

RSS is based on knowledge of the efficacy of religious ritual acts. The program performs several difficult tasks at expert levels in the domain of the ritual actions in a religious tradition. The categories of problems solved by RSS include interpretation, explanation, prediction and instruction (Lawson, 1985). RSS encodes knowledge of different religious traditions using a common underlying representation, and provides a scientific method for structural analysis and comparison.

By asking the system to show a described ritual to any level, a user can receive a justification for that ceremony—"how" the system decided the ceremony could be performed. The user can also ask "why" a particular question is being asked and the program will explain. The system can also predict what is possible from the information it has, so the user can ask what rituals a given
participant can perform or what rituals can be performed on a given participant. An important use of the self-knowledge the system has of its reasoning is the ability to ask, "what would happen if?" The user can change a rule or a fact and see the result of that change. RSS allows questions of the form why, how, and what if.

Traditional rule-based systems use rules, statements of fact, and an inference engine (Hayes-Roth, 1985). The grammar-based system, described in this thesis, uses syntactic patterns, transformations, facts stored in a graph, and a syntactic pattern matcher. This approach has advantages for certain types of knowledge and avoids some of the problems associated with a rule-based system, especially in the area of side effects caused by changes to the rules (Buchanan & Shortliffe, 1984).

The expert using a grammar-based system describes a grammar and then patterns in the form of a parse trees based on the grammar. A pattern matcher uses these patterns and known facts stored in a large parse tree to see if something is legal. The connection of one fact to another is done through the recursive productions in the grammar. A description of a ritual performance is only legal if it fits in the parse tree and satisfies the constraints imposed by the patterns.

In a rule-based system a fact that the inference engine determines is true may or may not be added to the knowledge base depending on the system, but the inferences used are not part of the facts. If the conclusion is added to the facts a change in one of the rules or another fact which should invalidate the original
conclusion will not, unless a separate mechanism is used. If the conclusion is not added to the known facts then the effect of a change in the rules can only be seen by testing. The latter is the normal route and in a large expert system with many rules and facts it is very hard to see the effects of a change unless all past conclusions are retried.

In the grammar-based system the relations between facts are stored with the facts in a graph. The effects of a change are much easier to see and propagate since all the interrelations are explicit. This avoids having to use a separate mechanism or retrying all past conclusions. The system needs only follow the connections in the graph to find all effects of a change.

Concepts Written Into the Program

There are certain principles hard coded into the RSS. The program was designed to function as much like the ritual syntax proposed by Lawson and McCauley (1980) as possible. Thus the program functions as the experimental stage of the theory of ritual syntax.

Action Structure with Qualities as Qualifiers

Lawson and McCauley (1980) state: "Any ritual, because it is an action, requires an agent, an act, and an object of that action. . . in rituals certain qualities are necessary for persons and objects to qualify as ritual participants." (p. 12)
For example: "You can read this sentence because Teacher1 taught you how to read; furthermore, you can read this sentence because I wrote the sentence. I could write the sentence because Teacher1 taught me to write (we both went to the same one room school)." The program, like an over-inquisitive child, keeps asking questions like: "How could Teacher1 teach you to write?" Where this line of questioning stops is interesting. Eventually it leads to an answer like: "Because it exists." If asked "why does it exist?", a cosmologist might use the phrase, "because of the Big Bang," to terminate the line of questioning. A formal system like mathematics uses "it is a basic assumption" or "it exists by definition" as terminating phrases. A religious person might terminate with the phrase "because an all powerful being created it." A parent would terminate the questioning long before it reached the "why does it exist?" stage by stating "just because, now go and play!" In the ritual structures system, it takes superhuman agency to terminate lines of inquiry about a ritual action.

Transference of a Quality to the Object

The ability of an actor to perform an action on the object of that action depends on qualities both participants have. Participants receive qualities either from other actions or by definition. When an action is performed on Bob, Bob receives that action as a quality. Because Bob has received this quality Bob can participate in other actions. Rules in the rule base are primarily specifications of the qualities required of the participants in
actions. These rules are expressed and stored as syntactic patterns.

A history is kept of commands and the time they are issued. The results of each command are marked with this time. A change made to a command or its results causes changes in the knowledge base. The program uses non-monotonic logic so these changes can propagate through the tree. The command time is used as a heuristic during this propagation, since any thing done before the time of the command being changed need not be checked for possible changes.

When the performance of an action is described, so is the simulation time of its performance. A hundred acts may be described as happening at the same instant. Then the user can change the current simulation time by some number of units. This allows the user to control the sequence of events being encoded independently of the order in which they are entered.

Acts have information about their lifetime stored as part of their rules or patterns. This time has four general forms. It may be: (1) a specific length of time, (2) until death, (3) past death, or (4) not appropriate. This is how long, in simulation time, the action is effective. It is well-known that the action of washing the car has an effective lifetime of about ten minutes, the length of time it takes for a clear sky to fill with clouds and start dumping rain.

In Black Elk Speaks (Neihardt, 1932) an American Indian religion is described which includes ritual re-enactment of visions. The power of a vision is available through re-enactment. The Indian
performing a vision can only pass on the vision's power for a certain length of time. A vision granting a shield, for example, deflects bullets and arrows for only a day. This information is represented as a specific time expressed in units of simulation time.

Many qualities of living beings are for life. Marriage is a ritual which is permanent "until death do us part." Holy mountains erode away to sand given enough time. The normal use of the word permanent embodies the idea that nothing lasts forever. All qualities with a permanent lifetime are removed upon a participant's death.

Some qualities are postulated by some cultures to last past death. This is another aspect of ritual actions which is different from a normal action. Superpermanent qualities are retained past death.

