VI. Template Analysis and its Application to Natural Language Processing

Stephen F. Weiss

Abstract

The first section of this paper introduces the three basic types of templates and the theoretical basics of template analysis. Building on this, the second section presents some practical considerations needed for the implementation of a general template analysis scheme, including the concept of template keywords. An actual implementation of template analysis is presented in Section 3. The scheme uses template analysis for the extraction of bibliographic phrases from natural language text. Basic implementation techniques as well as experimental results are presented.

1. The Basics of Template Analysis

A) Introduction

Template analysis is a technique for partial semantic analysis of natural language. It consists of matching a natural language input string against a predetermined set of templates. A template is a string of one or more words or special symbols. Its exact meaning will become clear in the examples that follow. If a match occurs between a complete template and some part of the input, a specified set of

actions is performed. These may involve direct effects which change the input string or side effects which work on other data. The process may terminate after the discovery of a single template match as in ELIZA [3], but more often it continues until all template matches in the input have been located and the associated actions performed.

Each template is a string, usually made up of very common words, which forms the framework of a natural language phrase. The template words generally convey little meaning. They do, however, create a context into which other words may fit and thus form a meaningful phrase. These words which fit into the template and bear the crux of the meaning of the phrase are called the template environment. The words in the template thus indicate the existence of a specific type of phrase as well as qualifying the environment. The template environment gives specific meaning to that phrase. Template analysis is basically a two step process. First, the input is compared with the template set. A match gives an indication that a certain type of phrase is present. Second, the environment of the matched segment of the input is checked. This either provides the exact meaning of the phrase or rejects the original analysis. The following examples serve to clarify the meaning of templates and template analysis. Consider the template:

BETWEEN 0 AND 0

The zeros indicate "holes" where one and only one word may occur. This template matches phrases A, B and C in Figure 1. It will not match D or E because the holes contain the wrong number of words.

A. BETWEEN A AND Z
B. BETWEEN HERE AND THERE
C. BETWEEN 1950 AND 1965
D. BETWEEN AND BEYOND
E. BETWEEN LAST YEAR AND NOW

Sample input phrases
Figure 1

A match of this template with an input string provides an indication of a date phrase. To verify this it remains only to determine if the words in the template environment, in this case the two words in the holes, are date indicators. If such is the case as in C, appropriate action is taken such as passing the dates to a search program. If as in A and B the environment check fails, the phrase is rejected. The environment check can sometimes be eliminated by the use of a more complete template. Consider the modified version of the previous template:

BETWEEN 19## AND 19##

The pound signs indicate the presence of any digit. Of the inputs in Figure 1, this template matches only C. No further testing of the environment is needed in this case and appropriate actions can be taken immediately.

B) Types of templates

There are three basic types of templates. They are distinguished by the processing required and results achieved in the two step template analysis process.

1. Certainty (C1). Under certain circumstances the template match phase provides all the information required to determine the meaning of a phrase. The environment test is thus not needed. An example of such a template is the second of the above examples:

BETWEEN 19## AND 19##

The template is complete within itself and the environment test may be omitted. While a high degree of accuracy in analysis is expected from them, C1 templates are not usable in practice. The large number of templates required to provide any degree of generality in allowable inputs, along with the huge task of creating the template list, makes C1 templates impractical for all but a few cases.

2. Conjecture-Certainty (C2-C1). The second type of template, the C2-C1, is typified by the following:

BETWEEN O AND O

where the zeros match any single word. In this case a template match is not a guarantee of a particular type of phrase. This example in particular matches inputs A, B, and C in Figure 1. The system thus only conjectures the existence of a specific phrase. This decision is subject to possible rejection depending on further examination of the template environment. Environment testing may consist of determining if the environment words are of a specified type or types. Or it may consist of testing environment words against a table. In all cases of C2-C1 templates, the template match produces a conjecture which is then completely resolved by the environment test, hence the name "conjecture-certainty". Because of this final certainty, the C2-C1 templates provide nearly the same high degree of accuracy as do the C1 templates. The analysis of the C2-C1 template matches, however, is more complex due to the extra testing needed to provide certainty. The prime advantage of the C2-C1 templates is that only the frame of a phrase need be specified. The job of determing the template set is greatly eased and the number of templates required for each phrase type is significantly reduced. As an example, in the small system implemented for this study, journal phrase detection is accomplished using 18 C2-C1 templates and a journal list. To do the same job with C1 templates alone

requires 450 templates, each longer than the present 18. Savings in search time, storage space as well as template preparation effort are substantial.

3. Conjecture-Conjecture (C2-C2). The third type of template is quite similar to the C2-C1 in that it consists of only the frame of a phrase, and a match yields only a conjecture that a phrase exists. Also like the C2-C1, an environment test is performed after the match is found. But in this case the environment test merely weakens or supports the original conjecture. The result is not certain, but is a second, and probably better, conjecture and hence the C2-C2 designation. It is apparent that because of the uncertainty in the final result, analyses with these templates cannot be expected to be as accurate as those of the C1 or C2-C1 types. The C2-C2 templates are necessitated whenever the environment for a phrase becomes too large or varied for practical use of a look-up table or environment testing rules. Such is the case of author phrases where a list of all possible forms of all author names is prohibitively large. A template such as:

WRITTEN BY

may give an indication of a possible author phrase; and there exist certain rules which may help the decision (see Section 3, part C). But the final decision of phrase type is still

only an educated guess. This, however, is better than no guess at all, and thus the C2-C2 templates serve a useful purpose when no other method is applicable.

From the standpoint of generality and ease of creation and use, the C2-C1 templates seem best. For this reason the vast majority of the templates used in this study are of this type. Figure 2 below summarizes the three types of templates.