Many times it makes no sense to talk of effective time in relation to a ritual act or the quality it bestows. The Roman Catholic idea of grace is a good example. A person can receive grace every three seconds or just once. Qualities with a lifetime "N/A" are effective as long as what they are associated with exists.

Types of Users: Theorist, Expert and Researcher

The RSS is designed to accommodate several types of users each with different expectations and competence. One of the primary uses of this system is testing theories about the underlying structure present in the knowledge base. This calls for a program which allows the non-programmer to change the control structures of the
system. The RSS allows the user to change most aspects of the system except the rewrite rules themselves. In order to change those, a programmer is necessary.

Users such as Lawson and McCauley (1980) are concerned with the structure used to represent knowledge. A theorist can define transformations, change rules, or change the basic grammar and test the effect of these changes. The challenge is to design the program to allow as much of this experimentation with the structure as possible without the intervention of a programmer or knowledge engineer. In this program, the model used to represent knowledge is as important as the information stored in it. This adds an extra layer to the program as compared to a rule-based system.

A religion expert defines ritual patterns. An expert may also describe actual performances of rituals and the system will help deduce the patterns or rules. In either case the expert is using the program to organize the ritual actions of a religion in terms of the grammar and structures previously set up by the programmer and the theorist. The expert user is actually putting knowledge into the program instead of playing with the structure of the program.

Once a ritual system has been encoded by an expert, that knowledge can be used by others for learning or research. A researcher can ask about what is stored, test new performances against the rules, or have the system generate new information from what is stored.
Uses of the System

There are three distinct ways the program is used: (1) structural, (2) ritual, and (3) ceremonial. These can be viewed as levels in a hierarchy where each level controls the levels underneath. At the first and highest level, the structure of the program is defined. At the second level, rituals are defined. And at the third level, ceremonies are described. The performance of a ritual is a ceremony. A "definition" of the marriage "ritual" includes the conditions necessary for a legal marriage. A "description" of a marriage "ceremony" includes the date, names of the people involved, and so forth. The marriage ritual can be defined only once, but can be performed many times. Each ceremony must obey the constraints defined for marriage at the ritual level and the general constraints defined at the structural level.

Structural Level

The basic purpose of this program is to explore methodologies which explicate and explain religious phenomena, specifically ritual actions. The system is designed to allow experimentation with the structure of the knowledge representation. A programmer can change the grammar and a theorist can change patterns and transformations. All of these change the basic functioning of the program. The structural theorists use the program at this level. Anything defined at the structural level will be universal across all religions so things defined at this level are called universal rules, variables, transformations, patterns, etc.
Information is entered in terms of a grammar and displayed as a parse tree. This information is parsed by a bottom up LR(1) parser. The grammar represents the rewrite rules in transformational generative grammars. Patterns and transformations are also defined in terms of the same grammar. Different grammars can be tried by a theorist with the help of a programmer. Since the language generated by the grammar can be parsed by a LR(1) parser, it is possible to use the tools developed for compilers greatly simplifying the testing of new grammars.

All legal rituals, whether as patterns or ceremonies, are required to match universal patterns. Universal patterns are always in effect whether the user is using predefined patterns for specific acts or not. So the patterns control the basic functions of the program.

Here is a short introduction to patterns which are explained in detail later. A pattern is defined using the same grammar as is used by the parser. Instead of specific lexical terminals, a pattern uses variables. The pattern with the single element (R) would match all complete parse trees since R is the start symbol of the grammar. The pattern (RP RA RP) would also match any legal string parsed. The pattern (PT Q1 RA PT Q2) would match any tree which had different lexical terminal strings for Q1 and Q2. Rules can be associated with any pattern in terms of the variables defined in that pattern and information stored for that variable type. Q1 and Q2 are qualities which have information concerning times so a rule might state Q1.life_time > 10 units or Q1.simulation_time <=
Q2.simulation_time. According to Lawson and McCauley, religious rituals must trace their efficacy to a superhuman (1980, p. 16). This rule can be defined as patterns. The patterns forcing a ritual to evoke superhuman agency in the initial (RP) node are: (PT Is RA RP) or (PT R RA RP). The patterns forcing a ritual to have a superhuman quality in the (AC) node are: (RP ACT C OBJ Is RP) or (RP ACT C OBJ R RP). These patterns are defined recursively the first of each pair being the base where the terminal for a superhuman (Is) appears. The recursive part is replacing the (Is) with (R) a ritual. A ritual matching any of these four patterns would conform to the constraints the theorists have imposed.

A pre-order traversal of the parse tree would match the pattern in the order indicated (see Figure 7). Following the parse tree the (PT 3) would match the pattern so the traversal would go to the (superhuman 5) which is the lexical type (Is) which matches the (Is) in the pattern and so forth.

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Parse Tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>PT Is RA RP</td>
<td>R (1)</td>
</tr>
<tr>
<td>(3) (5) (6) (7)</td>
<td>RP (2) RA (6) RP (7)</td>
</tr>
<tr>
<td>PT (3) Q (4)</td>
<td>ACT PT Q</td>
</tr>
<tr>
<td>Jesus superhuman (5) bless Person luns</td>
<td></td>
</tr>
</tbody>
</table>

Figure 7. Example of Universal Pattern

After a string is parsed and stored in a parse tree, it is subject to transformations. These are expressed as pattern ==> pattern. If the tree matches the left side, then it is transformed
into the pattern on the right side of the $\rightarrow$. The object agency and reversal transformations described by Lawson and McCauley (1980) are examples of universal transformations which are defined at the structural level. A user working at the structural level can define both obligatory and optional transformations along with rules controlling the use of these transformations. A rule might state that only one transformation out of a list can be performed on a given ritual.

**Ritual Level**

A religion expert will make much use of the ritual level because this is where ritual definitions are manipulated. There are several different parts to this level. Each religious tradition may need specific transformations not covered by universal ones so global transformations, specific to a religion, are possible. The lexicon and patterns for acts are also defined at this level. The lexicon and the global transformations form a context for a specific religious tradition. There are no global patterns at this level analogous to the universal patterns of the structural level. They can be added if they prove necessary. The constraints for specific rituals are defined from within the environment provided by the lexical definitions and global transformations. At the ritual level several sets of rituals may be defined within the same religious context, or the system may work without ritual definitions. All that is necessary is the lexicon.
Each word, in order to be recognized, is first stored in the lexicon along with the type of grammar symbol it is. In addition, participants which are either agents or objects of an action have a terminal quality associated with them. The string Jesus defined as the token (lpt) associated with the (ls) type quality superhuman is an example. If Jesus is used in the description of a ceremony and the user does not specify a quality and the pattern matcher either needs a (ls) quality or does not care, then the default quality superhuman is used. The system as presently specified allows only one default quality and the acquiring of other qualities by being the object of an action. However, there is a perceived need to allow many qualities to be associated with not only (lobj) and (lpt) types but also actions. These qualities would be similar to properties of an entity in data base terminology, or the value attribute pairs used in associative lists in LISP. Mechanisms to allow these other types of qualities may be implemented in version II of the system if they prove necessary.

The constraints defined for rituals are called "rules for acts." Each ritual defined is given the name of the action in that ritual. Specific constraints for each ritual are expressed in terms of patterns with rules referring to the variables in those patterns, required transformations, and other information. Every act has at least one pattern but it may have several joined by the logical binary operators. Required transformations, with variable substitutions referenced to variables in the patterns, are stored with the
patterns. Other information is stored with the rules for an act, such as the effective life of the act.

Ceremonial Level

Once the structure of the program is set and the ritual level defined, the user may describe and manipulate ceremonies at the ceremonial level. Ceremonies may be described either with or without previously defined rules for rituals. Likewise, ceremonies may be described either with or without previously described ceremonies. All ceremonies are described in an environment which includes the structural level plus at least the lexicon and any global transformations defined at the ritual level. While describing ceremonies the user has the option of adding to previous descriptions or simply building a temporary tree.

The system has the ability to deduce rules for acts from ceremonial descriptions. Each time a ceremony is described the program goes through the description with the user trying to generalize the description into a pattern. Each node in the ceremony’s parse tree is visited and the user asked if the information there is required. Using this process, the system can build rules for rituals from descriptions of actual performances of those acts.

The Pattern Matcher

Rules for acts are not necessary for the functioning of the program. An expert could encode the ritual system of a religion in one big string, as in the analysis section of this paper. When
using the patterns, the user just describes one ceremony at a time. Qualities are not entered by the user initially. The pattern matcher supplies the qualities using the information stored in the rules for the act under question (see Figure 8). The qualities used can come from either the lexicon, previously described rituals, or as a last resort, from the user.

The lexicon contains:

<table>
<thead>
<tr>
<th>string</th>
<th>lexical class</th>
<th>quality</th>
</tr>
</thead>
<tbody>
<tr>
<td>water</td>
<td>lobj</td>
<td>luns</td>
</tr>
<tr>
<td>Parishioner</td>
<td>lpt</td>
<td>luns</td>
</tr>
<tr>
<td>Church</td>
<td>lpt</td>
<td>luns</td>
</tr>
<tr>
<td>Priest</td>
<td>lpt</td>
<td>luns</td>
</tr>
<tr>
<td>Jesus</td>
<td>lpt</td>
<td>superhuman</td>
</tr>
<tr>
<td>purify</td>
<td>lact</td>
<td>N/A</td>
</tr>
</tbody>
</table>

The ritual act purify has these patterns:

- PT (purify or ls) ACT RP
- RP ACT C (OBJ purify) RP

Figure 8. The Pattern Matcher at Work

In Figure 8, the first pattern will match any ceremony where the agent of the act either has purify as a quality or is a superhuman. The second pattern will match any ceremony where the act is performed along with something which has purify as a quality. Notice that these patterns are for the act purify so the RA and ACT symbols in the patterns imply purify. The use of the word purify in these patterns refers to an act which imparts the quality purify.
In the parse tree this would be an embedded R with purify as the lexical class act.

The terminal sessions (Figures 9, 10, 11) which follow are not from an actual printout. The system starts with an empty knowledge tree each session. In Figure 9, the pattern matcher prompts for information it needs as it tries to match at least one of the patterns for purify. Note that since Jesus has the default quality superhuman, defined in the lexicon, the pattern matcher uses that information and does not prompt for it.

user: Parishioner purify with water Parishioner

system: what quality lets Parishioner purify with water ?
user: unspecified

system: what quality lets water purify?
user: Priest purify water

system: what quality lets Priest purify?
user: Church purify Priest

system: what quality lets Church purify?
user: Jesus purify Church

system: ceremony 1,2,3,4 recognized and added to the tree

Figure 9. Terminal Session #1
user: Parishioner purify with water Parishioner

system: what quality lets Parishioner purify with water?

user: why?

system: in p1 for purify: PT (purify or Is) ACT RP

PT = Parishioner, trying to match (purify or Is)

user: show patterns for purify

system: p1 = PT (purify or Is) ACT RP

p2 = RP ACT C (OBJ purify) RP

user: use p2

system: what quality lets water purify?