Template type	Decision after template match	Decision after environment test
C1	certain	not needed
C2-C1	conjecture	certain
C2-C2	conjecture	improved conjecture

Summary of template types

Figure 2

C) Applicability of template analysis

The use of template analysis requires an a priori knowledge of every type of natural language input that is to be analyzed. This is specified in the form of the templates themselves as well as environmental testing rules and look-up tables. If the input form is quite varied or unpredictable, then clearly template analysis cannot be used. In such cases more general and complex forms of syntactic and semantic analysis are required. But there are some

applications in which the input structure meets the criteria for template analysis. One such application is input from a console. While the semantic and syntactic structures that the average person can understand are almost without bound, the language that a person actually uses, called his performance grammar, is a small subset of his understandable language. This performance set is further restricted when the user must type rather than speak his input. The extra time and effort required to compose and type complicated sentence structure tend to discourage such actions. When the further restriction is added that for each input the user is limited to a specific set of input types, the inputs become sufficiently predictable to permit effective use of template analysis. The remainder of this paper explores the uses of template analysis and describes a working implementation.

2. An Implementation of Natural Language Analysis by Template Analysis

The theoretical basis of template analysis has been presented in the previous section. The present section explores the practical considerations used in the implementation of an experimental template analysis system.

A) Keyword analysis

Searching an input string against a template set can often be a very inefficient process. The vast majority of templates do not match a given input string, and of these nonmatching templates, the majority do not have even a single word in common with the input. These templates thus have no possible chance of matching the given input, and it would be advantageous to exclude them from the template matching phrase. For this reason the technique of keyword templates is introduced. This is similar to the scheme used by Weizenbaum in ELIZA [3]. The technique entails assigning one of the words in each template to be the keyword. As the templates are entered into the system, those with identical keywords are stored in a group. After all the templates have been entered, a keyword list is created. The list contains the keyword for each group, and the storage location of the first and last template for that particular group. The template matching process begins with a search of the keyword list. If the particular keyword is contained in the input, then each member of the associated template group is searched for a possible match. If the input does not contain the particular keyword, the associated template set is ignored. Thus with a single scan of the input, an entire group of obviously nonmatching templates is eliminated

from the set which must be searched. The savings in search time by keyword templates is quite substantial. These savings are further enhanced if the keyword is the least likely word in the template. "Least likely word" is defined as that word in the template which occurs least often in contexts other than that of the given template phrase. Consider the template:

DURING O TO O

Clearly the use of "DURING" as the keyword leads to fewer unsuccessful match attempts than would the use of "TO". Thus by eliminating those templates which have no chance for a match with a given input, the keyword template facility greatly streamlines the complex and fairly lengthy template matching process.

The keyword template technique has a second useful function; it permits a middle-out template matching scheme. Without keywords, template matching algorithms require a decision of where to start. The only solution is to try every word in the input as a possible first word of the template. If a match occurs, the next word of the input is tested against the second template word, and so on. For long input strings this can be a time consuming process. This is especially true for nonmatching templates since rejection

cannot occur until every input word has been tested. Since the keyword scheme tests only those templates which have at least one word in common with the input (i.e., the keyword) this common word provides an ideal place to start the search. For inputs with two or more occurrences of the same keyword, one is designated as the principal key and searching begins there. Words to the left of the keyword in the template are tested against words to the left of the keyword in the input, and likewise for the right; and hence the middle-outward strategy. Most templates used in this study have their keyword as either the first or last element, so that most searching is actually in one direction only. But the important thing about this technique is that the search seed, that is the starting point for the search, is uniquely determined. This also permits rejection of nonmatches to occur after only a very few tests rather than after exhaustion of the entire input string. A further advantage of the keyword process involves the inputs containing more than one substring which matches a given template. A normal matching process searches the input left to right and picks out the first of the substrings. This, however, may not be the intent of the system designer who in fact may want the second, or possible rightmost, sub-

phrase analyzed first. The solution is easy using keywords. Inputs with multiple matches of the same template must contain multiple occurrences of the keyword for that template. By simply designating the desired keyword as the principal key, the proper substring is analyzed first.

Thus the keyword technique helps in template matching on two levels. First, it narrows the set of templates which must be tested against the input, and second it provides a specific point in both the template and input for the start of the search. This permits the matching process to run faster and more efficiently, as required for on-line use.

The template matching process proceeds in a manner similar to that of a Markov algorithm [13]. The template list is scanned in top down order until a match with the input is found. The associated actions are then carried out. The process is then restarted at the top of the template list. Of course this necessitates removal from the input string of matched substrings in order to prevent repeated matching of that substring. This is generally done by replacing the substring with a nonterminal symbol appropriate to the type of template matched. This nonterminal may then be used in subsequent template matches. The process terminates when the end of the template list is reached without finding a match.

The advantage of the Markov approach is that it enables a specific hierarchy of templates and ordering of analysis to be imposed in the system, as well as allowing a desired template grouping to be maintained.

B) Implementation conventions

The templates shown in the previous examples are made up of words and are represented alphabetically. While these are easy to read and understand, they are not practical for an actual implementation. Alphabetic templates require large amounts of storage with a variable amount of space needed for each word. Second, alphabetic searches are quite complex and time consuming. And third, the use of alphabetics makes it necessary that even synonymous templates be represented individually. For example the templates below all have the same meaning and associated actions.

WRITTEN BY
AUTHORED BY
PRESENTED BY

The use of alphabetic templates would require that all three be in the template set. From actual experimentation it appears that there are in fact many such synonymous templates, and thus a very large template set can result. Clearly a

shorthand template representation which overcomes these problems is desired. For this reason numerical templates and input vectors are used. The conversion from a word to its associated numerical concept is achieved through a special template dictionary. The dictionary is quite small and is composed, for the most part, of functor type words, that is conjunctions, prepositions and a few other very common words. Words not found in the dictionary are assigned the concept "999" to indicate that they are unknown. The dictionary used in this implementation contains 140 words which map into 22 concept numbers. The problem of synonymous words is solved by assigning to the desired words the same concept number. The numeric input vector maintains the same word order as the natural language input. The example below shows two natural language inputs and the numeric vector that results from either of them. Note the treatment of synonyms and unknowns.

Input A:	PAPERS WRITTEN BY LESK ON CLUSTERING
Input B:	ARTICLES AUTHORED BY WEISS CONCERNING TEMPLATES
Concept Vector:	27 15 12 999 999 999

Sample natural language and numeric input strings

Figure 3

Templates are also stored in numeric form with the addition of some special conventions explained in the following paragraphs. In the remainder of this study all templates and stored input strings are assumed to be numeric. Occasionally, however, some alphabetic examples may be presented for purposes of clarity. The use of numeric representation of inputs and templates clearly solves the previously mentioned synonymy problem. The resultant reduction in the size of the template set varies with the number of synonymous words, in the present implementation 44 templates are required instead of more than two thousand. Numeric templates also make storage and searching easier. Unlike an alphabetic string, a number uses a fixed amount of storage. Comparison of numbers is also easier and faster than comparison of variable length alphabetic strings. In general the use of numeric vectors provides a more compact data representation and facilitates faster and more efficient processing.

Templates are stored numerically in a similar manner to that of input strings. They may be entered into the system in alphabetic form and run through a dictionary look-up, or they may be entered in numeric form. The latter is used in this implementation. The general form of a template is given in the BNF definition in Figure 4. The reason for the distinction of the digits in lines 5, 6

and 7 will become clear in what follows.

1. Template ::= <Parameter part> <Template elements>
2. Parameter part ::= <Action number> <Key position> <Length>
3. Template elements ::= <Template symbol> | <Template symbol> <Template elements>
4. Template symbol ::= <Concept number (i.e., a number ≥11)> | Minus sign <Concept number>
5. Template symbol ::= 0 (zero)
6. Template symbol ::= digit 1-9
7. Template symbol ::= 10 (ten)

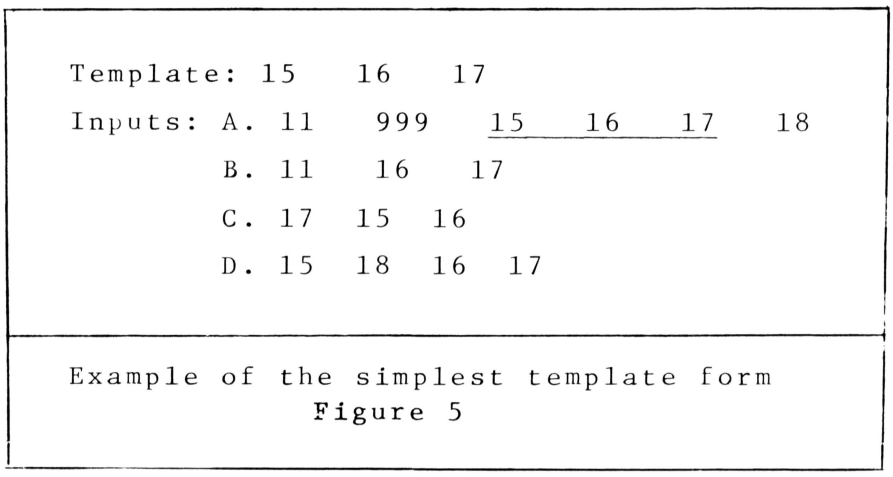
Formal definition of a template
Figure 4

The action number is the identification number of an action routine which is to be executed if that particular template is matched. More than one template can have a particular action numer. The key position indicates the keyword within the template. And length is merely the number of elements in the template. A general template is shown below:

55 3 7 15 -16 11 0 18 5 18

The three parameters here indicate that action routine number 55 is to be called if this template is matched. Also the template is seven elements in length and the keyword is in position three of the template elements (i.e., the 11).

As shown in Figure 4 the template elements consist of numbers and certain of these numbers have special meaning. The simplest form of a template is shown in Figure 5 along with several sample inputs. The template consists entirely of concept numbers, that is positive integers greater than ten. Such a template matches an input if and only if an exact match of concept numbers, ordering, and contiguity occurs. The sample template matches the underlined portion of input A. It does not match B because not all the concepts are present; C has incorrect ordering; and in D the 15 and 16 are not adjacent. In this example and in all subsequent template samples, the three parameters are omitted for simplicity.



Example of the simplest template form
Figure 5

Most desired templates can be specified in this form but there are several drawbacks. First, there is no way to specify templates with "holes" in them, that is a template with noncontiguous elements. And second the number of templates required by this format is fairly large. In particular there is no facility for combining similar templates, or templates of which one is a subset of another. Thus, a shorthand for representing numeric templates is needed. By combining some of the templates, the size of the template set is reduced, and search time is also saved since one search of the combined template replaces the several individual searches.

The present implementation allows four options in template specifications. The first is the optional concept. It is often useful to be able to specify a certain concept in a template as being optional. The template matches an input if all the required concepts match. If any optional concepts also match, this is noted for possible use in the action routines. An optional concept is denoted by a minus sign preceding the concept number. In the example below (given here in alphabetic) the template matches all inputs shown.

```
Template: -PUBLISHED IN -THE -YEAR
Inputs: A.  IN
        B.  PUBLISHED IN
        C.  PUBLISHED IN THE YEAR
        D.  PUBLISHED IN  YEAR
```

Example of optional concept facility
Figure 6

The example indicates the usefulness of this technique in combining several templates. This facility is also useful in combining the positive and negative forms of a phrase as is seen below.

```
Template: -NOT PUBLISHED IN
Inputs:  A.  PUBLISHED IN
         B.  NOT PUBLISHED IN
```

Positive and negative phrase combination
Figure 7

The indicated template matches both inputs. The occurrence or non-occurrence of the "NOT" is passed on to the action routine so that the proper action may be taken. The system always tries to match as many of the optional concepts as possible and of course the keyword concept may not be optional.