user: Priest purify water

system: what quality lets Priest purify?

user: Church purify Priest

system: what quality lets Church purify?

user: Jesus purify Church

system: ceremony 1,2,3,4 recognized and added to the tree

user:

Figure 10. Terminal Session #2
In Figure 11, the user enters the ceremonies in such an order that the pattern matcher always has what it needs to fill in the appropriate qualities. The ceremonies described in terminal sessions generate the same parse tree (see Figure 12). This is the basic structure used in the knowledge base for ceremonies and is called "the tree of knowledge."
Parishioner unspecified purify with water R Parishioner unspecified
   RP RA RP
   PT Q ACT OBJ OQ

Priest R purify Water unspecified
   RP RA RP
   PT Q ACT PT Q

Church R purify Priest unspecified
   RP RA RP
   PT Q ACT PT Q

Jesus superhuman purify Church unspecified

Figure 12. Tree of Knowledge

The complete ceremony of a Parishioner blessing himself with purified water can be described without the use of rules for acts at all, by putting all the information in one string, as was done in the analysis section of the paper. This use of the program may be all that is necessary to accomplish the goals of the religion expert, but the use of the pattern matcher is much more powerful.

Once a religion expert has described the ceremonies in the example, a researcher could expand on the general terms used by adding specifics. The system allows one level of inheritance so that a user can refer to Parishioner.Bob, a specific instance of a Parishioner. The string Bob must be defined in the same lexical class as the string Parishioner. Anything a Parishioner can do Parishioner.Bob can do, but not necessarily the other way around. This inheritance is based on the structure data type as provided by the language PC Scheme (Texas Instruments, TI Scheme Language Reference
Manual, 1985). The use of inheritance is at the ceremonial level and it is stored in the tree of knowledge.

Going Through Changes

A user can change anything previously entered into the system. This might be done to correct errors or to see the effects of a change. Ceremonies are stored in a graph where each node has pointers to its parent as well as its children, so that changes can propagate though the tree. Ritual level information is stored in a similar way. The extra overhead of using non-monotonic logic is justified because it allows easy experimentation with the structure of religious ritual systems. The program is used primarily by theorists and experts who enter a small system and then make changes and additions. The process of encoding information and discovering its underlying structure is the important thing, not the efficient recovery of large amounts of data.

The effects of the performance of a ritual may be changed by another ceremony. The main question is when, in simulation time, the change takes place. A divorce will reverse a marriage starting at the time of the divorce. That simply means that during the time between marriage and divorce the marriage is valid. Only after the time of the divorce are dependant rituals affected. An annulment however, states that the marriage was never valid. This means that the ceremony describing the marriage has to be kept, but the time it is effective reduced to zero, so that any rituals dependant on this specific marriage must be re-evaluated and perhaps removed.
Presently the information of how one act changes the effective time of another is stored with the links to the reversal transformation.

The person using the program may change things. A change in the description of a ceremony causes all ritual performances dependant on it to be re-evaluated. Any change to the patterns or their associated rules will cause the affected descriptions to be re-evaluated. The real time of the pattern's creation is used for heuristics, since only those ceremonies described since the definition of the pattern need to be checked.

The string used to represent any lexical item is stored in only one place. All strings stored in the lexicon are unique. Each use of a string is a pointer to it. Spelling changes are thereby trivial and cause no re-evaluation. A change is reflected in all output produced from the time of the change on. An act named bless changed to purify causes all references to that act to be renamed purify for all times. The system will lose all memory of the string bless.

Each participant and object defined in the lexicon has an initial quality associated with it. A change in this quality causes ceremonies dependant on it to be re-evaluated. Changing a quality from superhuman to non-superhuman causes the most changes in the tree of knowledge.

User Interface and Sub-Grammars

The user interface is interactive. Commands entered by the user are recognized by a parser. This parser is generated from a grammar organized into the command grammar and several sub-grammars.
The commands are used to set the environment and retrieve information. The creation of patterns, transformations, or ceremonies causes the use of the sub-grammars. These sub-grammars are different versions of the underlying rewrite rules. Any change made to the rewrite rules must be reflected in the sub-grammars.

Defining Patterns

The grammar for patterns is the same as the rewrite rules except it allows variables besides terminals. The parser does require enough information to build a complete parse tree out of the new pattern the user is entering and that parse tree must reduce to the same start symbol as the rewrite rules. Patterns are controlled by the patterns defined above them. After a new pattern is parsed it is checked against patterns defined immediately above it. Structural level patterns force all patterns defined at the global level to obey their constraints. Global patterns force all ritual patterns to match them. This means that all ritual level patterns have to be complete enough to match the higher level patterns and higher level patterns must be general enough to allow meaningful use of subordinate patterns. The hierarchy formed by patterns provides a mechanism to control the form patterns take. Also, if the parse tree of a ceremony matches a ritual pattern, it automatically matches all higher patterns up to and including the structural level patterns, an important heuristic used by the pattern matcher.
Defining Transformations

The grammar for transformations is very simple as it uses the pattern grammar for the left side, followed by the transformation symbol $\Rightarrow$, then the pattern grammar for the right side. After a transformation is parsed then checks must be done to see that the patterns on either side of the transformation symbol are consistent with each other.

When patterns are defined they are not transformed by transformations. Patterns for a ritual are used to fill in qualities and check legality before the required transformations defined for that ritual can be performed. This means that the ritual patterns should match the left side and must match the right side of those transformations. If the left side of a transformation does not match any pattern for its act, it will never be performed; if the right side of a transformation does not match a pattern for its act, it will create an illegal parse tree after it is performed. The same is true of patterns and transformations on any level. The patterns used in a transformation have to match patterns on the same level where as normal patterns have to match patterns on the level above.

Describing Ceremonies

The ceremony sub-grammar is exactly like the conceptual except it allows entering descriptions of ceremonies with or without qualities. The missing qualities are then supplied by the pattern matcher. If rules for acts are being used, the pattern matcher tries to find qualities called for by the patterns in those rules. The
pattern matcher will use universal patterns to decide which qualities to supply, if no ritual level patterns are used. The source for the missing qualities is either previously described ceremonies stored in the tree of knowledge or previously defined default qualities, or as a last resort, the user. After the missing qualities are filled in, the resultant parse tree must be legal in terms of the rewrite grammar.

The linking of a ceremony to the tree of knowledge is done after pattern matching, obligatory transformations, and the user's chance to perform optional transformations. In other words, the ceremony parser and the pattern matcher build a complete parse tree from what is entered. Each quality supplied from the tree of knowledge is just temporarily associated with the local tree. Only after the predefined constraints and the user are satisfied, is the local tree permanently linked to the big tree through those supplied qualities. Notice that only the pattern matcher can supply qualities from the tree of knowledge or link the current parse tree to the other ceremonies.

Detailed Specifications

Both the rewrite rules and the patterns allow (and) and (or) lists of selected variables in the grammar. Thus a minister can marry Bob and Fran. The syntax for the patterns allows complex use of these type of lists.

A pattern with an (and) list can specify that the elements must match in the same order given, or the candidate parse tree can have
the elements mixed up. The pattern (Bob AND Sam) would match (Bob
and Sam) but not (Sam and Bob). The pattern (Bob and Sam) would
match either. The all uppercase AND forces the order to be the same
where as the lower case allows any order. The AND is used in trans­
formations for rearranging elements of a list.

When variables are in an (and) list or an (or) list (see Fig­
ure 13) they may be listed as (P1 and P2) or (P-1 and P-2) each a
list of exactly two elements. A list of variable number of elements
may be expressed as (P1 and Pn) for one or more distinct P's or (P1
and Pn) for two or more distinct P's.

(P1 AND Pn) bless (P2 AND Pn )
(Bob and Fran) bless (Sam and Joe and Sue)

=> P1.1 bless ((P2 AND Pn) AND (P1.2 AND P1.n) )
Bob bless ((Sam and Joe and Sue) and Fran))

=> P2 bless (P2.2 AND (P1 AND Pn) AND P2.n)
Sam bless (Joe and (Bob and Fran) and Sue)

P1.1: refers to the first element in the list (P1 AND Pn)
P1.2: the second element in the list (P1 AND Pn)
(P1.2 AND P1.n): the second element to the end of the list
(P1.2 AND P1.4): 2 elements of list (P1 and Pn)
(P1.2 - P1.4): a range including elements 2, 3, and 4

Figure 13. Multiple Lists in a Pattern

The discussion of the qualification of the reversal transforma­
tion when used by divorce leads to the following addition to pat­
terns. Any where an ACT variable is used as a quality it may be
replaced by a R [ ACT ] variable. The reversal for divorce could
have the rule ACT 1 = R1 [ marry ] forcing all uses of ACT 1 to be
from the same ritual. The pattern (P1 R1 [ ordain ] and PN RN [ ordain ] ) would force each P to be distinct and each quality ordain
to come from distinct rituals. Where as the pattern (PI R-1 [ ordain ] and PN R-n [ ordain ] ) is the same as (PI ordain and PN ordain) the small -n allowing one or more of perhaps the same ritual. The pattern (PI R1 [ ordain ] and Pn RN [ ordain ]) would force two distinct ceremonies performed on perhaps the same participant.

Transformations are defined at either the structural or ritual level. Each rule for an act may have information which associates the act with a transformation to be performed. The reversal transformation is only performed on acts which reverse another act. This semantic information is stored with the act which does the reversing. The variables in the transformation are linked to the variables in the pattern for that act.

Let the universal reversal transformation be (RP1 ACT1 (PT1 and PTn) => RP1 ACT1 (PT1 ACT2 and PTn ACT2)). Let the following variable assignments be made for the act excommunicate: ACT 1 = excommunicate; ACT 2 = confirm. Given the ceremony (Priest confirm Tom) and the ceremony (Priest confirm Bob); the ceremony (Priest excommunicate Tom and Bob) is legal.

The divorce ritual is more complex. Here more is necessary than just the variable assignments: ACT 1 = divorce; ACT 2 = marry. Given the ceremony (Priest marry ((Tom and Sue) and (Dick and Cathy))), the ceremony (Priest divorce (Tom and Dick)) would be accepted but obviously more is needed. Either a transformation to (Priest divorce ((Tom and Sue) and (Dick and Cathy))) or to (Priest divorce (Tom and Sue)) is necessary.
Some sort of additional rule to force a proper transformation is necessary. This rule must say that the people being divorced must have been married in pairs in the same ritual. The phrase "in pairs" meaning only two elements in any (and) list on lowest level: either (Tom and Sue) or (Dick and Cathy). The pattern for marry must guarantee that people are married in pairs: (RP marry (P1 and P2)). Plus a global pattern in divorce: (RP divorce (P1 and P2)). Then a variable assignment in the rules for the act divorce forcing the quality (ACT 2) to be acquired in the same ritual for all PTs: ACT2 = R1 [marry]. The combination of these patterns and reversal assignments will place the proper constraints on the divorce ritual.