It is often desirable to specify templates with holes in them, that is having noncontiguous elements. This is made possible by the use of three types of skip operators. The first type, the single skip, is indicated by a zero in the template. This specifies that exactly one input word is to be skipped. The example in Figure 8 illustrates this operator. The template matches only those inputs with the words "COMPUTER" and "DEPARTMENT" separated by one arbitrary word. Thus a match occurs with input B only.

```
Template:  COMPUTER   0   DEPARTMENT
Inputs:    A.  COMPUTER  DEPARTMENT
           B.  COMPUTER SCIENCE DEPARTMENT
           C.  COMPUTER SCIENCE EDUCATION
                    DEPARTMENT
```

Single skip operator
Figure 8

A second type of skip operator is the bounded skip. This is indicated by a number i where $1 \leq i \leq 9$, and specifies that as few as zero, or as many as i, words are to be skipped at that point in an effort to match the template. If several matches are possible, the one involving the fewest number of skipped words is chosen. Figure 9 demonstrates this facility. The template matches any input with "COMPUTER"

and "DEPARTMENT" in that order and separated by two words or less. Matches thus occur with inputs A, B and C, but not with D because there are three separating words. A match also occurs with the underlined portion of E. Note that in input E, the second occurrence of "DEPARTMENT" instead of the first would also satisfy the template. This illustrates the usefulness of skipping the least possible number of words.

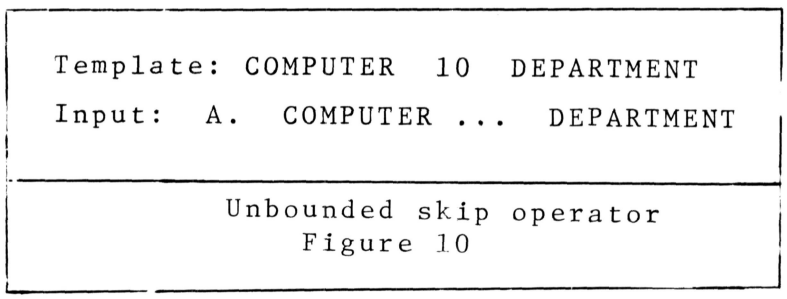
Template: COMPUTER 2 DEPARTMENT

Inputs:
A. COMPUTER DEPARTMENT
B. COMPUTER SCIENCE DEPARTMENT
C. COMPUTER SCIENCE EDUCATION DEPARTMENT
D. COMPUTER INFORMATION AND SCIENCE DEPARTMENT
E. COMPUTER SCIENCE DEPARTMENT DEPARTMENT

Bounded skip operator
Figure 9

The facility to skip from zero to nine words is adequate for most templates used in this implementation. For those few templates for which a skip greater than nine is needed, no upper bound on the number of skipped words is really desired. To indicate this, the unbounded

skip is used. This is specified by the number ten and causes an arbitrary number of input words to be skipped in an effort to match the template. Figure 10 demonstrates this. A match always occurs with input A. The ellipsis indicates a string of words of arbitrary length but of course not containing the word "DEPARTMENT".



Unbounded skip operator
Figure 10

For all the skip operators a record is kept of how many words are skipped. This information can then be used by the action routines. The use of the optional concepts and skip operators reduces the number of needed templates to 44 as opposed to the more than 81 which would otherwise be needed for this implementation.

While the previously presented conventions are adequate for the present system, several other facilities are suggested for future use. It is sometimes useful to be able to specify that a group of words may appear in any order and in an

assignable proximity to one another. For example the template in Figure 11 matches any input containing the words "INFORMATION" and "PROCESSING" within a three word group. It matches inputs A, B and C. This can be generalized to words in the same sentence or words in the same input.

Template: 3 (INFORMATION PROCESSING)
Inputs: A. INFORMATION PROCESSING
B. INFORMATION TEXT PROCESSING
C. PROCESSING OF INFORMATION

Arbitrary ordering and proximity
Figure 11

A second addition recommended for the future is the facility to indicate that a single template spot may be occupied by any member of a group of concepts. This can be partially accomplished in the present system by mapping the desired words into the same concept number. But there are some instances in which several nonsynonymous words (i.e., with different concept numbers) may appear in similar templates. To allow such a substitution would reduce the number of required templates. The example of Figure 12 shows two standard templates and a third which combines the first two.

Templates:	A.	COMPUTER PROGRAM
	B.	COMPUTER SYSTEM
	A+B.	COMPUTER <PROGRAM, SYSTEM>

Combination of similar templates

Figure 12

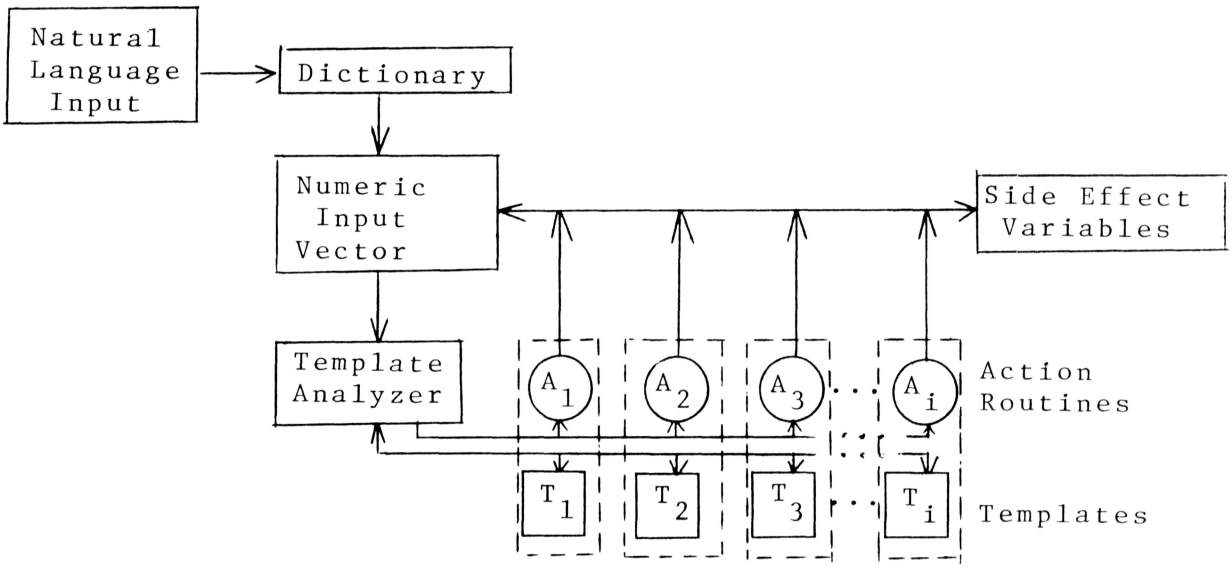
A third possible improvement would render whole sets of words as optional rather than single words only. Notice, for example, that in Figure 6 the template matches