It does not make sense for a person to be married and divorced at the same time. If an action is a reversal of another then it must fit the reversal transformation at this time period. A divor­cee must have been married before, and still be married at the time of the divorce. This constraint is a rule associated with the re­versal transformation at the structural level.

Comments and Open Problems

RSS is an experiment into an aspect of the general problem of knowledge representation. By working with actions the problem is reduced toward a manageable size. By studying a formally defined system of actions, as represented by religious ritual systems, the problem is reduced even more. Religious actions are very complex human activities, so the problem has not been reduced to a desert island case.
Like all theories (Popper, 1959), the theory of ritual syntax discovers new problems while solving old ones. One of the new problems involves the various ways one act can change the efficacy of another. The reversal transformation embodies the idea that Fran must be confirmed before excommunication. One act reverses the other. At that point the effective lifetime of the conformation starts when Fran was confirmed and ends when Fran is excommunicated. During that period of time Fran can participate in any action a confirmed person can but not outside that period. The reversal transformation can represent this direct influence of excommunicate on confirm.

But the system cannot represent the Bishop performing X on Fran and saying: "As long as you are confirmed you may participate in any action that a person with the quality X can." This statement ties the effective lifetime of X to the effective lifetime of confirm, at least as far as Fran is concerned. Fran is not required to have the quality X in order to be excommunicated so the reversal transformation does not apply.

Are there any connections like this in religious ritual actions? There are some in the secular world. Many people have had the effective lifetime of their imprisonment tied to the effective lifetime of the enthronement of "The King." Assassinations have, more than once, caused the release of political prisoners.
Appendix A

Parser for Original Grammar
/* Source File for LEX */
/* Ritual syntax Original Grammar */
/* Author: Bob Hardin, 1986 */

#include "y.tab.h" /* token declarations produced by YACC */
#include "Strings.h" /* string handling */
extern int yylval; /* value stack in yyparse declared by YACC */

ops [,.()]

/* token declarations produced by YACC */

[ \t\n\b]+ ;  
{ops} {yylval = StoreStrings(yytext); return(lact);}  
purify|purifies {yylval = StoreStrings(yytext); return(lact);}  
bless|blesses {yylval = StoreStrings(yytext); return(lact);}  
water|[Cc]hurch {yylval = StoreStrings(yytext); return(lobj);}  
hammer|fire {yylval = StoreStrings(yytext); return(lobj);}  
person [Pp]arishioner {yylval = StoreStrings(yytext); return(lpt);}  
[Bb]ob|[Ff]ran {yylval = StoreStrings(yytext); return(lpt);}  
[Tt]om|[Rr]uth {yylval = StoreStrings(yytext); return(lpt);}  
[Jj]esus|[Gg]od {yylval = StoreStrings(yytext); return(lpt);}  
[Pp]riest {yylval = StoreStrings(yytext); return(lpt);}  
with|to {yylval = StoreStrings(yytext); return(lc);}  
unspecified|luns {yylval = StoreStrings(yytext); return(luns);}  
superhuman|ls {yylval = StoreStrings(yytext); return(ls);}  
nonsuperhuman|lns {yylval = StoreStrings(yytext); return(lns);}  

#include <stdio.h>

yywrap() {return (1);}
/* YACC Source File */
/* Ritual Structures Original Grammar */
/* Author: Bob Hardin, W.M.U. 1986 */

%token lact lpt lc lobj Is Ins luns
%token AR1 AR2 A1 A2

%start R
%

R : RP RA RP
  { pchkerr(root = $$ = Push("R",$1,$2,$3)); }
  
RA : ACT
  { pchkerr( $$ = Push("RA",0,$1,0)); }
  | ACT AC
  { pchkerr( $$ = Push("RA",$1,0,$2)); }
  
RP : PT Q
  { pchkerr( $$ = Push("RP",$1,0,$2)); }
  | OBJ OQ
  { pchkerr( $$ = Push("RP",$1,0,$2)); }
  
Q : ls
  { pchkerr( t = Push(Strings[$1],0,0,0));
    pchkerr($$ = Push("Q",0,t,0)); }
  | lns
  { pchkerr( t = Push(Strings[$1],0,0,0));
    pchkerr($$ = Push("Q",0,t,0)); }
  | luns
  { pchkerr( t = Push(Strings[$1],0,0,0));
    pchkerr($$ = Push("Q",0,t,0)); }
  | R
  { pchkerr($$ = Push("Q",0,$1,0)); }
  
OQ : lns
  { pchkerr( t = Push(Strings[$1],0,0,0));
    pchkerr($$ = Push("Q",0,t,0)); }
  | luns
  { pchkerr( t = Push(Strings[$1],0,0,0));
    pchkerr($$ = Push("Q",0,t,0)); }
  | R
  { pchkerr($$ = Push("Q",0,$1,0)); }
  
AC : C OBJ OQ
  { pchkerr( $$ = Push("AC",$1,$2,$3)); }
  
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
OBJ : lobj
   {  pchkerr( t = Push(Strings[$1],0,0,0));
       pchkerr($$ = Push("OBJ",0,t,0));  }
;

PT : lpt
   {  pchkerr( t = Push(Strings[$1],0,0,0));
       pchkerr($$ = Push("PT",0,t,0));  }
;

ACT : lact
   {  pchkerr( t = Push(Strings[$1],0,0,0));
       pchkerr($$ = Push("ACT",0,t,0));  }
;

C : lc
   {  pchkerr( t = Push(Strings[$1],0,0,0));
       pchkerr($$ = Push("C",0,t,0));  }
;