PUBLISHED IN THE
PUBLISHED IN YEAR

both of which have dubious meaning. In this case it might be desired to make "THE YEAR" optional as a phrase rather than as individual words. This can eliminate undesirable matches with partial phrases. For the most part these suggestions do not add any power to the template, but they do allow a more compact template set and more efficient template searching.

This section has described the template analysis implementation under study. The complete system is summarized in Figure 13. Natural language input is converted to numeric form by the dictionary. The template analyzer then tries to match a template with the input. If

a match is found, the action routine associated with the template is executed. This process results in a modified input, and possibly in the setting of some side effect variables Control then returns to the template analyzer and the process continues until no further template matches exist. The process then halts. The template analyzer is a tool which can be used in a number of applications. One such application is discussed in Section 3.



Summary of template analysis system

Figure 13

3. An Implementation of Template Analysis

This section presents an application of the template analysis scheme for extraction of various forms of bibliographic information from natural language text. The template analysis section extracts date, journal, and author phrases from English input; and the action routines convert this information into a form usable by a search program. This system can be used as a query processor in an information retrieval system thereby allowing the user to include bibliographic information in his query. This system is chosen for implementation and study not only because of its obvious application to information retrieval but also because the techniques used cover the full spectrum of template analysis schemes. It is therefore expected that the problems encountered and results achieved in the operation of this pilot system are typical of those to be expected in any template analysis application. Thus while it might be argued that the present analysis facility is not practical for actual information retrieval, the implementation does provide a useful indication of the performance of template analysis in general. The following three sections deal respectively with the extraction of date, journal, and author phrases and their associated problems and techniques.

The order of presentation is the order in which the types of phrases are isolated from the input.

A) Date phrases

In this system a date phrase is defined to be any phrase which specifies a year or group of years. Single date phrases contain only one date reference while double date phrases contain two. Examples of date phrases are shown in Figure 14.

Single date phrases
IN 1965
NOT IN 1964
BEFORE 1967
SINCE 1960
Double date phrases
FROM 1955 TO 1967
NOT BETWEEN 1957 AND 1960
DURING 1950-1955

Examples of date phrases
Figure 14

The somewhat more complex problem of date phrases involving months or specific days is not considered here. To handle phrases a slight modification of the template dictionary look-up procedure is employed. The general procedure used

is a binary search on the first character of a word to get into the proper section of the dictionary, followed by a linear search for the whole word. This is altered slightly in the case of words beginning with the number one. When such a word is found, and before the linear search of that section of the dictionary is started, a check is made to determine if the word consists of exactly four numbers with the first one being a one and a nine. If so, the word is assumed to be a date and is assigned the date concept number 1900. If not, normal dictionary procedure continues. This actually provides a quick and compact way of storing all dates from 1900 to 1999 in the dictionary with the concept number 1900 for every one.

The alphabetic templates for date phrases are shown in Appendix A. The word "DATE" in quotes denotes the position of the date concept 1900. The "DP" is a nonterminal symbol denoting a single date phrase. Template analysis is much like context-free bottom-up analysis with each isolated phrase being replaced by a nonterminal. In this case the nonterminal for single date phrase enters into the analysis for the double date phrase. In all templates except the last two, the 1900 concept is the key. Thus a search of the date phrase templates is performed only when a word of the form 19ij appears; this provides a good indication of the presence of a date phrase. When a date phrase template is matched,

no further tests need to be made to validate the decision. It of course remains necessary to look at the input to determine exactly what year is specified; this however has no bearing on whether or not the matched phrase is a date phrase. The date phrase decision is certain at template match time, and thus is an example of a certainty or C1 template match. For this reason date phrases are easiest to recognize among the three phrase types, and provide the least potential for error. Performing the date phrase analysis first provides a way to accurately break up an input into smaller units which facilitates easier subsequent phrase isolation.

Once the date phrase has been recognized, there is still the problem of storing the indicated information in a useful way. For the most part dates appear in either single or double date phrases. It thus seems natural to store date specifications in pairs. Each element of the pair can contain an operator and a date. The operator part indicates the specified requirement concerning each date. Internally the operators are represented numerically and are shown in Figure 15.

Operator	Numerical Equivalent
=	3
≠	-3
<	2
≥	-2
>	1
≤	-1

Operators and their numerical equivalents

Figure 15

Single date phrases use only the first element of the pair, while double date phrases use both. More than one pair of date elements may be filled by a single input. Figure 16 gives some examples of input phrases and the resultant date pair elements.

The date pair form specified above is one that can easily be used by a search program. There is, however, one further process that must be carried out to coordinate the date specifications. This process need not be applied for those inputs which fill no more than one date pair. But for those that use two or more, three possible relations may occur and these necessitate appropriate modifications of the date specifications. First, two single date phrases may act in a bounding capacity, that is they act as a double date

Input	Result
General form	op | date
IN 1965	3 | 1965
BEFORE 1965	2 | 1965
BETWEEN 1964-1967	-2 | 1964 -1 | 1967
NOT DURING 1950-1951	2 | 1950 1 | 1951
DURING 1950-1955 OR DURING 1960-1965	-2 | 1950 -1 | 1955 -2 | 1960 -1 | 1965

Sample date representations

Figure 16

phrase. In this case the single date phrases are combined into the appropriate double specification. Second, one double date phrase may specify a date interval lying completely inside an interval specified by another double date phrase. Here the enclosed date interval is discarded and only the larger remains. Third, two double date phrases may specify overlapping date intervals. In this case the overlapping intervals are combined into a single interval which is the union of its constituents. The final date

specification is the union of those specifications which remain after the appropriate coordinations have been performed. Figure 17 summarizes this coordination phrase of the date phrase analysis and provides some examples.

Type	Example	Action	Result
Bounding single date phrases	AFTER 1965... BEFORE 1968	Combine into one double	1 1956 2 1968
Inclusive	IN 1965-1967 IN 1960-1968	Use larger one only	-2 1960 -1 1968
Overlapping	IN 1965-1969 IN 1963-1967	Take the union	-2 1963 -1 1969

Summary of date phrase coordination

Figure 17

B) Journal phrases

The determination of journal phrases is a more difficult task than that of date phrases, and requires that several assumptions be made. First, it is assumed that any user request for a publication relates to a periodical rather than to a single edition such as a book or specific paper. This seems to be a realistic assumption since a user who knows the specific publication he wants has no need of the

retrieval system at all. This section is therefore concerned only with extraction of journal names.

The determination of journal phrases consists of two parts. The first is a search for a journal phrase template match. The templates are shown in Appendix A and are for the most part of the form:

PUBLISHED IN
FOUND IN
NOT PUBLISHED IN
etc.

Clearly a match of one of these templates is no guarantee of a journal phrase. The actual decision is made using a journal name table look-up which constitutes the second part of the journal phrase analysis. The journal name table contains the names of all the journals in the collection and assigns a unique reference number to each. For each journal all anticipated forms of its name are included. For example, the "Journal of the Association for Computing Machinery" is listed as

JACM
ACM JOURNAL
JOURNAL of the ACM

in addition to the full name. The list also contains some journal family names which can specify a group of journals.

In this way an input which specifies

IN AN ACM PUBLICATION

is taken to mean both the ACM "Journal" and "Communications". Clearly in an operational system there will be some journal references which have not been anticipated. These can be added to the journal list periodically and hopefully the list would quickly stabilize. After a journal template is matched the words following the template are looked up in the journal table. If a match is found, the reference number of the specified journal is placed in a journal specification list along with an operator (3 for equals and -3 for not equals). This list may then be used by a search program. If the search of the journal table fails, the phrase is assumed not to be a journal reference. While this method will occasionally miss a proper journal reference, it does prevent searches of the collection keyed on journals which do not appear in the collection, thereby eliminating some wasted searches. Figure 18 includes some examples of journal phrases and their resultant entries in the journal specification table. For clarity the journal names rather than reference numbers are shown in the specification table.

The journal phrase analysis is a good example of a C2-C1 process (conjecture-certainty). The template match

Input	Resulting journal specification	
IN JACM	3	JACM
IN JACM	3	JACM
AND CACM	3	CACM
IN AN ACM	3	JACM
PUBLICATION	3	CACM
NOT IN JACM	-3	JACM

Journal phrases and specifications

Figure 18

provides an indication that a journal phrase may be present; and then a check of the environment against the journal name table determines the final decision.

C) Author phrases

At first the problem of determining author phrases looks similar to that of journal phrases. Template matches indicate possible author phrases and then an author table look-up can finalize the decision. However, upon further analysis it is determined that the author list is not feasible. In a typical collection dealing with a specific subject area, the number of journals included is fairly small. Furthermore both the discontinuation of present journals and the creation of new ones is quite infrequent. Thus a journal list is compact and varies little with time.

Author lists, however, have different properties. Few authors produce more than ten papers per year. A collection which grows by ten thousand documents each year can thus have thousands of authors. Also people are continually entering and leaving a given field. An author list would thus be too large and would require too much file maintenance in an information retrieval system. Several alternate schemes exist. One such method developed by Borkowsky [5] is successful in extracting personal names from newspaper text. The method centers on the recognition of capitalized words and on finding personal titles such as Mr., Miss, President, etc. This method, however, is not applicable to this particular study. User input is entered in a single type case because of the input devices used (keypunches and teletypes have only upper case). Thus recognition of capitalization is impossible. Also a user is unlikely to preface his author specification by a personal title simply because it represents more input work and adds nothing to the meaning of the input. Another method is therefore called for.

The method developed for author phrase determination makes use of the collection dictionary. This is the large dictionary used in a query and document text analysis and is not to be confused with the small dictionary used in template analysis. The method requires a slight modification

in the dictionary structure. The motivation and development of the author phrase extraction method is presented in the following paragraphs.

Two types of personal names are defined. The first type includes those which are used only as names. Some examples are Jones, Dattola, Borkowski, Fredrick, etc. These words are generally not found in the collection dictionary, and thus if they occur in the query, are classified as unknowns. If all names were of this type, author phrase determination would be relatively easy. An author name would simply consist of a string of unknown words following an author template match. Unfortunately a second type of personal name exists, including those names that can be used in capacities other than that of a name. These include Rose, Brown, Fox, Little, Smith, etc. These words may be found in the dictionary and thus would not necessarily be classified as unknown. Analysis of the ADI dictionary shows that very few words of this type are actually included in the dictionary. There are enough, however, to indicate that the performance of a name extractor which uses unknown words only would not be acceptable. The solution centers around tagging certain words in the dictionary. When documents are entered into the system each word must be looked up in a dictionary in order to compose the document vector.

There must also be a facility to perform a dictionary update at this time to ensure that the dictionary is adequate to handle the new material. The dictionary tagging process fits within this framework and thus imposes no new dictionary processing. The names of document authors are looked up in the same process in which the document words are looked up. If an author name is found in the dictionary, a dictionary update is required. During this update the specified dictionary entry is tagged as a possible author name. This can be done by considering the tag as an extra part of speech for the word or by the use of a special tag bit in the dictionary entry. If the dictionary does not contain the author name or if the name is already tagged, no update is required.