%%
# include <stdio.h>
# include ".../base/Strings.h"
# include ".../base/TriTree.h"
# define pchkerr(F) if (  (F) <0)return(yyerror("error in Push"))

extern yytext[];
int root = 0;
int t;

/* >>>>>>>>>>>>>>> yyerror <<<<<<<<<<<<<<<< */

/* this function is called by yyparse() for error handling */

/* output is to stderr */

yyerror(s)
char * s;
{
   fprintf(stderr,"
 %s %s
",s,yytext);
   return(-1);
}
Actual Output of Program

31 % parser

Bob
  Fran ins bless Fran ins
bless
person superhuman

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
32% parser

Bob
   Fran ins bless Bob ins
   bless person ins

PT
RP
Q

PT
RP
Q
ins

PT
RP
Q
ins

PT
RA
ACT
bless

PT
RA
ACT
bless

PT
person

RP
Q
ins

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
33 % parser

person ins

   purifies with water ins
person ins

R

   ACT

   RA

   C

      with

   AC

      OBJ

      water

      OQ

      ins

   PT

   person

RP

   Q

   ins
Appendix B

Parser Incorporating Transformations
/* Source File for Lex */
/* Incorporating Transformations and Unwritten Rule */
/* Author: Bob Hardin */

{%
#include "y.tab.h" /* token declarations produced by YACC */
extern char * malloc();
extern char * strcpy();

char * c_V;
#define SS(P) ((c_V=malloc(strlen(P)+1))==NULL)?NULL:strcpy(c_V,P)
#define Strcp(D,S) if((D=SS(S))==NULL)
    return(fprintf(stderr,"out of memory"),'e')
%
}
ops [,.()]
%
[ \t\n\b]+ ; /* ignore comments from ; to end of line */
(ops) {return(yytext[0]);}

purify|purifies {yyval.r.object = "purify";
    yylval.r.action = NULL; return(lact);}
bless|blesses {yyval.r.object = "bless";
    yylval.r.action = NULL; return(lact);}
marry|marries {yyval.r.object = "marry";
    yylval.r.action = NULL; return(lact);}
divorce|divorces {yyval.r.object = "divorce";
    yylval.r.action = "marry"; return(lact);}
confirm {yyval.r.object = "confirm";
    yylval.r.action = NULL; return(lact);}
excommunicate {yyval.r.object = "excommunicate";
    yylval.r.action = "confirm"; return(lact);}
water|church {Strcp(yylval.sval.yytext); return(lpt);}
hammer|fire {Strcp(yylval.sval.yytext); return(lpt);}


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Church
with\to
unspecified|luns
superhuman|ls
nonsuperhuman|lns
return(lns);

%
#include <stdio.h>

yywrap() { return (1);}

{Strcp(yylval.sval ,yytext); return(lpt);}
{Strcp(yylval.sval ,yytext); return(lc);}
{yyval.sval = "unspecified"; return(luns);}
{yyval.sval = "superhuman"; return(ls);}
{yyval.sval = "nonsuperhuman";
/* YACC Source File */
/* Object Agency and Reversal Transformations */
/* Incorporated Into To The Parser */
/* Author: Bob Hardin, W.M.U. 1986 */

{%
# include <stdio.h>
# include "TriTree.h"

# define pchkerr(F) if ( (F) == NULL)\  return(yyerror("error in Push"))

/* global variables */
extern yytext[];
struct node * root = NULL;
struct node * t;
%
    /* define the value stack of the parser */
    /* this is a union named yyval */
%
union {
    struct node * nval; /* pointer to node in parse tree */
    char * sval; /* pointer to string from LEX */

    struct { /* r (ritual) */
        /* stores context-sensitive info */
        struct node * nval; /* a node in the parse tree */
        char * object; /* the object of a ritual action */
        char * action; /* the act performed */
    } r;

    /* lex returns r along with token lact */
    /* with the act in object and the */
    /* act it reverses or NULL in action */
}

%token <sval> lpt lc lobj ls lns luns
%token <r> lact

%type <r> R RP RA PT OBJ ACT Q OQ
%type <nval> AC C P

%start R
R : '(' R ')'
/* allow placing rituals in parens */
{ $$.nval = $2.nval;
 $$.object = $2.object;
 $$.action = $2.action;
}

| P RA RP
/* before reducing store the action and its object */
/* for possible later use being reduced to a Q */
{
int mess;
pchkerr(root = $$, nval = Push("R",\$1,\$2.nval,\$3.nval));
$$, object = $3.object;
$$, action = $2.object;

/* check that if RA reverses an action */
/* then RP has quality of that action */
if ($2.action != NULL) /* RA must reverse something */
{ mess = 1;
 if ($3.action != NULL)
 { if (strcmp ($2.action, $3.action) == 0) mess = 0; }
 if (mess) return (fprintf (stderr, "error: attempt to give \$s the Q \$s which has \$s as its object\n",
 $1.object, $2.action, $2.object), -1);
 } /* end if RA reverses something */
}

P : PT Q
/* check that if Q is a ritual it has PT as object */
{ if ($2.action != NULL)
 { if (strcmp ($1.object, $2.object))
 return (fprintf (stderr, "error: attempt to give \$s the Q \$s which has \$s as its object\n",
 $1.object, $2.action, $2.object), -1);
 } pchkerr( $$ = Push("RP", $1.nval, NULL, $2.nval));
}

RA : ACT /* RA puts an r on value stack with */
/* object: the name of the action */
/* action: the name of what it reverses if anything */

{  pchkerr( $$.nval = Push("RA",NULL,$1.nval,NULL));
    $$ .object = $1 .object;  $$ .action = $1 .action;
}

| ACT AC
{  pchkerr( $$ .nval = Push("RA",$1 .nval,NULL,$2));
    $$ .object = $1 .object;  $$ .action = $1 .action;
}

| OBJ QO

/* check that if QO is a ritual it has OBJ as object */

{  if ($3 .action != NULL)
    {
    if (strcmp($2 .object,$3 .object))
        return( fprintf(stderr,
                    "\nerror: attempt to give %s the Q %s which has %s as its object\n", 
                    $2 .object,  $3 .action,  $3 .object),
                    -1);
    }
    pchkerr( $$ = Push("AC",$1,2 .nval,$3 .nval));
}

| OBJ QO

/* check that if QO is a ritual it has OBJ as object */

{  if ($2 .action != NULL)
    {
    if (strcmp($1 .object,$2 .object))
        return( fprintf(stderr,
                    "\nerror: attempt to give %s the Q %s which has %s as its object\n", 
                    $1 .object,  $2 .action,  $2 .object),
                    -1);
    }
    pchkerr( $$ .nval = Push("RP",$1 .nval,NULL,$2 .nval));
    $$ .object = $1 .object;  $$ .action = $2 .action;
}
{ if ($2.action != NULL)
  
  if (strcmp($1.object,$2.object))
    return( fprintf(stderr,
      "error: attempt to give %s the Q %s which has %s as its object\n",
      $1.object, $2.action, $2.object),
    -1);
  }
pchkerr( $$.nval = Push("RP",$1.nval,NULL,$2.nval));
$$.object = $1.object; $$.action = $2.action;
}

Q :  Is
{ pchkerr( t = Push($1,NULL,NULL,NULL));
  $$object = $1;
  $$action = NULL;
  pchkerr($$.nval = Push("Q",NULL,t,NULL)); }
| Ins
{ pchkerr( t = Push($1,NULL,NULL,NULL));
  $$object = $1;
  $$action = NULL;
  pchkerr($$.nval = Push("Q",NULL,t,NULL)); }
| lns
{ pchkerr( t = Push($1,NULL,NULL,NULL));
  $$object = $1;
  $$action = NULL;
  pchkerr($$.nval = Push("Q",NULL,t,NULL)); }
| R
{ $$object = $1.object; /* pass on info for ritual */
  $$action = $1.action;
  pchkerr($$.nval = Push("Q",NULL,$1.nval,NULL)); }

OQ :  Ins
{ pchkerr( t = Push($1,NULL,NULL,NULL));
  $$object = $1;
  $$action = NULL;
  pchkerr($$.nval = Push("OQ",NULL,t,NULL)); }
| lns
{ pchkerr( t = Push($1,NULL,NULL,NULL));
  $$object = $1;
  $$action = NULL;
  pchkerr($$.nval = Push("OQ",NULL,t,NULL)); }
| lns
{ pchkerr( t = Push($1,NULL,NULL,NULL));
  $$object = $1;
  $$action = NULL;
  pchkerr($$.nval = Push("OQ",NULL,t,NULL)); }
| R
{ $$object = $1.object; /* pass on info for ritual */
  $$action = $1.action;
  pchkerr($$.nval = Push("OQ",NULL,$1.nval,NULL)); }

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
OBJ : lobj
{ pchkerr( t = Push($1, NULL, NULL, NULL));
  $$ . object = $1;
  pchkerr($$. nval = Push("OBJ", NULL, t, NULL)); }

PT : lpt
{ pchkerr( t = Push($1, NULL, NULL, NULL));
  $$ . object = $1;
  pchkerr($$. nval = Push("PT", NULL, t, NULL)); }

ACT : lact
{ pchkerr( t = Push($1 . object, NULL, NULL, NULL));
  $$ . object = $1 . object;
  $$ . action = $1 . action;
  pchkerr($$. nval = Push("ACT", NULL, t, NULL)); }

C : lc
{ pchkerr( t = Push($1, NULL, NULL, NULL));
  pchkerr($$ = Push("C", NULL, t, NULL)); }

%%

/* >>>>>>>>>>>>> yyerror <<<<<<<<<<<<<< */

/* this function is called by yyparse() for error handling */

yyerror (s)
char * s;
{ fprintf(stderr,"\n %s %s\n", s, yytext);
  return(-1);
}

/* >>>>>>>>>>>>>>>>>>>>>>> main <<<<<<<<<<<<<<<<<<<<<< * /

/* This program builds and prints a parse tree */
/* using the function yyparse() built by YACC */
/* from the source file above */

main()
{ if (yyparse()) return(-1); /* parse input from stdin */
  TriPrint(root, 4); /* print the parse tree */
}
47 % parser

Bob

Fran lns bless Fran lns

error: attempt to give Bob the Q bless which has Fran as its object

48 % parser

Priest (Jesus ls purify Priest luns)
excommunicate
Fran (Priest luns marry Fran luns)

error: attempt to excommunicate Fran without quality confirm

49 % parser

Priest (Jesus ls purify Priest luns)
divorce
Fran (Priest luns marry Fran luns)
50 % parser
Parishioner purifies with water
Priest Church
Jesus purifies Church
purifies Priest
purifies water
Parishioner luns

R

PT Parishioner
RP Q unspecified
R

ACT purifies
RA with
AC OBJ water
OQ

PT Priest
RP Q
R

Church
PT Jesus
RP Q superhuman
R

RA ACT purify
PT Priest
RP Q nonsuperhuman
R

RA ACT purify
PT Priest
RP Q

RP Q nonsuperhuman
PT Parishioner
RP Q unspecified
BIBLIOGRAPHY