The algorithm for author phrase extraction uses both unknown and tagged words. When a match of an author template is found, the string of words immediately following the template is tested. A string of initials and unknown and tagged words is taken to be the author name. The string is terminated by the first known and untagged word. If the terminating word is a conjunction or a comma, a second test of the string following the conjunction is made to test for possible multiple authors or inverted ordering (i.e., Brown, T.). The great variety of possible ways used in specifying a name, as shown in Figure 19, presents a considerable problem in author

determination.

John Alan Jones
John A. Jones
J.A. Jones
John Jones
J. Jones
J. Alan Jones
Jones, John
Jones, J.A.

Example of name variations
Figure 19

The one invariant is the surname. For this reason the author extractor tries to isolate the last name. It uses the given names and initials for modifiers for the last name. Figure 20 shows some sample analyses performed by the author extractor.

Input	Output	
	Surname	Modifiers
John Jones	Jones	John
J. Jones	Jones	J
J.A. Jones	Jones	J A
J. Jones and S. Smith	Jones Smith	J S
Jones, J.	Jones	J
Jones, S. Smith and F. Brown	Jones Smith Brown	 S F

Sample author analyses
Figure 20

The templates for author phrases appear in Appendix A. In all except the last two, the author name is assumed to follow the tempalte. The last two deal with possessives of the form:

SMITH'S
JONES'

The template keyword is the apostrophe, and here the author name preceeds the template. As with the journal phrases, a match of an author template is only an indication of a possible phrase, not a guarantee. But unlike the journals, the environmental testing for an author phrase does not make the decision a certainty. This testing can reject or support the original conjecture but cannot guarantee it. Thus the author phrase extraction is an example of a C2-C2 (conjecture-conjecture) process.

The above method is chosen because a useful author name list would become excessively large and require continuous file maintenance. It may be argued that by adding tags to the dictionary structure and updating the dictionary whenever new author names are entered, no advantage is gained over the more straightforward author list method. This, however, is not the case. The tags themselves occupy a maximum of one bit per dictionary entry. This is quite small in comparison with the dictionary size. It may even be possible

to incorporate the tags into existing dictionary fields with no increase in size. Furthermore the number of words in the dictionary is not increased by the tagging process. Also, an analysis of the ADI dictionary shows that there are very few words which function as possible author names. Thus since the dictionary need be altered only when a new author name is found, dictionary updates are very infrequent. The proposed method thus provides a useful procedure for extraction of author names from natural language text and requires a minimum of extra storage space and file processing.

D) Experiments and results

The system performs the following operations; a query which may contain bibliographic information is entered. The query is analyzed and the date, journal, and author phrases are identified and removed, leaving only the subject part. Three types of errors can occur in this process. First, a phrase of one type may be improperly taken to be of another type; second, a bibliographic phrase may be missed; and third, a false drop may occur, that is, a set of words not denoting a bibliographic phrase may be taken as one. The following experiments are used as an initial test of the system. First, to check for false drops, a set of queries from the ADI collection is entered. These 35 queries contain no bibliographic information, and thus any

such phrases found by the system are clearly in error. The queries contain many words found in the template dictionary as well as words tagged as possible names (such as Field and Will). These result in matches of author and journal templates. However they were all rejected by the environmental tests. The final results indicate that no bibliographic phrases are found, and in every case the query which remains after analysis is identical to the original, thus indicating that the system is not prone to false drops.

The second experiment involves the use of the same ADI queries but modified to include bibliographic information. These queries were constructed by the author and by several others who knew nothing of the structure of the system. The informants were merely given some ADI queries and told to rewrite them incorporating bibliographic information, and to phrase them as if they were typing at a teletype console. 31 such queries were used, including a total of 63 bibliographic phrases in this set. The phrase extractor correctly located 60 of these phrases, and made no false drops. The three phrases misses are all journal phrases. The errors are caused because the system is unable to read the journal list backwards and thus cannot cope with journal templates which occur to the right of a journal name. Thus a journal

phrase such as

DATAMATION ARTICLES

is missed. This problem is corrected by slight modification in the structure of the journal name list, and will be implemented in a future edition of the system.

A second experiment using the modified ADI queries makes use of what remains after the bibliographic phrases are removed from a query for a normal SMART run. By comparing these results with those achieved using the unmodified queries, it can be determined how much, if any, the retrieval quality is hurt by the added information and processing. For several reasons this experiment was abandoned. First, the performance of the system is such that the processed modified queries are almost identical to the unmodified. And second, nearly all the words in the bibliographic phrases are either high frequency or unknown, and are therefore ignored in the construction of concept vectors. Thus the concept vectors resulting from the processed modified queries are identical to the unmodified queries. Obviously the retrieval results are the same in both cases, so that an actual retrieval test need not be performed. Thus although the results are based on a fairly small data set, they indicate that it is possible to extract bibliographic information from natural

language queries via template analysis with little or no loss in retrieval power of the query.

E) Conclusion

It may be argued that extraction of bibliographic information from natural language queries is not a useful facility in an actual information retrieval system. Many working systems such as RECON [12] successfully use a more structured input method which requires the various parts of the query to be specified individually. An example of such an input is shown in Figure 22. From the results obtained by the RECON study, only a small percentage of users object to the structured input.

```
AUTHOR: JONES
AUTHOR: SMITH
JOURNAL: JACM
SUBJECT: INFORMATION RETRIEVAL
DATE: AFTER 1965
```

Structured input format
Figure 22

This is probably due to the fact that a user generally structures his query in his head in much the same way as it is structured in the input format. Be that as it may, this study indicates that bibliographic information can be extracted from natural language text with a high degree of

accuracy. Of more importance is the fact that this study is a successful test of the general template analysis schemes and it indicates that template analysis has definite potential in natural language analysis.

Bibliography

1. M. Rubinoff, S. Bergman, W. Franks, E. Rubinoff, "Experimental Evaluation of Information Retrieval through a Teletypewriter", CACM, Vol. 11, No. 9, Sept. 1968.

2. M. Rubinoff, S. Bergman, H. Cautin, F. Rapp, "Easy English, A Language for Information Retrieval through a Remote Teletypewriter Console", CACM, Vol. 11, No. 10, Oct. 1968.

3. J. Weizenbaum, "ELIZA - A Computer Program for the Study of Natural Language Communications between Man and Machine", CACM, Vol. 9, No. 1, Jan. 1966.

4. J. Weizenbaum, "Contextual Understanding by Computers", CACM, Vol. 10, No. 8, Aug. 1967.

5. C. Borkowski, "An Experimental System for Automatic Identification of Personal Titles in Newspaper Text", American Documentation, Vol. 18, No. 3, July 1967.

6. B. Lampson, B. Dimsdale, "A Natural Language Information Retrieval System", Proceedings of the IEEE, Vol. 54, No. 12, Dec. 1966.

7. IBM Systems/360 Document Processing System; Applications Descriptions. IBM Document H20-0315-0, 1967.

8. M. Halpern, "Foundations of the Case for Natural Language Programming", IEEE Spectrum, Vol. 4, No.3, March 1967.

9. W. D. Mathews, "TIP Reference Manual", TIP-TM-104, Technical Information Program, The Libraries, Massachusetts Institute of Technology, Cambridge Mass., 1968.

10. E. Parker, "SPIRES User's Manual", Stanford Physics Information Retrieval System, Institute of Communication Research, Stanford University, Palo Alto, California.

11. R. Williamson, "A Prototype Document Retrieval System", unpublished.

12. D. Meister, D. J. Sullivan, "Evaluation of User Reaction to a Prototype On Line Information Retrieval System" (Appendix: "Recon User's Manual"), Report NASA-CR-918. Prepared by Bunker-Ramo Corp. Canagn Park, California.

13. E. Mendelson, Introduction to Mathematical Logic, D. Van Nostrand C., New York, 1964.

Templates

Appendix A

The templates shown below are in alphabetic form with the parameter part omitted. The underlined words are the keywords. The word "DATE" indicates the position of the date concept 1900. The order shown here is the order in which the templates are scanned in the analysis process.

I. Housekeeping templates

OTHER THAN

LATER THAN

II. Date phrase templates

A. Double date phrases

DP[1] UNTIL "DATE"

DP AND -THE -YEAR "DATE"

DP "DATE"

B. Negative double date phrases

NOT 1 DDP[2]

DDP EXCLUDED

C. Negative single date phrases

NOT 1 DP

DP EXCLUDED

---

1. DP is a nonterminal which replaces an analyzed single date phrase.

2. DDP is a nonterminal which replaces an analyzed double date phrase.

Appendix A (Cont'd)

D. Single date phrases

-WRITTEN IN -THE -YEAR "DATE"

-WRITTEN BEFORE -THE -YEAR "DATE"

-WRITTEN AFTER -THE -YEAR "DATE"

-WRITTEN BETWEEN -THE -YEAR "DATE"

E. Current year phrases

-WRITTEN IN 1 PRESENT YEAR

-WRITTEN BEFORE 1 PRESENT YEAR

III. Journal phrase templates

A. Multiple journals

JP[3] AND

B. Journal phrases

EXCEPT THOSE -PUBLISHED IN

-PUBLISHED NOT IN

NOT -PUBLISHED IN

WHICH ARE -NOT IN

-PUBLISHED IN

EXCEPT THOSE -PUBLISHED BY

-PUBLISHED NOT BY

NOT -PUBLISHED BY

WHICH ARE -NOT BY

-PUBLISHED BY

DP

DDP

---

3. JP is a nonterminal which replaces an analyzed journal phrase.

Appendix A (Cont'd)

IV. Author phrase templates

A. Author name follows template

-ARTICLES NOT -PUBLISHED BY -MR.

-ARTICLES -PUBLISHED NOT BY -MR.

ARTICLES -PUBLISHED BY MR.

-PUBLISHED BY MR.

B. Author name occurs inside template

WHAT HAS 10 WRITTEN

WHAT HAS 10 DP

WHAT HAS 10 DDP

WHAT HAS 10 JP

WHAT HAS

C. Author name precedes template

NOT 10 'S

'S -NOT

Template dictionary used for implementation in Section 3

Appendix B

Concept No.	Word
11	ABOUT
	AROUND
	DURING
	FROM
	IN
	OF
	WITH
12	BY
	BEFORE
	PRIOR
13	AFTER
	FOLLOWING
	LATER
14	BETWEEN
	SINCE
15	APPEAR
	APPEARED
	APPEARING
	AUTHORED
	DEVELOPED
	FOUND
15(cont'd)	PRESENT
	PRESENTED
	PRINTED
	PUBLICATION
	PUBLISHED
	SEEN
	WRITTEN
	WROTE
16	EXCLUDING
	EXCEPT
	NOT
	NOTHING
17	EXCLUDED
	EXCEPTED
18	TO
	TILL
	UNTIL
	- (dash)
19	AND
	BUT
	OR
	, (comma)

Appendix B(Cont'd)

Concept No.	Word
20	CURRENT
	PRESENT
	THIS
21	YEAR
	YEARS
25	WHAT
	WHICH
26	ARE
	BE
	BEEN
	IS
	WERE
27	ARTICLE
	ARTICLES
	BOOK
	BOOKS
	DATA
	INFORMATION
	PAPER
	PAPERS
	PRESENTATION
	PUBLICATION
	PUBLICATIONS
	REPORT
27(Cont'd)	REPORTS
	THESES
	THESIS
28	HAD
	HAS
	HAVE
	DID
	DO
	DOES
50	A
	AN
	THE
51	Any initial;
	A...Z or
	A. ... Z.
52	Personal
	titles;
	MR.
	MRS.
	PROF
	etc.
53	OTHER

Appendix B (Cont'd)

Concept No.	Word
54	THAN
55	' (apostrophe) 'S
1900	Any word of the form 19ij where i and j are digits

Nonterminal Concept No.	Phrase Type
100	Single date phrase
101	Double date phrase
110	Journal phrase
120	Author phrase